LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH | TRINITY COLLEGE LIBRARY DUBLIN
Ollscoil Átha Cliath | The University of Dublin

Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

COMPUTATIONAL APPROACHES TO THE DISPLAY AND MANIPULATION OF METABOLIC PATHWAYS: THEORY AND PRACTICE

by

James Joseph Connolly

A dissertation submitted to the University of Dublin in candidature for the degree of Doctor of Philosophy

School of Biochemistry and Immunology
Trinity College
Dublin 2
Ireland

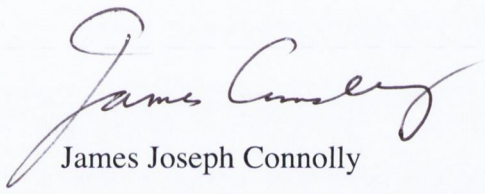
October 2008

TRINITY COLLEGE
27 JUL 2011
LIBRARY DUBLIN
THESIS
9253

Declaration

I certify that none of the work presented in this thesis has been submitted for any degree at this, or any other University, and that the work described in this thesis is entirely my own, except where otherwise stated.

I agree that the library may lend or copy this thesis upon request.



James Joseph Connolly

ACKNOWLEDGEMENTS

I am most grateful to Professor Keith Tipton for his expert supervision of the project. I thank him for his advice and guidance over the years and the many hours he spent correcting my thesis. I always found him in good humour. I am also very grateful to Dr. Andrew Mc Donald for his help in sorting out computer problems. The many discussions I had with both of them were interesting and inspirational. I wish to thank Michael Brady, Glenn Strong, Hugh Gibbons and Carol O'Sullivan of the Computer Science Department, for allowing me to attend their courses on computer programming. I thank Martha Motherway for looking after administrative affairs. I am grateful to Dr. Gavin Davey for the use of printing facilities.

CONTENTS

ABSTRACT 1

ABBREVIATIONS 3

CHAPTER 1 INTRODUCTION 7

§1.1 Preamble 8

§1.2 Defining metabolic pathways and networks 11

1.2.1 Textbook pathways 11

1.2.2 Metabolic networks 12

1.2.3 Pathways by synthesis 13

1.2.4 Elementary flux modes 14

1.2.5 Pathway modelling 17

§1.3 Regulation of metabolic pathways 18

1.3.1 Enzyme activity 18

1.3.2 Enzyme concentration 18

1.3.3 Hormonal control 19

1.3.4 Cellular compartmentation 19

§1.4 Data models for metabolic pathways 20

1.4.1 Stoichiometry matrices 20

1.4.2 Graphs 22

1.4.3 Bipartite graphs 26

1.4.4 Representations of graphs 27

1.4.5 Graph models 30

1.4.6 Bipartite graphs and stoichiometry 35

1.4.7 Graph layout 37

§1.5 Computational representation of pathways 38

1.5.1 Encoding and inference 38

1.5.2 Multiple occurrences 40

1.5.3 What to include 40

1.5.4 Avoiding confusion 41
1.5.5 Alternative view 41
§1.6 Computational display of pathways 42
1.6.1 Static drawings 42
1.6.2 Automated drawing 43
§1.7 Managing large networks 47
§1.8 Storage of pathway information 48
1.8.1 Databases 48
1.8.2 Spreadsheets 50
1.8.3 Metabolic databases 51
1.8.4 Searching metabolic databases 56
1.8.5 Computational limitations 59
§1.9 Aims 60

CHAPTER 2 SYSTEMS AND METHODS 63

§2.1 Systems 64
2.1.1 Computer system 64
2.1.2 Programming environment 64
2.1.3 Programming language 65
2.1.4 Information storage 65
§2.2 Methods 65
2.2.1 Mathematical methods 65
2.2.2 Java programs 66

CHAPTER 3 RESULTS: THEORY 68
§3.1 Definitions 69
§3.2 Graph models of biochemical events 72
3.2.1 Simple events 72
3.2.2 Complex events 73
3.2.3 Description of the representation 73
§3.3 Graphs of metabolic pathways 78

3.3.1 Glycolysis 78
3.3.2 Glycolysis and gluconeogenesis 80
§3.4 Simple object bipartite graphs and stoichiometry 82
§3.5 Non-enzyme operators 90
3.5.1 Transport across membranes 90
3.5.2 Receptor-ligand interactions 92
3.5.3 Spontaneous reactions 93
§3.6 Graph of a regulatory cascade 94
§3.7 Miscellaneous 97
3.7.1 Modulators 97
3.7.2 Extended dense operators 98
§3.8 Binary relations 99
§3.9 Conclusion 103

CHAPTER 4 RESULTS: PRACTICE 105
§4.1 Automated generation of biochemical pathways 106
4.1.1 General principles 106
4.1.2 The input format 107
4.1.3 Database 109
4.1.4 Software and system 112
§4.2 Implementation 124
4.2.1 General description 124
4.2.2 Classes and objects 124
4.2.3 Packages 125
4.2.4 Layout managers 125
§4.3 The RSD Java application 126
4.3.1 File input and output 126
4.3.2 Class TrivialApplication 128
4.3.3 Class DataFile 131
4.3.4 File Class FileView 132
4.3.5 File Class Reaction 132
4.3.6 File Class GuiSelection 133

4.3.7 StandardPathway and StandardSubnet classes 135
4.3.8 Class UserSearch 137
4.3.9 Class NewPathway 138
4.3.10 SubString classes 139
4.3.11 Class ClassicalPathway 140
4.3.12 ReactionFinder and ReactionExcluder classes 141
4.3.13 Class MessageaDialog 141
4.3.14 Class Analysis 142
4.3.15 Class PathwayInfo 143
4.3.16 Class Reference 143
4.3.17 Class ReactionString 144
4.3.18 Class ReactionParser 144
4.3.19 Class HTML 148
4.3.20 Class Relation 149
4.3.21 Drawing classes 151
4.3.22 Information flow 153
§4.4 Output from the application 158
4.4.1 Output from ReactionParser class 158
4.4.2 Output from the drawing classes 160
4.4.3 Additional output 172
§4.5 The WP applet 175
4.5.1 General description 175
4.5.2 Classes 175
4.5.3 Modifications to the original applet 175
4.5.4 Running the applet 177
4.5.5 Output from the WP applet 177
§4.6 Conclusion 181

CHAPTER 5 EXTENSIONS 183
§5.1 Independent applets 184
§5.2 Glycolysis pathway applet 184
5.2.1 General description 184
5.2.2 Data output 185

5.2.3 Classes 185
5.2.4 Class TrivialApplet 186
5.2.5 Class Information 186
5.2.6 Class InfoOption 186
5.2.7 Class InfoText 187
5.2.8 Class GraphPanel 187
5.2.9 Output from the glycolysis pathway applet 192
§5.3 Multi-pathway applet 200
5.3.1 General description 200
5.3.2 Data input 201
5.3.3 Classes 201
5.3.4 Class TrivialApplet 201
5.3.5 Class GraphPanel 202
5.3.6 Output from the multi-pathway applet 203
§5.4 Conclusion 209

CHAPTER 6 DISCUSSION 211
§6.1 Discussion 212
§6.2 Future developments 226

REFERENCES 231

APPENDICES 247
Appendix A1 248
General.dat 250
User.dat 292
Appendix A2 332
Glycolysis.dat 333
Gluconeogenesis.dat 334
PPP.dat 335
UreaCycle.dat 336
Trehalose.dat 337

Effectors.dat 338

Appendix A3 339

SpecRef.txt 340

GenRef.txt 378

Appendix A4 382

GlycolInfo.txt 383

PPPInfo.txt 383

GluconeoInfo.txt 384

UreaCycleInfo.txt 385

EC Pathways.txt 386

DataFileNames.txt 386

Appendix A5 387

Source code for the RSD Java application 387

Appendix A6 490

Source code for the WP applet 490

Appendix A7 505

Source code for the Glycolysis pathway applet 505

Appendix A8 553

Source code for the Multi-pathway applet 553

Appendix A9 571

HTML source for the Glycolysis pathway applet 571

Appendix A10 573

HTML source for the Multi-pathway applet 573

ABSTRACT

Since metabolic pathways constitute a complex system, a mathematical description, even a structural one, must be simplified. Yet, the model must be sufficiently general so that it will encompass the system that is to be represented. An approach is presented for representing metabolic reaction systems. It uses simple object bipartite graphs, with *event operators* (mediators of events) representing one species and *non-operators* (the non-mediators in events) the other. Edge colours describe relations between the two species and shared objects connect events together. The operators (e.g., enzymes) are integral components of the system interacting with the non-operators (e.g., substrates). This facilitates inclusion of receptors and transporters (operators), and effectors (non-operators) in the same model. A membrane operator separates two compartments and mediates the transport of compounds between them. Furthermore, enclosing a sequence of events between two membrane operators represents compartmentalisation of that sequence. *Extended operators* can include additional information such as the species and chromosome. Non-enzymatic reactions are included in the model by treating the activation energy as the operator. An operator in one event may be a non-operator in another, and *vice versa*. This is useful for addressing conceptual issues such as representing an enzyme as substrate (e.g., phosphorylation) or a metabolite (e.g., cyclic AMP) as mediator in a signal transduction pathway.

A one-to-one correspondence was established between the simple object bipartite graph and the stoichiometry matrix for a system of biochemical reactions. An element of the matrix is a property of the system represented as an edge label in the graph, and *vice versa*. The concept is generalised so that any edge property of a simple object bipartite graph of a metabolic reaction system can be expressed in matrix form. Binary relations of operators with their non-operators drop naturally out of the adjacency list representation of the simple object bipartite graph. Extending binary relations to higher order relations (n-ary relations), encodes additional information such as stoichiometry.

By comparison with other models, the present model was found most suitable for representing metabolic reaction systems because it (1) is unambiguous, (2) makes it possible to display complex reactions, (3) allows the use of graph algorithms and standard drawing techniques, (4) can represent reaction stoichiometry and other properties of

reactions as edge labels, (5) does not contain multiple edges, (6) does not contain artefacts, and (7) can model various biochemical processes.

The links in the bipartite structure allow the display of properties, such as stoichiometry, metabolite roles, etc. as edge labels, which is not possible with classical drawings. The representation of enzymes by vertices can transform the representation to a gene-centric one, through the substitution of enzymes for their associate gene(s). In addition, the model facilitates a metabolite- or enzyme-centric view of metabolism, which can show all reactions and/or all pathways using a particular metabolite or enzyme.

A Java-based system, consisting of an application and applets, was developed. User-friendly interfaces allow easy access to the system. Using the object bipartite graph model, the application produces pathway maps from stored files of reaction data. The Java code converts the input data in chemical notation to n-ary relations, which are used to produce the graphical output. Any sequence of reactions, including a pathway, can be searched for. Furthermore, pathways can be created from reactions in the files, lessening the need to store them. In addition, classical pathways, with or without modulators, can be accessed from pathway files, which can contain special features, such as membrane operators. The application is not restricted to named pathways, but can generate reactions in any combination from scrolling lists of codes, operators or reactions, or based on selected criteria for inclusion or exclusion using substring searches. An application of the substring searches allows the user to select any compound (enzyme or metabolite) and view all the reactions it is involved in. The system can generate and display useful information in the form of numeric data, colour coding and text. Reactions that are displayed in diagrams are also shown as reaction equations. Information on using it is available as part of the system. General and reaction specific references are also available.

The application generates a file in HTML format, which produces a display of a pathway in an applet for viewing and manipulation online. Links to external databases are available in the applet. Drop down lists and check boxes facilitate the visualisation of substantial amounts of information as required, and without clutter, in another applet designed for a single pathway. A multi-pathway applet allows the display of several built-in pathways and reactions. A user-defined option for creating a pathway makes it extendable, while an automated method for the selection of subcellular components greatly enhances it.

ABBREVIATIONS

ADP	adenosine diphosphate
ACA	adenylate cyclase (active)
ACI	adenylate cyclase (inactive)
Adr	adrenaline
AR	adrenergic receptor
AR-Adr	adrenergic receptor-adrenaline complex
ALA	δ-aminolaevulinic acid
ALAS	δ-aminolaevulinic acid synthase
ATP	adenosine triphosphate
AMP	adenosine monophosphate
BioPAX	Biological Pathways Exchange Language
BND	Biochemical Names Database
BPG	1,3-bisphosphoglycerate
C	currency metabolite
cAMP	cyclic adenosine monophosphate, or cyclic AMP
CAS	Chemical Abstracts Service
ChEBI	Chemical Entities of Biological Interest
CO	carbon monoxide
CO_2,	carbon dioxide
CoA	coenzyme A
CPU	central processing unit
DBMS	database management system
DHAP	dihydroxyacetone phosphate
DPPPK	dephspho-phosphorylase kinase
E4P	erythrose 4-phosphate
EC	Enzyme Commission
EMP	Enzymes and Metabolic Pathways
F	cofactor
F1,6BP	fructose 1,6-bisphosphate
F2,6BP	fructose 2,6-bisphosphate
FBPase	fructose bisphosphatase
F6P	fructose 6-phosphate

GPPa	phosphophorylase a
GPPb	glycogen phosphophorylas b
glc	glucose
G1P	glucose 1-phosphate
G3P	glyceraldehyde 3-phosphate
G6P	glucose 6-phosphate
G6Pase	glucose 6-phosphatase
G3PDH	glyceraldehyde-3-phosphate dehydrogenase
GSH	glutathione (reduced form)
GSSG	glutathione (oxidised form)
GDP	guanosine 5'-diphosphate
GTP	guanosine 5'-triphosphate
GUI	graphical user interface
H^+	hydrogen ion
HK	hexokinase
HCO_3^-	bicarbonate ion
HMB	hydroxymethylbilane
HMBS	hydroxymethylbilane synthase
H_2O	water
HTML	HyperText Mark-up Language
IDE	integrated development environment
InCHI	International Chemical Identifier
IUBMB	International Union of Biochemistry and Molecular Biology
IUPAC	International Union of Pure and Applied Chemistry
K^+	potassium cation
KEGG	Kyoto Encyclopaedia of Genes and Genomes
KGML	KEGG Markup Language
LHS	left hand side
Mg^{2+}	magnesium cation
MM	mitochondrial membrane
MPW	Metabolic Pathways Database
M_r	relative molecular mass
NAD^+	nicotinamide adenine dinucleotide
NADH	nicotinamide adenine dinucleotide (reduced form)
NADP	nicotinamide adenine dinucleotide phosphate

NADPH	nicotinamide adenine dinucleotide phosphate (reduced form)
NA	not available
NH_3	ammonia
NOP	non-operator
O_2	oxygen
OAA	oxaloacetate
OC	operator code
OP	operator
P	product
PBG	porphobilinogen
PBGS	porphobilinogen synthase
PC	pyruvate carboxylase
PEP	phosphoenolpyruvate
PEPCK	phosphoenolpyruvate carboxykinase
pf	Plasmodium falciparum
PFK-1	phosphofructokinase-1
2PG	2- Phosphoglycerate
3PG	3-Phosphoglycerate
PGDB	pathway/genome database
PGI	phosphoglucose isomerase
PGK	phosphglycerate kinase
PGM	phosphglycerate mutase
Pi	inorganic phosphate
PK	pyruvate kinase
PKI	protein kinase (inactive)
PKA	protein kinase (active)
PPi	pyrophosphate
PPPKA	phospho-phosphophorylase kinase
Pt	platinum
RHS	right hand side
R5P	ribose 5-phosphate
RSD	ReactionString Database
S	substrate
SBML	Systems Biology Markup Language
SCoA	succinyl-CoA

S7P	sedoheptulose 7-phosphate
TC	transporter classification
TD	thermodynamics
TCA	tricarboxylic acid
TPI	triose phosphate isomerase
TK	transketolase
TK1	transketolase1
TK2	transketolase2
UM-BBD	University of Minnesota Biocatalysis/Biodegradation Database
UPP III	uroporphyrinogen III
UPPS	uroporphyrinogen-III synthase
WIT	What Is There?
WP	WebPathway
Wt	weight
XML	Extensible Markup Language
Xu5P	xylulose 5-phosphate

CHAPTER 1

INTRODUCTION

§1.1 Preamble

"*A very important ingredient of mathematical creativity is the ability to formulate useful definitions, ones that will lead to interesting results*" (Fraleigh, 1982).

According to Lehninger et al. (1993), metabolism is the sum of all the chemical transformations that occur in a cell or organism. When dealing with metabolism, the notion of a metabolic pathway is often called upon. However, there is no agreed formal definition of this concept. Several definitions exist in the literature and it depends on the author's point of view how the term is defined. Traditionally, metabolic pathways have been defined on the basis of their progressive discovery. However, in the current post-genome era there is a paradigm shift to network-based definitions (for review see Papin et al., 2003).

A computer-aided approach to the display and manipulation of metabolic pathways requires a computational model for the generation and display of the pathways. Although definitions and models of metabolic pathways are not mutually exclusive, it is convenient to deal with them separately. Nevertheless, there is a need for both to be in mathematical and thus computer digestible form. Furthermore, it is important to use standard mathematical models to which well-established techniques can be applied.

The biochemical structure of many metabolic pathways is now well known. In most cases, all the chemical reaction steps and effectors are available from textbooks (Voet and Voet, 1995) and information systems (Schomberg et al., 2002). On this basis, structural models of metabolic networks can be constructed. However, the structural modelling of such networks poses certain challenges. For example, the division of compounds into enzymes and substrates is complicated by the fact that certain enzymes assume the roles of substrates when other enzymes are activating them in regulatory cascades, while the nucleotide cyclic adenosine monophosphate (AMP), which is not an enzyme, can activate some protein kinases. In these cascades, molecular interactions can be described in two different ways, namely, biochemical (e.g., phosphorylation) and logical (e.g., activation). Also, many proteins bind small molecules but do not catalyse a reaction of the ligand. Examples include the binding of hormones and drugs by specific receptor proteins. Furthermore, there is increasing evidence of moonlighting among biological molecules. For example, according to Tipton et al. (2003), many enzymes have alternative non-enzymic functions, while several proteins that were identified as having non-catalytic

functions were later found to have enzyme activities. Finally, transport steps carried out by transport proteins or pores that link intracellular compartments, and effectors that alter the rates of reactions, need to be incorporated in the model. Thus, trying to render each compound's role in a mathematical model is difficult. Yet, the model must be sufficiently general so that it will encompass the network that is to be represented. Furthermore, it is preferable if the model mirrors the biochemical structure of the network (Kohn and Letzkus, 1983).

Diagrammatic representations play an important role in biochemistry. They are used as visual aids in communication, learning and research. Metabolic pathway diagrams, as in biochemistry textbooks (Lehninger et al., 1993), in wall charts (Nicholson, 2000) and in some information systems such as the Kyoto Encyclopaedia of Genes and Genomes (Kanehisa et al., 2002), are produced *manually*. On the other hand, the *automated* generation of metabolic pathways offers new possibilities for their display and manipulation such as different views and different levels of detail (Karp and Paley, 1994b), and operations like adding or removing reactions (Goesmann et al., 2002). In either case, many authors aspire to achieve results similar in style to classical drawings as may be seen in Lehninger et al. (1993). However, classical drawings do not adequately represent the network of interacting molecules that forms the living cell, since they do not emphasise the structural aspect of the network or the interactive nature of its parts. For example, enzymes are not displayed as integral components of a system interacting with substrates, but rather as labels on links between metabolites. Thus classical drawings need to be modified to generate realistic displays of biochemical interactions in a living system that could lead to greater insight. However, it is desirable that the representation should still remain intuitive to biologists.

Proteins synthesised from genes may function as enzymes, thus directly relating structural properties of metabolic networks to features of the genome encoding them. Thus, it is important to be able to transform the representation to a gene-centric one, through the substitution of enzymes for their associate gene(s). Also, genome-scale metabolic networks can now be reconstructed (Ma and Zeng, 2003). Therefore, it is important to be able to display and manipulate them. Even though the results of methods for assigning genes to enzyme functions may not always be completely reliable, such reconstructions may be justified on the grounds that for organisms not having their biochemistry extensively studied, it may be useful or desirable to determine their metabolic networks

using annotated genome sequence data. This obviously precludes uncatalysed reactions and those for which no enzyme has been annotated in the corresponding genome. However, these metabolic networks are very large and entangled, and their representation becomes cluttered as a result of overlapping objects on the diagram. The generation and layout of genome-wide metabolic networks is difficult. Furthermore, the usefulness of such an undertaking is doubtful. Additionally, analysis of large metabolic networks has the problem of combinatorial explosion of pathways. Therefore, it is necessary to develop methods that can deal with these problems so that a user can comfortably explore large data sets of biochemical reactions. In particular, since biochemical knowledge is difficult to conceptualise, a researcher who has sequenced the genome of an organism may wish to visually explore its reaction system. The development of tools to deal with the display and manipulation of large-scale data sets of reactions and associated information is a continuing challenge.

It is important to include the regulation of reactions in a display of metabolic pathways (Michal, 1998). Pathways can be regulated by changes in either the activity or the amounts of the enzymes involved. The regulating compounds are often formed far away from the regulation site, as in feedback inhibition, where the end product of a pathway inhibits the first reaction. According to Michal (1998), if one wants to indicate the regulation by arrows between the regulating compound and the regulation site, it is necessary to resort to a special drawing in order to avoid many crossing lines. In a drawing emphasising the pathways he suggests naming the regulating compounds only at the regulation site. Therefore, it would be useful to develop computational methods to address these situations.

A mathematical model capable of structurally representing metabolic networks in a form that can be readily processed by computers, easily understood by humans, and that can realistically and consistently describe the network of interacting molecules, which constitutes the living cell, would be of value. The model should support the production and display of the networks from a collection of biochemical reactions stored in a computer. Furthermore, it should facilitate the development of methods to optimise the amount of information available without clutter, to handle and manipulate large-scale data sets of reactions, and to represent the control of metabolic pathways. Finally, it would be useful to develop computer programmes to implement these proposals, especially as applications suitable for individual use.

§1.2 Defining metabolic pathways and networks

"In the light of recent challenges in bioinformatics, it has turned out to be instrumental to use the pathway concept and phrase it in a formalised way" (Schuster et al., 2002).

1.2.1 Textbook pathways

The term metabolic pathway is frequently used in the 'classical' sense of a named pathway as given in a biochemistry textbook. A textbook pathway describes the sequence of reactions that constitutes the pathway and gives rise to its catabolic or anabolic function(s). For example, in *glycolysis*, glucose is degraded in ten sequential enzyme-catalysed reactions to two molecules of pyruvate, supplying energy in the form of adenosine triphosphate (ATP), and reducing equivalents in the form of nicotinamide adenine dinucleotide, reduced form (NADH) as described by Lehninger et al. (1993). Pyruvate is at a crossroads in metabolism and may be converted to lactate, alanine, acetyl coenzyme A (CoA) or oxaloacetate (OAA), while many reactions require second substrates that are converted from a high- to a low-energy form, as in the case of ATP and adenosine diphosphate (ADP), or from a reduced to an oxidised form, as in the case of NADH and nicotinamide adenine dinucleotide (NAD^+).

However, a named pathway may not be the same in all organisms. In some cases there are significant differences in the biosynthetic pathways for certain amino acids in bacteria, fungi, and plants. For example, biosynthesis of methionine from aspartate requires six steps in the yeast *Saccharomyces cerevisiae*, and seven steps in the bacterium *Escherichia coli*, with just four reactions in common, while in the same organisms, completely different pathways are used for lysine biosynthesis, with no reaction in common (see van Helden et al., 2001). In other cases, pathways that occur in some organisms do not occur in others. For example, most bacteria and plants can synthesise all 20 amino acids but mammals can synthesise only about half of them (see Lehninger et al., 1993). In several respects glycolysis in trypanosomes differs from its namesake in other eukaryotes. For example, in these parasites most of the reactions occur in a special organelle called a *glycosome* (Opperdoes and Borst, 1977). Even within the same organism there may be different versions of a named pathway. For example, in plants there is a sequence of glycolytic enzymes in plastids (e.g., chloroplasts), distinct and spatially separated from the glycolytic enzymes in the cytosol. Cytosolic glycolysis of plants may make use of several enzymes

other than the usual ones found in yeasts, muscle tissue and plant plastids (for review see Givan, 1999).

Thus, for a named pathway, it would seem necessary to indicate which species and sub cellular space is under consideration. Furthermore, the differences in the pathways suggest the need for accepted standard named pathways to which variations can be compared. In any case, many authors simply refer to textbook pathways as sequences of reactions organized to accomplish specific chemical goals. For example, "*Metabolic pathways are series of consecutive enzymatic reactions that produce specific products*" (Voet and Voet, 1995).

1.2.2 Metabolic networks

While individual named pathways are useful concepts that allow one to focus on a specific part of metabolism, all pathways are connected to other pathways (see Karp and Paley, 1994b). Furthermore, there is a paradigm shift away from individual reactions and pathways towards systems of interacting biochemical reactions. This is due to the availability of a rapidly increasing volume of molecular data on biochemical reactions and the more complete sets of data that have arisen with genome annotation efforts (see Papin et al., 2003). The notion of a metabolic network is important because an enhanced perspective of metabolism and cellular function can be obtained by considering a system of reactions in its entirety rather than reactions in isolation from one another.

A *metabolic network* may be defined as a regulated and highly connected system of enzyme-catalysed reactions and selective transport systems, in which the outputs of some reactions are inputs to others. An aspect specific to metabolic networks is reaction reversibility. Although, in principle, all chemical reactions are thermodynamically reversible, certain biochemical reactions, including those of most kinases and phosphatases, are essentially irreversible. In most cases the latter is achieved as the product is either removed from the cell or is rapidly converted further down an enzymatic chain (see Schmidt and Dandekar, 2002). In the case of reversible biochemical reactions, an enzyme will catalyse them from a higher to a lower concentration. Biochemical networks can be extracted from information systems such as the Kyoto Encyclopaedia of Genes and Genomes (Kanehisa et al., 2002) and MetaCyc (Karp et al., 2002).

Stephanopulous et al. (1998) define a metabolic pathway to be *"any sequence of feasible and observable biochemical reaction steps connecting a specified set of input and output metabolites"*. Thus, they define glycolysis as the sum total of all biochemical reactions by which glucose is converted to pyruvate. This can be accomplished by more than one route in a metabolic network.

Since cellular metabolism is considered more flexible than a set of predefined pathways, a number of different but mathematically related approaches, which do not require kinetic information, have been proposed for mapping biochemical reaction networks without preconceptions. These include pathways by synthesis and elementary flux modes.

1.2.3 Pathways by synthesis

Stoichiometry indicates the proportions of molecularities with which compounds participate in reactions (Heinrich and Schuster, 1996). Metabolic *pathway synthesis* deals with the construction of stoichiometrically consistent routes of enzyme-catalysed reactions that meet certain specifications (Stephanopoulos et al., 1998). Seressiotis and Bailey (1986) developed a computer programme, which systematically finds possible pathways that transform a given substrate to a selected product by successive addition of reactions from a database of enzymes and metabolites. Starting with all reactions that consume a specific substrate, they find all the products of these reactions, which in turn become substrates for other reactions. However, this approach leads to a combinatorial explosion of pathways. To address this problem, Seressiotis and Bailey (1988) introduced improvements to the procedure, which included a means for the elimination of redundant pathways. However, their programme involved the specification of only one required substrate and one required product and did not include allowed substrates and by-products. Furthermore, it did not guarantee that all possible pathways were synthesised.

Mavrovouniotis et al. (1990) presented an algorithm for the solution of the problem of synthesising biochemical pathways that satisfied linear stoichiometric constraints. These constraints designate biochemical reactions and metabolites as required, allowed or prohibited. The algorithm is claimed to be correct, complete and computationally tractable. Mittenthal (1996) also used this algorithm to assemble molecular networks in which reactions modify proteins. However, the initial classification of metabolites as

required (Seressiotis and Bailey, 1988; Mavrovouniotis et al., 1990), allowed or prohibited (Mavrovouniotis et al., 1990) is goal-oriented and thus narrows the scope of the approach.

The previous algorithms require explicit lists of compounds and reactions. However, the compounds and reactions in molecular networks other than metabolism may not be completely known. Furthermore, the algorithms only accommodate local constraints, i.e., restrictions on individual compounds or reactions. However, there may also be global constraints, such as minimising the number of reaction steps or the number of kinds of enzymes used. To deal with these, Mittenthal et al. (2001) presented a new method for assembling metabolic networks. Their algorithm uses generic reactions rather than a reaction list and generates networks with fewer reaction steps earlier.

Another limitation of these methods is their inability to produce chemicals that have not been identified in biological systems. To this end, a computational framework for the construction and evaluation of metabolic pathways given input substrates and knowledge of enzyme-catalysed reactions was developed (Li et al., 2004; Hatzimanikatis et al., 2005). Application of the framework creates new and existing routes to chemicals known to exist in, and novel to, biological systems. Molecules are represented as graphs (see Section 1.4.2), or equivalently, matrices (see Section 1.4.4). Each vertex of a graph corresponds to an atom and each edge corresponds to a bond between different atoms. The graphs are stored in a computer using matrices and enzyme-catalysed reactions are implemented via matrix addition. Pathways are generated automatically through repetitive application of a set of matrices (called reaction operators), representing the enzymatic reactions, to the reactants. Thermodynamic properties are calculated to assess the relative feasibility of novel pathways.

1.2.4 Elementary flux modes

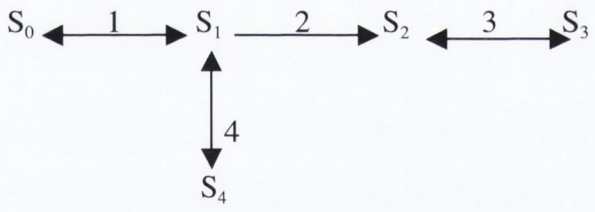
Elementary flux modes require the classification of metabolites as either external or internal to a system (i.e., substances with either constant or variable concentrations, respectively) and the definition of reactions as either reversible or irreversible (see Papin et al., 2003). It is assumed that the formation of each internal metabolite is exactly balanced by its consumption (steady-state assumption).

In this approach, the stoichiometry of a metabolic network is investigated to find out which direct routes from one external metabolite to another are possible. A flux mode **M** is a set of flux vectors that represent such direct routes through a metabolic network. Formally, it is defined as the set:

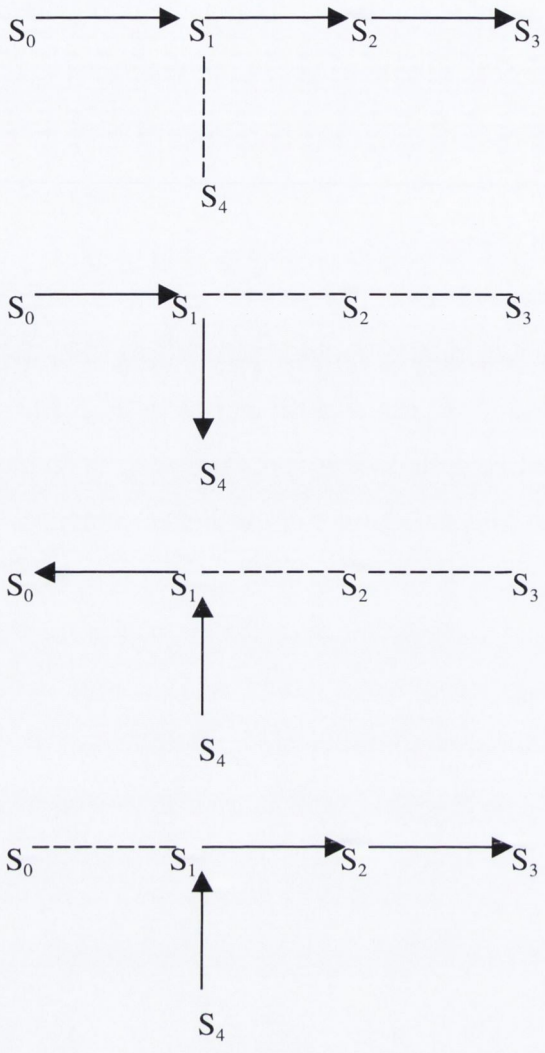
$$\mathbf{M} = \{\mathbf{V} \in R^r \mid \mathbf{V} = \lambda \mathbf{V}^*, \lambda > 0\}$$

where $\mathbf{V}^*$ is an r-dimensional vector (unequal to the null vector) fulfilling two conditions: (1) steady state and (2) sign restriction, i.e., the flux directions in $\mathbf{V}^*$ fulfil the prescribed irreversibility relations (Schuster and Hilgetag, 1994). A flux mode is an elementary flux mode if it uses a minimal set of reactions and cannot be further decomposed (see Klipp et al., 2005).

Consider, for example, the following simple network of four reactions, 1 through 4, and assume reaction 2 is irreversible in the direction indicated, while the net flux of the other reactions can go in any direction:



The elementary flux modes connect the external metabolites S_0 and S_3, S_0 and S_4, or S_4 and S_3. They are $(1\ 1\ 1\ 0)^T$, $(1\ 0\ 0\ 1)^T$, $(-1\ 0\ 0\ -1)^T$, and $(0\ 1\ 1\ -1)^T$, where T denotes the transpose. The transpose of a matrix is obtained by interchanging its rows and columns. In particular, the transposes of row matrices are columns. Solid arrows indicate the possible routes shown below:



Thus an elementary flux mode is a minimal set of enzymes that can operate at steady state with all irreversible reactions proceeding in the appropriate direction, and the enzymes weighted by the relative flux they need to carry for the mode to function (see Schuster et al., 1999, 2000). Any steady-state flux pattern can be expressed as a non-negative linear combination of these modes to form a unique set. Elementary flux modes represent a systematic and analytical definition of a biochemical pathway (Schuster et al., 2000). They can be calculated using an algorithm that is similar to the simplex algorithm. This algorithm was implemented in a computer program named Metatool (Pfeiffer et al., 1999). Since the number of elementary flux modes grows nonlinearly with network size, the calculation of these modes can be computationally difficult for large networks (see Klamt and Stelling, 2003). A new version 5.0 of Metatool (von Kamp and Schuster, 2006) is based on an algorithm proposed by Urbanczik and Wagner (2005), which empirically shows a higher performance. Recently, new algorithmic concepts that enable large-scale

computation of elementary flux modes constitute an important step toward genome-scale applications (Terzer and Stelling, 2008). Nevertheless, the number of network-based pathways, compared with that of traditional pathways, is regarded as a deterrent for their use as a conceptual framework in research, but their use as a tool for mathematically describing metabolic networks is considered an advantage (see Papin et al., 2003).

1.2.5 Pathway modelling

There are also definitions that are important in the context of metabolic pathway modelling. For example, the term pathway is used for a chain of enzyme-driven chemical reactions where the input of one reaction is the output of the previous reaction. Thus, according to Ochs et al. (1994) " *pathways are a conglomerate of connected enzymes, with the primary constraint that the product of one reaction is the substrate of the next*". They refer to the metabolites forming the backbone of substrate-product sequences as the *pathway substrates*, connected in series, and metabolites such as ATP and NAD as *pathway cofactors*, which connect pathways in parallel. Likewise, according to Karp and Paley (1994a), the nature of a pathway is that adjacent reactions operate on shared compounds that are the main substrates, which lie along the backbone of the pathway. The compounds that are not shared are called the side substrates. These definitions have been incorporated into computer programs by the authors to automatically generate metabolic pathway diagrams. However, they do not include biochemical regulatory cascades (see Section 1.3.3). In such cascades the product of one reaction serves as a catalyst for another reaction rather than the substrate. This type of definition both implies and requires the exact linking of compounds. This is very important in computer-aided generation of pathways.

Similarly, Sirava et al. (2002), define a metabolic pathway as " *a sequence of events or reactions that lead from one compound to another*". The analysis part in their algorithm generates biochemical pathways in the sense of this definition. Although the definition is said to comprise both metabolic and regulatory pathways, it does not give an explicit criterion for connecting the reactions.

§1.3 Regulation of metabolic pathways

" *Mechanisms that regulate the enzymes balance metabolic supply and demand*" (Campbell, 1996).

Metabolic regulation refers to the regulatory actions that operate at the level of the genes and enzymes involved in a pathway, thereby modulating the pathway output (see Deville et al., 2003). There is rapid regulation of enzymatic activity by activation or inhibition of enzymatic reactions through regulatory enzymes that respond to effector concentrations by increase or decrease in their reaction rates and slower regulation that changes the amount of enzyme protein.

1.3.1 Enzyme activity

Allosteric enzymes are capable of changing their catalytic activity in response to stimulatory or inhibitory modulators (effectors). An allosteric inhibitor binds to the enzyme at a site distinct from the substrate-binding site and influences conformational changes that may alter the binding characteristics of the enzyme for the substrate or the subsequent reaction characteristics, or both (see Palmer, 1995). Production of excess end product in a biosynthetic pathway may be prevented, and supplies of initial substrate conserved, by feedback inhibition, where the end product acts as an allosteric inhibitor of an early enzyme in the pathway. In contrast, steps subject to allosteric activation often occur late in a metabolic sequence. An example of allosteric activation is the effect of fructose 1,6-bisphosphate on pyruvate kinase, the last enzyme in glycolysis.

1.3.2 Enzyme concentration

Regulating the concentration of its enzyme in the cell may control the rate of a metabolic step. The concentration of an enzyme is the result of a balance between its rate of *synthesis* and its rate of *degradation*, both of which are subject to regulation on a time scale of minutes to hours. The former may be influenced by *induction* and *repression* and the presence of hormones. Enzymes are broken down more rapidly (by proteolytic enzymes) when they are the only source of energy available to the cell (e.g. in starvation) or when some change in function is taking place that requires the synthesis of different proteins, as for example, in germinating seeds (see Palmer, 1995).

1.3.3 Hormonal control

In higher organisms, *hormones* serve to coordinate the metabolic activities of different tissues, and their actions and effects are generally on a longer time scale than those of allosteric effectors (see Lehninger et al., 1993). This could involve changes in enzyme levels (i.e., 1.3.2 above), covalent modification (e.g., phosphorylation of enzyme proteins) or stimulation of the synthesis of allosteric regulators. Each time a hormone molecule such as adrenaline binds to its specific receptor in the plasma membrane of its target cell, it activates a number of stimulatory G proteins. This effect starts an amplification of the hormonal signal, which, in the case of adrenaline, leads to increased levels of glucose in the blood. The series of amplifying steps is called a *cascade* (see Campbell, 1995). In these regulatory cascades a protein at one level catalyses interconversion at the next level. Therefore, one level does not exchange matter with the next, and the levels affect the state of each other solely via regulatory effects (see Kahn and Westerhoff, 1991). The receptors for steroid hormones occur within the cell, where steroid-receptor complexes affect the transcription of specific proteins.

1.3.4 Cellular compartmentation

Catabolic and anabolic pathways that connect the same two endpoints (e.g., a fatty acid and acetate) may employ many of the same enzymes, but at least one of the steps is catalysed by different enzymes in the catabolic and anabolic directions, and these enzymes are the sites of separate regulation. Such paired pathways commonly occur in different cellular compartments. For example, fatty acid catabolism occurs in mitochondria, whereas the synthesis of fatty acids takes place in the cytosol. The concentrations of intermediates, enzymes and regulators can be maintained at different levels in different compartments, contributing to the separate regulation of catabolic and anabolic reaction sequences (see Lehninger et al., 1993).

§1.4 Data models for metabolic pathways

" *Mathematical models are the most precise representation of knowledge because they have a unique and objective interpretation"(*Wiechert, 2002).

The increasing volume of genomic and biochemical data available creates the need for computational methods to generate and display metabolic pathways and networks. To develop computer applications that aid in this task, a data model that can represent metabolic systems is needed. Furthermore, it is desirable that the representation should be intuitive. Metabolic reaction systems are often represented in a graphical way as in metabolic wall charts (Nicholson, 2000). Thus, it would seem useful to model such systems as graphs (see Section 1.4.2). However, according to Heinrich and Schuster (1998), a graph-theoretic approach would not be appropriate, since reactions other than monomolecular cannot simply be represented by a line connecting two vertices. Instead, they recommend an algebraic treatment using stoichiometry matrices (see Section 1.4.1), especially if the approach is to be computer-aided. Both of these approaches are reviewed in this section.

1.4.1 Stoichiometry matrices

The stoichiometry of a system of reactions can be represented in a matrix format (Heinrich and Schuster, 1996). A matrix is mathematically defined as a rectangular array of elements arranged in rows and columns. A *stoichiometry matrix* consists of one column for each reaction and one row for each metabolite. The entries in each column represent the stoichiometric coefficients of the substrates (negative numbers) and products (positive numbers) for each reaction. If a metabolite and a reaction are unrelated, the corresponding entry is zero. Reversible reactions can be treated as two separate irreversible reactions, in which case the reversible reactions are present as a pair of columns in the matrix, one the negative of the other (see Képès, 2007; Palsson, 2006). When several enzymes catalyse the same reaction, the latter is entered in the matrix multiple times. Conversely, some enzymes can catalyse different reactions, and this can also be accounted for in the matrix.

Thus the *i*th row shows the participation of a particular metabolite across all reactions and the *j*th column gives the stoichiometry of all metabolites in that reaction. For example, the system of reactions:

glucose (glc) + ATP → glucose 6-phosphate (G6P) + ADP (1)

G6P → fructose 6-phosphate (F6P) (2)

F6P + ATP → fructose 1,6-bisphosphate (F1,6BP) + ADP (3)

catalysed by the enzymes hexokinase (HK), phosphoglucose isomerase (PGI) and phosphofructokinase-1 (PFK), respectively, is represented in the following annotated stoichiometry matrix:

	HK	**PGI**	**PFK**
glucose	-1	0	0
G6P	1	-1	0
F6P	0	1	-1
F1,6BP	0	0	1
ATP	-1	0	-1
ADP	1	0	1

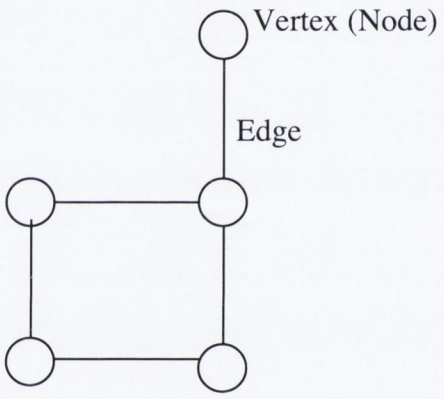
The value of 1 for G6P in the HK column indicates it is a product of reaction (1), -1 in the PGI column that is a substrate of reaction (2), and 0 in the PFK column that it is neither a substrate nor a product of reaction (3). Matrix elements also indicate relationships that are not one-to-one. For example, if two substrate molecules are used to form one product molecule, the loss in substrate is represented as -2.

Thus, the stoichiometry matrix can represent a metabolic network by recording the participation of the different metabolites and their transformations in the various reactions and by showing how the reactions are connected. Furthermore, the dynamics of metabolite concentrations can be expressed in terms of the stoichiometry matrix and a vector of reaction rates by the equation $d\mathbf{S}/dt = \mathbf{N}\mathbf{v}$, where $\mathbf{v}$ and $\mathbf{S}$ denote the vector of reaction rates and the vector of concentrations of internal metabolites, respectively, and $\mathbf{N}$ is the stoichiometry matrix (Heinrich and Schuster, 1996). However, when dealing with large matrices, it is difficult to visualise the pathway connectivity and the relationships between the chemical components. Furthermore, for a large metabolic network the matrix is sparse, i.e., it mostly comprises zero elements (see Palsson, 2006).

1.4.2 Graphs

A commonly used form of graphic representation is a vertex and edge graph. Vertices are objects that can have names and other properties. An edge is a connection between two vertices. The most important feature of a graph is the way in which the vertices are connected by its edges. For a lucid description of graph theory, see Aldous and Wilson (2000).

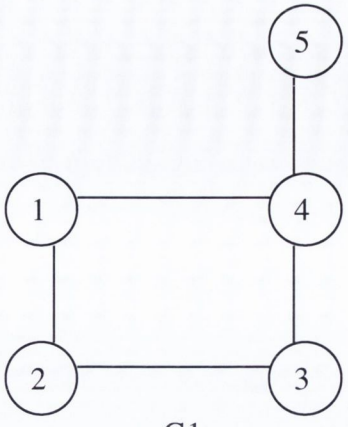
One can picture a graph by drawing geometric shapes for the vertices and lines (not necessarily straight) for the edges as follows:



Graph G

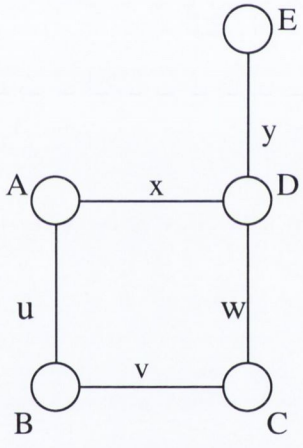
However, a graph is defined independently of the representation: a graph G consists of a finite nonempty set of objects called *vertices* (or *nodes*) and a (possibly empty) subset of unordered pairs of the vertices called *edges*.

Vertices are usually named by integers. An edge between vertices i and j is then represented by the unordered pair (i, j). Thus, in the graph G1 below, the edge connecting vertices 1 and 2 is (1, 2), and the graph can be described by the set of edges {(1, 2), (1, 4), (2, 3), (3, 4), (4, 5)}.



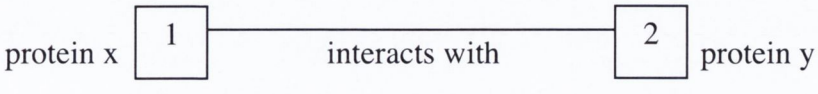
G1

Labels may be attached to vertices and edges. Any type can be used for labels:



G2

According to Aho and Ullman (1995), vertex names must be unique in a graph, but two or more vertices can have the same label. However, if the name of the vertex is not important, its label can represent it. A vertex may be drawn with both its label and its name. For example, the graph G3 below shows a vertex named 1 with the label " protein x", a vertex named 2 labelled "protein y", and an edge (1, 2) with the label " interacts with".

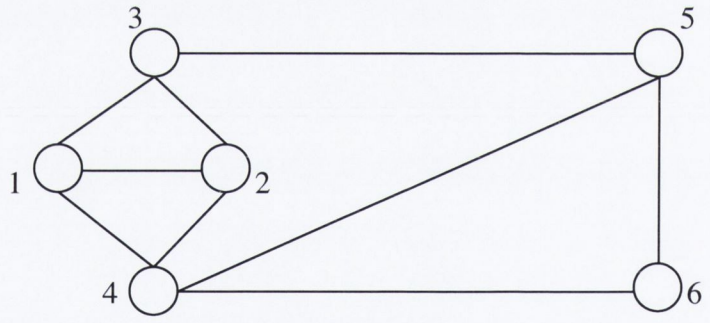


G3

An edge is said to be *incident* to the vertices to which it is attached. In graph G2 above, edge u is incident to A and B. Two vertices joined by an edge are said to be *adjacent:* A and B are adjacent, while A and C are not. The *neighbours* of B are its adjacent vertices. The *degree* of a vertex is the number of edges incident on it, or equivalently, the number of its neighbours. The degree of B, denoted by *deg*(B), is 2, while deg(D) is 3 and deg(E) is1 (see Grossman, 1995).

Two or more edges joining the same pair of vertices are called *multiple edges.* A graph with multiple edges is called a *multigraph*, while a graph with no multiple edges is called *simple* (see Asratian et al., 1998).

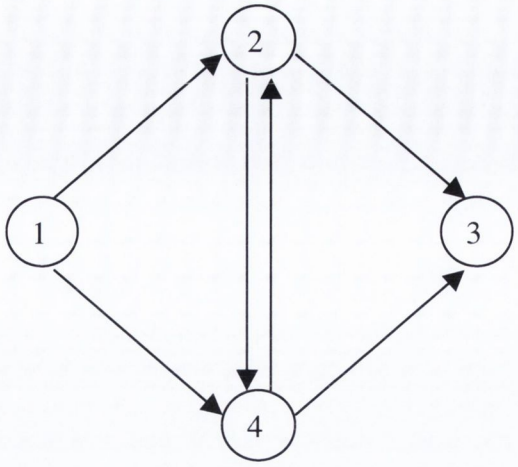
A *path* is a sequence of vertices such that each pair of successive vertices is joined by an edge and no vertex is passed through more than once. The *length* of a path is the number of edges in it. The set of edges {(1, 2), (2, 4), (4, 5), (5, 6)} in G4 below is a path from vertex 1 to vertex 6. A graph is *connected* if there is a path between every pair of vertices; otherwise, it is *disconnected.*



G4

A *cycle* is a path that starts and finishes at the same vertex. In G4 above, the set of edges {(1, 3), (3, 5), (5, 4), (4, 1)} is a cycle (see Barnett, 1995).

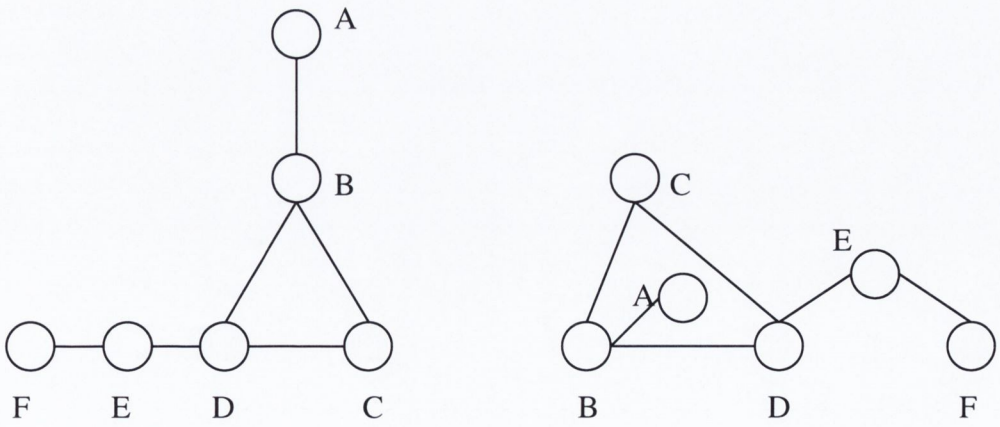
A *directed graph* consists of a set of vertices and a set of directed edges that join pairs of distinct vertices. Thus a directed edge is an ordered pair of vertices (i, j) from vertex i to j. An arrow from i to j representing each edge indicates this, as shown in G5 below.



G5

Vertices 2 and 4 have a directed edge in both directions between them, in which case it is usual to show a single undirected edge (see Luenberger, 1989). When i → j is an edge, i is called a *predecessor* of j, and j a *successor* of i. The *out-degree* of i is the number of edges incident from i and the *in-degree* of i is the number of edges incident to i (see Aho and Ullman, 1995).

Two graphs are identical if they have the same set of vertices and the same set of edges. Two graphs that are identical, but are perhaps drawn differently, are called *isomorphic*. This means that the two graphs have essentially the same structure (see Grossman, 1995). For example, the following two diagrams in G7 below represent the same graph.

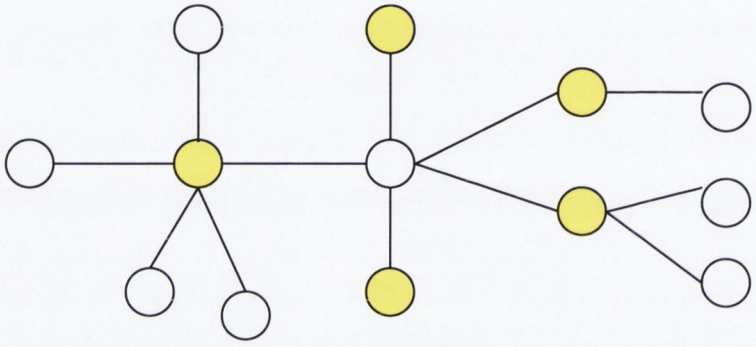


G7

1.4.3 Bipartite graphs

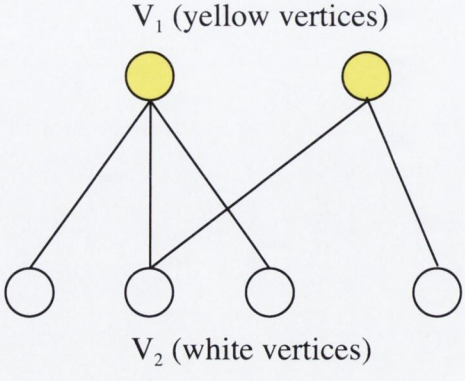
Of particular importance in applications are the bipartite graphs. Bipartite graphs have the property that their vertices can be divided into two parts such that no two vertices in the same part are joined by an edge. For a comprehensive treatment of this topic, see Asratian et al. (1998).

Formally, a graph is called *bipartite* if and only if the vertices can be coloured with two colours in such a way that no pair of vertices joined by an edge has the same colour. For example, the following graph B1 is a bipartite graph:



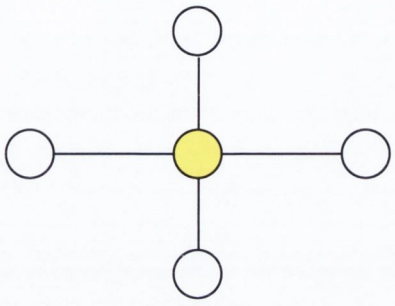
B1

In other words, a bipartite graph is a graph whose vertex set can be split into sets V_1 and V_2 in such a way that each edge of the graph joins a vertex in V_1 to a vertex in V_2. The sets V_1 and V_2 are called the *partite sets* of a graph. This is clearly shown in graph B2 below.



B2

There are several important classes of bipartite graphs (see Aldous and Wilson, 2000). These include complete bipartite graphs and trees. A *complete bipartite graph* is a bipartite graph in which each yellow vertex is joined to each white vertex by exactly one edge. The complete bipartite graph with m yellow vertices and n white vertices is denoted *by* $K_{m,n}$. A complete bipartite graph of the form $K_{1,n}$ is called a *star graph*, e.g., $K_{1,4}$ is shown in B3 below.



B3

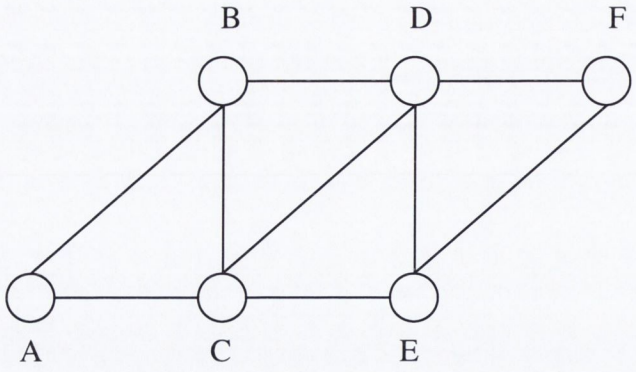
A *tree* is a connected graph with no cycles (e.g., B1 above). Every tree is a bipartite graph and every tree with n vertices has exactly n – 1 edges.

König (1916), obtained one of the most widely used characterisations of bipartite graphs i.e., a graph G is bipartite if and only if G has no cycle of odd length. Equivalently, in a bipartite graph every cycle has even length, and conversely, if every cycle in a graph has even length, then the graph is bipartite.

1.4.4 Representations of graphs

Although it is possible to store a digital image of a graph in a computer, it takes up a lot of space and cannot be manipulated. Alternatively, a graph may be completely determined by specifying its adjacency structure. From this information a visual representation can be constructed.

The *adjacency matrix* of a graph with vertices 1, 2, 3, …, n is an n x n matrix in which the entry in row i and column j is the number of edges joining the vertices i and j. In particular, if the graph is simple, then the entry in the *i*th row and the *j*th column will be 1 if i and j are adjacent, and 0 if they are not (see Grossman, 1995). For example, consider the following labelled simple graph G8:



G8

The adjacency matrix for graph G8 is:

	A	**B**	**C**	**D**	**E**	**F**
A	0	1	1	0	0	0
B	1	0	1	1	0	0
C	1	1	0	1	1	0
D	0	1	1	0	1	1
E	0	0	1	1	0	1
F	0	0	0	1	1	0

A simpler representation is to list each vertex in the vertex set along with those vertices that are adjacent to the listed vertex (see Tucker, 2002). This *adjacency list* contains all the structure of the graph, but has no extraneous information about non-adjacencies (like the zeros of the adjacency matrix). The adjacency list for the above graph is:

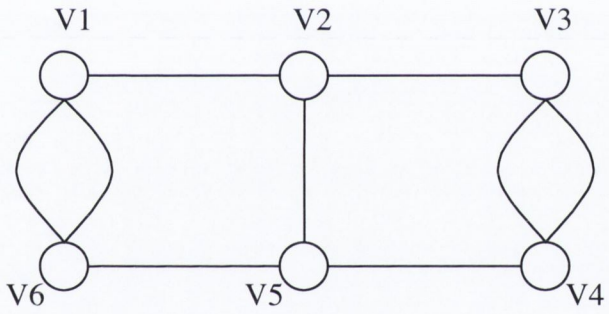
Vertex	**Adjacent Vertices**
A	B, C
B	A, C, D
C	A, B, D, E
D	B, C, E, F
E	C, D, F
F	D, E

If a bipartite graph is suitably labelled, its corresponding adjacency matrix can be transformed as follows:

0	**B**
$\mathbf{B}^T$	0

where **B** is an m x n sub-matrix, $\mathbf{B}^T$ is its transpose, and 0 is a matrix in which each entry is zero. Recall that the transpose of a matrix is obtained by interchanging its rows and columns. Conversely, a graph is bipartite if and only if it can be labelled so that its adjacency matrix has this form.

For example, consider the following multigraph G9:



G9

The adjacency matrix of G9 is:

	V1	**V2**	**V3**	**V4**	**V5**	**V6**
V1	0	1	0	0	0	2
V2	1	0	1	0	1	0
V3	0	1	0	2	0	0
V4	0	0	2	0	1	0
V5	0	1	0	1	0	1
V6	2	0	0	0	1	0

The transformed matrix of G9 is:

	V1	V3	V5	V2	V4	V6
V1	0	0	0	1	0	2
V3	0	0	0	1	2	0
V5	0	0	0	1	1	1
V2	1	1	1	0	0	0
V4	0	2	1	0	0	0
V6	2	0	1	0	0	0

According to Asratian et al. (1998), this transformation suggests a simpler way to represent bipartite graphs as matrices. The *reduced adjacency matrix* (or *biadjacency matrix*) of a bipartite graph is the sub-matrix of its adjacency matrix indexed by the rows of one partite set and the columns of the other. The reduced adjacency matrix of G9 is **B**, where

B =

	V2	V4	V6
V1	1	0	2
V3	1	2	0
V5	1	1	1

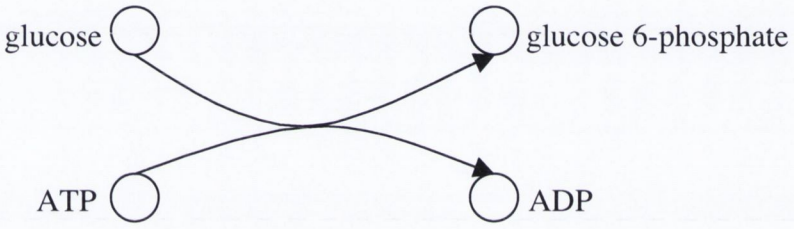
An element of the reduced m x n adjacency matrix **B** is the number of edges joining a vertex in one partite set to a vertex in the other. Thus, the reduced adjacency matrix records the edges of the bipartite graph. Furthermore, because there is no edge between any two vertices of the same partite set, the reduced adjacency matrix contains all the information about the graph G9.

1.4.5 Graph models

To build a graph model, one determines what the vertices and edges represent. The edges usually indicate a relationship between the vertices. Biochemical reactions have been modelled using graph theoretic formalisms. For example, in Wagner and Fell (2001), a substrate (or compound) graph is defined as having the metabolites as vertices and an edge connecting two metabolites if they take part in the same reaction. In reaction graphs the

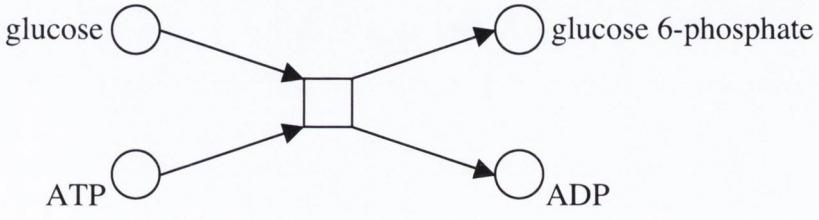
reactions are vertices, and there is an edge between two reactions if a compound is both a product of one reaction and a substrate of the other. However, substrate and reaction graphs are ambiguous and therefore unsuitable for the display of metabolic pathways (for review see Deville et al., 2003).

Since many metabolic reactions have more than one substrate and/or product, they violate the condition that an edge is a connection between two vertices. This was the problem encountered by Heinrich and Schuster (1998). However, hypergraphs, in which edges are allowed to connect more than two vertices, can be used to represent these types of reactions (Klamt and Stelling, 2003). For example, a single hyperedge relates the two substrates and the two products of reaction (1), in Section 1.4.1, as follows:



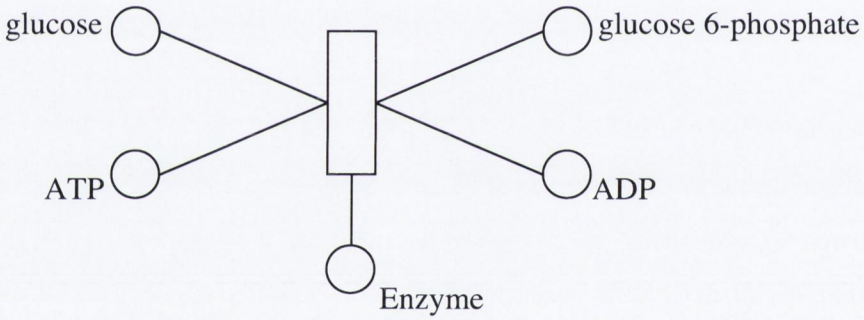
Hypergraphs have been used to model metabolic pathways, with vertices representing metabolites and hyperedges representing reactions or enzymes (Krishnamurthy et al., 2003). However, according to Benkö et al. (2003), a well-developed theory for the structural analysis of directed hypergraphs is not available, so that common methods for the analysis and visualisation of directed graphs are not immediately applicable to hypergraphs (Junker et al., 2006).

On the other hand, directed bipartite graphs, in which metabolites and reactions are vertices, and edges are connections between them, obviate the need of modelling reactions as hypergraphs. In the following directed bipartite graph, circles represent metabolite vertices and a rectangle represents the reaction vertex:



This representation has the advantage that graph algorithms and standard drawing techniques can be used. The arrow directed *from* a metabolite vertex to a reaction vertex indicates that the metabolite is a *substrate* for the reaction. Conversely, the arrow from a reaction vertex *to* a metabolite vertex indicates that the metabolite is a *product* of the reaction. Gagneur et al. (2003) adopted this formalism to deal with metabolic networks. They considered reaction vertices more generally as chemical processes and were thus able to include transport events. However, unless a reaction vertex explicitly represents a material entity (e.g., an enzyme), the model contains an artefact that may interfere with the analysis of metabolic networks and make their display non-intuitive.

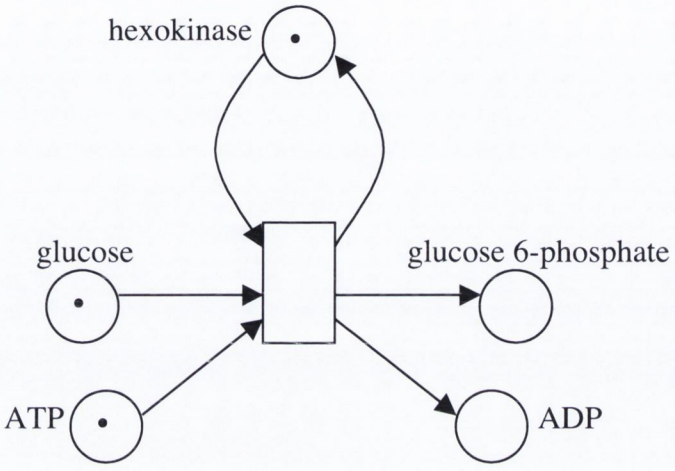
Although a vertex explicitly represents an enzyme in the bipartite graph used by Kazic et al. (2000), the model also contains a display artefact in the form of a reactive conjunction (rectangle) that has no counterpart in a biological system:



The enzyme is hexokinase. Again, the artefact may complicate analysis and make display non-intuitive. According to Benkö et al. (2003), the distinction of two classes of nodes with completely different interpretations may lead to artefacts in the data.

Directed bipartite graphs have also been used in Petri net representations of pathways (Reddy et al., 1996). A *Petri net* consists of transitions (rectangles) and places (circles), which are connected by arcs (arrows). A single arc with an arc-weight can represent the presence of multiple arcs between a place and a transition. Places may contain tokens, which are drawn as dots. A transition is enabled if the number of tokens in each input place is greater than or equal to the weight on the arc connecting that place to the transition. An enabled transition may fire by removing a token from each place connected with an input arc and adding a token to each place connected with an output arc. The numbers of tokens removed and added are determined by the arc-weights. Formal

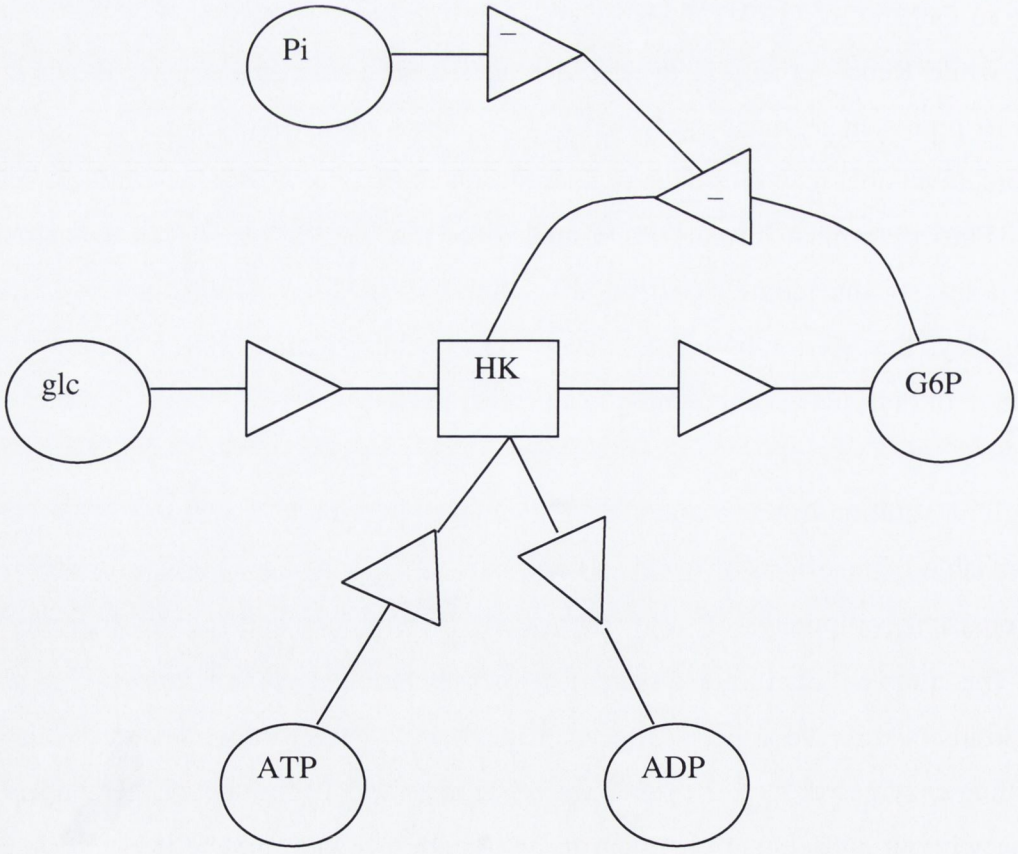
definitions of Petri nets and theoretical results about their structural properties can be found in Peterson (1981). For metabolic pathways, places represent compounds, including enzymes, a transition represents a reaction, and tokens indicate the presence of a compound in certain proportions:



According to Ochs et al. (1994), transporters share the essential properties of enzymes. However, while Reddy et al. (1996) use a place to represent an enzyme, they use a transition to represent a transport system. Thus, their representation is inconsistent. Furthermore, reversible reactions have to be represented by two separate transitions, since individual Petri nets must have a predefined direction. Additionally, the transitions constitute artefacts that could interfere with the analysis of metabolic networks and produce displays that are not intuitive. According to Schuster et al. (2002), the graphical representation of Petri nets is uncommon to a biochemist's eye.

Interestingly, a solution to the problem mentioned by Heinrich and Schuster (1998) was already available in the efforts of Kohn and Letzkus (1983), who used vertices of different types to represent compounds. Circles represented metabolites and squares represented enzymes. This allowed them to distinguish between metabolites and enzymes. They also used two kinds of directed edge. Substrate edges connected substrate vertices with the corresponding enzyme vertices. Regulator edges represented the binding of metabolites to an enzyme without subsequent chemical transformation of the metabolites. Thus, the biochemical meaning of the vertices was more intuitive.

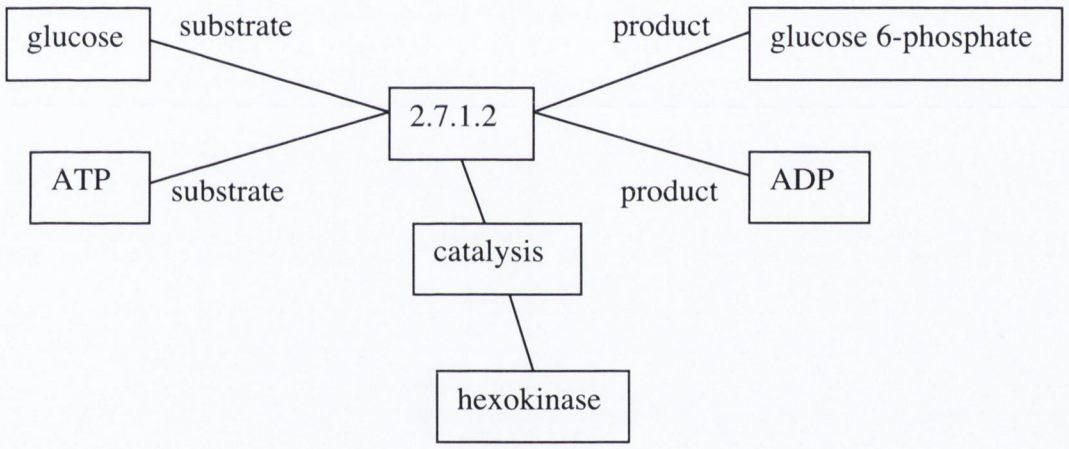
In modifications and extensions to this model (Kohn and Lemieux, 1991) two categories of vertices were defined. Chemical vertices represented compounds (as in Kohn and Letzkus, 1983) and relational vertices (drawn as triangles) represented the relationships among compounds. Relational vertices contained symbols to indicate the nature of the functional relationship between their respective predecessor and successor vertices. For example, if a relational vertex contains a minus sign, the corresponding chemical vertex is bound as an inhibitor. According to Kohn et al. (2002), a modifier always has a stoichiometry of zero. However, a zero in a stoichiometry matrix implies that the corresponding compound and reaction are unrelated, i.e., there is no connection between them in a metabolic network. Cellular compartments were indicated by the prefixes m- for mitochondrial and c- for cytosolic. The graphs were not bipartite, since some vertices of the same type (e.g., triangles) were adjacent, as shown in the following example (Kohn and Lemieux, 1991), which demonstrates how the binding of inorganic phosphate (Pi) to HK inhibits the binding of the inhibitor G6P:



Furthermore, the relational vertices, which have no counterparts in a biological system, significantly increase the size of the graph, thus adding considerable complexity to the reaction model, and hence to the display of a metabolic pathway.

According to Deville et al. (2003), object-oriented models, which are a generalisation of bipartite graphs where the vertices are typed, permit a more detailed description. An object-oriented model of a biochemical pathway requires the definition of the objects and their attributes, the design of a hierarchy between the different objects, and the definition of all the relations between the objects.

An object representation of the hexokinase reaction, based on Deville et al. (2003), would be as follows:



However, this representation has three vertices associated with the enzyme, hexokinase. These would make the pathway display complex and difficult to follow.

1.4.6 Bipartite graphs and stoichiometry

According to Kazic et al. (2000), a compound's stoichiometry in a reaction is a property of the edge connecting it with the relevant reactive conjunction. However, they do not justify this assertion. It is necessary to do this when writing for biochemists, since they may not be familiar with the relationship between bipartite graphs and rectangular matrices. On the other hand, Zeigarnik and Temkin (1994) demonstrated a one-to-one correspondence between the algebraic description (stoichiometric matrices) and the graph-theoretical

description (bipartite graphs) of chemical reaction mechanisms. In their bipartite graph of a mechanism, one type of vertex represented compounds while the other type represented reactions. Using columns for compounds and rows for reactions, the multiplicity of an edge e_{ij} equalled the stoichiometric coefficient of the *j*th compound in the *i*th reaction.

To illustrate this, they showed how the following two-route mechanism for the carbon monoxide (CO) oxidation on platinum (Pt) could be described:

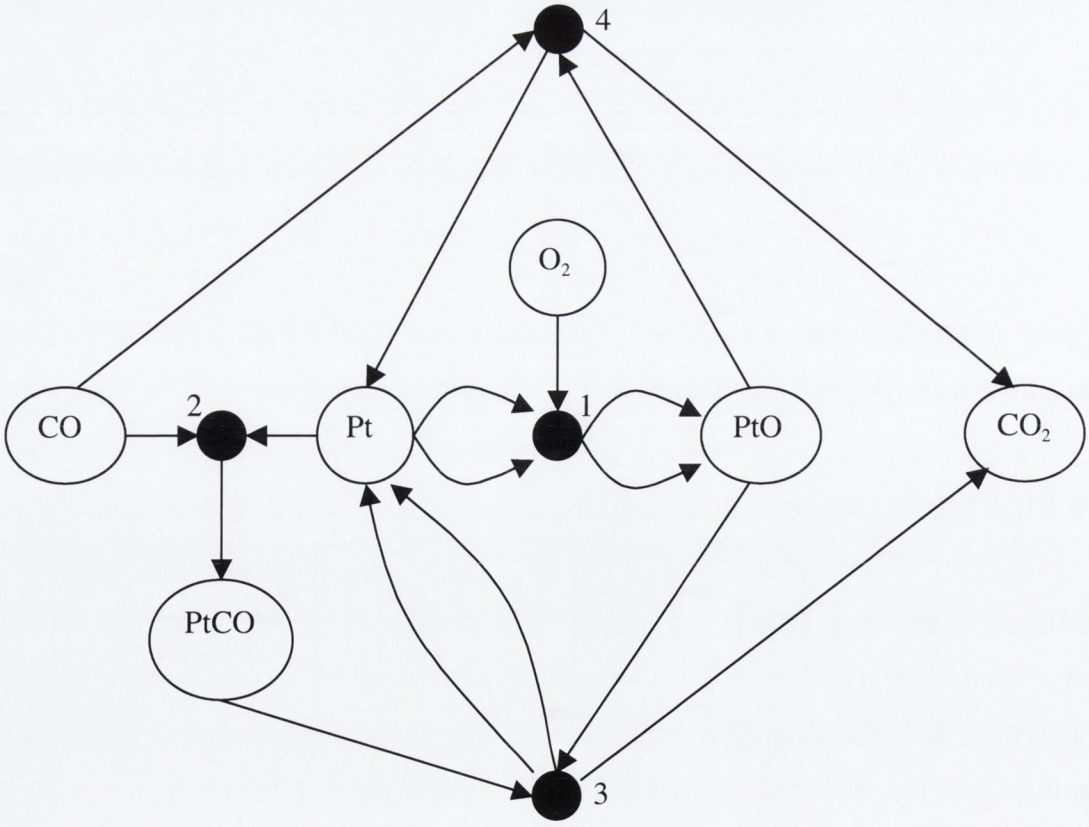
(1) 2Pt + oxygen (O_2) → 2PtO

(2) Pt + CO → PtCO

(3) PtO + PtCO → 2Pt + carbon dioxide (CO_2)

(4) PtO + CO → Pt + CO_2

The bipartite graph for this mechanism was described in the form of the following graph (with corrections to their original graph), where solid circles denote steps and open circles denote compounds:



For *directed* bipartite graphs the reduced adjacency matrix (see Section 1.4.4) is defined so as to account for the adjacency direction: the sign of an entry is *negative* if a vertex in one partite set is adjacent *to* a vertex in the other, and the sign of an entry is *positive* if a vertex in one partite set is adjacent *from* a vertex in the other (Temkin et al., 1994).

Since one partite set represents the compounds and the other partite set represents the reactions, and the absolute value of a reduced adjacency matrix entry equals the multiplicity of the respective edge, they thus arrived at a one-to-one correspondence between the directed bipartite multigraph and the stoichiometric matrix. However, multiple edges are computationally more difficult to deal with than single edges, and in the case of metabolic pathways, the reaction vertices would make the display non-intuitive.

1.4.7 Graph layout

A graph is a topological but not a geometric object. An algorithm is needed to produce graph layouts automatically, i.e., given a formal vertex and edge set representation of a graph, an algorithm assigns planar coordinates to the vertices and edges. The layouts should be easily viewed and aesthetically pleasing to the eye (see Di Battista et al., 1999). Relevant aesthetic criteria include: minimising the number of crossings of edges, the area of the drawing, and emphasising the main direction. In graph drawing, several major techniques have been elaborated, such as hierarchical (Sugiayama et al., 1981) and force-directed (Fruchterman and Reingold, 1991) layout for the placement of objects.

However, it is generally considered that the known graph drawing algorithms are inadequate for visualising biochemical reaction networks according to established conventions of biochemistry, although good results have been obtained by using labelling techniques, where the enzymes and side compounds are treated as edge labels and placed separately near the edges representing the reactions in which they take part (Karp and Paley, 1994b). Some solutions concentrate on the placement of the main substrates and products only (Becker and Rojas, 2001). Other solutions consider main and side compounds as nodes during the layout process but with different priorities (Rojdestvenski, 2003) or placed simultaneously (Wegner and Kummer, 2005). See Section 1.6.2 for further details.

The layout of biological network diagrams can be improved by including contextual information. For example, Cerebral (Barsky et al., 2007) uses subcellular localisation annotation to automatically position nodes in layers. For a discussion of layout algorithms used in various tools for visually exploring biological networks, see Suderman and Hallett (2007).

§1.5 Computational representation of pathways

"The automatic generation of drawings of metabolic pathways is a challenging problem that depends intimately on exactly what information has been recorded for each pathway, and on how that information is encoded" (Karp and Paley, 1994a).

1.5.1 Encoding and inference

Karp and Paley (1994a) represent a metabolic pathway as a graph whose vertices consist of enzymes, main compounds, and side compounds. The edges of the graph are straight lines that connect main compounds, and curved lines that connect groups of side compounds. They encode a pathway as a list of tuples, where each tuple is a pair consisting of a reaction, and the predecessor of that reaction in the pathway. For example, ((predecessor 2 1) (predecessor 3 2) (predecessor 4 3)) is the *predecessor list* for a pathway of four consecutive reactions. Each number in the list refers to a reaction, e.g., the first tuple encodes the fact that the predecessor of reaction 2 is reaction 1. From the predecessor-list form of a pathway stored in a database and the reaction equations for each reaction in the predecessor-list, they then compute the graph representation in which a direction is assigned to each reaction, main compounds are distinguished from side compounds, and the enzymes that catalyse each reaction in the pathway are known. This is a computationally complicated process consisting of rules, heuristics and assumptions. The algorithms for drawing the pathways are described in Karp and Paley, 1994b (see Section 1.6.2).

Compounds in the substrate-product overlap between two consecutive reactions are treated as mains; otherwise, they are regarded as sides. Two functions called *left* and *right* return the compounds on the left and right sides of a reaction, as it is stored in the database. Two other functions called *reactants* and *products* return the compounds on the left and right sides of a reaction given a putative direction assignment by an algorithm.

Some rules handle the case where the direction of a reaction can be immediately determined with respect to its predecessor because there is an overlap between either the left or the right substrates of the reaction (but not both), with some of the substrates of its predecessor. However, in some cases the direction of a given reaction is not determined solely by its predecessor, if, for example, there is overlap between products and substrates on both sides of both reactions. Additional rules backtrack to determine the direction of a neighbour of the predecessor. Once the direction of a neighbour of the predecessor is known, the direction of the predecessor itself can be inferred. Groups of compounds such as ATP and ADP, nicotinamide adenine dinucleotide phosphate (NADP) and nicotinamide adenine dinucleotide phosphate, reduced form (NADPH), and individual compounds such as P_i, CO_2, and water (H_2O) are removed from consideration as overlapping compounds between a pair of reactions, unless this would remove all overlap between two reactions.

If a reaction has no predecessors or no successors, they cannot rigorously determine mains and sides. However, two heuristics enabled them to make the correct determination most of the time. The first heuristic is to remove a set of inorganic molecules, such as bicarbonate ion (HCO_3^-), CO_2, P_i, H_2O, etc., from consideration as mains for a given side of a reaction if other alternative compounds exist. The second heuristic deals with pathways in which common cofactors (such as ATP) are mains. For example, when ATP is paired with ADP, they infer that it is serving as a phosphate donor and is a side, whereas without ADP, ATP must be contributing structurally to the product, so it is a main. However, the reaction ATP + AMP = 2 ADP, catalysed by the enzyme adenylate kinase (EC 2.7.4.3), could be problematic for this rule. To avoid having all sides and no mains, they choose to select as a main the compound most similar to the main on the other side of a reaction, using, for example, the number of carbons as a measure of similarity. This heuristic is applied only in the case when there would otherwise be no main. They considered it unusual for a compound to serve as both a main and a side in different reactions in the same pathway. Thus if they are left with more than one main compound after applying the previous heuristics, they check to see if any of them occur as sides elsewhere in the pathway. If so, they assume that the compound is a side in the reaction in question also.

1.5.2 Multiple occurrences

According to Sirava et al. (2002), representing ubiquitous compounds such as ATP and ADP by multiple occurrences makes a diagram more readable. Furthermore, according to Koyutürk et al. (2004) an enzyme may appear more than once in the same pathway implying that it takes part in the process at different times. Thus, more than one main vertex of a pathway graph could have the same enzyme label. One can preserve these temporal relationships or merge vertices in the graph with identical labels. Merging vertices with identical labels simplified their graph analysis problem. Schuster et al. (2000) distinguish between the two different reactions transketolaseI and transketolaseII in the pentose phosphate pathway, since they regard this as necessary in the case of bi- and multifunctional enzymes because, in principle, the different reactions might participate in different pathways. Alternatively, in the case of parallel reactions catalysed by a single enzyme, Kohn and Lemieux (1991) showed both sets of reactants and products connected to the same enzyme vertex but with their relational vertices (see Section 1.4.5) numbered appropriately to distinguish them.

1.5.3 What to include

Ochs et al. (1994) regard separation of pathway substrates from cofactors, reactions involving binding proteins, and the recognition of which species, cell type, and sub cellular space is under consideration at each step of a generated sequence of reactions as important for representing metabolism. Since a pathway need not be contained in a single tissue but can span several, there is a need to represent the boundary between what is in a system and what is not. Such pieces of information should be specified for each biochemical event, which represents a link in a complex network.

According to Michal (1998), it is helpful in many cases to supplement drawings with short texts and to use tables for the definition of components shown in the diagrams. He regards the chemical formulae of the reactants and the enzyme names as most important in demonstrating metabolism. A modulator may be an enzyme or an inhibitor, or it may represent effectors like temperature or pH. If a modulator has an inhibiting effect, a minus sign may be associated with the arrowhead that represents this modulation and a plus sign with activating influences (Voit, 2000).

Graphical visualisation of metabolic pathways should be at the same time information dense but not overloaded. A data collection representation and visualisation should satisfy a set of many, often conflicting, requirements: compromise between overview and detail, Internet-readiness, standard graphical user interface, and so on (see Rojdestvenski, 2003). A wide choice of visual attributes for the displayed objects (textures, colours, shapes, sizes, etc.) could be used for encoding different properties of objects. For example, different classes of reactions could be ascribed different colours. The use of visual attributes such as shape and colour enhances the information content of a graph. When using a 2-D representation, one can choose to limit the amount of information on the graph so that the picture fits in a single screen, or extend the amount of information presented, in which case an extensive use of scrolling is required, with the viewer loosing the perception of the pathway as a whole.

1.5.4 Avoiding confusion

In traditional diagrams, arrows are used with various meanings from transcriptional activation or inhibition to signal transduction via protein-protein interaction and transport. This may result in confusion, and researchers have to read parts of a text to understand what a diagram describes. In addition, it is often the case that both state transitions and relationships are described within one diagram, which makes the meaning of the vertices and edges confusing. One arrow may mean activation, but another arrow in the same diagram may mean transition of a state or translocation of materials. Without consistent and unambiguous rules for representation, not only is information lost, but also the wrong information could be disseminated (see Kitano, 2003).

1.5.5 Alternative view

In a standard view, a metabolic pathway is a graph with chemical compounds as vertices and reactions (enzymes) as edges. Ogata et al. (2000), take an alternative view in which a metabolic pathway is treated as a graph with enzymes as vertices and metabolites as edges. In this view, two adjacent vertices representing successive enzymes or reaction steps in the pathway are connected by at least one edge representing a specific metabolite that is a substrate of one reaction and a product of the other reaction. According to Home et al. (2004), the creation of such an enzyme-centric view of metabolism would provide an essential precursor to the extrapolation of physiological meaning from genomics data, by

allowing one to determine metabolic distances between enzymes and further transform the representation to a gene-centric one, through the substitution of enzymes for their associated gene(s).

§1.6 Computational display of pathways

"*Hand-crafted HTML is still the tool of choice for concise, well-formatted pathways of complicated metabolism*" (Ellis et al., 1998).

"A system of graphical representation should be powerful enough to express sufficient information in a clearly visible and unambiguous way and should be supported by software tools" (Kitano, 2003).

1.6.1 Static drawings

Manually produced drawings, as they appear in biochemical textbooks (Voet and Voet, 1995), on wall charts (Nicholson, 2000), or in electronic information systems like the University of Minnesota Biocatalysis/Biodegradation Database (Ellis et al., 1998) are considered to be well developed for the visualisation of metabolic pathways. However, several authors have referred to the limitations of these static drawings. For example, according to Sirava et al. (2002), static representations are often inconsistent with the more recent data present in the database itself. Furthermore, according to Schreiber (2002), static pathway diagrams cannot be updated with a reasonable amount of work and are often overloaded with information. According to Michal (1998), one frequently has to exclude a number of details in order to avoid such overloading. A mass of information may be given by different text colours and by the style and colour of arrows. Such visualisations are hard to read, especially for beginners, and it is difficult to find specific information such as all pathways between two substances. Also, there is no way to specify the amount of detail of each reaction to be displayed. For example, the user cannot choose between the use of enzyme names or enzyme classification numbers in the pathway diagram. Finally, static visualisation is restricted to only one level of representation providing either an overview or detailed information. However, different views are needed. Using static visualisation the user is not able to interactively change the depth of information in the diagram and

browsing through pathways from abstract overview diagrams to detailed pictures is not possible.

Most systems like KEGG (Kanehisa and Goto, 2000) employ static methods for the visualisation of metabolic data. This not only limits the usefulness of the systems, but also increases maintenance costs. Furthermore, these methods are unsuitable for interactive information systems because they cannot keep pace with the rapid increase in the knowledge of biochemical processes and cannot be manipulated in any way nor integrated in a programmatic environment. For reasons of economy of effort in redrawing the maps, automated drawings are preferable. Ellis et al. (1998) offer generated pathway maps as a supplement to, but not a replacement for, static HyperText Mark-up Language (HTML) pathway maps.

1.6.2 Automated drawing

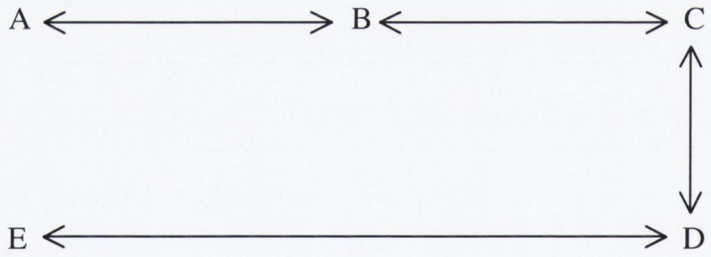
Karp and Paley (1994a) presented an inference procedure for the conversion of a predecessor list to a graph representation that serves as input to a pathway-drawing algorithm (see Section 1.5.1). This is the starting point for the algorithms described in Karp and Paley (1994b) for drawing metabolic pathways, which were subsequently developed for online biopathway visualisation tools (Paley and Karp, 1996). The authors use complicated graph-layout algorithms (see Karp and Paley, 1994b) to compute the positions of all the components of the pathway automatically. This frees them from entering position information manually and from re-entering position information when a pathway definition changes. They take the topology of the graph into account and decide which algorithm to use based on the graph being circular, branched, linear or complex. According to the authors, it is best to visualise graphs of different topologies using different layout algorithms. They divide the overall pathway-drawing problem into three steps:

1. Determination of the topology of the input pathway, as linear, cyclic or branched.
2. Application of the layout algorithm that is appropriate to the topology of the pathway to the main nodes of the pathway, thus assigning positions to them.
3. Assignment of positions to the side nodes and the enzyme nodes of the pathway.

Step 1 examines the connections among the main nodes of the pathway so that in step 2 it is known which layout algorithm to apply to the pathway. The topology tests employed are:

- *Circular:* A standard cycle-detection algorithm is applied to the pathway graph. If a single cycle is found, and all main nodes are members of the cycle, then the pathway topology is circular.
- *Branched:* If the pathway contains no cycles, and any main node has more than one incoming edge or more than one outgoing edge, or the pathway graph is not fully connected, then the pathway topology is branched.
- *Linear:* If the pathway contains no cycles, and it is not branched, then the pathway topology is linear.
- *Complex:* If the above cycle-detection algorithm finds more than one cycle, or one or more nodes are not members of the cycle, then the pathway topology is complex.

They developed layout algorithms for drawing *linear pathways* in three ways. The main nodes in the pathway can be drawn along a straight horizontal or vertical line, or a linear pathway can be drawn using a *snake layout* as follows:

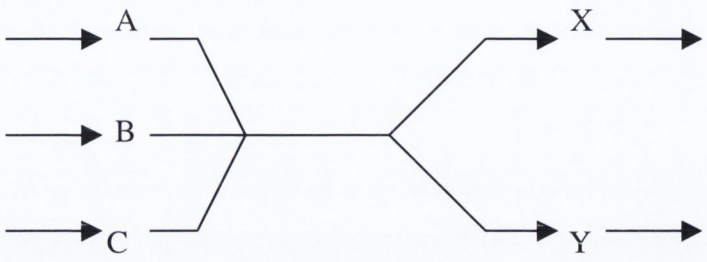


This layout maximises the usage of screen space. When a line is full, a node is placed on a new line centred under the last node on the previous line, but with the direction reversed. Their implementation separates pairs of main nodes on one line by the minimum space needed to hold the intervening side nodes and enzyme name(s).

They applied a tree layout algorithm from the Grasper-CL toolbox (Karp et al., 1994) to the layout of *branched pathways* and circular layout algorithms (from the same toolbox) to the layout of *cyclic pathways*, e.g., tricarboxylic acid (TCA) cycle (cyclic system of enzymatic reactions for the oxidation of acetyl residues to carbon dioxide, in which

formation of citrate is the first step). The layout for *complex pathways* is derived by partitioning the graph into sub-graphs, each of which can be laid out using one of the above algorithms, and using a hierarchical layout facility to place the sub-graphs relative to each other.

The authors also consider *complex junctions*, e.g., a single reaction in which multiple substrates and multiple products are all shared by other, separate reactions:



A user can choose different visual presentations of the same pathway. For example, main and side compounds can be drawn as names, structures, or both, or side compounds can be absent from the drawing. Enzyme names can be included or omitted, and connections to other pathways can be shown. In addition, they can display either single pathways or groups of connected pathways called super-pathways. The representation for super-pathways is an extension of the predecessor-list representation used for individual pathways. The predecessor list for a super-pathway can be constructed as the union of the predecessor lists of its component pathways, sometimes with adjustments. For example, one can form a super-pathway from glycolysis and the TCA cycle, but since neither of these pathways as commonly defined contains the reaction that converts pyruvate to acetyl-CoA, two items must be added to the predecessor list for this super-pathway.

Their system fulfils the special requirements and graphical conventions of classical drawings of metabolic pathways. Their approach assumes that the user is interested in the relationships among historically defined metabolic pathways, as opposed to the problem of finding new potential pathways in a large metabolic network defined by a database of biochemical reactions.

Several other methods that dynamically draw metabolic maps were developed (Becker and Rojas, 2001; Goesmann et al., 2002; Demir et al., 2002; Holford et al., 2005). Dynamic visualisation is the generation of a diagram on demand at the time the drawing is needed.

For example, Becker and Rojas (2001), presented an algorithm based on the work of Karp and Paley (1994b), supplementing it with a spring-embedder model (Eades, 1984), which is a force-directed layout algorithm. However, they concentrated on the placement of the nodes and edges representing main substrates and products only. The Java-based graph library Yfiles was used to create, manipulate and view the graph. In cases where a reaction had more than one main substrate or product, it was represented by a hypergraph. However, since the Yfiles library does not support hyperedges, inserting dummy nodes at the front and back of the edges that forked out to the multiple target and source nodes, respectively, simulated these. The sizes of the dummy nodes were set to zero. However, this did not always give optimal results because algorithms treat dummy nodes like normal nodes.

According to Rojdestvenski (2003), the topology of a traditional metabolic graph depends on the distinction between main and side compounds. However, it takes a lot of inference to determine whether substances are mains or sides. A problem arises when dealing with large-scale data sets, with many pathways to be drawn automatically. For this situation, the formulation of certain rules is recommended. If the system of rules is limited, it may lead to graph layouts that differ significantly from traditional ones. On the other hand, establishing a set of rules individually for each pathway destroys any advantage an automated pathway layout system might have, because the complexity of drawing pathways one by one manually is moved to the complexity of creating a unique set of drawing instructions for each graph. Before processing their data, they created a list of metabolites that would be cofactors in all the pathways of a given database, and during automatic conversion all of them were assigned as such. These included ATP, ADP, NADH, and NAD. However, this solution lacks generality. The compounds ATP and ADP are considered as sides in the majority of reactions in which they participate, but in ATP synthesis pathways, ATP is a product. This situation should be avoided, as it leads to individual cofactor lists for different groups of pathways (Rojdestvenski, 2003). A predefined manual assignment of sides and mains is unnecessary in a new algorithm for finding a valid connection between metabolites in a metabolic network (Rahman et al., 2005). It was designed to use chemical structure information between metabolites for identification of shortest paths.

Rojdestvenski (2003) used a modified spring embedding layout algorithm (Kamada and Kawai, 1989). The algorithm considers main and side compounds as nodes during the

layout process. Firstly, only the main compounds are placed. Secondly, the algorithm is started again with main and side compounds at the same time, but with frozen coordinates for the main compounds. The side compounds are not repeated in the map but occur once only. This leads to many edge crossings due to the high connectivity of the sides.

Wegner and Kummer (2005) developed a new layout algorithm based on that of Becker and Rojas (2001). It differs from the latter in being able to join and split nodes and to detect cycles of arbitrary size. It reduces the number of edge crossings and edge length. Like Becker and Rojas (2001), it uses dummy nodes and is independent of database information. Furthermore, it can include side compounds, but unlike Rojdestvenski (2003), sides and mains are placed simultaneously.

§1.7 Managing large networks

"Information visualisation of networks is a major challenge because of the elaborate topology, the large numbers of nodes, and the complexity of the tasks" (Card et al., 1999).

The approaches outlined in Section 1.6.2 are suitable for drawing a limited number of reactions. However, for extensive networks these representations quickly become too large. Nodes and edges are so numerous that the layout becomes cluttered and relationships are hidden. Nevertheless, tools for the visualisation and analysis of large molecular interaction networks are available. Cytoscape (Shannon et al, 2003) and VisANT (Hu et al., 2005) are well-known examples. For a comprehensive review of tools for visually exploring biological networks, see Suderman and Hallett (2007). See also Sorokin et al. (2006).

Luyf et al. (2002) developed an application representing reaction data in a genome-wide metabolic map. It provides a new way of drawing user-specified pathways that does not adhere to classical drawings or other representations. The reactions are stored in a database and sorted prior to map generation. This determines the order of ligands. The result is a 'metabolic map' with chains of successive reactions displayed in text format. However, according to Michal (1998), it is more appropriate to use graphical means for a general survey of reaction sequences. It is difficult to extract an overview from long lists of single reactions.

Another approach divides large metabolic networks into subsystems. Ravasz et al. (2002) proposed a hierarchical modularity for metabolic networks. According to this model, metabolic networks of organisms are organised into many small, highly connected modules that combine in a hierarchical manner into larger, less cohesive units. Methods have been developed for the decomposition of these networks into sub-networks based on the local connectivity of metabolites (Schuster et al., 2002), the global geometry of the networks (Holme et al., 2003), and the combined properties of global network structure and local reaction connectivity (Ma et al., 2004). Yoon et al. (2007) extended the use of graph-based models to include quantitative data on connection diversity (biochemical activity distribution) in addition to connectivity distribution in the characterisation of modular structures in these networks. Their results underscore the inhomogeneous distribution of both connectivity and connection strengths.

User control to limit the number of nodes displayed seems an essential part of an effective user interface. Nodes or edges could be restricted to only those connected to a given node or those whose attributes satisfy certain criteria. Major challenges exist for researchers who wish to make browsing of large networks convenient (Card et al., 1999).

§1.8 Storage of pathway information

"*The keystone to the entire bioinformatics effort is the existence of vast amounts of biological information in large electronic repositories around the world*" (Mullan, 2003).

1.8.1 Databases

A database is an organised collection of related information consisting of one or more files. In computer files, a *field* is a space for a specified item of information (such as an EC number). A set of related named fields is grouped as a *record*, and a *table* is a collection of records of the same type. A simple database consists of a single table (see Munnelly, 1999). In a relational database, there are separate tables containing related information. The tables are called relations. Each row of a table corresponds to a record and each column corresponds to an attribute of the record (Rolland, 1998). Where it is necessary to process data from different tables, one can join two or more of them together, and in so doing construct a new table (Rothwell, 1993). With most databases one can view and

manipulate information in two ways: in a data-sheet, or in forms. A *data-sheet* shows the information arranged in columns (one for each field) and rows (one for each record). A *form* presents selected information for one record (Munnelly, 1999).

However, relational databases have shortcomings, many of which are tackled by object-oriented systems (Rolland, 1998; Connolly and Begg, 1999). In a relational system, all data are held in relations composed of simple attributes. There are many types of information that are not conveniently represented in such a way. An example is a parts explosion where it is desired to represent an object (e.g., an aircraft) as being composed of parts and composite parts, which in turn are composed of other parts, and so on. The process of normalisation (i.e., ensuring that the data structures in a relational database are 'efficient') leads to the creation of relations that do not correspond to entities in the 'real world'. The fragmentation of a 'real world' entity into many relations, with a physical representation that reflects this structure, is inefficient, leading to many joins during query processing. The join is one of the most costly operations to perform. The data in a relational system are, in most respects, passive. Application programs are required in order that the data are given some sort of active meaning. In an object-oriented database, it is possible to capture the idea of how data 'behave' as well as its structure.

Computer-based database systems provide the means by which access to a set of data may be shared and integrated amongst a set of applications. The use of the data is not tied to or controlled by any single application. Instead, it is shared amongst different applications. This is achieved by removing the responsibility for creating and maintaining the stored data away from the individual applications and passing this to an underlying layer of software known as a 'database management system' (DBMS) (Rolland, 1998; Munnelly, 1999). However, a DBMS is expensive and complex, and access to data may be slow. Thus, it may be better to work with file-based data if the costs arising cannot be justified (see Orengo et al., 2003) or multiple-user access to data is not required (Elmasri and Navathe, 1994). Alternatively, MySQL (http://www.mysql.com) community edition is available as a free DBMS program. Online documentation is extensive but intended for a technology-oriented reader. Commercial object-oriented databases are a relatively recent innovation, and industry standards are still being developed. This can cause problems if moving data from one vendor's DBMS to another. Nevertheless, they can be very useful in managing complex data structures (see Ye, 2008).

1.8.2 Spreadsheets

To cope with very large datasets, various computing algorithms have been developed. However, the majority of these require specific bioinformatics training for potential users. Thus, if biologists become dependent on bioinformatics assistance, the flexibility of their research can be reduced. However, a spreadsheet can be employed for data storage, manipulation and basic analysis. Thus, while waiting for detailed bioinformatics analysis, they can evaluate major trends in the data and proceed accordingly (see Ye, 2008).

A spreadsheet program manipulates and presents data found in tabular form. Built-in formulae can be caused to act on any range of the worksheet allowing complex calculations to be carried out. The industry standard for spreadsheets is Microsoft Excel. Add-in programs can make Excel even more powerful in solving quantitative analysis problems (Render and Stair, 2000).

Although Excel does not offer all the features of a DBMS, one can perform basic database tasks such as finding individual records, adding and deleting records, editing existing records, and sorting records. More complex tasks are also possible, such as sorting records into a different order, or extracting records meeting a particular search criterion. However, the relational facilities of databases, e.g., multi-table queries and reports, are not possible in Excel (see Muir, 1997).

Another main element of Excel is a *macro*. This is a sequence of commands grouped together and run as a single command (see Diamond and Hanratty, 1997). A macro can automate sets of commands and complicated tasks, reduce the number of steps in a complex operation, and make complex formula bar entries more efficient, thus saving time and reducing error.

Sometimes a researcher wishes to answer a specific question that may not be addressed by any program. It is therefore an advantage to be able to program oneself or to have access to someone who can. However, a modern spreadsheet program may provide an alternative. Spreadsheets are versatile, widely available, easy to use, and include a variety of graphical and statistical procedures. Furthermore, they are relatively inexpensive and may already have been purchased for other purposes. Most, if not all, biologists are familiar with and frequently use Excel. For example, Ma and Zeng (2003) used a reaction

database in Excel format. Also, Dickerson et al. (2001) enter their node and link data on spreadsheets, and so are able to sort, share and post the data on the web.

1.8.3 Metabolic databases

Metabolic databases are a relatively new type of bioinformatics resource that can make biochemical information, including metabolic pathways, readily available via the Internet. Metabolic databases describe collections of enzymes, reactions and biochemical pathways. Some deal specifically with certain organisms while others are more general. The visualisations available with each database are either drawn manually or produced automatically by computer algorithms. For brief reviews, see Karp (1998) and Ellis (2001). For a more detailed review, see Wittig and De Beuckelaer (2001).

All newly discovered enzymes, when classified, are displayed on the *Enzyme Nomenclature* web site (http://www.chem.qmul.ac.uk/iubmb/enzyme/). The data that it contains are fundamental to many existing metabolic databases such as ENZYME, BioCyc, UM-BBD and KEGG (see below). Static diagrams of pathways in which certain enzymes participate are available. An Enzyme Commission (EC) number consisting of four elements separated by points represents each enzyme (e.g., 2.7.1.1 represents hexokinase). The first number shows to which of the six main classes the enzyme belongs, the second indicates the subclass, the third gives the sub-subclass, and the fourth is the serial number of the enzyme in its sub-subclass. The main classes from one to six are oxidoreductases, transferases, hydrolases, lyases, isomerases and ligases, respectively (for further details, see Webb, 1992).

The *ENZYME* database (http://www.expasy.ch/enzyme/) contains useful information relative to the nomenclature of enzymes. It is based on the recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (IUBMB), which is the main source of the data. Additional information is taken from the literature. The database contains the following data for each type of characterised enzyme for which an EC number has been provided: EC number, recommended name, alternative name(s), catalytic activity, and cofactor(s) (Bairoch, 2000). It also has access to Boehringer Mannheim's Biochemical Pathways on the ExPASy server. This is an electronic version of the Biochemical Pathways Wallchart, provided as a series of linked images. An electronic index allows localization of any metabolite or enzyme on the chart.

Each enzyme name in the chart is linked to its corresponding entry in ENZYME, and vice versa.

ExplorEnz – The Enzyme Database (http://www.enzyme-database.org) was developed at Trinity College Dublin in 2005 as a new way to access the data of the IUBMB Enzyme Nomenclature List (McDonald et al., 2007). The data, which are stored in a MySQL database, preserve the formatting of chemical names according to the International Union of Pure and Applied Chemistry (IUPAC) standards. Information on an enzyme is divided into the following fields: EC number, accepted name, reaction, glossary, synonyms, systematic name, comments, references and links to other databases. Diagrams of reactions or related metabolic pathways, which show metabolite structures, are provided in many cases. The database is the primary source of new EC numbers from which all other databases containing the Enzyme Nomenclature data can be updated.

The *EcoCyc* database (Karp et al., 1997, 1998) describes the genome and biochemical systems of *E.coli*. It contains annotations of *E.coli* genes, as well as the DNA sequence of each *E.coli* gene. EcoCyc describes all known pathways of *E.coli* small molecule metabolism. Each pathway and its component reactions are annotated in detail, with references to the biomedical literature. The database can be queried and explored using visualisation tools such as automatic layouts of metabolic pathways. *MetaCyc* (Karp et al., 2002) describes pathways, reactions, and enzymes of a variety of organisms. The pathways in MetaCyc were determined experimentally, and are labelled with the species in which they are known to occur based on literature references. These databases are accessible through BioCyc (http://BioCyc.org/). The BioCyc database collection is a set of 160 pathway/genome databases (PGDBs) for most eukaryotic and prokaryotic species whose genomes have been completely sequenced (Karp et al., 2005). Each PGDB in the BioCyc collection describes the genome and predicted metabolic network of a single organism, inferred from the MetaCyc database, which is a reference source on metabolic pathways from multiple organisms.

The *University of Minnesota Biocatalysis/Biodegradation Database* (UM-BBD) (http://umbbd.ahc.umn.edu) provides information on microbial catabolism and related biotransformations, primarily for environmental pollutants (Ellis et al., 1998, 2003). Compound information is taken from the scientific literature or Internet resources and verified. Reaction information is also based on the scientific literature. Contributions

from workers in the field are accepted. Each personally communicated pathway is reviewed before it is installed on the database. The database contains information on over 130 metabolic pathways. Pathway maps are available in text-format and in graphic format. One can click on compound or enzyme names for further information. The generated pathway view allows one to select any reaction within the UM-BBD and dynamically construct a pathway map from that point to intermediary metabolism and other compounds.

KEGG (http://www.genome.ad.jp/kegg/kegg.html) (The *Kyoto Encyclopaedia of Genes and Genomes*) is the primary database resource of the Japanese GenomeNet service. Its main objective is to link genomic information with higher-level functional knowledge. It contains three main databases: PATHWAY for the representation of pathways and complexes, GENES for information about genes and proteins, and LIGAND for knowledge on chemical compounds and reactions (Goto et al., 1996; Goto et al., 1998; Kanehisa et al., 2002). The KEGG databases are intended not only for searches and browsing but also for computation of network data (Ogata et al., 1998; Ogata et al., 2000). The PATHWAY database contains at least 200 manually drawn reference pathway diagrams. These represent all feasible metabolic pathways rather than a consensus view of known pathways. Not all the reactions depicted in such diagrams occur in all organisms. Matching enzyme genes in a gene table with enzymes on the reference pathway automatically generates organism-specific pathways, and following the connections of enzymes that are coloured green allows the organism-specific pathway to be visualised. The reference pathways were first derived from the Boehringer Mannheim metabolic pathways wall chart and from a collection of the Japanese Biochemical Society. They are continuously modified and updated according to other sources. The database describes the reactions within each pathway, and the compounds within each reaction.

The KEGG resource has been made more accessible to automated analysis (Kanehisa et al, 2004). For example, the XML (Extensible Markup Language) representation of KEGG pathway diagrams is useful for automatic drawing of potential networks identified by two-hybrid experiments as an extension to known networks. Also, two new database components have been released: KO for ortholog grouping and hierarchical classification of genes, and GLYCAN for carbohydrate structures. Recently, Klukas and Schreiber (2007) presented a new approach for dynamic visualisation and interactive navigation especially tailored to KEGG pathway diagrams. These diagrams are given as KEGG

Markup Language (KGML) files. Their methods use a combination of semi-static and dynamic visualisation. KGML-ED, a program implementing the methods, allows the dynamic exploration and editing of KEGG pathway diagrams. Pathway files are loaded and transformed into a graph network, which may be modified to fulfil user-specific needs, such as deleting or adding network elements and changing labels and colours.

Metabolic Pathways Database (PathDB) (http://www.ncgr.org/pathdb/) is a research tool for scientists interested in analyzing their experimental or computational data in the context of biological pathways and networks. In addition to data on metabolism, it contains signal cascade data, gene regulatory data, protein-protein interaction data, and protein-small molecule binding data. Predefined pathways are available, e.g., glycolysis. Not only can a user display pathways that are defined as such in the database, but also any set of steps (metabolic reactions, protein-protein interactions, protein-gene interactions) can be sent to a PathwayViewer. For example, all the cellular interactions in an organelle or all cellular interactions associated with DNA repair can be represented. In addition, custom pathways can be discovered based on queries made with a QueryTool. A user can merge a number of different networks together and view them as a single network. For example, one can view a number of the pre-defined pathways integrated and connected together, or expand them by selecting the neighbourhoods within a cellular compartment. Vertices that are expanded can be metabolites or proteins or ribosomes or other biochemical objects.

The *Enzymes and Metabolic Pathways* (EMP) database (http://www.empproject.com) is a comprehensive source of biochemical data covering all aspects of enzymology and metabolism. The database format has about 300 subject fields to encode the following categories: entry identification, bibliographic description, biological source, host, biochemical genetics, cell cultivation conditions, metabolism, enzyme and reaction, enzyme assay and purification, enzyme kinetics, enzyme regulation, enzyme modification, enzyme structure, equilibrium and thermodynamics, physical chemistry and spectral properties, immunochemistry, and common fields. The format allows different types of tables and stoichiometric matrices to encode metabolic pathways, reaction mechanisms, rate laws and a very wide spectrum of numeric data. The *Metabolic Pathways Database* (MPW) is a derivative of the larger EMP database and contains over 3,000 pathway diagrams.

The *WIT* (What Is There?) system (http://wit.mcs.anl.gov/WIT2/) supports the comparative analysis of sequenced genomes and generates metabolic reconstructions based on chromosomal sequences and metabolic modules from the EMP/MPW family of databases (Overbeek et al., 2000). A metabolic reconstruction is defined as a model of an organism's metabolism derived from sequence, biochemical, and phenotypic data. The system contains over 2900 diagrams of metabolic pathways from the Metabolic Pathway Database. Each pathway diagram includes a set of functional roles. The goal of producing a metabolic reconstruction is to identify which pathways are present in an organism and which genes implement the functional roles. A relational table representing the pathway collection includes two columns: a pathway identifier and a functional role. Additionally, for each organism, there is a list of pathways present in the organism and a table connecting genes to hypothesized functional roles.

The *Agora* (http://www.the-agora.org/) is a medium for sharing curatorial functions among independent databases, and includes the UM-BBD and *Moirai* as participants. Klotho is one of three components of the Moirai project. It is a biochemical compounds declarative database of structures of biological molecules, viewable as static images or in PDB format, and contains computational tools to generate and recognise the structures of molecules and groups of molecules. In turn, these tools interact with the other elements of Moirai that specify reaction information to provide a suite of searching tools.

BRENDA (http://www.Brenda.uni-koeln.de/) (BRaunschweig ENzyme DAtabase) is a comprehensive relational database containing functional and molecular information on enzymes, which has been extracted from the primary literature. The database includes biochemical information on classification and nomenclature, reaction and specificity, functional parameters, occurrence, enzyme structure, stability, disease, isolation and preparation, ligands, organism, application and engineering, literature references and links to other databases (Schomburg et al., 2002). Useful additions include a new EC-tree browser, a taxonomy-tree browser, a chemical substructure search engine for ligand structure, a controlled vocabulary, an ontology for some information fields and a thesaurus for ligand names (Schomburg et al., 2004).

Reactome (http://www.reactome.org/) is a relational database of human biopathways. However, it also includes many biochemical reactions from rat, mouse, fugu fish and zebra fish. The information in the database is provided by expert biologists and is peer-reviewed

prior to release on the Reactome website (Joshi-Tope et al., 2005). A reaction is the basic unit in Reactome. It is defined as any event that converts inputs to outputs, where these are physical entities such as small or large molecules, or complexes of these. This definition encompasses a broad range of reaction types, including translocation and polymerisation. Physical entities are cross-referenced to relevant external databases, such as UniProt (http://www.uniprot.org/) for proteins and ChEBI (http://www.ebi.ac.uk/chebi/) for small molecules. A reaction also includes information on the species, sub-cellular location and the experimental evidence in the form of literature citation(s). Other attributes of a reaction may include catalysis and regulation. Different states of a molecule are explicitly annotated. For example, extracellular and cytosolic glucose are represented as separate entities. A pathway is defined as any group of related reactions.

The top of the homepage shows an interactive reaction map, which is a graphical summary of all the reactions in the database. A user can navigate the database by clicking on the map, by selecting a topic of interest from a hierarchical table of contents, or via several structured searches. All data can be downloaded in a variety of standard formats, including BioPAX (Biological Pathways Exchange Language – http://www.biopax.org/) and SBML (Systems Biology Markup Language – http://www.sbml.org/).

1.8.4 Searching metabolic databases

In ENZYME, one may search for an enzyme by EC number (or partial EC number), by enzyme class, by description (official name) or alternative name(s), by cofactor or in comment lines. A partial EC number gets a list of all ENZYME entries whose EC numbers begin with the given pattern. Searching by chemical compound (or partial word(s)) returns a list of enzymes that contain the search string as a sub-string of the enzyme names.

In the University of Minnesota Biocatalysis/Biodegradation Database, one can search for (1) compounds by full or partial name or synonym, by CAS registry number, or by formula, (2) enzymes by full or partial name or alternative name, or by full or partial EC number, (3) reactions by full or partial primary name of the starting or ending product(s), (4) pathways by full or partial name.

In the KEGG database, the pathways can be searched for using an enzyme's EC number, a compound's number, or a gene name. For example, the compound number for glucose is C00031, and a search for that number in LIGAND returns a link to pathway maps for glucose metabolism. Following those links retrieves pathway maps containing names for compounds and EC numbers for enzymes, with the name for glucose coloured red. Clicking on a metabolite name in a pathway map is a convenient way of finding information on that metabolite and its shared pathways. The same can be done for an enzyme by clicking on the EC number box or for an intersecting pathway indicated in a round rectangle (see Rashidi and Buehler, 2000). Searching LIGAND for a compound by name returns a list of names containing that compound name as a sub-string with links to the corresponding pathway maps. Clicking on Metabolic pathways in the Pathway Map Ortholog Table returns links to pathway maps by name, and clicking on a specific name returns the reference pathway. The user can then choose an organism from a list, thereby instantiating the organism-specific pathway. Also, searching PATHWAY using DBGET for a pathway by name returns links to that pathway for specific organisms. The DBGET/LinkDB (http://www.genome.ad.jp/dbget/dbget.links.html) facility is a retrieval system for all GenomeNet databases.

In BioCyc, clicking a pathway name accesses the corresponding diagram. A pathway can be displayed at different levels of detail. An intermediate level includes enzymes, EC numbers, genes and the chemical structures of the main compounds. A lower level gives the same information without chemical structures. Within the graphical representation of the pathways, inhibitors and activators of reactions are represented. Placing the mouse over an object on the map gives information on it. For an enzyme, this includes name, species, synonyms and evidence (e.g., inferred from experiment) and for its EC number, the name of the enzyme and the reaction it catalyses. For a compound, one gets its name and synonyms, while for a gene the information given is its name, species, product and synonyms. References and a description of the pathway are given below the map.

In BRENDA, Enzymes can be searched by their EC numbers, their names or synonyms, or by the organisms, in which the enzyme reaction is detected. All other information can be searched individually or by combination searches, which can be performed organism specifically. Therefore, it is possible to find a specific enzyme for a specific organism or even for a specific organ or tissue. A search for ligands may be performed. Kinetic data for enzyme-ligand interaction can be searched. The ligands are stored as compound

names, SMILES (Weininger, 1988) strings and Molfiles. The two-dimensional chemical structures of these compounds can be displayed as images.

One can query The Agora using predefined queries. Queries about reactions, networks and paths are as follows:

- Find all reactions involving a particular molecule.
- Find the network of reactions centred on a particular molecule for N steps.
- Find all paths between two molecules within N steps.

Queries about classes or groups of reactions take the form of:

- Find all reactions in an EC class, subclass, or sub-subclass.
- Find all reactions between two EC numbers.

Queries about reactions involving particular molecules or reactions are:

- Find all reactions involving lists of co-reactants.
- Find all reactions in a nominal pathway or sub-reactions of a reaction.

One can also query a specific reaction, find the Klotho entry for a molecule and find all synonyms for a particular term.

The following are typical questions a biochemist might like metabolic database systems like those above to answer (assuming they have the necessary data):

- Where does a particular metabolite come from and go to?
- Is it the same in all species?
- Does it regulate any enzymes?
- Are any enzymes that are involved in its metabolism involved in any other pathways?

In general, one might want a database that gives all information on a compound and traces its activities in a particular type of cell. For example, the compound could be an enzyme, inhibitor, substrate, product, or cofactor. What actions does the compound have? For example, does it stimulate a receptor, and if so, what receptor? Does it catalyse one or more reactions? Does it inhibit? What consequences does its action have in cells? What are all the actions the compound has inside (or outside) a particular cell? In addition, one might want to know what effect a mutation would have on the activity (active site) of an enzyme and thus on a metabolic pathway containing the enzyme.

1.8.5 Computational limitations

The large volume of information on biochemical pathways can be displayed and manipulated only with the help of computers. However, this contribution of computers was initially not recognised by biologists. Hence in many cases, the data stored in a computer are not suitably structured for display and manipulation by computer programmes.

In general, metabolites have not been named consistently and there is difficulty in linking large-scale network representations to sequence data, according to Home et al. (2004). The MetaCyc database breaks a network down into pathways and this destroys the global connectivity of the network. In the ENZYME database a number of reactions are available only as natural language descriptions. Furthermore, some entries contain alternative substrates in the same equation (e.g., 'aldehyde or ketone'). Other equations contain indefinite articles, e.g., an alcohol, or case differences, e.g., alcohol and Alcohol. Finally, some compounds contain information as to the source of the metabolite. For example, oxidized adrenal ferredoxin, EC 1.14.15.4 (see Home et al., 2004).

According to Ma and Zeng (2003), there are mistakes in the KEGG LIGAND reaction database such as inconsistencies in compound names and mistakes in the reaction equations. They recall that one enzyme may catalyse different reactions and different enzymes may catalyse the same reaction. They give as examples the enzyme fatty-acid synthase (2.3.1.85), which catalyses 31 reactions in the fatty acid synthesis pathway, and the reaction: 3-Hydroxydecanoyl-[acyl-carrier protein] = trans-dec-2-enoyl-[acp] + H_2O, which is catalysed by five different enzymes (2.3.1.85, 2.3.1.86, 4.2.1.58, 4.2.1.60, 4.2.1.61). In most enzyme databases, only the main reaction is listed for each enzyme. Thus, according to the authors, a complete metabolic network could not be constructed from such enzyme databases.

Wittig and De Beuckelaer (2001) indicate, with reference to KEGG, that reactions catalysed by an enzyme with wide substrate specificity, such as alcohol dehydrogenase (EC 1.1.1.1), are often given as general reaction equations. This results in the loss of detailed substrate-product information. Sometimes a compound identity is the same for two different compounds (e.g., C00023 for Fe^{2+} and Fe^{3+}) or different compound identities represent the same compound (e.g., C02038 and C02156 for glycyl-peptide).

However, with respect to the above computational problems, the following sites can be valuable: ChEBI (http://www.ebi.ac.uk/chebi/), InCHI (http://www.iupac.org/inchi/) and BND (http://www.biocheminfo.org/). For metabolic reconstruction, it is important to correctly define chemical compounds involved in enzyme reactions. So, Chemical Entities of Biological Interest (ChEBI) is a useful, freely available dictionary of molecular entities focused on small chemical compounds (Degtyarenko et al., 2008). The IUPAC International Chemical Identifier (InCHI) is a non-proprietary identifier for chemical substances that can be used in printed and electronic data sources, thus enabling easier linking of diverse data compilations. This is probably the best because it has unique identifiers. The Biochemical Names Database (BND) is a curated, public domain database of terms and synonyms for small molecules of biochemical interest.

There is a need to provide biologists with the ability to view and analyse their own findings together with information from the public domain, since any one database may not serve their purpose. The user would supply the information and therefore it would not be limiting, as for example in MetaCyc (Karp et al., 2002), which has a microbial focus and only contains limited information for human and other mammalian organisms. Finally, independent applications are necessary if one is not using the built-in ones of other databases. These should be able to create traditional pathways such as glycolysis.

§1.9 Aims

"*Problems worthy of attack, prove their worth by hitting back*" (Hein, 1966).

The overall aim was the development of a procedure for the automatic production of metabolic pathway graphs from a collection of enzymes and reactions stored in a computer. It should be possible to draw and redraw the metabolic pathways not only in the 'classical' way, e.g., glycolysis, citric acid cycle, etc., but also to allow all reactions involving any particular component (e.g., enzyme or metabolite) to be displayed. An advantage of this is that the user will be able to examine all pathways and/or reactions that produce and consume a given compound, and/or asses the overall metabolic effect of an enzyme inhibitor or an inherited deficiency of an enzyme. Such a system must ultimately be Web-based and therefore efficiency and speed-of-operation is essential.

Automated layout of the graphs is not a goal of this work, but a longer-term objective. Layout imposes the programmer's view on the users. This may not be exactly the format that individual users require, as will be discussed later (see Sections 6.1 and 6.2). In the present case users must be allowed the freedom to decide how a pathway should be laid out for their own specific needs. Manual layout is a stepping-stone to the larger goal of automatic layout that can be tweaked by the user.

This approach necessitates:

1. The use of a mathematical model to structurally represent pathways in a form that can be readily processed by the computer. The model should support the automatic production of pathway graphs from the stored reactions. Furthermore, it should facilitate the development of methods to optimise the amount of information available without overloading pathway maps, and to handle and manipulate large-scale data sets of reactions.

   This involved a graph-theoretic approach or the use of an algebraic treatment using stoichiometry matrices. However, it should be possible to incorporate both of these approaches into an integrated, unified, object model, which would algebraically represent the structure of the reaction system and graphically display it, as will be described later.

2. Use of a database (flat file and/or Excel) to contain information on the pathways. The database should describe the reactions within each pathway, and the compounds within each reaction. The database may contain information on some of the following:
   - *Enzymes:* EC number, name, cell space, cofactor(s), substrate(s), product(s), activator(s), inhibitor(s).
   - *Metabolites:* Name, molecular weight, structure.
   - *Thermodynamic data:* $\Delta G^{0'}$.
   - *Comments:* Isoenzymes
   - *References:* Archival, modern.
   - *Links to other databases:* Internal, external.

It should be possible to interrogate the system to get a metabolic pathway or an enzyme on a pathway, to get a biochemical reaction by name, EC number or pathway, and to get information on the control of a pathway.

3. The development of the software. The Java programming language was used for this purpose. This involves:
   - Developing a format describing data on the pathways, the main events occurring in them, and their input to the software system.
   - Developing approaches for visualisation of the pathways from their formalised description.
   - Making possible selection of the data to be visualised depending on the target enzyme or metabolite, or other criteria.

   A user interface is important in exploiting information in the system and should be designed to satisfy the user's needs. In the present case, it should probably be both textual and graphic. The interface could search the system in response to mouse clicks, selections from lists, and textual input.

4. The automated connection of metabolites and/or enzymes in the form of graphs, and the representation of this material in readily accessible and attractive ways.

The problem can be divided into:

- Modelling.
- Storage.
- Software development.
- Manipulation – this is the core and also raises the problems of flexibility, redrawing and extensibility.
- Display.
- Provision of interactive access to the system via the Internet.

CHAPTER 2

SYSTEMS AND METHODS

§2.1 Systems

2.1.1 Computer system

The Apple PowerBook G4 1Ghz CTO was used for the project. It has the following specifications:

- 256K Level 2 and 1MB Level 3 Cache
- 512MB SDRAM Memory
- 60GB IDE Hard Drive
- Built in Gigabit (1000 BaseT Ethernet)
- DVD-ROM /CD-RW Combo Drive
- 15.2" TFT display
- ATI RADEON Mobility 9000 Video Card
- 64MB DDR Video Memory
- Firewire Port
- S-Video Output / VGA Output
- DVI output for connection to Apple DVI TFT Screens
- 2.45kg with 5 Hrs Battery life
- Airport Card Pre-installed
- Built in 56k V90 Modem

Mac OS X (Version 10.2.8) is the default operating system. Classic is an environment for using Mac OS 9 applications while the computer is running Mac OS X. If the special features of Mac OS X are not needed, the computer can be started up in Mac OS 9.

2.1.2 Programming environment

The CodeWarrior Professional programming package (release 4) from Metrowerks (www.metrowerks.com) was used throughout the project. This is an integrated development environment (IDE), i.e., all the pieces of a classic programming environment (text editor, compiler, linker, makefiles, debugger, etc.) are combined in a cohesive unit controlled by a project manager. Professional includes C, C++, Java, and Pascal compilers for Windows 95/98/NT on x86 and Mac OS on PowerPc and 68K. The software was used on Mac OS 9.2.

2.1.3 Programming language

The Java programming language was used because it is *platform-independent*, *object-oriented* and *Internet-ready*. A Java program consists of the definition of classes that contain data structures and *methods* associated with them. With Java, one can produce two main types of programs – *applets* and *applications*. Applets are programs that operate within a browser, and hence can appear in a web page. Applications are stand-alone programs that run on one's own computer.

Java 1.2 and higher became available for Macintosh with Mac OS X. However, some web browsers may still not be capable of running applets written with 1.2. Therefore, *in the spirit of true platform independence*, as Hammond (2002) puts it, the work described in this dissertation used Java 1.1. For information on object-oriented programming using Java 1.1, see Bishop, 1998 and Holmes, 1998.

2.1.4 Information storage

BBEdit Lite 6.1.2 for Mac OS X was used to contain the file-based reaction data that is used directly by the Java programs. BBEdit Lite is a product of Bare Bones Software (www.barebones.com).

§2.2 Methods

2.2.1 Mathematical methods

Simple object bipartite graphs, in which one type of vertex is assigned to *event operators* (mediators of events) and a second type to *non-operators* (the non-mediators in events), were found most suitable for representing and displaying metabolic reaction systems and associated information (e.g., stoichiometry). Edge colours describe relations between the two species and shared objects connect events together. The operators (e.g., enzymes) are integral components of the system interacting with the non-operators (e.g., substrates). Operators are not restricted to enzymes, but can also be receptors or transporters, or other biochemical objects. An operator in one event may be a non-operator in another, and *vice*

versa. This is useful for addressing conceptual issues such as representing an enzyme as substrate (e.g., phosphorylation) or a metabolite (e.g., cyclic AMP) as mediator in a signal transduction pathway.

A one-to-one correspondence was established between the simple object bipartite graph and the stoichiometry matrix for a system of biochemical reactions, as shown in Section 3.4. An element of the stoichiometry matrix is a property of the system represented as an edge label in the bipartite graph, and *vice versa*. The concept is generalised so that any edge property of a simple object bipartite graph of a metabolic reaction system can be expressed in matrix form. This leads to a unified concept of simple object bipartite graphs of metabolic reaction systems and *associated matrices*. The advantage of the matrix approach is its underlying mathematical representation in terms of matrix equations. On the other hand, well-established techniques from graph theory can be applied. An edge to node ratio (edge/node) was used as a measure of the interconnectivity of nodes in a pathway graph.

The *binary relation* (binary) of an operator with its non-operator, defined as a connected operator_non-operator pair, dropped naturally out of the *adjacency list* representation of the simple object bipartite graph of a reaction. This allowed biochemical reactions and pathways to be represented computationally as specific sets of binaries. The representation was extended to higher order relations (n-ary relations), which included stoichiometry. A simple frequency histogram of operator code classes was used to display the frequency of occurrence of each class in a pathway.

2.2.2 Java programs

A Java-based system for the display and manipulation of reaction systems was developed. One part consists of an application with an applet. The second part consists of two applets. The system was designed and implemented using an object-oriented approach. Using a simple object bipartite graph model, the application automatically generates pathway graphs from stored files of reaction information, which was downloaded from the *Enzyme Nomenclature* site (http://www.chem.qmul.ac.uk/iubmb/enzyme/). These are manually laid out as maps of varying complexity and style. A powerful *ReactionParser* class breaks down the reaction information into basic components and concatenates them into n-ary relations. These are sent to the *Relation* class, which decodes them and sends the resulting

data to the drawing classes for display in a window. The program generates graphical user interfaces on demand, which permit reactions and pathways to be selected in different ways. The interfaces search the system in response to mouse clicks, selections from lists, and textual input. Standard pathways may be generated and classical ones, with or without modulators, may be accessed from pathway files. Any sequence of reactions, including a pathway, may be searched for on submission of a file containing them. Pathways may be created from a list of enzymes (or EC numbers). The application is not restricted to named pathways, but can generate reactions in any combination from scrolling lists of codes, operators or reactions, or based on selected criteria for inclusion or exclusion using substring searches.

Reactions that are displayed in diagrams are also shown as reaction equations. Information, including stoichiometry and modulators of reactions, is displayed on the maps as required. Nodes and edges are coloured for ease of navigation. In addition, the system generates and displays information in the form of numeric data and text. Information on using it is available as part of the system. General and reaction specific references are also available. The application generates a file in HTML format, which produces a display of a pathway in an applet that has been designed for viewing and manipulation online. The applet has links to external databases. Finally, specialised applets dealing with detailed aspects of glycolysis and its regulation, and the display of selected pathways, including a regulatory cascade, were developed. In the former, the display of detail without clutter is made possible by the use of drop down lists and check boxes. In the latter, a "User defined" option makes it extendable, while an automated method for the selection of subcellular components greatly enhances it.

CHAPTER 3

RESULTS: THEORY

§3.1 Definitions

The following metabolic pathway components are distinguished: biochemical objects and biochemical events. They are defined as follows:

DEFINITION 3.1. A *biochemical object* is any entity that can exist within or as a biochemical system. Each object (e.g., metabolite or organelle) may have one or more *attributes* that describe it. For example, an enzyme has an EC number.

DEFINITION 3.2. A *biochemical event* is any controlled or uncontrolled process consisting of biochemical objects, which can occur in a biochemical system.

Two components of an event may be distinguished: reaction and regulatory action. A *reaction* is defined as an interaction between objects that leads to the generation of a new object (e.g., product), binding of objects, dissociation of objects, transport of an object, and object modifications (e.g., protein phosphorylation), etc. A *regulatory action* is defined as the effect of an object (e.g., modulator) on a particular reaction. For convenience, an event is described as follows:

$$\begin{array}{c} \\ n_1 X_1 + \ldots + n_i X_i \overset{C}{<S>} n_1 Y_1 + \ldots + n_j Y_j \\ n_1 A_1 + \ldots + n_i A_i <S> n_1 I_1 + \ldots + n_j I_j \end{array}$$

The first line encodes the reaction, while the second line encodes the modulators of that reaction. Each object plays a certain *role* in the event. If the X's are the substrates in the reaction, then the Y's are the products of the reaction, and vice versa where the reaction is reversible, while C is a catalyst for the reaction. The A's are the objects activating this reaction and the I's are the objects inhibiting it. The X's, Y's, A's, I's and C are all *participants* in the event. There will always be at least one X and one Y. However, there may be no A or no I. In this case, the left hand side or the right hand side, respectively, of the second line is left empty. If a reaction is not controlled or the representation of control is not required, the second line is omitted. The stoichiometry is represented by n ($n \geq 2$). The case $n = 1$ is not explicitly represented but is understood as such. The symbol <S> can be an equals sign (=) or an arrow ($\rightarrow$, $\leftrightarrow$ or $\leftarrow$) in a reaction, or a colon (:) in the case of the modulators. The function of the colon is to separate activators and inhibitors and to

distinguish them from metabolites in an equation. The concept of a biochemical event operator is introduced in the next definition.

DEFINITION 3.3. A *biochemical event operator* (or simply an *operator*) is a biochemical object that *is mediating* a biochemical event. The non-mediators in this event are called *non-operators*. An operator (OP) in one event may be a non-operator (NOP) in another event, and *vice versa.*

In mathematics, an operand is a quantity or entity on which an operation is applied. However, the term " operand" is not used here since, while a substrate can be considered as an operand in a reaction because an enzyme operates on it, a product in the same reaction cannot, because it is not operated on in that reaction. The converse applies for the reverse reaction. However, a non-operator can be a substrate or a product in the *same* reaction. It is important to note that a biochemical object is only an operator while it is in the act of mediating a particular biochemical event, i.e., an object is an operator only in the context of a specific event. Operators may be proteins (e.g., enzymes or receptors) or non-proteins (e.g., cyclic AMP). Definition 3.3 permits role switching between operators and non-operators.

DEFINITION 3.4. A *simple* biochemical event is a single input-output event, which is mediated by a single operator and has no modulators. An event that is not simple is said to be *complex*. For example, a simple reaction is a single substrate-product reaction catalysed by a single enzyme that is not modulated.

The data format used to describe an operator-mediated reaction is as follows:

Operator <tab> $n_1 X_1 + \ldots + n_i X_i$ <S> $n_1 Y_1 + \ldots + n_j Y_j$

The <tab> represents a tab delimiter and is used for convenience, as described later.

An operator may have an attribute associated with it:

Attribute <tab> Operator <tab> $n_1 X_1 + \ldots + n_i X_i$ <S> $n_1 Y_1 + \ldots + n_j Y_j$

For example, using this notation, the glucokinase reaction is represented as follows:

2.7.1.2 glucokinase glucose + ATP → ADP + glucose 6-phosphate

where the attribute is an EC number, <S> is →, and the stoichiometry is implicitly understood to be 1 for all the metabolites. An operator-mediated event is then represented as follows:

Attribute <tab> Operator <tab> $n_1 X_1 + \ldots + n_i X_i$ <S> $n_1 Y_1 + \ldots + n_j Y_j$
Attribute <tab> Operator <tab> $n_1 A_1 + \ldots + n_i A_i$ <S> $n_1 I_1 + \ldots + n_j I_j$

The following example illustrates this:

2.7.1.11 PFK F6P + ATP → ADP + F1,6BP
2.7.1.11 PFK AMP + F2,6BP : ATP + citrate

DEFINITION 3.5. A *metabolic pathway* is any defined or meaningful sequence of operator-mediated events in which shared objects link the events.

The term " any defined or meaningful sequence" can be interpreted in the light of a textbook pathway or an elementary flux mode, etc. The notion of 'meaningful' covers physiological as well as biotechnological aspects such as the expression of enzymes in a genetically manipulated organism to achieve the synthesis of novel products (see Stephanopulous et al, 1998). Furthermore, by Definition 3.2 the control of a pathway is implicit in the term 'event'. This general definition (3.5) comprises both metabolic and regulatory pathways and facilitates the generation of a wide variety of pathways.

DEFINITION 3.6. A *metabolic subnet* is a system of connected pathways that conforms to thermodynamic constraints.

Three basic graph models were investigated for their suitability to represent biochemical events consisting of operators and non-operators (Section 3.2).

§3.2 Graph models of biochemical events

3.2.1 Simple events

Three basic models were evaluated (Fig.3.1) in terms of known pathways.

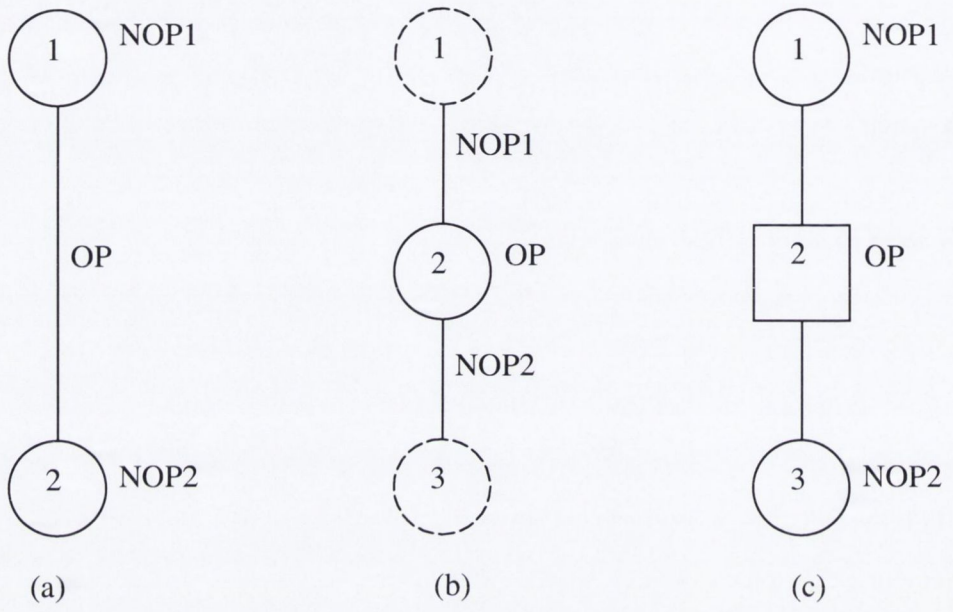


Fig.3.1. Graphs of simple events where OP = operator and NOP = non-operator: (a) NOP on node and OP on edge, (b) NOP on edge and OP on node, (c) NOP on node and OP on node.

The nodes and edges have names (integers for nodes and pairs of integers for edges) and labels. In model (a), the nodes represent different non-operators while the edge (1, 2) represents both an operator and a relationship (e.g., transformation). In model (b), node 2 can represent both an operator and a relationship (e.g., transformation) while edges (1, 2) and (2, 3) represent different non-operators. Nodes 1 and 3 are dummy nodes. These are included to satisfy the definition of a graph in which an edge joins two nodes. In the bipartite graph model (c), nodes 1 and 3 represent non-operators only, node 2 represents an operator only, while edges (1, 2) and (2, 3) represent relationships only, i.e., the properties of being a substrate or a product. In this case, non-operators, operators and relationships are clearly separated. In the above models, an edge (i, j) is equivalent to an edge (j, i), e.g., the edge (1, 2) is equivalent to the edge (2,1).

3.2.2 Complex events

Using model (a) above, it was not possible to represent a complex event because an edge can only join two nodes. It was possible to represent a complex event with model (b), but the model was unsuitable because all external nodes were dummy nodes. Model (c) proved best for modelling complex events. Also, using this model, it was possible to clearly distinguish between operators, non-operators and relationships between them.

The object bipartite graph of the complex event NOP1 + NOP2 = NOP3 + NOP4 mediated by the operator OP is shown in Fig.3.2 below. Colour distinguishes the OP (yellow) from the NOPs (white). The reactants are at the top or the bottom of the diagram depending on the direction of the reaction.

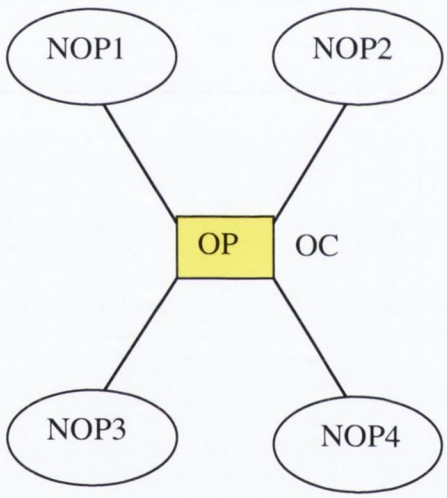


Fig.3.2. Object bipartite graph of an operator-mediated complex event: OP = operator, NOP = non-operator, and OC = operator code. The operator node is coloured yellow while the non-operator nodes are white.

Since the names (integers) of the vertices are not important, their labels represent them. These labels are inserted inside the vertices. In general, labels that identify biochemical objects (usually by their non-integer names) will be placed inside the vertices, and other attributes will be placed adjacent to the vertices. The concept of an *operator code* is introduced. An operator code (OC) is more general than an EC number. An OC would contain all six classes of the EC numbers and all seven classes of the recently developed transporter classification (TC) system (Busch and Saier, 2004) as well as additional classes, which could possibly include uncatalysed reactions and receptors, etc. However,

the development of operator codes is beyond the scope of this thesis. The terms 'event' and 'operator-mediated event' will be used interchangeably.

Using the notation developed above, the complex event of Fig. 3.2 may be described as follows:

OC OP NOP1 + NOP2 = NOP3 + NOP4

Fig.3.3 below shows the directed simple object bipartite graph of three events that are linked by a common NOP (NOP4):

$$NOP1 + NOP2 \rightarrow NOP3 + NOP4$$
$$NOP4 \leftrightarrow NOP5$$
$$NOP4 + NOP6 \rightarrow NOP7 + NOP8$$

Coloured nodes represent operators, where yellow indicates reversibility, and orange otherwise. White nodes represent the non-operators. Since an arrow may give the impression of the conversion of a NOP (e.g., a metabolite) into an OP (e.g., an enzyme), the use of coloured edges is convenient, where blue represents consumption of substrate(s) and green generation of product(s). Thus the coloured edges indicate the directions of the reactions. A black edge can represent either consumption or production. Thus, one can go from OP1 to OP2 and from OP2 to OP3, but not from OP2 to OP1.

This system of events may be described as follows:

OC1	OP1	$NOP1 + NOP2 \rightarrow NOP3 + NOP4$
OC2	OP2	$NOP4 \leftrightarrow NOP5$
OC3	OP3	$NOP4 + NOP6 \rightarrow NOP7 + NOP8$

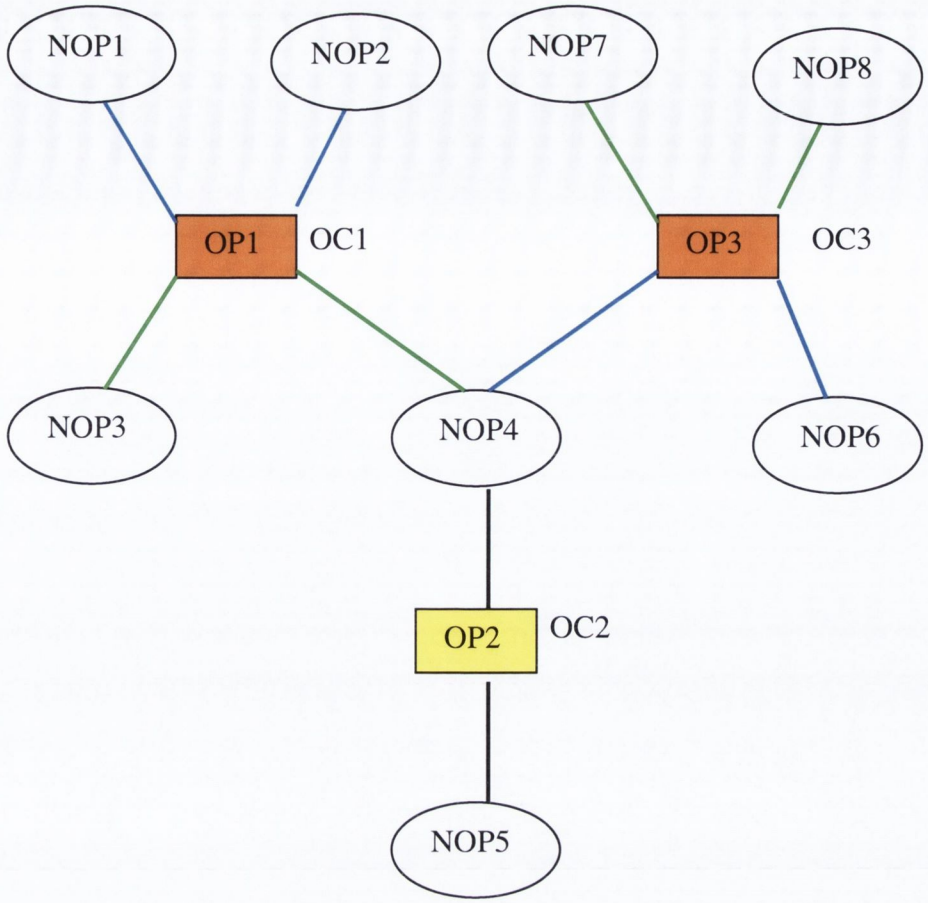


Fig.3.3. Directed simple object bipartite graph of two linked events: OP = operator while NOP = non-operator, and OC = operator code. Coloured nodes represent operators, where yellow indicates reversibility, and orange otherwise. White nodes represent the non-operators. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s). A black edge can represent either consumption or production of metabolite(s).

3.2.3 Description of the representation

There is a space of non-operators (see Definition 3.3) that can be represented as a space of *states* of non-operators (NOPs). There is also a space of *operations* defined on the space of NOPs that is called the space of biochemical events (see Definition 3.2). An operator (OP) acting on the space of states of NOPs, represents an elementary act of event, i.e., a change in the state of NOPs. All such operators belong to the space of events. An event is a basic functional unit in which OPs change the states of NOPs.

The term 'metabolic pathway' is used as defined previously (see Definition 3.5), with the linking of events by shared objects interpreted as a chain of biochemical events constructed

such that the output of one event is the input NOP to, or the OP for, another event, which is usually the next event. Thus, pathways are interconnected arrangements of events, where events are the building blocks of the pathways and objects are the building blocks of the events. The set of event operators from the pathway could be regarded as an operator that represented this pathway in the event space.

Events are represented mathematically by simple object bipartite graphs, in which one type of vertex corresponds to operators and the other type to non-operators. Edges correspond to relationships between the two types. An event may be described in terms of objects and binary interactions between them by noting that binary relations (binaries) of an operator with its non-operators (Section 3.8) drop naturally out of the adjacency list representation (see Section 1.4.4) of the simple object bipartite graph. For example, in Fig.3.2 above, the binary relations are OP_NOP1, OP_NOP2, OP_NOP3 and OP_NOP4, where '_' is a delimiter. A ternary relation, which includes stoichiometry (n), would be OP_NOP1_ n, while an n-ary relation (> 3 components) could be OP_NOP1_ n_L, where L means that NOP1 is on the left hand side of the reaction (as represented in a database). So, given the n-ary relations for some event(s), the corresponding simple object bipartite graph may be constructed. This graph will constitute the actual visual display of the event(s), in which stoichiometry, for example, will be an edge label.

An event is represented in the database (Section 4.1.3) for the Java application (Section 4.3) in a biochemist-friendly way using the notation developed in Section 3.1, and is input as such to the program (Section 4.1.2). With the exception of EC numbers, the input data are converted to a string of n-ary relations (as described above) by the Java code (see Section 4.3.18, especially the example at the end of it). A semi-colon separates each n-ary relation, and different delimiters separate each component of an n-ary relation. The n-ary relations are subsequently separated from the string, using the semi-colon as delimiter, and the components of each n-ary relation are separated from each other using their different delimiters (see Section 4.3.20). The program then uses the various components to construct a simple bipartite graph (see Section 4.3.21), which is the display object. Thus, the n-ary relations connect the input data (converted to n-ary relations) to the graphical output (the graph constructed from the n-ary relations) via the Java code. In other words, the data are the raw inputs, which are then converted to representational form that may, in turn, be displayed. The data input format and the graphical output are biochemist-friendly while the n-ary relations are not.

Certain node and edge attributes are encoded in the n-ary relations. For example, stoichiometry and reaction side (L or R) are edge labels, while edge and edge label colour is blue if the edge label is L and green if it is R. Operator nodes are always coloured (usually yellow). The representation facilitates the inclusion of effectors, which have special features to distinguish them from metabolites (see Section 3.1). The EC numbers relate the files for reactions, reaction-specific references and effectors. Certain options are implicit in the representation, e.g., edge thickness (draw double edge) and node border visibility (draw rectangle).

The data are input to the Applets directly as n-ary relations. These are generated by the application for the WebPathway applet (see Section 4.5). For the other applets, they are partly produced by the application and partly done manually. This is because they use data not present in the Enzyme List, for which the application was designed. The applets generate some simple statistics derived from the representation, i.e., the number of nodes and edges, and the edge to node ratio (see Section 4.5). The node and edge attributes encoded in the n-ary relations in the application are the same for the WebPathway applet.

In the Glycolysis applet (see Section 5.2), the colours for stoichiometry and metabolite role (edge labels) are computed in the program from the sign of the stoichiometric coefficient (blue for negative and green for positive). The edge colours for metabolite flow are also computed from the sign of the stoichiometric coefficient. The program also computes the colours for reversibility. If reversibility is "y" (yes), the corresponding edge is black, otherwise, if it is 'n' (no), the corresponding edge is coloured according to the sign of the stoichiometric coefficient, as already described. However, the Glycolysis applet uses some data (contained within the Java code) that are not part of the n-ary relation representation, e.g., genes, ions and atomic weights (see Section 5.2). In the MultiApplet (see Section 5.3), the colours are assigned to the edges and some of their labels as in the Glycolysis applet. The cell space letters 'C' (for cytosol) and 'M' (for mitochondrion) in the n-ary relations are assigned to the edges. These enable reactions of a pathway to be selected on the basis of cell space, e.g., urea cycle (see Section 5.3).

§3.3 Graphs of metabolic pathways

This section deals with operators and non-operators as components of metabolic pathways in the form of enzymes and metabolites, respectively. The following additional abbreviations (Table 1) are used in the glycolysis pathway and in the glycolysis and gluconeogenesis subnet maps:

Table 1: Compound abbreviations.

Abbreviation	**Name of compound**
BPG	1,3-bisphosphoglycerate
DHAP	dihydroxyacetone phosphate
FBPase	fructose bisphosphatase
G3P	glyceraldehyde 3-phosphate
G6Pase	glucose 6-phosphatase
G3PDH	glyceraldehyde-3-phosphate dehydrogenase
OAA	oxaloacetate
PC	pyruvate carboxylase
PEP	phosphoenolpyruvate
PEPCK	phosphoenolpyruvate carboxykinase
2PG	2- Phosphoglycerate
3PG	3-Phosphoglycerate
PGK	phosphglycerate kinase
PGM	phosphglycerate mutase
PK	pyruvate kinase
TPI	triose phosphate isomerase

3.3.1 Glycolysis

The object bipartite graph of the complete *glycolysis* pathway (glucose to pyruvate) is shown in Fig.3.4 below.

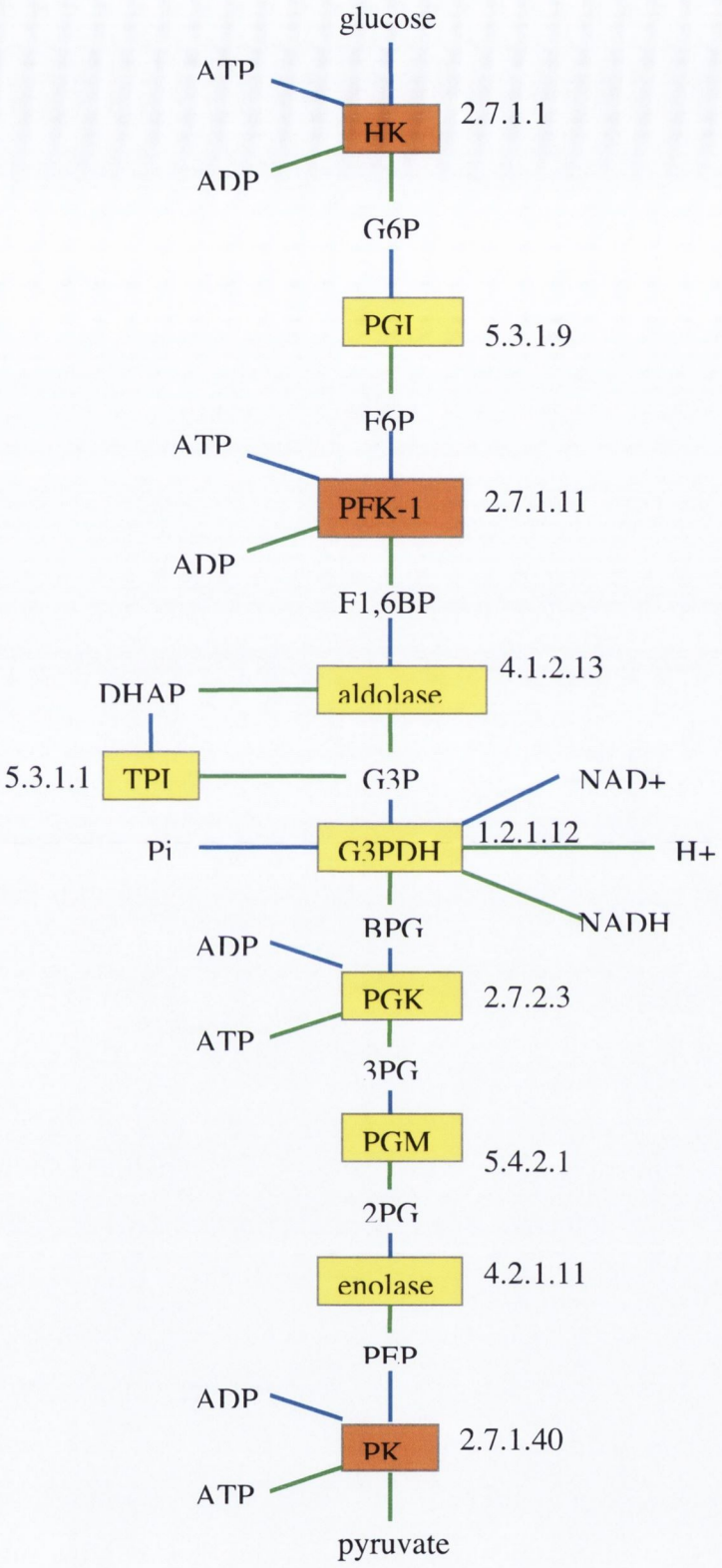


Fig.3.4. Object bipartite graph of glycolysis. Coloured nodes represent enzymes, where yellow indicates reversibility and orange otherwise. The rest are metabolites. The borders of the shapes of the metabolite nodes are not visible. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s). EC numbers are labels adjacent to the enzyme nodes. Abbreviations: see Table 1 above.

The graph of glycolysis conforms to the bipartite model (Fig.3.4) since all adjacent vertices are enzyme-metabolite pairs and the cycle at aldolase in the underlying undirected graph is even. Some compound names are represented many times, e.g., ATP. This is permissible, since two or more vertices can have the same label. Attributes (EC numbers) are displayed as additional labels adjacent to the enzyme vertices. Coloured nodes represent enzymes, where yellow indicates reversibility and orange is otherwise. The rest are metabolites. Edge colours indicate the roles of the metabolites as substrate (blue) or product (green) in the execution of the pathway. The outlines of the shapes of the metabolite vertices are not visible. This appears to make the diagram more intuitive. However, it is better to retain the colour of the enzyme vertices as this emphasises the interactive nature of the bipartite structure.

3.3.2 Glycolysis and gluconeogenesis

The object bipartite graph of the *glycolysis* (glucose to pyruvate) and *gluconeogenesis* (pyruvate to glucose) subnet is shown in Fig.3.5 below. Gluconeogenesis in higher animals begins with pyruvate in the mitochondrial matrix. Pyruvate is carboxylated to oxaloacetate (OAA) by pyruvate carboxylase (PC), which is present only in mitochondria (see Lehninger et al., 1993).

The graph conforms to the bipartite model, since all adjacent vertices are enzyme-metabolite pairs and all three directed cycles, and the cycle at aldolase in the underlying undirected graph, are even. Coloured nodes represent enzymes, where yellow indicates reversibility and orange is otherwise. The rest are metabolites. Coloured edges indicate the roles of the metabolites as substrate (blue) or product (green), while black edges can be either. The letters L (left) and R (right) on the black edges connecting the side compounds to their respective enzymes in the reversible reactions indicate that if a compound belongs to an 'L' group as substrate, then it will belong to an 'R' group as product, and vice versa. This helps to avoid mixing up the side compounds as substrates with the side compounds as products when dealing with reversible reactions in subnets such as this one. In the case of glycolysis, L is substrate and R is product, and vice versa for gluconeogenesis.

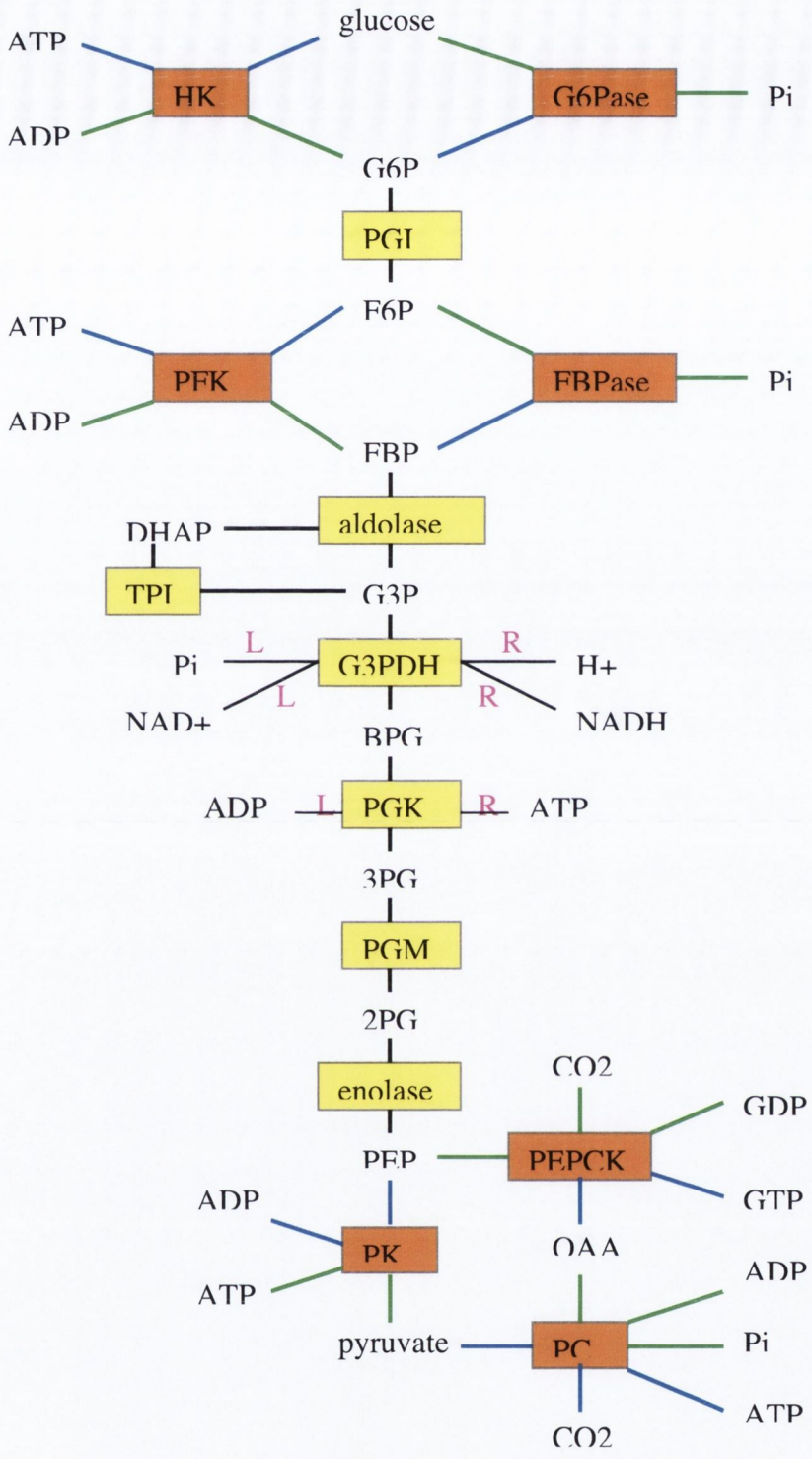


Fig.3.5. Object bipartite graph of the glycolysis (glucose to pyruvate) and the gluconeogenesis (pyruvate to glucose) subnet. Coloured nodes represent enzymes, where yellow indicates reversibility, and orange otherwise. The rest are metabolites. Coloured edges indicate the roles of the metabolites as substrate (blue) or product (green), while black edges can be either. In the case of glycolysis, L is substrate and R is product, and vice versa for gluconeogenesis. Abbreviations: see Table 1 above.

§3.4 Simple object bipartite graphs and stoichiometry

Following on the work of Zeigarnik and Temkin (1994) on bipartite graphs and the stoichiometry of complex reaction mechanisms, a similar approach (with some differences, which will be described) is used here to show the equivalence of the simple object bipartite graph representation of metabolic pathways and their stoichiometric representation.

Consider the following two consecutive reactions on the *haem biosynthesis pathway*:

4 porphobilinogen (PBG) + H_2O → hydroxymethylbilane (HMB) + 4 ammonia (NH_3)

HMB → uroporphyrinogen III (UPP III) + H_2O

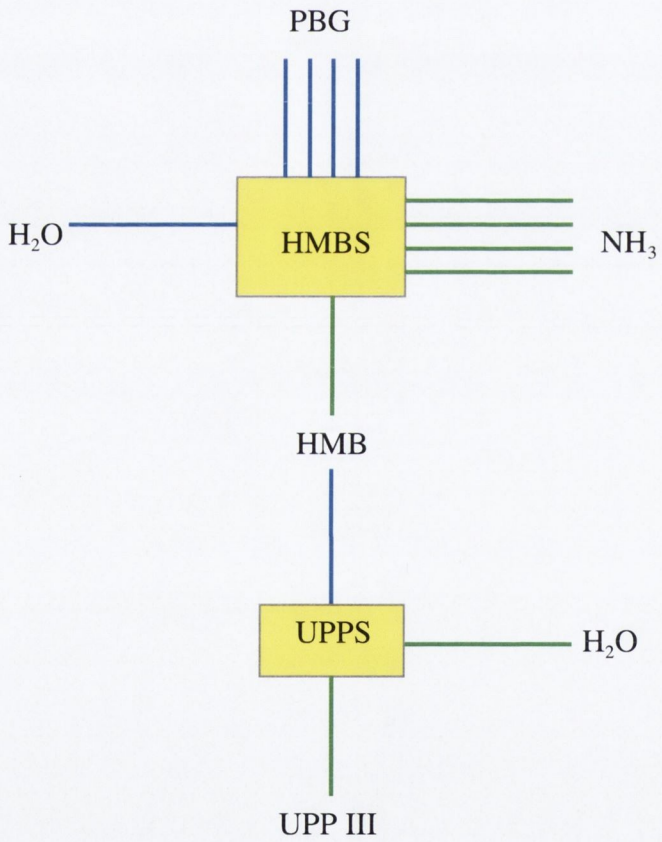


Fig.3.6. Directed bipartite multigraph of two consecutive reactions on the haem biosynthesis pathway. The yellow vertices represent enzymes and the rest are metabolites. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s). The number of edges connecting a metabolite with an enzyme represents the stoichiometry of that metabolite in the reaction.

They are catalysed by the enzymes hydroxymethylbilane synthase (HMBS, EC 2.5.1.61) and uroporphyrinogen-III synthase (UPPS, EC 4.2.1.75), respectively. The object bipartite multigraph for these reactions is shown in Fig. 3.6 above. The number of edges connecting a metabolite with an enzyme represents the stoichiometry of the metabolite in the reaction (see Section 1.4.6).

Firstly, since the graph is bipartite, its reduced adjacency matrix (see Section 1.4.4) represents it as follows:

	HMBS	**UPPS**
PBG	4	0
H_2O	1	1
HMB	1	1
NH_3	4	0
UPP III	0	1

Secondly, the bipartite graph is directed. Therefore, taking the sign of an entry as *negative* for a vertex in the non-operator (metabolite) set adjacent *to* a vertex in the operator (enzyme) set, and the sign of an entry as *positive* for a vertex in the non-operator set adjacent *from* a vertex in the operator set, and the absolute value of a reduced adjacency matrix entry equal to the multiplicity of the respective edge, results in the following matrix:

	HMBS	**UPPS**
PBG	-4	0
H_2O	-1	1
HMB	1	-1
NH_3	4	0
UPP III	0	1

This is the stoichiometry matrix for the reactions in question. Therefore, without loss of generality, there is a one-to-one correspondence between the directed object bipartite multigraph and the stoichiometric matrix for a system of metabolic reactions. However,

multiple edges are computationally more difficult to draw and aesthetically less pleasing to the eye than single edges. Therefore, instead of using several edges between an operator and a non-operator or vice versa, it is proposed to use a single edge that is weighted by an appropriate integer, which is signed if the direction is known. This edge label indicates the number of edges connecting a metabolite to an enzyme, which in turn represents the stoichiometry of the metabolite that is connected to the enzyme. Additionally, the corresponding colour and the sign of the integer(s) on an edge enhances the notion of direction. These proposals give rise to the following structure shown in Fig. 3.7 below, which represents the same two reactions of Fig. 3.6 above.

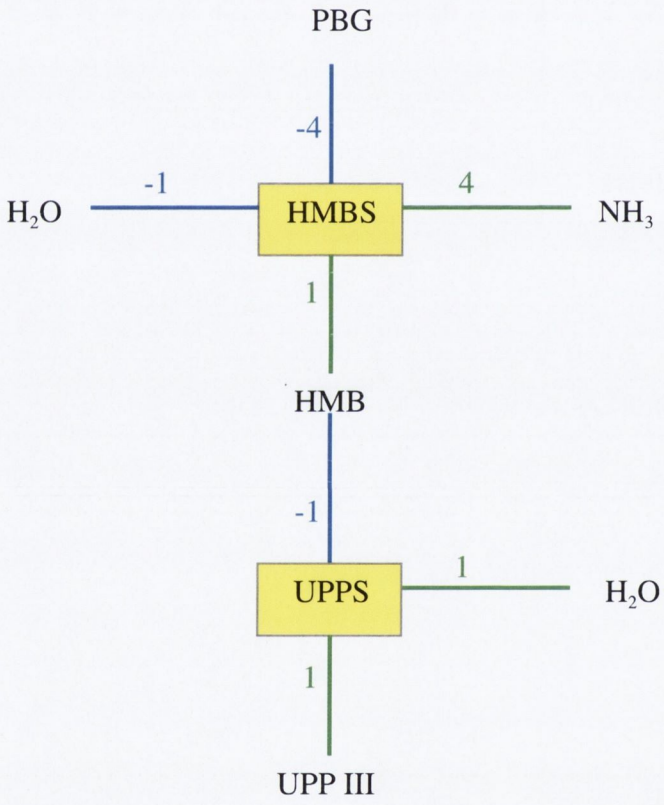


Fig.3.7. Simple directed bipartite graph of two consecutive reactions on the haem biosynthesis pathway, showing the stoichiometry as edge labels. The yellow vertices represent enzymes and the rest are metabolites. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s).

Thus, the stoichiometry matrix and the simple object bipartite graph representations are mutually consistent and complementary. In the latter, the sign of an entry is an edge colour where for any given colour that represents a negative sign, a different colour represents a positive sign, and no incoming edge has the same colour as an outgoing edge. Thus an entry of the stoichiometry matrix is a property of an edge of the simple object

bipartite graph in which vertices represent biochemical objects. The magnitude of an entry represents the number of metabolite vertices (or molecules) of the same type linked to an enzyme vertex in a reaction. Conversely, any property of an edge in the simple object bipartite graph can be represented as an element of the corresponding matrix. Thus the stoichiometry matrix is the matrix representation of the simple object bipartite graph, including edge multiplicity that is represented by edge labels. This is consistent with the fact that bipartite graphs and rectangular matrices are interchangeable (see Beineke et al., 2004).

It is also necessary to take into consideration the number of times a reaction occurs in a pathway (see Seressiotis and Bailey, 1986), defined here as the *reaction occurrence number*. For example, consider the first two reactions on the haem biosynthesis pathway. In terms of the notation developed above (3.1), they may be represented as follows:

2.3.1.37 ALAS SCoA + glycine → δ-aminolaevulinic acid (ALA) + CoA + CO_2

4.2.1.24 PBGS 2 ALA → PBG + 2 H_2O

These reactions are catalysed by the enzymes δ-aminolaevulinic acid synthase (ALAS, EC 2.3.1.37) and porphobilinogen synthase (PBGS, EC 4.2.1.24), respectively. The abbreviation 'ScoA' stands for succinyl-CoA. Obviously, the first reaction must occur twice to enable the second one to take place. Since a reaction that is catalysed by several enzymes is entered in the matrix multiple times (see Section 1.4.1), it follows that the same enzyme-catalysed reaction, which has to occur twice for a pathway to be viable, can be represented twice for convenience in the matrix. Here, this is referred to as the *pathway stoichiometry*. Thus, the annotated stoichiometry matrix for these reactions is as follows:

	ALAS	**ALAS**	**PBGS**
SCoA	-1	-1	0
glycine	-1	-1	0
ALA	1	1	-2
CoA	1	1	0
CO_2	1	1	0
PBG	0	0	1
H_2O	0	0	2

The bipartite graph for the same reactions is shown in Fig. 3.8 below.

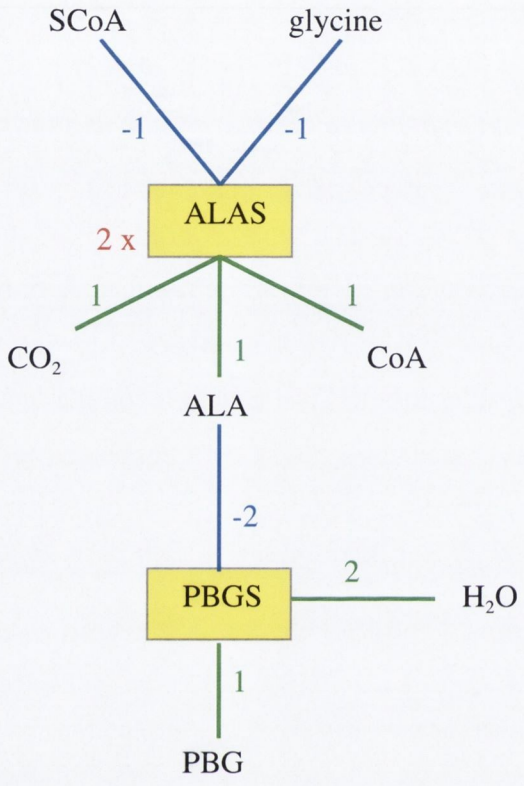


Fig.3.8. Simple directed object bipartite graph of the first two reactions on the haem biosynthesis pathway, showing the reaction stoichiometry as edge labels. The yellow vertices represent enzymes and the rest are metabolites. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s). The reaction occurrence number 2 followed by a multiplication sign placed as a red label adjacent to the enzyme node ALAS indicates that this reaction occurs twice.

If a reaction occurs more than once in a pathway, a reaction occurrence number followed by a multiplication sign is placed as a red label adjacent to the corresponding enzyme node, otherwise the reaction occurrence number is assumed to be one. For example, the label '2 x' adjacent to the 'ALAS' node indicates that this reaction occurs twice in the execution of the pathway, each time converting one molecule of SCoA and one of glycine into one molecule each of CO_2, ALA and CoA. This is necessary since two molecules of ALA are required for the formation of PBG.

Multiplication of the reaction stoichiometry by the reaction occurrence number gives the pathway stoichiometry, shown as edge labels in Fig. 3.9 below. This represents the number of molecules reacting in one turn of the sequence to convert two molecules of SCoA and two molecules of glycine to one molecule of PBG.

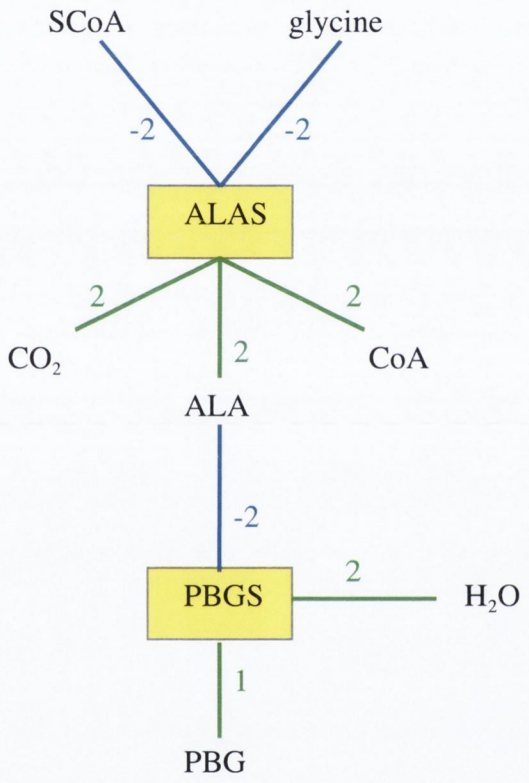


Fig.3.9. Simple directed object bipartite graph of the first two reactions on the haem biosynthesis pathway, showing the pathway stoichiometry as edge labels. The yellow vertices represent enzymes and the rest are metabolites. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s).

Alternatively, the use of a number in brackets after the enzyme name, e.g., ALAS (2), to represent the reaction occurrence number, with a corresponding multiplication of the stoichiometric coefficients, would reduce the number of columns in the matrix. The corresponding simple bipartite graph would be the same as in Fig. 3.9 above except that the name of the ALAS node would now be ALAS (2) (see Beasley and Planes, 2007; Planes and Beasley, 2008).

Other matrices can also be derived from the graph, e.g., a role matrix, which gives the roles of the metabolites in the various reactions. For example, consider the graph of Fig. 3.10 below.

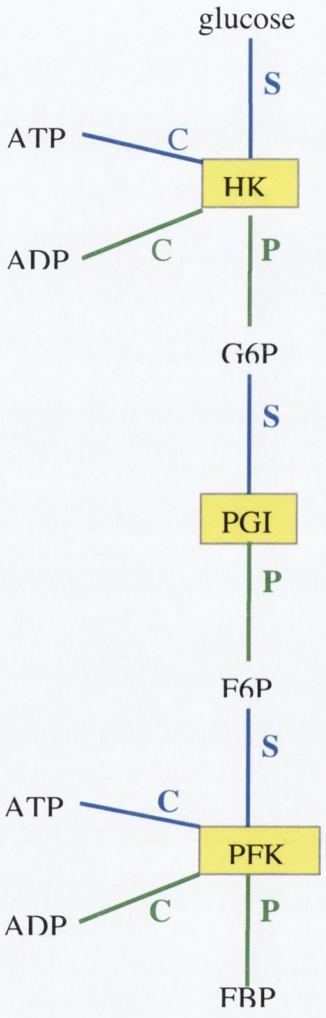


Fig.3.10. Simple directed object bipartite graph of the first three reactions on the glycolysis pathway, showing the roles of the metabolites as edge labels. The yellow vertices represent enzymes and the rest are metabolites. A blue edge represents consumption of substrate(s) or cosubstrates(s) and a green one represents generation of product(s) or co product(s). The edge labels are S, P and C, which represent substrate, product and currency metabolite, respectively.

In the graph of Fig. 3.10, the roles of the metabolites are shown as edge labels. Although the colouring of an edge can be the same as labelling it, this is not true in general. For example, the edge label 'C' gives additional information on the role of ATP as a currency metabolite, which is not evident by colour alone. Since a property of an edge of the object

bipartite graph is an element of the corresponding matrix, the roles of the metabolites in the various reactions can be represented in a *role matrix* as shown in Fig. 3.11 below:

	HK	PGI	PFK
glucose	S	0	0
G6P	P	S	0
F6P	0	P	S
FBP	0	0	P
ATP	C	0	C
ADP	C	0	C

Fig.3.11. Role matrix for the graph of Fig. 3.8. S is substrate, P is product, and C is currency metabolite. Colour distinguishes substrates (blue) from products (green).

For example, G6P is a product (P) of the HK reaction and a substrate (S) for the PGI reaction. Furthermore, although ATP and ADP are both currency metabolites (C), the colour coding indicates that ATP is consumed and ADP is produced.

The present model differs from that of Zeigarnik and Temkin (1994) with respect to the following: (1) it represents metabolic pathways rather than chemical reaction mechanisms, (2) there are no artefacts in the form of reaction nodes, (3) an edge label is used to indicate the stoichiometric coefficients, (4) an operator node label is used to indicate the number of times a reaction occurs in a pathway and (5) the graphs are simple. Simple graphs are less difficult computationally to draw and aesthetically more pleasing than multiple edges. Furthermore, this concept is generalised so that any edge property of an object bipartite graph of a biochemical pathway is represented as an entry in its corresponding matrix. Thus, a metabolic pathway may be represented by a set of annotated matrices of the same size, where each matrix represents a different edge property of the corresponding object bipartite graph.

§3.5 Non-enzyme operators

So far, the operators have been enzymes and the non-operators have been metabolites. However, since operators constitute a general concept, they should also be able to mediate other processes, e.g., transport across membranes, receptor-ligand interactions and spontaneous (uncatalysed) reactions.

3.5.1 Transport across membranes

Multicellular organisms have multiple tissue types and a subcellular structure. This information can be encoded in the operators by specifying the boundary between compartments, e.g., a membrane operator separates two compartments and mediates the transport of compounds between them. Consider again the first two reactions on the haem biosynthesis pathway. These reactions occur in different cellular compartments, which are separated by the mitochondrial membrane (MM). This situation, in which the membrane is the operator and OC represents an operator code, can be represented as follows:

2.3.1.37	ALAS	SCoA + glycine → m-ALA + CoA + CO_2
OC	MM	2 m-ALA → 2 c-ALA
4.2.1.24	PBGS	2 c-ALA → PBG + 2 H_2O

Reaction 2.3.1.37 takes place in the mitochondrion while reaction 4.2.1.24 occurs in the cytosol. Cellular compartments are indicated by the prefixes m- for mitochondrial and c- for cytosolic. The corresponding graph is shown in Fig. 3.12 below.

In this thesis a compartment is not treated as a separate object but may be represented by a section of a graph that is sandwiched between two membrane nodes (one entry node and one exit node) for convenience.

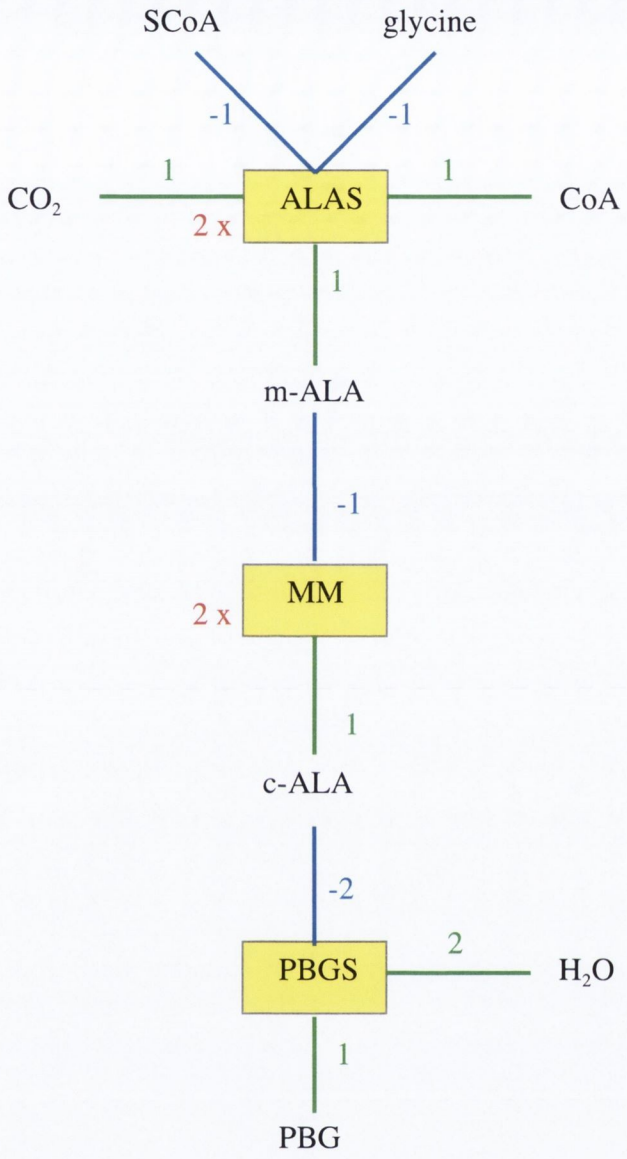


Fig.3.12. Simple directed object bipartite graph of the early reactions on the haem biosynthesis pathway, showing the transport of ALA across the mitochondrial membrane (MM). The yellow vertices represent operators and the rest are metabolites. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s). The red labels adjacent to the operator nodes indicate the number of times these reactions occur.

3.5.2 Receptor-ligand interactions

The following is a representation by Ochs et al. (1994) of a reaction of a receptor with its ligand:

$$\text{Epinephrine} + \text{Receptor} \xrightarrow{\text{Receptor}} \text{Epinephrine-Receptor}$$

According to the authors, the term receptor is shown in the reaction both as substrate and as "enzyme". Using the present notation, where adrenaline is used instead of epinephrine and OC stands for operator code, the same reaction is represented as follows:

OC receptor receptor + adrenaline = receptor-adrenaline

Similar to Ochs et al. (1994), this representation shows the receptor as both an operator and a non-operator in the same reaction. The receptor binds the ligand to produce the receptor-adrenaline complex. In this case, it is useful to think of the operator as adding itself to the ligand. The graph of this reaction is shown in Fig. 3.13 below.



Fig.3.13. Simple directed object bipartite graph showing the binding of adrenaline by its receptor to give the receptor-adrenaline complex. The yellow vertex represents a receptor operator and the rest are non-operators. A purple edge represents binding and an orange one represents formation (of the receptor-adrenaline complex).

Interestingly, even though the receptor is both an operator and a non-operator in the same reaction, the graph remains bipartite as shown above (Fig. 3.13). The receptor-adrenaline complex becomes an operator in a subsequent reaction (see 3.6 below). Although this raises conceptual issues, such as the same object being both an OP and a NOP in the same reaction, the representation is convenient and consistent with the use of bipartite graphs. Definition 3.3 does not deal with this situation.

Alternatively, adrenaline may be considered as an effector (see Boyer, 1999), which the receptor operator binds. The binding of adrenaline induces a conformational change in that part of the receptor that is positioned inside the cell (see Stryer et al., 2007). Thus the

activated receptor becomes the operator for the next event (activation of a G protein) due to its altered cytoplasmic domain. In its simplest form, this can be represented here as follows:

OC receptor adrenaline :

Recall that the colon (:) used to separate activators from inhibitors (see Section 3.1). The corresponding bipartite graph is as follows:



This works, but the receptor-ligand equation is omitted.

3.5.3 Spontaneous (unanalysed) reactions

Activation energy is the energy that a molecule must possess above the energy of its ground state in order to be transformed into some other species (see Carey, 1996). A catalyst acts by lowering the activation energy for a reaction. In an uncatalysed reaction, the activation energy itself could be treated as the operator for convenience. Alternatively, the name of the process or even the operator code may represent the operator. For example, in the last step of glycolysis the product pyruvate is believed to first appear in its enol form. However, the enol form tautomerises rapidly and nonenzymatically to yield the keto form of pyruvate. The bipartite graph of this spontaneous reaction is shown in Fig. 3.14 below.

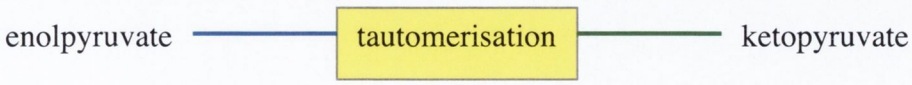
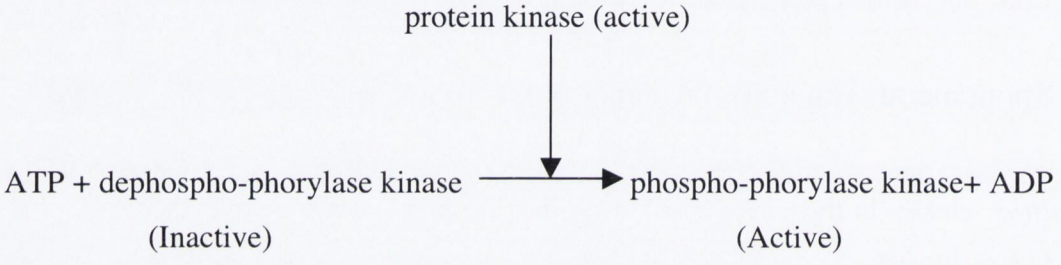


Fig.3.14. Simple directed object bipartite graph of a spontaneous reaction in which the process of tautomerisation is the operator. The yellow vertex represents the operator and the rest are metabolites. A blue edge represents consumption of substrate and a green one represents generation of product.

§3.6 Graph of a regulatory cascade

There are also regulatory pathways that contain hierarchical structures in which the product of one reaction serves as a catalyst for another reaction. This situation challenges the bipartite model, since one is dealing with the concept of an enzyme as substrate (and product). However, since a biochemical object is only an operator while it is in the act of mediating a particular biochemical event (see Definition. 3.3), an enzyme is a non-operator in a reaction in which it is being activated. Intermodular links (dashed edges) are formed by the product (non-operator) of one module acting as a catalyst (operator) for the next module (Fig. 3.15). This is not unusual, since in classical diagrams (see Hames et al., 1997), the following equivalent scenario may be encountered:



The product (active protein kinase) of one reaction catalyses the next reaction. However, in Fig. 3.15 below, compounds are shown twice (once as non-operator and once as operator). It was chosen this way because it is convenient to progress towards a simpler representation. This may appear cumbersome but it allows later, ready production of some graphs.

The graph conforms to the bipartite model, since all adjacent vertices are OP-NOP pairs. Also, the graph is a tree and every tree is a bipartite graph (see Section 1.4.3). It shows a cascade of events amplifying the response to adrenaline and seamlessly integrating with core metabolism via the conversion of glycogen to GIP. Yellow vertices represent operators while the rest are non-operators. The latter include enzymes as substrate (inactive state) in red font, enzymes as product (active state) in green font, metabolites in black font and protein-metabolite complexes in orange font. It is not necessary to have different font colours. However, they are used here for convenience. Blue edges represent transformation or consumption of substrate, while green edges represent formation of product. A purple edge represents the binding of a ligand to its receptor and an orange one represents the formation of a receptor-ligand complex.

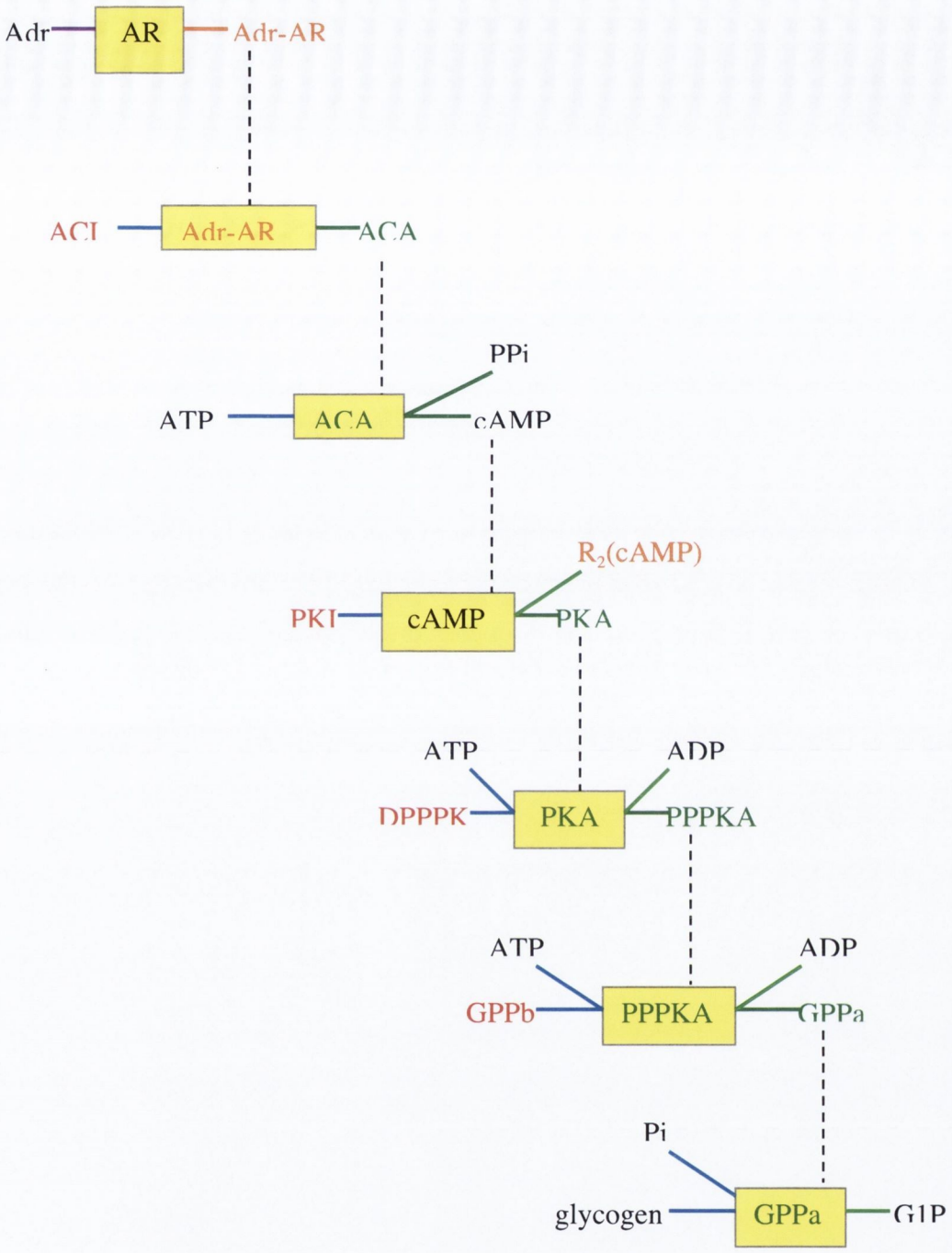


Fig. 3.15. Object bipartite graph of a regulatory cascade for the activation of glycogen degradation in liver. Metabolites are in black font and protein-metabolite complexes are in orange font. Compounds in red font are enzymes as substrate (inactive state) and compounds in green font represent enzymes as product (active state). These are non-operators. Yellow rectangles represent operators that are actually mediating the reactions. The intermodular links (dashed edges) are formed by the non-operator of one module acting as an operator for the next module. Blue edges represent transformation or consumption of substrate, while green edges represent formation of product. A purple edge represents binding of a ligand and an orange one represents formation of a complex. See the text for an explanation of the cascade and the abbreviations.

In the simple object bipartite graph of the regulatory cascade for the activation of glycogen degradation in liver (Fig. 3.15 above), the hormone adrenaline (Adr) binds to an adrenergic receptor (AR) in the plasma membrane of the target cell to produce the adrenaline-adrenergic receptor complex (Adr-AR). This causes a conformational change in the receptor protein, which in turn converts inactive adenylate cyclase (ACI) to the active form via a G-protein (not shown). Then activated adenylate cyclase (ACA) converts ATP to cyclic AMP (cAMP) and pyrophosphate (PPi). The cAMP binds to inactive protein kinase A (PKI), which consists of two regulatory (R) and two catalytic (C) subunits in a complex R_2C_2. The binding of two molecules of cAMP to each of the regulatory subunits leads to dissociation of the complex into an R_2 complex and two free C subunits that are active. The active protein kinase A (PKA) phosphorylates the inactive dephospho-phosphorylase kinase (DPPPK) to the active phospho-phosphophorylase kinase (PPPKA). The latter now phosphorylates the glycogen phosphophorylase b (GPPb), converting it to the active phosphophorylase a (GPPa), which carries out a rapid degradation of glycogen to glucose 1-phosphate (G1P). See Hames et al. (1997). Dashed edges connect the reactions by linking shared objects. It is noted that the hormone adrenaline (a first messenger) is a non-operator, while cAMP (a second messenger) is an operator, in this model.

§3.7 Miscellaneous

3.7.1 Modulators

An allosteric enzyme is a regulatory enzyme, with catalytic activity modulated by the noncovalent binding of a specific metabolite at a site other than the active site. Consider the following reaction on the glycolysis pathway mediated by the operator (enzyme) 6-phosphofructokinase (EC 2.7.1.11):

ATP + D-fructose 6-phosphate = ADP + D-fructose 1,6-bisphosphate

The modulators of this reaction are represented as follows (see Section 3.1):

AMP + fructose 2,6-bisphosphate : ATP + citrate

However, in order to distinguish metabolites from modulators in the graph models, the representation of the modulators is modified as follows:

{+}AMP + {+}fructose 2,6-bisphosphate : {-}ATP + {-}citrate

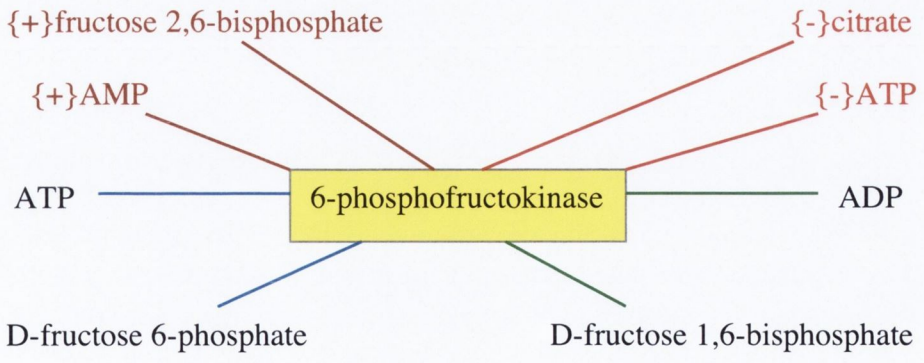


Fig. 3.16. The object bipartite graph of a modulated reaction. The activators (brown) are connected to the enzyme (yellow) by a brown activation edge while the inhibitors (red) are connected to the enzyme by a red inhibition edge. Metabolites are in black font. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s).

The activators are prefixed by {+} and the inhibitors are prefixed by {-}. The graph of this modulated reaction is shown in Fig. 3.16 below. It is noted that the modulators are non-operators in an event. It may be recalled that hormones are also non-operators in an event (see 3.6). This is consistent, since hormones may be considered as effectors (see Boyer, 1999).

3.7.2 Extended operators

Extended operators can include additional information such as the organism and the chromosome of origin of the operator. This does not imply that genes catalyse reactions. The following example shows the chromosomes of origin of the operators in the organism *Plasmodium falciparum* (Fig. 3.17):

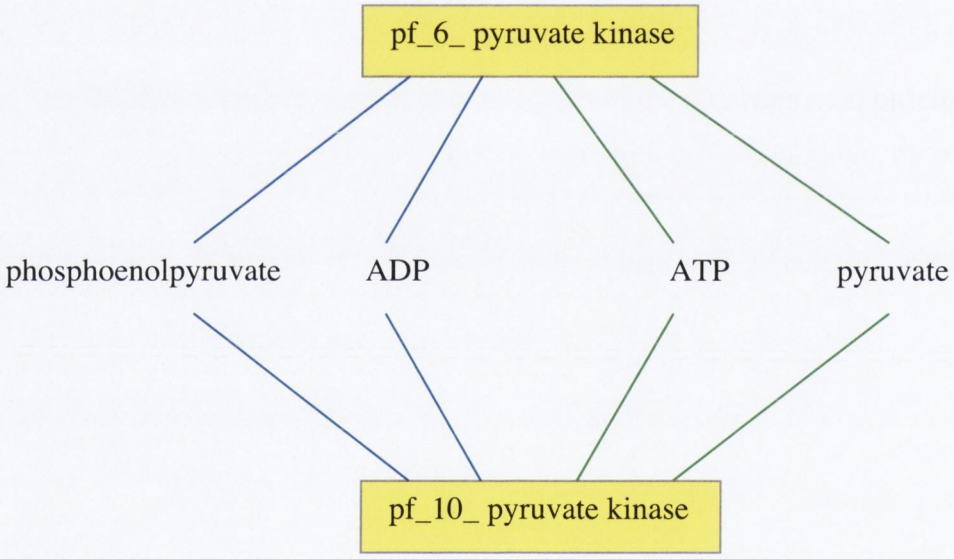


Fig. 3.17. The object bipartite graph of the pyruvate kinase reaction showing the chromosomes (6 & 10) of origin of the operators in the organism *Plasmodium falciparum* (pf). The yellow vertices represent extended operators and the rest are metabolites. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s).

This may be a useful representation for some situations. For example, a pathway may be considered as an event, in which source metabolites are converted into target metabolites via a specific sequence of reactions. This is consistent with Definition 3.2. The extended operator for this event would be the set of individual operators specific to each reaction, which is consistent with Definition 3.3. For example, the glycolytic operator (GOP)

mediates the conversion of glucose, 2 NAD^+, 2 ADP, 2 Pi into 2 pyruvate, 2 NADH, 2 H^+, 2 ATP and 2 H_2O, which are the non-operators in the glycolysis event. This effectively is the directed event representing the net reaction.

§3.8 Binary relations

DEFINITION 3.7. The *binary relation* (or binary) of an operator with its non-operator is defined here as a connected operator_non-operator pair.

The binaries of an operator with its non-operators drop naturally out of the adjacency list representation of the bipartite graph of a biochemical reaction. For example, consider the following bipartite graph of the hexokinase reaction (Fig.3.18):

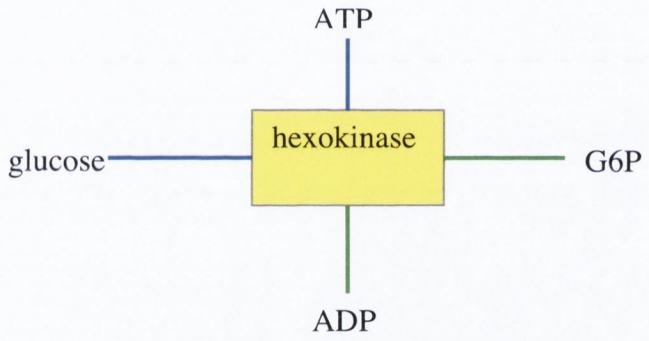


Fig.3.18. Object bipartite graph of the hexokinase reaction. The yellow vertex represents the enzyme and the rest are metabolites. The blue edges represent consumption of substrates and the green ones represent production of products.

The adjacency list for the graph of Fig Fig.3.18 is:

Vertex	Adjacent Vertices
hexokinase	Glucose, ATP, ADP, G6P
glucose	hexokinase
ATP	hexokinase
ADP	hexokinase
G6P	hexokinase

Hexokinase is the only vertex adjacent to each of the other vertices. This results in the following binaries:

BINARIES
Hexokinase_glucose
Hexokinase_ATP
Hexokinase_ADP
Hexokinase_G6P

Thus, a biochemical reaction can be represented as a specific set of binaries. This leads to the following computational definition:

DEFINITION 3.8. A biochemical reaction is a specific set of operator_non-operator binaries, where all operator components are the same and all non-operator components are different.

Now consider the short pathway of Fig.3.19.

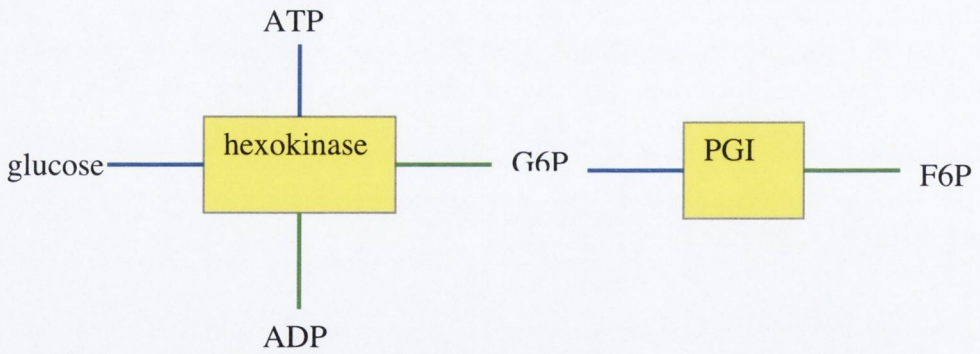


Fig.3.19. Bipartite graph of a short pathway (consistent with Definition 3.5). The yellow vertices represent enzymes and the rest are metabolites. Blue edges represent consumption of substrates and the green ones represent production of products.

The binaries for the graph of Fig.3.19 are:

BINARIES
hexokinase–glucose
hexokinase–ATP
hexokinase–ADP
hexokinase–G6P
PGI–G6P
PGI–F6P

Therefore, a pathway can also be represented as a specific set of binaries. If two or more binaries have a right component in common (e.g., hexokinase–G6P and PGI–G6P), then the reactions to which they belong are connected. Thus, in general, a set of intra-reaction binaries is characterised by having a common enzyme while a set of inter-reaction binaries is characterised by having a common metabolite. A metabolic pathway is thus computationally represented as a specific set of binaries. These binaries may be considered as fundamental "building blocks" of reactions and pathways, where a pathway is a super-set of binaries. This leads to the following computational definition of a metabolic pathway:

DEFINITION 3.9. A metabolic pathway is a specific set of operator_non-operator binaries, where intra-reaction binaries are characterised by having a common enzyme and inter-reaction binaries are characterised by having a common metabolite.

However, consider the binaries of the transaldolase (TA) reaction and transketolase (TK) reactions on the pentose phosphate pathway as shown in Fig.3.20 below. The TK enzyme is involved in two different reactions. With binaries 'A' it is not possible to differentiate between these reactions. However, the reactions are distinguished by appending a different number to the name of the enzyme for each reaction. This gives transketolase1 (TK1) and transketolase2 (TK2). Thus, with binaries 'B' it is possible to distinguish between them. Furthermore, it is more intuitive to distinguish between these two reactions when displaying the pathway as shown in Fig.3.21 below.

BINARIES A
TK_R5P
TK_Xu5P
TK_S7P
TK_G3P
TK_Xu5P
TK_E4P
TK_F6P
TK_G3P
TA_G3P
TA_S7P
TA_E4P
TA_F6P

BINARIES B
TK1_R5P
TK1_Xu5P
TK1_S7P
TK1_G3P
TK2_Xu5P
TK2_E4P
TK2_F6P
TK2_G3P
TA_G3P
TA_S7P
TA_E4P
TA_F6P

Fig.3.20. Sets of binaries for the transaldolase (TA) reaction and the transketolase reactions (TK1 and TK2) on the pentose phosphate pathway. With binaries 'A' it is not possible to differentiate between the two TK reactions but with binaries 'B' it is possible to do so. Abbreviations: E4P = erythrose 4-phosphate, F6P = fructose 6-phosphate, G3P = glyceraldehyde 3-phosphate, R5P = ribose 5-phosphate, S7P = sedoheptulose 7-phosphate, Xu5P = xylulose 5-phosphate.

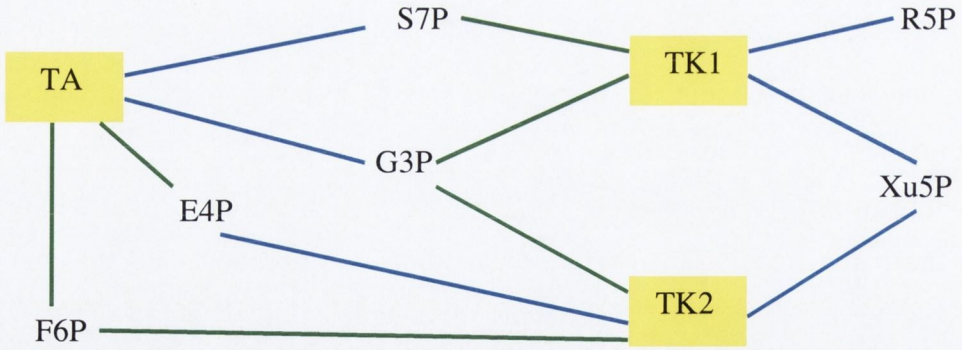


Fig.3.21. Object bipartite graph of the transaldolase (TA) reaction and the transketolase reactions (TK1 and TK2) on the pentose phosphate pathway. The yellow vertices represent enzymes and the rest are metabolites. A blue edge represents consumption of substrate(s) and a green one represents generation of product(s). Abbreviations: see Fig.3.20 above.

For a specific reaction, one can list all the members of a set of binaries. For example, if the hexokinase reaction is the set H and the phosphoglucose isomerase reaction is the set I, then:

$$H = \{HK\text{–glucose}, HK\text{–ATP}, HK\text{–G6P}, HK\text{–ADP}\}$$
$$I = \{PGI\text{–G6P}, PGI\text{–F6P}\}.$$

and the union of the sets, P, represents the pathway of Fig.3.19**:**

$$P = H \cup I = \{HK\text{–glucose}, HK\text{–ATP}, HK\text{–ADP}, HK\text{–G6P}, PGI\text{–G6P}, PGI\text{–F6P}\}.$$

In addition to binary relations, it is useful to consider other relations, such as ternary relations (e.g., enzyme_metabolite_stoichiometry) and even higher order relations (n-ary relations), e.g., enzyme_metabolite_stoichiometry_L, where L denotes the left hand side of the reaction. These relations are important in practice as will be seen in chapter 4.

§3.9 Conclusion

For the purpose of representation and display of biochemical reactions and metabolic pathways, the method of simple object bipartite graphs, in which one type of vertex is assigned to operators and a second type to non-operators, was found most suitable. This decision resulted from the fact such graphs (1) make it possible to display complex reactions, (2) can model various biochemical processes, (3) can represent reaction stoichiometry and other properties of reactions as edge labels, (4) can represent operator and non-operator attributes as node labels, including reaction occurrence number as an operator node label, (5) are unambiguous, (6) are simple, and (7) can generally separate operators, non-operators and the interactions between them. In this scheme the 'material' aspect is emphasised, with the edge colours describing relations between non-operators and operators (e.g., the property of being either a substrate or a product in a reaction, etc.).

A one-to-one correspondence was established between simple object bipartite graphs and stoichiometry matrices for a system of biochemical reactions. This allowed entries of the matrices to be represented as edge properties of the graphs, and vice versa. Thus, the stoichiometry matrix is a representation of the object bipartite graph where an entry of zero

indicates that there is no connection between operators and non-operators. Conversely, an entry that is not equal to zero indicates a connection between them, and is also an edge property of the corresponding graph. Other matrices can also be derived from the graph, e.g., a role matrix, which gives the roles of the metabolites in the various reactions. Therefore, a metabolic pathway may be represented by a set of annotated matrices of the same size, where each matrix represents a different edge property of the corresponding object bipartite graph. In such matrices an entry may not only represent stoichiometry but other properties as well. The model was used to represent and display reactions, pathways and subnets of biochemical reactions. The binaries of an operator with its non-operator(s) dropped naturally out of the adjacency list representation of a simple bipartite graph of a biochemical reaction. Furthermore, it was possible to represent reactions and pathways in terms of these binaries, and to extend the representation to higher order relationships (n-ary relations), which can include additional information, such as stoichiometry.

The model resulted in drawings that were different from classical ones with respect to the representation of enzymes as vertices rather than labels and coloured edges instead of arrows. The links in the bipartite structure allow the display of properties of reactions such as stoichiometry, metabolite roles, etc. as edge labels, which is not possible with classical drawings. Furthermore, since it uses vertices for enzymes it can easily be transformed to a gene-centric representation. The model was able to represent other biochemical processes, such as membrane transport, receptor-ligand interactions and spontaneous (uncatalysed) reactions. Additionally, it adequately describes the network of interacting molecules that forms the living cell.

The directed object bipartite graph establishes a basis for a unified model of metabolic pathways and regulatory cascades, and supports their display and manipulation. Finally, the representation in terms of both matrices and graphs confers considerable power to this system of modelling metabolic pathways. This may not necessarily completely solve the situation with respect to a standard model, but it may be a way to move forward.

CHAPTER 4

RESULTS: PRACTICE

§4.1 Automated generation of biochemical pathways

4.1.1 General principles

The following metabolic pathway components were used: biochemical objects and biochemical events, as defined in §3.1. Two types of object were employed: operators that are mediating biochemical events and non-operators that are participants in operator-mediated events (see Definition 3.3). Operators may be proteins (e.g., enzymes or receptors) or non-proteins (e.g., cyclic AMP). Non-operators are usually metabolites, but can also be enzymes (as substrates) or other compounds. Two components of an event were distinguished: reaction and regulatory action, as described in §3.1. Each event represents a link in a pathway via its shared objects. Enclosing a sequence of events between two membrane operators (membrane proteins or lipid bilayers) represents compartmentalisation of that sequence. The information was stored in text files (which constitute the database). These include two main files: a general data file including the reactions of the Nicholson Metabolic Pathways wall chart (Nicholson, 2000) and a user data file (see Appendix A1). Smaller data files contain the reactions of classical pathways and regulatory actions (see Appendix A2). Further information on these and other files is given below (see Section 4.1.3). This list of files can be readily expanded. The hierarchy of pathway components is shown in Fig. 4.1 below.

Data are automatically extracted from the data files, processed and sent to drawing classes, which display the reactions and pathways in the form of object bipartite graphs. One type of vertex represents the operators and the other type represents the non-operators, while the edges represent interactions between them. Colour distinguishes operators from non-operators and certain information is displayed as edge labels, e.g., stoichiometry. These graphs are automatically produced "on the fly". A separate window gives the equations and associated information for the reactions that are displayed. Information on selected metabolic pathways and on using the database is available as part of the system. The system was designed and implemented using an object-oriented approach. Graphical user interfaces allow the exploration, selection, and visualisation of the reactions and pathways as a Java application or through the Internet as a Java applet.

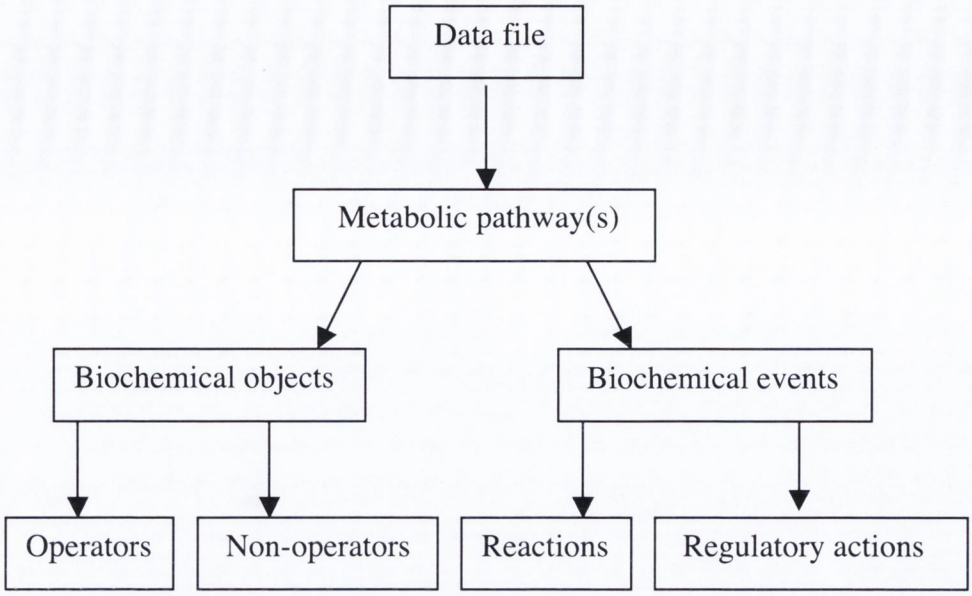


Fig.4.1. The hierarchy of metabolic pathway components in the data files.

4.1.2 The input format

An operator-mediated biochemical event was specified in a user-friendly way using the notation developed in §3.1. The events were manually copied and pasted or typed into the data files (see 4.1.3 below). These files are automatically input to the system in the format that the program can recognise and manipulate. The input data have to be in plain text and therefore Greek letters are replaced, e.g., alpha and italics are lost. The event information for each operator is contained in a string that occupies a single line. Each line contains the following information: the operator code, the operator name and the reaction or regulatory action. These are tab delimited, so they can be obtained from a database that can be incorporated into a spreadsheet such as Excel. For example, the information for the acetolactate synthase and hexokinase reactions is encoded as follows:

4.1.3.18 acetolactate synthase 2-acetolactate + CO_2 = 2 pyruvate

2.7.1.1 hexokinase D-glucose + ATP = ADP + D-glucose 6-phosphate

Starting from the left, the first piece of information is the operator code (EC number). This is followed by white space in the form of a tab. Next comes the name of the operator (enzyme), also followed by white space in the form of a tab. The last item of information is the reaction. A reaction is divided into a left hand side (LHS) and a right hand side

(RHS) by an equals sign with a space on each side of it. Compounds are separated by a plus sign with a space on each side of it, except where there is a single compound on either or both sides. Each compound is preceded by its stoichiometric coefficient and a space if the coefficient is greater than one, for example, "2 pyruvate". Otherwise, the stoichiometry is implicit and is understood to be one (e.g., D-glucose). This information is input as a single string with one such string per line of input. A metabolic pathway is encoded as a specific list of such strings. One can also access a single operator-mediated event (unit of input) or any combination of such events.

For computational purposes and display, modulators for the phosphofructokinase (PFK) reaction, for example, are represented in the following way (see 3.7.1):

2.7.1.11 PFK {+}AMP + {+} F2,6BP : {-}ATP + {-}citrate

This representation is used to distinguish modulators from metabolites in the graph models and is the input format for regulatory actions. The format for events (regulatory actions and/or reactions) must be strictly adhered to if the incoming information is to be processed correctly. However, certain violations of the format can be detected by the system. If an EC number is not classified, one may enter a generic term, e.g., a.b.c.d. If an EC number is only partially classified, the digit(s) that are known may be entered, e.g., 2.b.c.d, 3.4.c.d, etc. Uncatalysed reactions, receptors and membrane operators are assigned to arbitrary classes 0, 8 and 9, respectively. These are not internationally accepted codes but are defined here for the purpose of the project. A transporter classification (TC) number consists of five elements separated by points (Busch and Saier, 2004). For example, 2.A.1.1.28 represents the erythrocyte/brain hexose facilitator, Glut1 of Homo sapiens. However, the information on transporters as represented in the Membrane Transport Protein Classification Database (www.tcdb.org) is unsuitable for drawing pathways. Therefore, reaction 2.A.1.1.28 was converted into a suitable format as follows:

2.A.1.1.28 hs_Glut1 e-D-glucose = D-glucose

The 'Example' field in the transporter database includes a specific reference to species. It was possible to represent this information here in the extended operator (see Section 3.7.2) hs_Glut1, where 'hs' represents Homo sapiens. Furthermore, the transporter database does not have a field for specific transport reactions but only contains generalised transport

reactions as comments. However, it was necessary to represent this reaction explicitly here for the purpose of pathway drawing. The prefix 'e' on the left hand side of the reaction denotes the extracellular form of D-glucose and prevents D-glucose from joining with itself in the same reaction in the graph representation. On the other hand, a corresponding prefix to denote intracellular D-glucose was not included in order to link the transporter reaction with the first reaction (2.7.1.1 above) on the glycolysis pathway. Once transporter activities are encoded as transport reactions, they can be incorporated into automatically generated metabolic maps (Lee et al., 2008).

4.1.3 Database

The 'ReactionString Database' contains seventeen primary files. These are divided into data, references and information (on selected classical pathways). Two large data files (General.dat and User.dat) and a variable search file (UserSearch.txt) contain operator-mediated reactions, while a small data file (Effectors.dat) contains effectors. Another variable search file (Pathway.txt) contains operator codes and operator names only. There are, at present, four classical pathway data files, one each on glycolysis, gluconeogenesis, the pentose phosphate pathway, and the urea cycle. There is also a file (see Appendix A4) containing a list of the data files (DataFileNames.txt). There are five pathway information files, one each on the aforementioned pathways, and one containing pathway EC numbers (see Appendix A4). There are two reference files. One contains general literature references while the other contains references that are specific to reactions in the data files. The contents of the primary files were created manually and can be edited by altering, deleting or adding information. However, the contents of pathway data files can be created automatically as described in 4.3.9 below. Five secondary files are automatically generated when the program is run. These files contain: a copy of one of the data files (at any one time), HTML reaction code, search specific references, operator names in alphabetical order, and primary substring search results.

The general data file (General.dat) includes the reactions of the Nicholson Metabolic Pathways wall chart (Nicholson, 2000). These were downloaded from the *Enzyme Nomenclature* web site (http://www.chem.qmul.ac.uk/iubmb/enzyme/) using the EC numbers from the chart. This is not a simple data set. For example, a number of reactions are available only as natural language descriptions. Furthermore, some entries contain alternative substrates in the same equation (e.g., 'aldehyde or ketone'). Other equations

contain indefinite articles, e.g., an alcohol. Several equations contain the terms 'reduced acceptor' and/or 'acceptor' while some reactions have unidentified products. In addition, certain representative reactions contain terms of wide specificity, e.g., D-hexose. These are all referred to as *unsuitable reactions* (for drawing metabolic pathways).

Glucose-1-phosphatase had the incorrect EC number 3.1.3.1, which, according to the Enzyme Nomenclature handbook (Webb, 1992) is 3.1.3.10. This has been corrected. There was only one reaction for transketolase. The user data file is a simple version of the general data file, i.e., the 'unsuitable reactions' have been removed or replaced manually. Furthermore, several reactions of trehalose metabolism, an additional reaction for transketolase, two ligand-receptor reactions, one transporter reaction and two spontaneous (uncatalysed) reactions have been added to this file. The pathway data files (see Appendix A2) contain the reactions of the relevant pathways in the correct sequence for the convenience of the user. Including membrane operators in the urea cycle represents a compartment (mitochondrion). In the above data files a record contains an operator-mediated reaction as described in 4.1.2 above. The effectors data file (Effectors.dat) contains certain modulators including those for glycolysis and gluconeogenesis. (see Appendix A2) In this file each record contains a regulatory action as described in 4.1.2 above. With few exceptions, the names of compounds in the data files are in accordance with the notation used in the Enzyme Nomenclature Database, i.e., Recommendations (1992) of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (Webb, 1992). These do not necessarily match the corresponding names on the Nicholson Metabolic Pathways wall chart (Nicholson, 2000). By default, all the information in these files is accessed by the program. However, a system of filters enables a user to select only the reaction(s) of interest.

The "UserSearch.txt" search file contains reaction strings, which are searched for in the data files. The other search file (Pathway.txt) contains operator codes and operator names only, which are also searched for in the data files. The use of the former includes the elucidation of biochemical function in the data file of a given organism. The use of the latter includes the automation of specific pathway file creation, i.e., given a set of operators (and/or their codes) that constitute a pathway, find the corresponding reactions and thus make the pathway. The contents of these two files vary according to the specific search in question.

The pathway information files contain a brief textual description of the pathways in the four classical pathway data files, in addition to a file containing specific sequences of EC numbers, which correspond to reaction sequences in these pathways. The general reference file (GenRef.txt) contains references to textbooks and journal articles on biochemistry and bioinformatics (see Appendix A3). The other reference file (SpecRef.txt) contains references that are specific to reactions in the data files (see Appendix A3). These were taken from the handbook on Enzyme Nomenclature (Webb, 1992) and, as stated before, were manually entered into the file. PMID numbers were not available at the time of compiling the file; otherwise, the process would have been automated using the EC list. However, now the process may be automated because of developments in the Enzyme List.

Selecting a file from the " Files" menu (see Fig.4.2 below) generates the secondary Reactions.txt file to which the selected file is written. The program then reads the secondary file. In this way the same functions of the program can be used with different files and the user can experiment with the secondary file (e.g., remove a reaction) without affecting the primary one. The HTML file (html.txt) is generated when the reactions are displayed. This is used to produce a display of the selected reactions in an applet. The file containing search specific references (selectedRef.txt) is generated when individual reactions or groups of reactions are selected via menu items in the "Operators" menu or via the 'Reactions or regulatory actions' menu item in the "Events" menu. Primary substring search results are written to a file (Inclusion.txt) for further processing. While operator codes are in order of increasing code number in the General.dat, User.dat and Effectors.dat files, the operator names are not in alphabetical order in these files. Therefore, when access to the Reactions.txt file (to which the data files are written) via operator names is requested, the program first sorts these names in alphabetical order and writes them to another file (OperatorNames.txt) for further processing.

The database has limited information on reactions and pathways. Therefore, since the results of a search are dependent on the content of the database, pathways that involve enzymes that are not included in the database cannot be generated. However, the purpose of the database is simply to illustrate how the system works. This is the situation at present, but the database can be expanded in the future, i.e., it can readily be extended with new reactions and other data and, indeed, can contain all known reactions, regulatory

actions and pathways, including transporter reactions suitable for drawing metabolic pathways.

4.1.4 Software and system

The system was designed to provide a diagrammatic construction of biochemical events and metabolic pathways using Java-based software. For this purpose, a Java program drawing reactions from their formalised description (see 4.1.2 above) was developed. This program generates reaction sequences from the lists of reactions (and regulatory actions) contained in the data files and displays them on computer. It assembles the reaction equations into a graph in an operator-centric paradigm. Shared objects link the nodes and, thus, the events. The program consists of several modules. These include a parser for the decomposition of reaction information, a drawing system for the visualisation of the reactions, and a front end for interaction with the user. The latter consists of a number of graphical user interfaces that were designed to allow the user easy access to the system. A graphical user interface (GUI) typically consists of a container (e.g., a window or frame) into which components (e.g., buttons, menus and text fields) are added.

The initial GUI consists of a window with menus (see Fig. 4.2 below). The menus available are: Files, View File, Operators, Events, Systems Search, Substrings, Classical Pathways, Information, References, Tools, and Show. A data file may be selected from the "Files" menu and the selected file may be viewed on the Java console or in the text area of another scrolling window as chosen from the "View File" menu. Operators may be accessed by code or by name. The reactions and regulatory actions are accessed via the "Events" menu. A set of reactions, a metabolic pathway or subnet (as defined in the program or in the search files) may be searched for in a target data file via the "Systems Search" menu. Substring searches allow the user to find reactions that exclude and/or include a specific substring, and the option of visualising them in the form of maps. Classical pathways (with or without effectors) may be accessed from individual pathway files via the "Classical Pathways" menu. The brief descriptions of the pathways are also available from this menu. The user may gain access to general, reaction-specific and search-specific references via the "Reference" menu. Information on the system is displayed in a text area. There is only one tool, i.e., to clear the information on how to use

the database. The 'Show' menu allows the user to view and copy the HTML file that is automatically generated when the reactions/regulatory actions are displayed.

A file is selected via a GUI (Fig. 4.3) that is generated by clicking the 'Select any data file' menu item in the "Files" menu. The name of the file (e.g., User.dat) is entered into a text field. Then pressing a button, selects the file. However, the general and user data files may be accessed directly by clicking the corresponding menu item in the same menu. In either case, the selected file is first read and then written to the 'Reactions.txt' file, which is used by the program. The user is alerted to particular errors in the input, should they occur, and corrective procedures are suggested (see 4.3.3. below).

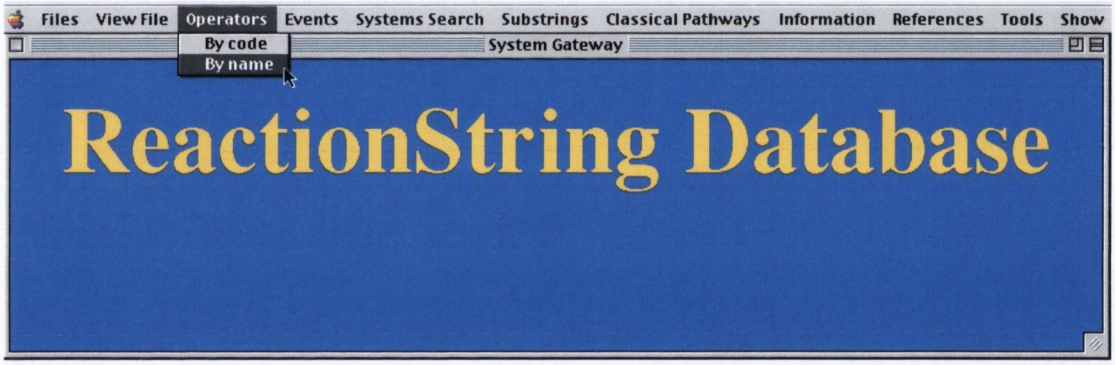


Fig.4.2. The main GUI consists of a window with menus. The ReactionString Database (RSD) may be searched using the menus, the results of which are given as text and/or displayed in the form of maps.

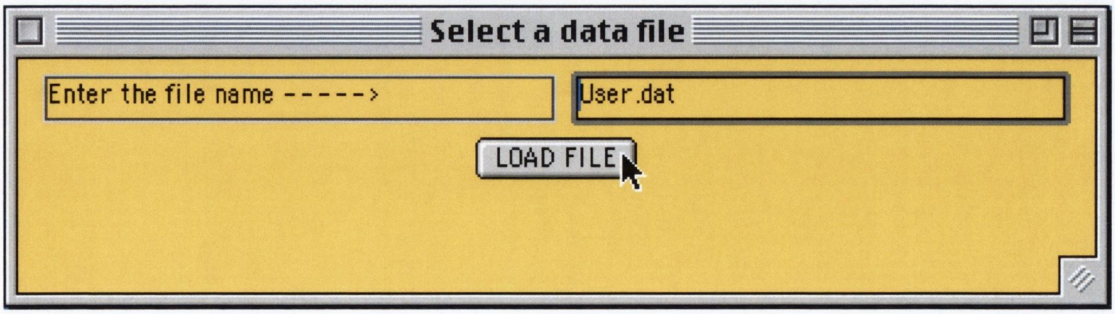


Fig.4.3. GUI for the selection of data files. The name of the file (e.g., User.dat) is entered into the text field on the right hand side. Then clicking the LOAD FILE button writes the selected file to the 'Reactions.txt' file, which is accessed by the program.

Three separate graphical user interfaces allow access to individual operators and reactions or regulatory actions via scrolling selection lists (see Sections 4.3.5 and 4.3.6). One or more reactions may be selected from these lists using the operator codes (Fig. 4.4), which include EC numbers, or the operator names (Fig. 4.5), which include enzyme names, or by directly clicking the reactions (Fig. 4.6). Thus, a user may build up reaction sequences and increment them piecemeal, or generate textbook pathways, by selecting items from these lists. The operator codes are in order of increasing code number while the operator names are presented in lexical order. Furthermore, these lists allow easy control of large data sets. For example, the user only selects the reactions of interest and thus can build up particular sequences of reactions. An advantage of such lists is that they display all the items in a particular file – if they are not in the list, they are not in the file, and if the file is edited, they will automatically reflect the changes. Another advantage is that they permit easy selection of the items – simply click one or more items. Furthermore, such lists are not fixed, but depend on the file that is being used.

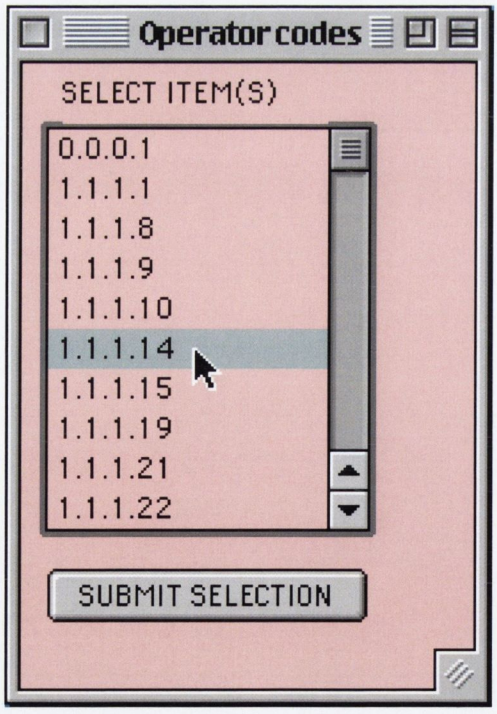


Fig.4.4. GUI for the selection of operator-mediated reactions via operator codes. Clicking on one or more codes (e.g., the EC number 1.1.1.14) selects them. Then clicking the SUBMIT SELECTION button displays a map of the corresponding reaction(s) in a window and shows the reaction information in a separate window.

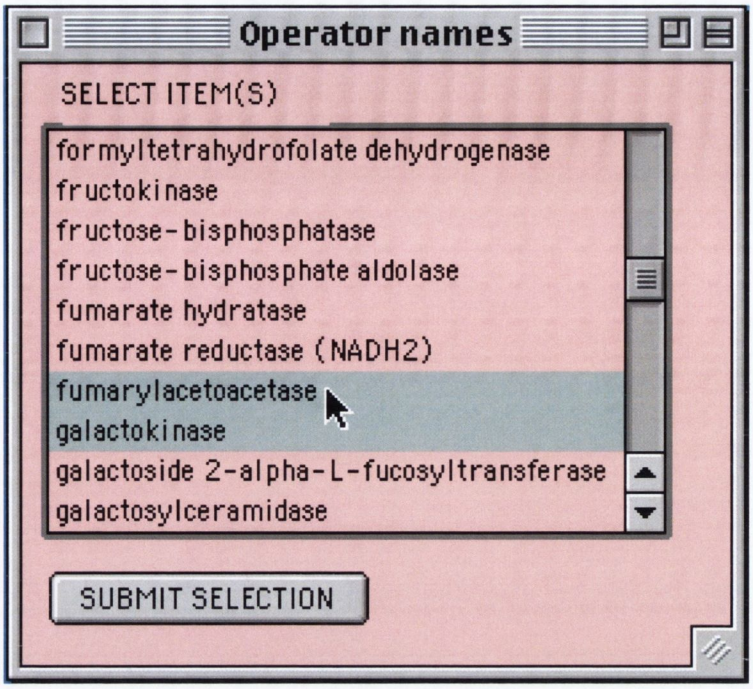


Fig.4.5. GUI for the selection of operator-mediated reactions via operator names. Clicking on one or more names (e.g., fumarylacetoacetase and galactokinase) selects them. Then clicking the SUBMIT SELECTION button displays a map of the corresponding reaction(s) in a window and shows the reaction information in a separate window.

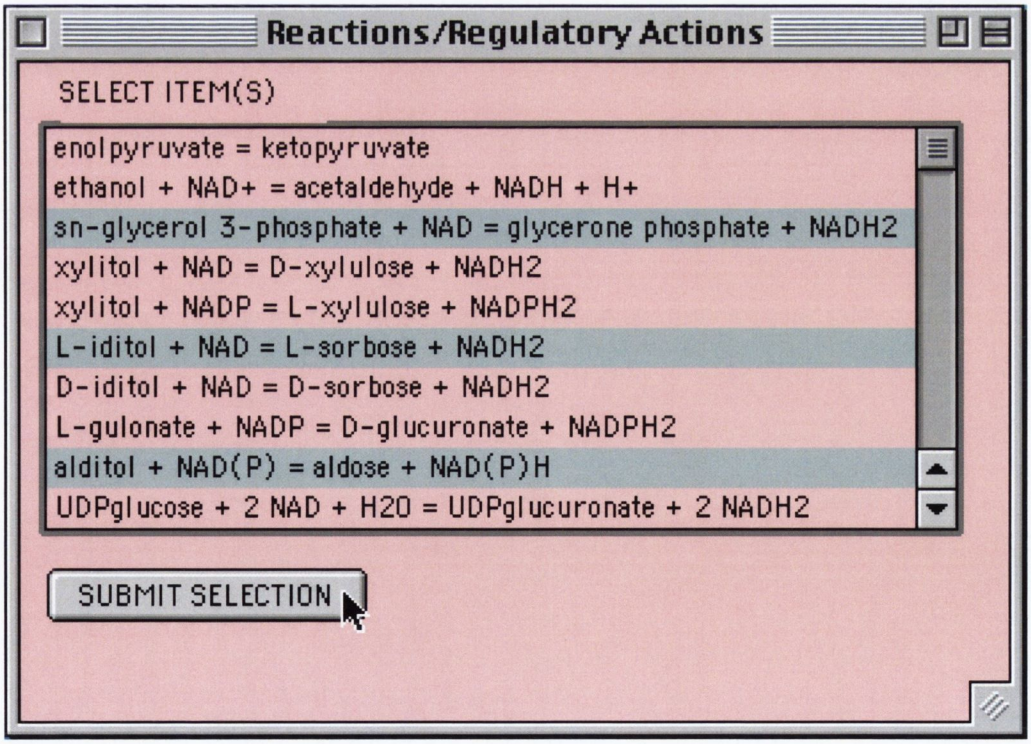


Fig.4.6. GUI for the selection of operator-mediated reactions or regulatory actions. Clicking one or more reactions (e.g., the reactions that are highlighted) selects them. Then clicking the SUBMIT SELECTION button displays a map of the corresponding reaction(s) in a window and shows the reaction information in a separate window.

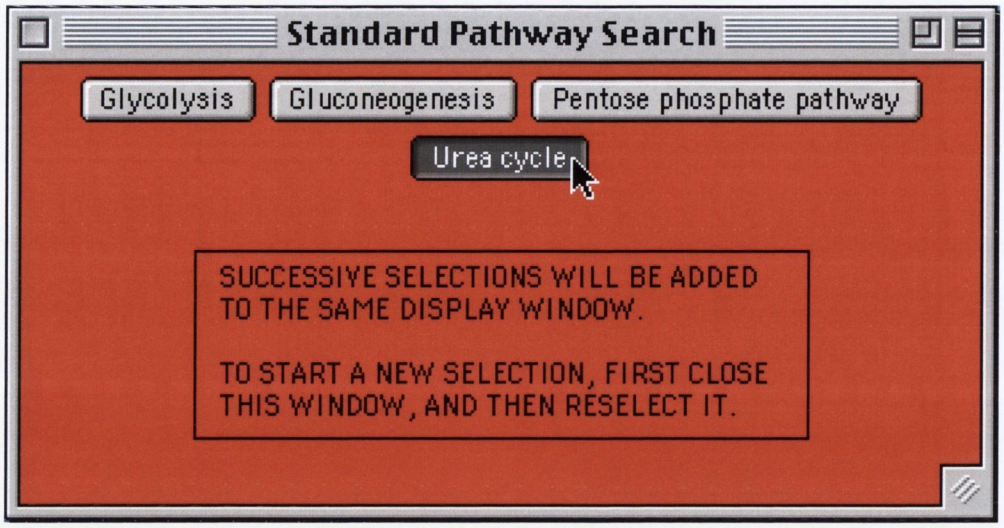


Fig.4.7. A search GUI for the extraction of 'standard' metabolic pathways from the "Reactions.txt" file. Clicking a button (e.g., Urea cycle) displays the corresponding pathway in a window and shows the reaction information in a separate window.

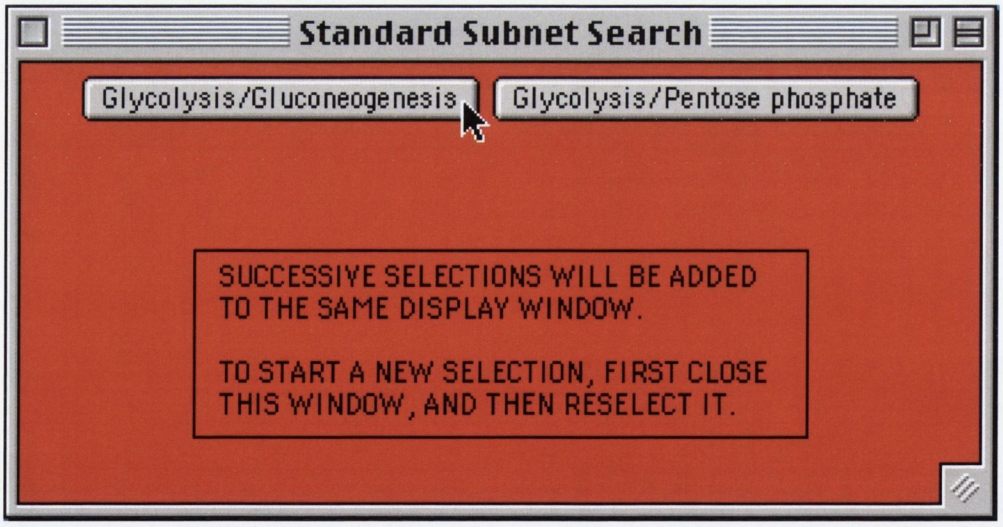


Fig.4.8. A search GUI for the extraction of 'standard' metabolic subnets from the "Reactions.txt" file. Clicking a button displays the corresponding subnet in the form of a map in a window and shows the reaction information in a separate window.

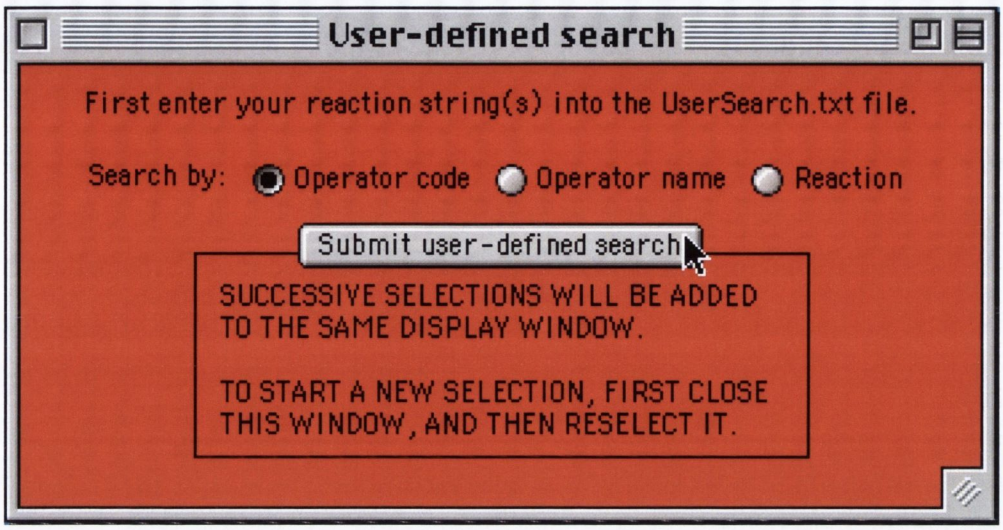


Fig.4.9. A search GUI for the extraction of user-defined reaction sequences from the "Reactions.txt" file. The radio buttons are used to search by code, operator or reaction. Clicking the 'Submit user-defined search' button enters the reaction strings that are to be searched for. The reactions that are found are displayed in the form of a map in a window and the corresponding reaction information is shown in a separate window.

The program contains methods to search for 'standard' pathways and subnets in the "Reactions.txt" file. A *standard pathway* may be considered as an accepted defined pathway that is not necessarily specific to a particular organism, but which may be contained within an organism in the same or a different (modified) form. A *standard subnet* is a collection of such pathways. These methods facilitate the searching of an organism's genome for certain metabolic functions. The searches are performed via graphical user interfaces. The pathways that can be searched for by this procedure are glycolysis, gluconeogenesis, the pentose phosphate pathway, and the urea cycle (Fig. 4.7). The subnets that can be searched for are glycolysis with gluconeogenesis, and glycolysis with the pentose phosphate pathway (Fig. 4.8). The number of different pathways (and subnets) that can be searched for in this way is limited in the present program but could readily be altered to increase the range. However, the available pathways and subnets illustrate how this part of the system works. A broader range of pathways, including user-defined ones, can be searched for by using the multiple-choice scrolling selection lists already referred to above (Figs. 4.4 – 4.6).

It was considered important to include functionality that allowed the user to enter reaction strings into a working file that could be searched for in the "Reactions.txt" file. This allows the user to conveniently search for any sequence of reactions, including organism-specific metabolic pathways as, for example, when the "Reactions.txt" file contains all the reactions of a particular organism. This search is performed via a GUI (Fig. 4.9 above). Radio buttons allow the user to search by code, operator or reaction. The 'reaction' option finds a reaction with the LHS and RHS as specified in the search file, whereas the other options do not discriminate between left and right sides. Furthermore, it was decided to make the creation of pathways more convenient by automating their retrieval from one of the large data files (e.g., User.dat). This is achieved via the GUI shown in Fig. 4.10 below. In this case the search file contains the operator codes and names only, i.e., from given a set of codes or operators, find the corresponding reactions (pathway) in a target file. When the reactions are displayed, the pathway reactions with their connectivity and directionality can be checked with reference to the literature. If the user is satisfied with the results, the pathway (with or without modifications) may be saved as text to a named pathway data file.

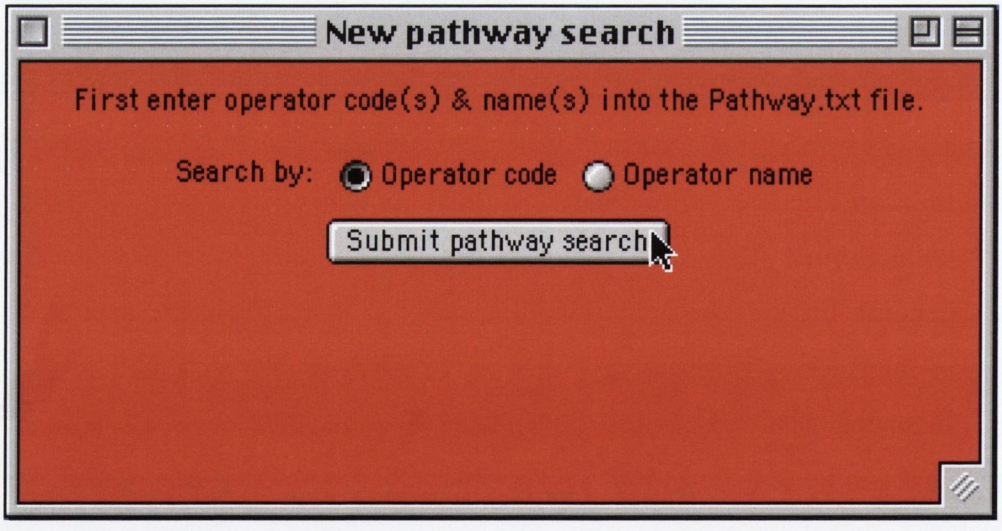


Fig.4.10. A search GUI for the retrieval of pathways from the "Reactions.txt" file. The radio buttons are used to search by code or operator. Clicking the 'Submit pathway search' button enters the codes or operators that are to be searched for. The corresponding reactions that are found are then displayed in the form of a map in a window and the corresponding reaction information is shown in a separate window.

As stated before, four separate classical pathway files are included, with one pathway per file. The urea cycle includes membrane operators while glycolysis includes an uncatalysed reaction. They can be accessed directly, with or without effectors, via the "Classical Pathways" GUI (Fig. 4.11 below). A class frequency histogram, including EC classes, is automatically generated for each pathway (see Fig. 4.27 below). These single pathway files can also be searched (via the 'Reactions.txt' file) for standard pathways. This allows the pathway search capability to be checked, since searching a pathway for itself should return the pathway. Also, searching one pathway for another (e.g., gluconeogenesis for glycolysis) returns those reactions that are common to both pathways. One can access information on these pathways via the GUI shown in Fig. 4.12 below. Clicking on a pathway button gives a description of the corresponding pathway in a window. By clicking on the 'Pathway EC numbers' button one can access specific sequences of EC numbers that correspond to reaction sequences in the pathways. Selecting the operator codes according to these EC numbers (Fig. 4.4) allows the user to build up the relevant pathway(s) either step by step or all at once.

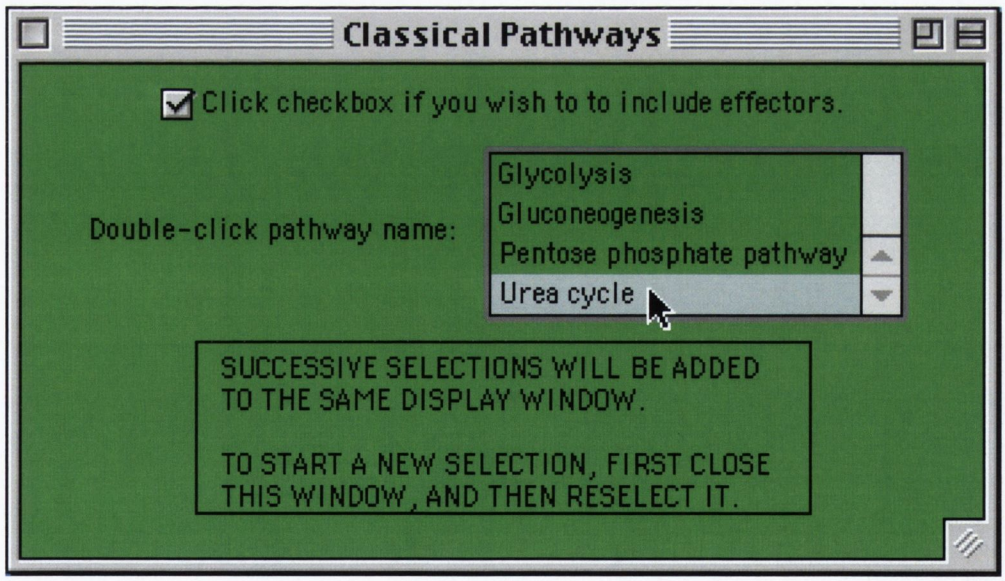


Fig.4.11. GUI for selection of classical pathways from the pathway files. Double-clicking a pathway name (e.g., Urea cycle) displays the pathway in a window and shows the reaction information in a separate window. Clicking the checkbox (at the top of the GUI) before selecting the pathway includes the corresponding effectors. A class frequency histogram, including EC classes, is automatically generated for each pathway.

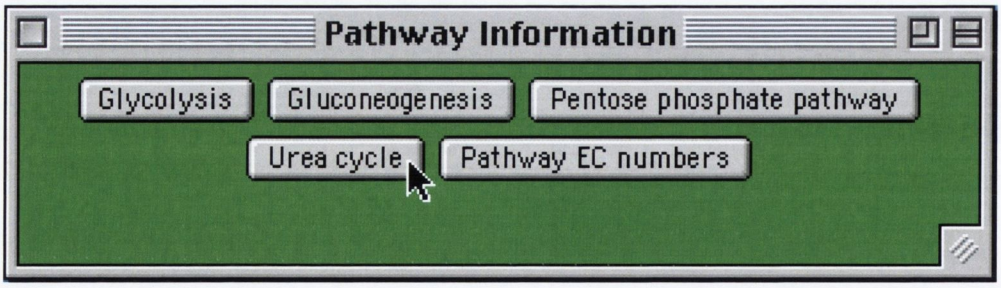


Fig.4.12. GUI for retrieval of information on selected classical pathways. Clicking a button gives the corresponding information.

Information dialogue (or dialog) boxes are repeatedly used in the program to display messages to the user. These are *modal* dialogue boxes, which block input to the rest of the application until they are dismissed (Fig. 4.13 below). Clicking the OK button removes the dialogue from the screen and relinquishes any operating system resources that were allocated for it.



Fig.4.13. An information dialogue box displays a message to the user and sits there until it is dismissed by clicking the OK button.

Information on using the system may be displayed, as required, in a text area in the 'System Gateway' GUI of the ReactionString Database, as shown in Fig. 4.14 below. Vertical scrollbars are used to display further parts of the text. A description of each menu is given. Instructions on how to clear the text area are also included.

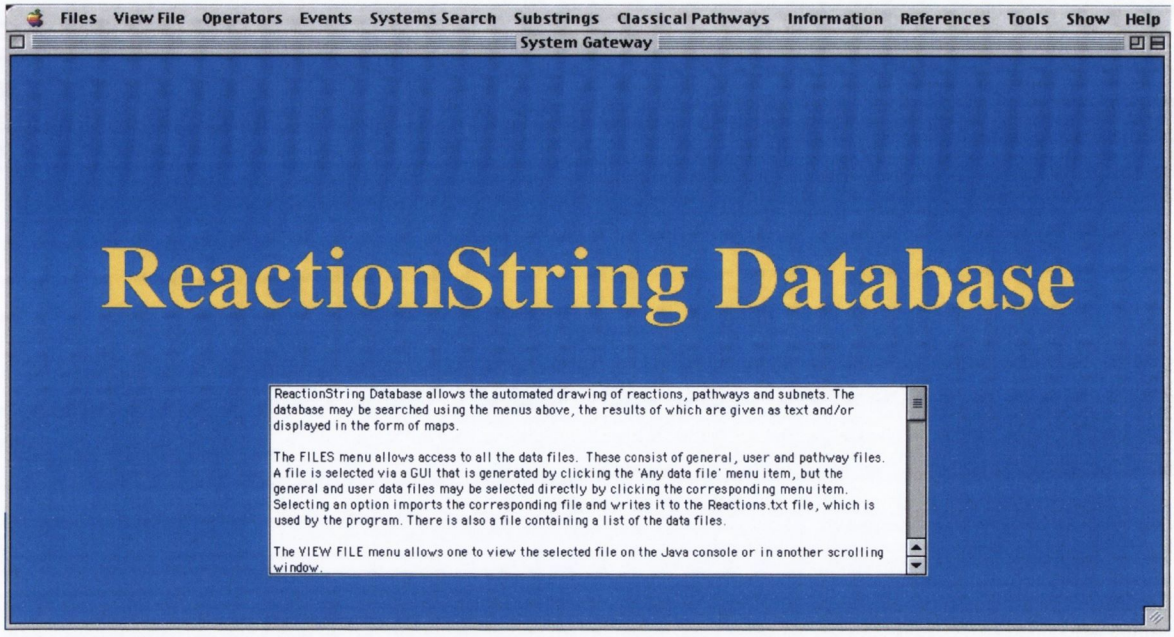


Fig.4.14. Text area containing information on using the system is shown in the main 'System Gateway' GUI of the ReactionString Database. Further parts of the text can be displayed using the vertical scrollbars. Instructions on how to clear the text area are included.

Four separate graphical user interfaces allow four different options for the substring searches. These may be called up collectively as shown in Fig. 4.15, or separately. The substring searches use filters, which allow the user to find reactions that include or exclude a specific substring. The 'Find' option allows one to view the result of a search as text only. Although this may give useful information in its own right, it also allows the user to assess the extent of a dataset before displaying the reactions. Just finding reactions containing substrings avoids the danger of selecting too much for display. In addition, the 'Find and display' option allows the reactions to be displayed as maps. These searches also allow the user to select any compound (enzyme or metabolite), e.g., D-glucose 6-phosphate, and view all the reactions it is involved in. Typing a number followed by a space helps to find reactions containing compounds with a stoichiometry of that number. For example, typing the number 4 followed by a space finds the following reactions in either the general data file or the user data file:

$$4 \text{ ferrocytochrome c} + O_2 = 4 \text{ ferricytochrome c} + 2\ H_2O$$

$$4 \text{ porphobilinogen} + H_2O = \text{hydroxymethylbilane} + 4\ NH_3$$

$$\text{uroporphyrinogen III} = \text{coproporphyrinogen} + 4\ CO_2$$

.

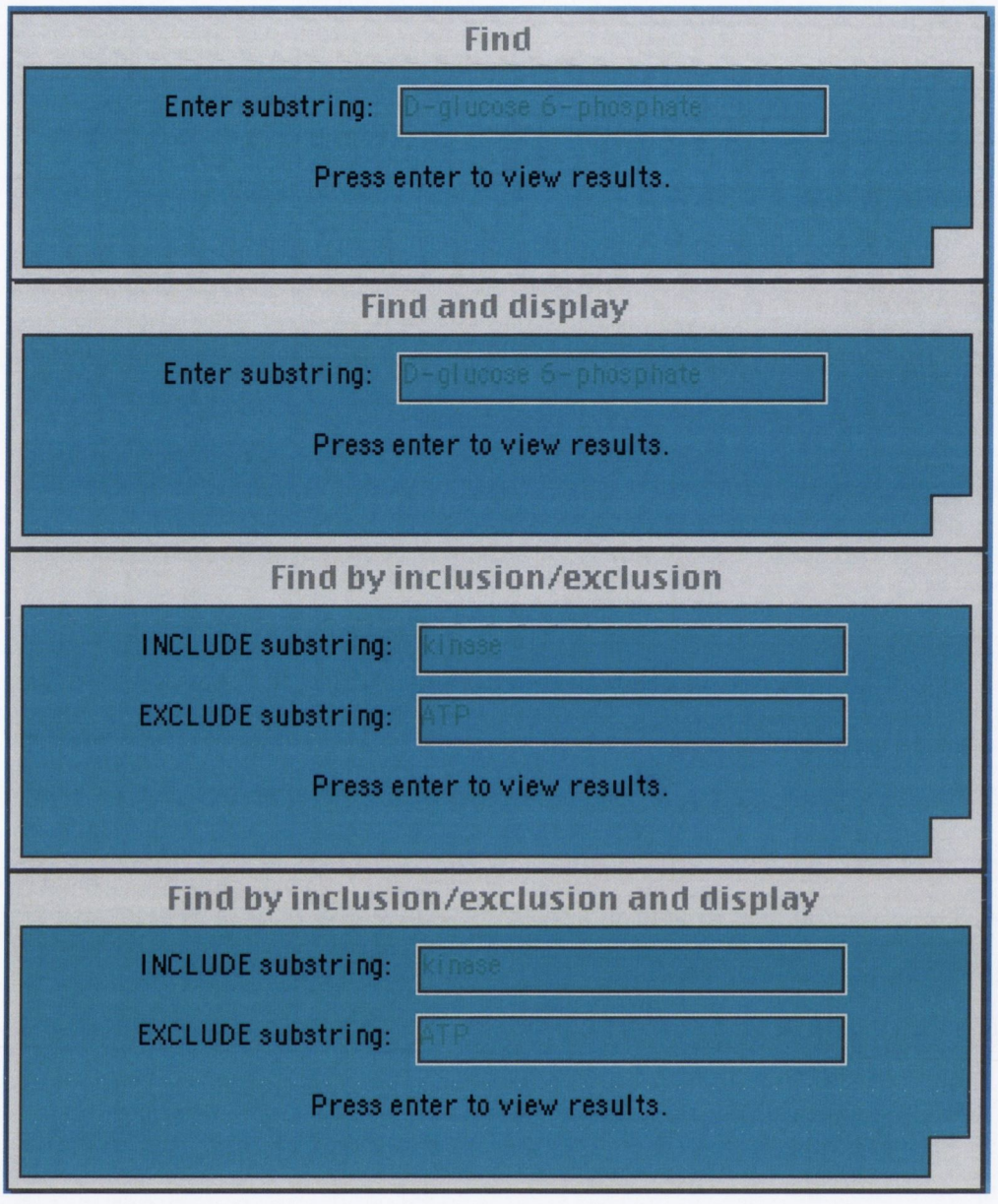


Fig.4.15. Graphical user interfaces for the execution of substring searches in the application. The 'Find' option returns the result of a search as text only. In addition, the 'Find and display' option allows the reactions to be displayed as maps. The inclusion/exclusion GUIs refine searches by also excluding a particular substring. The above GUIs may be called up separately or all together (as shown above).

There is at least one compound with a stoichiometry of 4 in each one of these reactions. A substring search of a pathway file helps to find a particular compound on that pathway. The inclusion/exclusion GUIs refine searches by also excluding reactions containing a particular substring. For example, the inclusion of the substring 'kinase' finds all reactions catalysed by a kinase, and exclusion of the substring 'ATP' returns a subset of these that

do not involve ATP. The inclusion/exclusion GUIs can also be used to exclude reactions that are in a form unsuitable for drawing pathways. For example, typing 'acceptor' in the exclusion substring field filters out reactions containing this named entity as a participant

A graphical user interface for running and interacting with the applet (applet viewer) is shown in Fig. 4.16 below. A brief set of instructions is given across the top of the window. The reactions and/or regulatory actions generated by the application and copied to the applet are accessed via the 'GET_EVENT(S)' button. The GUI includes a display area for the selected reaction(s) and/or regulatory action(s), and check boxes to show certain parameters as required (e.g., stoichiometry) and to control particular aspects of the display (e.g., motion of non-operator nodes). It also contains links to selected external metabolic databases and a drop down list for selection of simple layouts for the graphs. The system restricts the use of menus with applets. However, the drop down list provides something similar. A more detailed description is given in section 4.5 below.

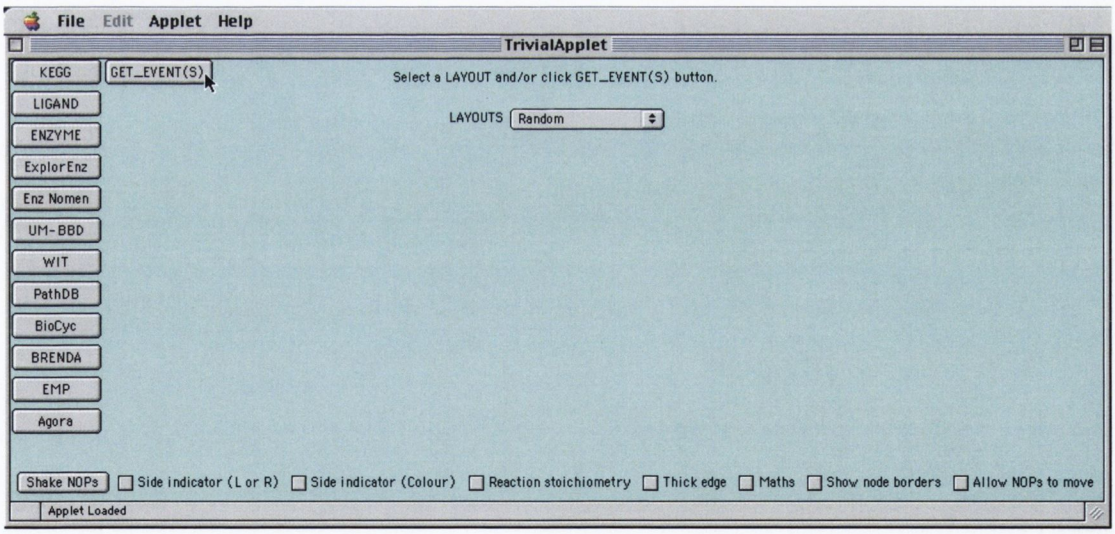


Fig.4.16. Graphical user interface for interaction with the applet. Instructions are given across the top. A drop down list at centre top allows selection of simple layout algorithms. The reactions and/or regulatory actions generated by the application and copied to the applet are accessed via the 'GET_EVENT(S)' button inside top left (indicated by the mouse pointer). The button and check boxes at the bottom are connected with certain aspects of the display, which will be explained later. The buttons at the extreme left allow access to external databases, which are listed later.

§4.2 Implementation

4.2.1 General description

A Java application and applet were developed to facilitate the automated connection of compounds in the form of maps. The primary tasks in the application are reading and writing reaction data from and to text files, parsing the input files and visualising complex reaction sets. Graphical user interfaces search the system in response to mouse clicks, selections from lists, and textual input, and generate a structural and textual representation of reaction sets, pathways and subnets from the input files. An HTML file (html.txt) is generated by the application. This may be manually copied and entered into the applet so that the output from the application can be viewed and interacted with over the Internet.

4.2.2 Classes and objects

The application and the applet consist of sets of inter-dependent classes. A *class* is a template from which like objects are created. An *object* (often referred to as an *instance* of a class) is a combination of data (variables) and actions (methods). Information passed to a method is received by variables called *parameters*, which are specified within parenthesis following the name of the method. A reference to an object is stored in a variable of the same type as the class. An object is created using the 'new' keyword, which calls a *constructor* method of the class (with or without parameters) to initialise the new object. Constructors have the same name as the class. If a constructor is not declared in a class, the compiler automatically declares one with no parameters. Creating an object is also called *instantiating* it. Objects of type *String* store character strings. A *String* may be created in the same way as any other object:

```
String str = new String("Reactions");
```

The 'str' variable is declared and initialised to 'Reactions'. However, it is more usual, as in the present application and applet, to initialise the *String* by setting it equal to "Reactions" after the declaration, as in: String str = "Reactions";.

4.2.3 Packages

Java has a number of core packages that can be used by any program. A *package* is a collection of classes that have similar roles. The packages used in the application, "ReactionString Database" (RSD), are 'java.lang' (routine language functions), 'java.util' (additional useful utilities), 'java.io' (input and output), 'java.awt' (graphics and graphical user interfacing), and 'java.awt.event' (handling events from the keyboard, mouse, etc). The packages used in the applet, "WebPathway" (WP) are 'java.applet.Applet' (creating Java programs to run on the Web) and 'java.net' (networking), in addition to 'java.util', 'java.awt', and 'java.awt.event' as already described. Access to packages is gained using the following form: import java.package.*;. The 'java.lang' package is automatically imported. Therefore, it is not included in the import list, which is placed at the top of the Java code.

4.2.4 Layout managers

Layout managers control the positioning of components in a window. Java has five such managers in the package 'java.awt', but only two of them (flow and border) are used in the present system. Both of these are used in the application but only the border layout is used in the applet.

When using the *FlowLayout* class, components are added to a frame at their preferred size, one after another in rows, and when a row is full the next component is added to the next row. The *FlowLayout* class has three constructors but only two of them are used in the RSD application. These are:

- *FlowLayout()*. Components are aligned in the centre of each row. Horizontal and vertical gaps are both 5 pixels (see Fig. 4.12 above).
- *FlowLayout(int align)*. Components are aligned at the left, in the centre or at the right of each row, according to whether align is *FlowLayout.LEFT*, *FlowLayout.CENTER*, or *FlowLayout.RIGHT*. Horizontal and vertical gaps are both 5 pixels.

The *BorderLayout* class allows up to five different zones (in which to place components) to be arranged within a container: four narrow, fixed-width zones at the edges and one large zone in the middle. The four edge zones are named: North, South, East and West.

The middle zone is called Center. North and South span the entire width of the top and bottom of the container, respectively. East, Center and West then occupy the space between North and South. In the WP applet, East and West are not used, so Center expands to fill in these zones.

§4.3 The RSD Java application

4.3.1 File input and output

The package 'java.io' supports Java's basic I/O (input/output) system. Fundamental to this package is the concept of a stream. A *stream* is a flow of information between a program and an external data repository, such as a file on disk. An *input stream* flows from the external source into the program. An *output stream* flows from the program to the external destination. The general format used in the present RSD application to *read* a file and process its contents is:

```
try{
    File f = new File("filename");
    FileReader fr = new FileReader(f);
    BufferedReader in = new BufferedReader(fr);
    String st = in.readLine();
    while(st!= null) {
        //process st
        st = in.readLine();
    }
    in.close();
}
 catch(Exception e) {
        System.out.println(e.getMessage());
 }
```

The *FileReader* class opens the text-based stream for reading. It has two main constructors, both of which throw a *FileNotFoundException* and consequently, is used within a *try-catch* block. An *exception* may be regarded as a run-time error. Before an

exception can be caught, some code somewhere must throw one. To read the data stream, an instance of the *BufferedReader* class is used. This class takes an instance of the *FileReader* class as a parameter in its constructor and creates a buffer to facilitate processing of the input stream. The *readline()* method reads the file line by line. When the end of the stream is reached it returns *null*. Because *readline()* throws an *IOException* it is also used within the *try-catch* block. Two forward slashes (//) begin the comment that indicates where to place the code to process the input. When processing is complete, the input stream is closed by the *close()* method. The dot operator (.) calls a method for a particular object. A semi-colon terminates an instruction and two curly brackets enclose a *block* of code.

The general format used in the present RSD application to *write* to a file is:

```
try{
    FileOutputStream ostream = new FileOutputStream("filename");
    PrintWriter pw = new PrintWriter(ostream);
    pw.println(" Hello");
    pw.flush();
    ostream.close();
    }
    catch(Exception e) {
        System.out.println(e.getMessage());
    }
```

The *FileOutputStream* writes binary data to a file. To write text, it is used with a *PrintWriter*. Its constructors do not throw a *FileNotFoundException*. If the file does not exist, the *FileOutputStream* creates it. However, the *FileOutputStream* can throw an *IOException* if some other I/O error occurs, so this exception has to be handled. The *println()* method takes a string of characters as a parameter (e.g., the string literal " Hello"), writes it to the file and finishes the line. The *flush()* method forces any buffered output to be written and the *close()* method closes the stream.

The flow of information between classes and between classes and files is shown in Fig. 4.15 and described in 4.3.22 below. The source code for the application may be viewed in Appendix A4. A description of the classes follows.

4.3.2 Class *TrivialApplication*

The class *TrivialApplication* inherits variables and methods from the class *Frame* in the package 'java.awt', i.e., the existing class *Frame* is extended to provide additional features in *TrivialApplication* (*TrivialApplication* extends Frame). This class also implements an *ActionListener* and a *WindowListener*. An *ActionListener* defined in 'java.awt.event' is an interface that that must be implemented in order to handle an *ActionEvent*, such as that generated by pressing a button. It contains a single method called *actionPerformed()*, which is automatically called when the button is pressed, to do whatever is supposed to be done. The 'java.awt.event' package also contains the interface *WindowListener*, which defines seven methods. All of these are necessarily implemented. However, in the present class, only the *windowClosing()* method is used. The rest are implemented as empty methods. The *windowClosing* event occurs when the user shuts down a window by using the small square button at the upper left corner of the window on the Mactinosh (or the equivalent on a PC). The *windowClosing()* method is used in many of the classes that follow, but will not be referred to again. The function of the present class is to set up the main GUI with menus (see Fig. 4.2 above) that are the starting points for the exploration of reactions and pathways. The contents of the menus are stored in arrays, which are collections of similar objects. For example, the contents of the "Operators" menu are stored in a one-dimensional array that is declared and initialised as follows:

```
String menuOperator[ ] = {"By code","By name"};
```

The class *TrivialApplication* contains the *main()* method, which is the entry point for the application. The computer starts the execution of the program at the first statement in this method and creates a graphics frame as follows:

```
public static void main(String args[ ]) {
    Frame f = new TrivialApplication();
    int w = 900;
    int h = 500;
    f.setSize(w,h);
    f.setVisible(true);
}
```

After the frame object is created, it invokes its *setSize()* method to set the size of the frame window to 900 pixels wide and 500 pixels high. The method *setVisible()* displays the frame on the screen provided the Boolean parameter is set to true. The display window in which the reactions are to be drawn is created separately as required (see Section 4.3.20). The *main()* method is declared *public*, since it is called by code outside of its class when the program is started. The keyword *static* allows *main()* to be called without having to instantiate a particular instance of the class, because the Java interpreter invokes it before any objects are made. The keyword *void* informs the compiler that *main()* does not return a value. The method accepts one parameter called 'args' that is an array of *String* objects. However, the present application does not make use of this information.

In the present class various objects are created as instances of several other classes. For example, the object named reaction is created as an instance of the *Reaction* class as follows:

```
Reaction reaction = new Reaction();
```

The constructor method *TrivialApplication()* sets the title ('System Gateway') of the window, adds a *WindowListener* to it, sets the layout manager to null (since no manager is being used), sets its background colour to blue using the method *setBackground(Color.blue)* with parameter of type Color, and sets up a text area (for multi-line entry) to display information about using the system. The constructor also creates the menu-bar and the drop-down menus with their titles. It then adds the menu items to their respective menus, adds the menus to the menu-bar, and registers the menus as listeners. The system is operated and interrogated via these menus.

A version of the method *actionPerformed()* is implemented to detect which item is chosen from a particular menu. It then handles the situation by passing instructions to the relevant classes, invoking methods in their objects. For example, if the 'By code' item in the "Operators" menu is chosen, it calls the *opCode(String str, int counter)* method of the Reaction object named reaction (referenced by 'reaction') in the class *Reaction*, passing the parameters 'str' and 'counter' to it, where 'str' contains the string "Reactions.txt" and 'counter' contains the number of records in the file of that name. Similarly, the 'By name' item in the same menu and the 'Reactions or regulatory actions' item in the "Events" menu call their corresponding methods in the class *Reaction* and pass the same parameters to them.

When selecting items from the "Operators" and "Events" menus, the "Reactions.txt" file is automatically read to ascertain the number of records in it (which is indicated on the Java console) and to check that the correct number of tabs (2) is present in each record. This is done for all data files irrespective of how they are accessed. If, for example, the number of tabs in the first record is not correct, the following message is output to the Java console: "Input format error in tab(s) in line 1 of Reactions.txt file!" This is then corrected manually. A Boolean variable (that holds either the value 'true' or 'false') is set to 'false'; otherwise it is 'true' when the number of tabs is correct. In the latter case, the program continues as the *actionPerformed()* method creates a dialogue box and invokes *Show()* to make it visible. It also passes the relevant parameters to the constructor method *MessageDialog()* in the class of the same name. The constructor creates the user interface with a button and a label that displays a message requesting the user to wait for the complete list of codes, names or reactions/regulatory actions (e.g., Fig. 4.13 above). Clicking on the 'All regulatory actions' menu item in the "Events" menu generates a visual display of all the contents of the "Effectors.dat" file.

Any data file may be selected via a GUI (Fig. 4.3) by clicking the 'Select any data file' menu item in the "Files" menu. However, the general data file or the user data file may be selected directly by clicking the corresponding item in the same menu. The user is informed (on the Java console) which file has been selected and how to view it. In any case, the selected file is first read and then written to the "Reactions.txt" file, which is accessed by the program. A list of the data files may be accessed via the 'View data file names' menu item in the "Files" menu. Clicking on items in the "View File" menu, calls methods in the 'FileView' object to display the contents of the "Reactions.txt" file (the name of which is passed as a parameter) on the Java console or in another scrolling window.

Clicking on items in the "Systems Search", "Substring" or "Classical Pathways" menus, or the 'Any data file' menu item in the "Files" menu, creates windows without components for the corresponding items. The components (e.g., buttons and text fields) are subsequently added to these windows in the relevant classes described below. Selection of items from the "References" menu, invokes methods in the 'Reference' object to display the contents of the various reference files. Clicking on the 'HTML code' menu item in the "Show" menu accesses the HTML code generated in the HTML class.

The class *TrivialApplication* uses the *paint()* method to draw an orange-coloured string " ReactionString Database " in the window. Clicking on the item 'About using the system' in the "Information" menu, displays this information in the text area set up by the constructor method. Clicking on the item in the "Tools" menu, calls the *remove()* method to clear this text area. Finally, the *aboutApplication()* method displays information about the application in a new window.

4.3.3 Class *DataFile*

The class *DataFile* inherits the characteristics of class *Frame* and implements an *ActionListener* and a *WindowListener*. The function of this class is to allow the user to select any data file via a GUI (Fig. 4.3). Its constructor uses *super()* to call the superclass constructor, which in this case is Frame, since *DataFile* extends Frame. Then *super()* passes the *String* variable 'title1' and initialises the title bar for this frame with the string "Select a data file", which is contained in 'title1'. The call to the superclass constructor appears as the first statement within the constructor method so that the superclass constructor executes before the subclass constructor. The constructor for *DataFile* sets the background colour to orange and determines the layout pattern for the components in the frame, i.e., basic window created by the class *TrivialApplication* when the 'Select any data file' menu item in the "Files" menu is clicked. The constructor for the class is also responsible for creating the instances of TextField and adding a button and the text fields to the frame. The button has a label (LOAD FILE), which is a String object that appears on the face of the button when it is displayed. After the button is created, the DataFile constructor invokes the *add()* method to make the Button object a component of the DataFile object. A version of the method *actionPerformed()* is implemented to detect when the button is pressed.

The primary data files end with '.dat' while the rest end with '.txt'. A message on the Java console informs the user to select a file with '.dat' appended to it. The name of the file is entered into the appropriate text field, either by typing it or copying it from the data file list and pasting it. Then clicking the 'LOAD FILE' button, selects the file. The selected file is first read and then written to the 'Reactions.txt' file, which, as stated before, is accessed by the program. If a user enters the wrong file type in the text field, he is advised via the Java console that an invalid choice of file has been made. If the file is not present or the file name is not correct, he is informed (on the Java console) that it cannot be found and is

prompted (in the message text field) to try again. In addition, he is informed that file names may be accessed via the 'View data file names' menu item in the 'Files' menu. Finally, if a file name is not present in the text field when the 'LOAD FILE' button is pressed, the *actionPerformed()* method creates a dialogue box and invokes *Show()* to make it visible. It also passes the relevant parameters to the constructor method *MessageDialog()* in the class of the same name. The constructor creates the user interface with a button and a label that displays a message requesting the user to enter a file name.

4.3.4 Class *FileView*

A window object named view is created as an instance of the *ReactionString* class. The function of the *FileView* class is to allow the user to view any data file that has been selected. It has two methods, both of which receive the string "Reactions.txt" as a parameter from the class *TrivialApplication*. The *consoleView()* method simply reads in the "Reactions.txt" file, adds a space between each line (to enhance visibility) and displays it on the Java console. The method *windowView()* also reads in the same file, but sends it to the *showReactions()* method in the class *ReactionString*, which allows it to be viewed in the text area of another scrolling window.

4.3.5 Class *Reaction*

The class *Reaction* carries out some primary processing for the selection of operator codes and names, and reactions/regulatory actions via scrolling lists. The methods *opCode(String str)*, *opName(String str)* and *event(String str)* in this class respond to calls from the *actionPerformed()* method in the class *TrivialApplication* by importing the reaction strings from the "Reactions.txt" file and the number of records in the file, setting up the respective windows for the operator names and codes, and the reactions/regulatory actions, and parsing the reaction strings. They then pass the operator code and name, and reaction/regulatory action substrings as parameters to the *reactions1()*, *reactions2()* and *reactions3()* methods, respectively, of the GuiSelection objects (referenced by 'window5', 'window6' and 'window7', respectively) in the class *GuiSelection*. The *opName(String str)* method, however, sorts the operator names in lexical order and writes the sorted names to a secondary file (OperatorNames.txt), from which it accesses them again, before sending them to the *GuiSelection* class.

"An array is a sequence of contiguous elements that are accessible by means of a subscript operator" as defined by Hubbard and Huray (2004). To carry out the sorting, the operator names are entered into an array and the array is then sent as a parameter to the sorter method *namesSort(String[ ] nameArray)*. The algorithm used in this method is known as *selection sort*. Its strategy is to find the smallest element in the array and exchange it with the element in the first position, and then find the second smallest element and exchange it with the element in the second position, and so on until the entire array is sorted. Since the mathematical operator '<' (less than) cannot be used on *String*, the *compareTo()* method is used instead. A string is less (greater) than another if it comes before (after) the other in dictionary order. The *String* method *compareTo()* serves this purpose. It has this general form:

int compareTo(String str)

The *compareTo()* method returns an integer result, with the following meaning: 0 if the strings are equal, a negative value if the string object precedes the parameter, a positive value if the string object follows the parameter. This method takes into account uppercase and lowercase letters, with the former coming out before the latter.

4.3.6 Class *GuiSelection*

The class *GuiSelection* inherits the characteristics of the class *Frame* and implements an *ActionListener* and a *WindowListener*. It declares and initialises one-dimensional arrays to hold the incoming items (operator codes or names, or reactions/regulatory actions) from the class *Reaction*. The constructor for the present class calls a constructor of Frame that takes a string argument, sets the background colour to pink and sets the layout manager to null (since no manager is being used). It also displays a label ('SELECT ITEM(S)') and a button ('SUBMIT SELECTION'). The function of this class is to allow the user to select reaction strings via scrolling selection lists displaying operator codes or names, or reactions/regulatory actions.

The methods *reactions1(String enzCode, int nrs)*, *reactions2(String enzName, int nrs)* and *reactions3(String ev, int nrs)* receive the codes, names and reactions or regulatory actions as parameters from corresponding methods in the class *Reaction* and display them on screen in the multiple-choice scrolling selection lists from which the user may select one or more items by clicking them (Figs. 4.4 – 4.6). A list is created as follows:

List list = new List(10,true);

The first parameter specifies that only ten items in the list are visible at once. The list scrolls if it contains more items. The second parameter, true, signifies that multiple selections are allowed in the list. Next, items are added to the list and then the list is added to the window. While the items are being added to a list, the *getItemCount()* method counts them and displays the count in the GUI below the 'SUBMIT SELECTION' button. When the value of the array subscript operator (assigned to the variable 'index') is greater than or equal to the number of records in the "Reactions.txt" file (received as a parameter via the variable 'nrs' and assigned to the variable 'nreactions'), the *reactions1()*, *reactions2()* or *reactions3()* method creates a dialogue box and invokes *Show()* to make it visible. It also passes the relevant parameters to the constructor method *MessageDialog()* in the class of the same name. The constructor creates the user interface with a button and a label that displays a message informing the user that the list is complete. When the 'SUBMIT SELECTION' button is clicked, the *actionPerformed(ActionEvent ae)* method responds by calling the *getSelection()* method. Here, *ae* is a reference to the action event.

The instance method (i.e., it relies on being told which particular object it must access) *getSelectedIndexes()* from the class java.awt.List returns an integer array containing the indexes of the currently selected items. The indexes stored in the array returned by the *getSelectedIndexes*() method are used to look up the names of the items from the array of codes, names or reactions/regulatory actions. The *getSelection()* method gets the selected items and passes them to the *getReactions()* method together with the parameter 's', which contains the string "Reactions.txt". This method accesses the "Reactions.txt" file, parses the incoming reaction strings and compares the results with the items selected from the GUIs. If there is a match between the two, the complete reaction strings are passed to the *ReactionParser* and *ReactionString* classes. The *getReactions()* method also sends the operator code (OC) number as a parameter to the *getOC(String oc)* method of the object named ref in the *Reference* class. Finally, the number of reactions or regulatory actions filtered is computed and printed on the Java console.

4.3.7 *StandardPathway* and *StandardSubnet* classes

These classes inherit the characteristics of the class *Frame* and implement an *ActionListener* and a *WindowListener*. Their purpose is to search a reaction data file for 'standard' pathways and subnets as defined in 4.1.4 above, extract them from the incoming reaction strings and send them to the relevant classes to be displayed on screen in the form of maps and text. The layout of the maps will be described later (see Sections 4.3.21 and 4.4.2). In both classes the objects named panel, parser and view are created as instances of the *GraphPanel* class, the *ReactionParser* class and the *ReactionString* class, respectively.

The constructor for the class *StandardPathway* calls a constructor of Frame that takes a string argument, sets the background colour to red and determines the layout pattern for the components in the frame that is created in the class *TrivialApplication* when the 'Search for pathways' menu item in the "Systems Search" menu is clicked. The constructor for the class is also responsible for adding the buttons to the frame. Each button displays the name of a pathway (see Fig. 4.7). Clicking a button displays the relevant pathway in a window and shows the reaction information in a separate window. The *paint()* method displays information to the user in the same window.

The class filters out the reactions of the named pathways, each pathway having a method of its own. This is achieved by checking the EC numbers as the reactions are passed through a pathway method. An EC number was preferred to an enzyme name for this purpose since there is greater potential for variability in the latter, e.g., the name may have synonyms. The first method that was tried, used an 'if statement' containing the sequence of EC numbers to be extracted. For example, if the EC number was 2.7.1.2 or 5.3.1.9 or 2.7.1.11, etc. for glycolysis, then the reactions of the glycolysis pathway were extracted from the file. This method filtered all the reactions for any given EC number contained in the 'if statement'. However, the reactions were filtered and displayed in the order in which they occurred in the file and not in the pathway. To get the reactions in pathway order, the EC numbers were used to assign the reaction strings to an array in the correct sequence for a given pathway. However, for an EC number with more than one reaction, the next reaction replaced the previous one. A hybrid method was finally used. This uses an EC number where the number has a single reaction associated with it and an enzyme name instead of EC number that represents more than one reaction, for example, the enzyme transketolase catalyses two different reactions on the pentose phoshate pathway. The

reactions are distinguished by appending a different number to the name of the enzyme for each reaction. For transketolase, this results in transketolase1 and transketolase2.

A version of the method *actionPerformed()* is implemented to detect which pathway is chosen when a button is pressed. It responds by calling the hybrid method corresponding to the name of the pathway on the button. The method extracts the relevant reaction strings from the "Reactions.txt" file and assigns them to the array as described above. The reaction strings are then accessed from the array and assigned to the variable 'st'. If this variable is not null they are passed to the *ReactionParser* and *ReactionString* classes. Otherwise the method creates a dialogue box, invokes *Show()* to make it visible and passes the relevant parameters to the constructor method *MessageDialog()* in the class of the same name. The constructor creates the user interface with a button and a label that displays a message informing the user that a certain enzyme for a particular standard pathway was not found. The identity of the missing enzyme(s) is ascertained by using an associated array containing the names of the enzymes corresponding to the entries in the hybrid array. The number of reactions filtered, selected, expected and missing is computed and displayed on the Java console. If one or more reactions are selected, a dialogue box requests the user to click in the drawing area to view the diagram. A similar approach was used in the class *StandardSubnet*, which, as stated before, extracts subnets from a file and displays them on screen in the form of maps and text (see Fig. 4.8).

The number of available pathways that can be extracted in this way in the present application is limited. However, the purpose of these classes is simply to illustrate how the system works. A commercial version could contain methods for the extraction of all known 'standard' pathways. If the pathways exist, standard subnets can be easily obtained by continuing to extract the pathways that constitute the subnets. However, if the definition of a particular 'standard' pathway changes, the Java code would have to be modified to accommodate the change. Nevertheless, this methodology constitutes a rapid way to search, for example, a newly sequenced genome for 'standard' pathways. However, the application contains an additional class (see Section 4.3.8) that overcomes these limitations.

4.3.8 Class *UserSearch*

The class *UserSearch* inherits the characteristics of the class *Frame* and implements an *ActionListener*, an *ItemListener* and a *WindowListener*. The *ItemListener* interface defines the *itemStateChanged()* method that is invoked when the state of an item changes. The purpose of this class is to search the "Reactions.txt" file for defined reaction sequences as entered into the "UserSearch.txt" file by the user and to send those that are found to the relevant classes to be subsequently displayed on screen in the form of maps and text. The objects named parser and view are created as instances of the *ReactionParser* class and the *ReactionString* class, respectively.

The constructor for the class calls a constructor of Frame that takes a string argument, sets the background colour to red and determines the layout pattern for the components in the frame that is created in the class *TrivialApplication* when the 'User-defined search' menu item in the "Systems Search" menu is clicked. The constructor for the class is also responsible for adding three radio buttons and the 'Submit user-defined search" button to the frame (Fig. 4.9). The radio buttons have the property that only one can be set at any one time. Switching one button on switches the others off. Thus they allow the option of searching by operator code or operator name or reaction. The 'reaction' option finds a reaction with the LHS and RHS as specified in the search file, whereas the other options do not discriminate between left and right sides of a reaction. An *ItemEvent* is generated when a radio button is selected or deselected. A check is made within the *itemStateChanged()* method to ascertain which radio button triggered the *ItemEvent*. The basic idea here is to check whether a condition is true or false and to set the *Boolean* variable to true or false accordingly. Thus, for example, the *Boolean* variable 'opcode' is set to *true* if the 'Operator code' radio button is selected, otherwise it is set to *false*. Clicking the "Submit user-defined search" button displays in a window the reactions that are found (based on the search criterion) and shows the reaction information in a separate window. A version of the *actionPerformed()* method is implemented to detect when this button is pressed. The *paint()* method displays information to the user in the same window.

Parsing of the reaction strings separates the operator code, the operator name and the reaction or regulatory action. These are then entered into their corresponding arrays. The Reactions.txt file is also read and similar parsing takes place. The *String* method

compareTo() is used to determine if the items from the parsed strings in the "UserSearch.txt" file are equal to those from the "Reactions.txt" file, respectively (see 4.3.5 above). If so, they are sent to the *ReactionParser* class and the *ReactionString* class. The number of records in the "UserSearch.txt" file is ascertained and indicated on the Java console and a dialogue box requests the user to click in the drawing area to view the diagram. Additional arrays are used to check for operator codes or operator names that are not found (and hence for the corresponding reactions that are not found) and also for reactions with specific metabolites on the left or right hand sides that are not found. For example, if an operator name from the "Reactions.txt" file and an operator name from the user search file are the same, this name is stored at position index (where index = 0, 1, 2, ..., 99) of the array 'nArray[index]', otherwise, position index is set to null. Therefore, if 'nArray[index]' equals null, this indicates that the operator, and hence the reaction, was not found. The same process is used to check for an operator code (and hence reaction) that is not found, as well as for a specific reaction that is not found. The resulting information is displayed on the Java console. For a given enzyme, the generic reaction A + B = C + D is the same reaction in reverse as the reaction C + D = A + B. However, these are two different strings. Therefore, the *compareTo()* method detects them as being different and will only find one of them. However, if the user wishes to utilise the reaction option to find either of them, he may enter both in the user search file. Operators may have synonyms and this may complicate searching by operator name.

4.3.9 Class *NewPathway*

This class is similar to the *UserSearch* class (4.3.8). However, its function is different. The function of the *NewPathway* class is to make a 'new' pathway from a data file. It makes the creation of pathways more convenient by automating their retrieval from one of the large data files (e.g., User.dat). In this case the search file (Pathway.txt) contains the operator codes and names only as entered by the user, i.e., given a set of codes and operators, find the corresponding reactions (pathway) in a target file and send those that are found to the relevant classes to be displayed on screen in the form of maps and text. When displayed, the pathway reactions with their connectivity and directionality can be checked. If the user is satisfied with the results, the pathway can be saved to a named pathway file. If an operator code or name is unknown, it is replaced with four asterisks (****) in the "Pathway.txt" file.

4.3.10 *SubString* classes

There are four substring classes, all of which inherit the characteristics of class *Frame* and implement an *ActionListener* and a *WindowListener*. When a menu item in the "Substrings" menu is clicked, the *actionPerformed()* method in the *TrivialApplication* class sets up a window for that item, and a constructor method in the corresponding substring class creates and adds to it one or two text fields for user input and labels to prompt the user.

The options for substring searches are: (1) class *SubStringOne* finds reaction strings containing a particular substring and displays them on the Java console and in another scrolling window, (2) class *SubStringTwo* performs a similar function to (1) but also displays the resulting reactions in the form of maps, (3) class *SubStringThree* finds reaction strings containing a particular substring but excluding another given substring and displays the results on the Java console and in another scrolling window, and (4) class *SubStringFour* performs a similar function to (3) but also displays the resulting reactions in the form of maps. There is a separate GUI for each of the four options. These may be called up collectively (Fig. 4.15) or individually as required.

In (1) and (3), the object named view is created as an instance of the *ReactionString* class. In (2) and (4), the objects named parser and view are created as instances of the *ReactionParser* and *ReactionString* classes, respectively. In (1) and (2), a text field allows a user to enter one line of text, and it generates an action event when the enter key is pressed. The *actionPerformed()* method is invoked when the action event occurs. It receives an *ActionEvent* object as a parameter. This parameter is not used. Instead, *the getText()* method of *TextField* is called to obtain the contents of the text field, i.e., the substring. The substring is then assigned to the variable 'subStr' and the text field is cleared. The *actionPerformed()* method creates a FileReader object (which includes the file name "Reactions.txt") as a parameter to the *ReactionFinder()* method in the class of the same name and also passes the substring as a parameter to the same method. In (3) and (4) there are two text fields: one for reaction strings containing (including) a given substring and the other for reaction strings not containing (excluding) the substring. These options involve writing the 'inclusion' substring search results to the file "Inclusion.txt" and then using this file for the exclusion search. The *actionPerformed()* method creates FileReader objects (which include the file name "Reactions.txt") as parameters to the

ReactionFinder() and *ReactionExcluder()* methods in the classes of the same names, respectively, and also passes the substring as a parameter to these methods. The *ReactionFinder* and *ReactionExcluder* classes (4.3.12) contain filters that return the reaction strings containing or excluding the substring, respectively. In all the above cases the selected reaction strings are passed to the *ReactionParser* class and/or the *ReactionString* class.

If there is no text present in the text field when the enter key is pressed, the *actionPerformed()* method creates a dialogue box and invokes *Show()* to make it visible. It also passes the relevant parameters to the constructor method *MessageDialog()* in the class of the same name. The constructor creates the user interface with a button and a label that displays a message to the user. In the case of (1) and (2), the message requests the user to enter a substring. In the case of (3) and (4), the message requests the user to enter a substring for inclusion and a substring for exclusion.

The provision of find only filters allows the user to simply view the result of the search as text. Just finding substrings avoids the danger of selecting too much for display. However, it also facilitates the user in making an initial assessment of the extent of the data set to be displayed, otherwise he may be confronted with a combinatorial explosion of reactions. The inclusion/exclusion filters help to narrow down the dataset retrieved. Another use of the latter is to find a subset of reactions that does not use a metabolite commonly used, e.g., kinases generally use ATP – are there any that do not? The substring searches also allow the user to select any compound (enzyme or metabolite) and view all the reactions it is involved in. If a database contains all reactions catalysed by an enzyme, the substring search will find all of these.

4.3.11 Class *ClassicalPathway*

The class *ClassicalPathway* inherits the characteristics of the class *Frame* and implements an *ActionListener*. The function of this class is to allow the user to select a named classical pathway, with or without effectors, via a GUI (Fig. 4.11). The objects named panel, parser and view are created as instances of the *GraphPanel*, the *ReactionParser* and the *ReactionString* classes, respectively. An anonymous inner class handles window events. The constructor for the class *classicalPathways* calls a constructor of Frame that takes a string argument, sets the background colour to green, determines the layout pattern for the

components in the frame and adds a checkbox and a label to it. It also sets up a list of the pathways available for selection. In this case, the second parameter is set to false, which signifies that only one item can be selected from the list (compare with 4.3.6 above). The *paint()* method displays a message to the user in the same window.

A version of the method *actionPerformed()* is implemented to detect which pathway is chosen from the list. A Boolean variable is set to 'false'; otherwise it is 'true' when the checkbox for the inclusion of effectors is clicked. With respect to the latter, both the relevant pathway file and the "Effectors.dat" file are read. Otherwise, the pathway file only is read. In either case, a check for the correct number of tabs is made and the reaction strings are passed to the *ReactionParser* and *ReactionString* classes. The *ClassicalPathway* class also sends the name of the pathway file and the name of the pathway itself to the *Analysis* class.

4.3.12 *ReactionFinder* and *ReactionExcluder* classes

These classes perform filtering, but they filter text a line at a time, rather than a character at a time. Thus BufferedReader is extended instead of FilterReader. The class *ReactionFinder* reads lines of text from a specified Reader and filters out all reaction strings that do not contain a given substring. The filter calls the *readline()* method of the superclass to get the actual lines, but only returns lines that *contain* the substring, and returns null when the superclass's *readline()* method returns null at the end of the file. The class *ReactionExcluder* is similar, but only returns those lines that *exclude* a specified substring. These classes receive input from and return output to the relevant substring classes.

4.3.13 Class *MessageDialog*

As stated before, information dialogue boxes are repeatedly used in the application to display messages to the user. These are *modal* dialogue boxes, which block input to the rest of the application until the user dismisses them. The dialogue boxes are implemented using the class *Dialog*.

The present class *MessageDialog* extends *Dialog* and implements an *ActionListener*. The default layout manager for *Dialog* (*BorderLayout*) is used. The *MessageDialog* constructor has four parameters one of which is a string containing the message (received from another class) to be displayed in the dialogue box and a second one is a Boolean (true) that renders the dialogue modal. The constructor creates a user interface with a label and a button. The *pack()* method of *Window* is called to initiate layout of these components and set the initial size of the dialogue. An anonymous inner class is created to handle window events. The *getInsets()* method returns an Insets object (named insets) of the *Insets* class that represents the margins around the dialogue. The insets add extra space around the outside of the dialogue and are set using the constructor of the *Insets* class. The parameters to the Insets constructor are the number of pixels of space on the top, left, bottom, and right, respectively. The *dispose ()* method is called when the button generates an action event. This is done to remove the dialogue from the screen and relinquish any operating system resources that were allocated for it.

4.3.14 Class *Analysis*

The function of this class is to display a simple frequency histogram of operator code classes for classical pathways selected via the 'Classical pathways' menu item from the menu of the same name. The *ClassicalPathway* class sends the name of the pathway file and the name of the pathway itself to the *histogram()* method in the present class. The reaction-strings are then imported from the relevant pathway file into the *Analysis* class, which isolates the class numbers as strings and converts them to integers. The String variable 'output' is declared and initialised to:

"Class\t\tFrequency\tHistogram"

A ten-element *frequency* array is used to count the occurrences of each operator class. The class number is used as a subscript for the *frequency* array to determine which class should be incremented during each turn of an outer 'for' loop. The results are then appended to the String variable 'output'. The numeric data are displayed graphically with a histogram that simply shows each class frequency value as a bar consisting of that many asterisks (*). In the nested 'for' loop, the bars are appended to the String 'output'. Each time the program reaches the inner 'for' structure, the loop counts from 0 to 'frequency[classNumber]' thus using a value in the array to determine the final value of the variable 'star' and the number

of asterisks to display. Then the String variable 'output' is sent to the class *ReactionString* and the results are displayed in the text area of a scrolling window.

4.3.15 Class *PathwayInfo*

The purpose of this class is to access information on named textbook pathways. It extends *Frame* and implements an *ActionListener* and a *WindowListener*. The object named pathInfo is created as an instance of the *ReactionString* class. Buttons with the pathway names and associated information are also created. The constructor for *PathwayInfo* calls a constructor of *Frame* that takes a string argument, sets the background colour to green and determines the layout pattern for the components in the window instantiated by the *actionPerformed()* method of class *TrivialApplication*. It also adds the buttons to the window (Fig. 4.12).

A version of the *actionPerformed()* method is implemented to detect which pathway button is pressed. The method then accesses the corresponding file containing the pathway information and sends it to the *showReactions()* method in the class *ReactionString*, which displays it in the text area of a scrolling window.

4.3.16 Class *Reference*

The purpose of this class is to access references from a general literature reference file and from a reaction-specific reference file. It also accesses references from a file containing search-specific references that is generated when individual reaction strings or groups of reaction strings are selected via menu items in the "Operators" menu or via the 'Reactions or regulatory actions' menu item in the "Events" menu. The object named view is created as an instance of the *ReactionString* class.

The methods *genRef()* and *specifRef()* access the general and reaction-specific references from the files "GenRef.txt" and "SpecRef.txt", respectively, and send them to the *showReactions()* method in class *ReactionString*, which displays them in the text area of scrolling windows. When a user selects operators by code or name, or selects events via the 'Reactions or regulatory actions' menu item in the "Events" menu, the *getReactions()* method in class *GuiSelection* passes the codes (e.g., EC numbers) as parameters to the method *getOC()* in the present class. This method accesses reaction-specific references

from the "SpecRef" file and compares the EC numbers. If these match, the method *getOC()* writes the results to a temporary file ("selectedRef.txt"). Then when the user clicks the 'Search-specific' menu item in the "References" menu, the *actionPerformed() method* of the class *TrivialApplication* retrieves these references and passes them as parameters to the *showReactions()* method in class *ReactionString*, which displays them in the text area of a scrolling window.

4.3.17 Class *ReactionString*

The function of this class is to display reaction-string information in the text area of a scrolling window. The class *ReactionString* inherits the characteristics of class *Frame* and implements a *WindowListener*. The constructor for the class *ReactionString* calls a constructor of *Frame* that takes a string argument, sets the layout manager to null (since no manager is being used) and sets up a window. The constructor for the class is also responsible for creating an instance of *TextArea*, which is then added to the window. The method *showReactions()* receives reactions strings and other strings as parameters from several classes and appends them to the text area.

4.3.18 Class *ReactionParser*

It is necessary to parse the input reaction strings and reassemble the component parts in a form suitable for use by the *HTML* and *Relations* classes. A *token* is a string of characters separated by a delimiting character. The process of separating individual tokens from a string (parsing) is normally achieved using methods from the *StringTokenizer* class, which is part of the 'java.util' package. However, only characters can be used as delimiters. The use of '+' as a delimiter to separate compounds in a reaction equation, resulted in the Stringtokenizer removing the '+' from compounds such as NAD^+. Ad hoc solutions were tried, but lacked generality. Hence, the present *ReactionParser* class was designed to provide a general solution to this problem and to perform other relevant functions.

The class *ReactionParser* may be regarded as the centre of the system since all items to be drawn and displayed in the drawing area must pass through it. The objects named code and bin are created as instances of the *HTML* and *Binaries* classes, respectively. The function of the *ReactionParser* class is to break down reaction information into its basic components and concatenate them in suitable ways for further processing. Reaction

information is passed to the class as a tab-delimited reaction string with the tabs separating operator code (e.g., EC number), operator name (e.g., enzyme name) and reaction or regulatory action.

Methods from the class *String*, which is part of the Java system, are used to help parse and manipulate this information. The method *length()* is used to get the number of characters in a string. All characters, including spaces, special symbols, and repeated characters are counted. Positions in the string are counted starting with 0. A position is referred to as an *index*. The method *indexOf()* returns the index of the substring given as its one argument. If the substring occurs more than once, *indexOf()* returns the index of the first occurrence of its substring argument. It returns -1 if the substring is not found. The method *lastIndexOf()* returns the position of the last occurence of its substring argument. The method *substring(Start, End)* returns the substring of the string object starting from Start through, but not including, position End of the string object.

The *ReactionParser()* method, to which the reaction string is passed as a parameter, does the actual parsing. The length of the reaction string is computed and subsequent substrings are re-evaluated as necessary. The original string is split into EC number and enzyme reaction, both of which are stored in their respective variables. Then the enzyme reaction substring is divided into enzyme and reaction, again both of which are stored in their respective variables. For computational purposes, the *replace()* method, which replaces one character by another throughout a string, is used to change ':' to '='. This is necessary in order to deal with the input format for regulatory actions (see 4.1.2). A reaction is first checked for the presence of the 'equals' sign. If it is not present, a dialogue box informs the user that this reaction is only available as a natural language description, and additional information, including the natural language description itself, is given on the Java console. A reaction is also checked for the presence of a space on each side of the 'equals' sign. If there is no such space(s), a dialogue box informs the user of an input format error in the reaction, and additional information, given on the Java console, indicates that the error is in the substring containing the equality sign. In either case, clicking the OK button removes the dialogue box from the screen and the program continues with the rest of the reactions (if any). A reaction is then split into left and right sides, both of which are stored in their respective variables. These are dealt with separately, the left-hand side being parsed first.

There may be any number of pluses, or none, on either side of the reaction. A *part* (of a reaction) is defined as a compound with its stoichiometric coefficient. The left-hand side of a reaction is broken down into parts, compounds and stoichiometric coefficients, all of which are stored in their respective arrays. The case where there is no plus sign is dealt with first. The stoichiometry may be implicit (when it is one) or explicit (all other cases). If the stoichiometry is implicit, the reaction part is equivalent to the compound. The situation is divided into the case where there is no space in the compound name (e.g., glucose) and the stoichiometry is implicit, and the case where there may be a space in the compound name (e.g., glucose 6-phosphate). The latter is subdivided into three cases: (1) explicit stoichiometry between 1 and 9, (2) explicit stoichiometry between 10 and 99, and (3) implicit stoichiometry. To accommodate stoichiometry in the form of 'm' or 'n', it is necessary to represent a coefficient as a string rather than an integer. For example, in the following reaction:

L-arginine + n $NADPH_2$ + m O_2 = citrulline + nitric oxide + n NADP

the stoichiometry is given as 'm' with respect to O_2, and 'n' with respect to $NADPH_2$ and NADP. However, this produces problems when 'an' or 'a' occurs in the name of a metabolite, as for example, in the following reaction:

an L-amino acid + H_2O + O_2 = a 2-oxo acid + NH_3 + H_2O_2

Although the stoichiometry is implicitly one, the program will interpret it as 'an' and 'a' with respect to L-amino acid and 2-oxo acid, respectively. This is exacerbated by the fact that the letter 'a' may also be in upper case. The Java programming language is case sensitive. In an attempt to solve this problem, a condition is included in the program that sets the stoichiometry equal to the string value "1" if it encounters these in either upper or lower case. Although this returns the stoichiometry as "1", it removes the 'an' and 'a' bits. However, it still returns the actual name (e.g., L-amino acid) of the respective compound intact. Typographical errors can also produce problems for stoichiometry, e.g., the following apparent error in the original dataset: N -(L-arginino)succinate, i.e., the space between the leading N and the hyphen, resulted in the 'N' being treated as stoichiometry.

The next situation involves the case where there is at least one plus sign. A loop is used here. The loop continues until there are no more plus signs left. Methods from the string

class are used to find the first occurrence of the substring " + " (i.e., space+space). This requires re-evaluating the length of the new string each time a substring is removed. The length of the string changes continuously, decreasing as the loop progresses. So the index of the first occurrence of the substring " + " and the index of its last occurrence change accordingly. The program exits the loop when both are equal. During this time the components of the string are removed step-by-step and stored in their respective arrays. All the stoichiometry cases above apply to the loop. The case where there are no more plus signs is dealt with outside the loop. Again, the previous stoichiometry conditions apply here also. The RHS of reaction goes through the same procedure as the LHS, except that the storage variables and arrays have different names. The string literals "| L" and "| R" are appended to the compound names on the LHS and RHS, respectively, to indicate the side of the reaction to which they belong.

The same processing applies to effectors. However, while a reaction has at least one substrate and one product, a regulatory action may have inhibitor(s) without an activator, or activator(s) without an inhibitor. The *ReactionParser()* method contains code to deal with these situations. The logic behind the code is that if an effector is missing, the side indicator (L or R) will exist on its own in the relevant array rather than being appended to an effector name. If this is the case, the left or right binary relation is set to empty. Obviously, this code is implemented only in the case where there is no plus sign present in the regulatory action. The stoichiometry for effectors is given as 'm' with respect to activators and 'n' with respect to inhibitors.

Finally, the binary relations of each enzyme with its metabolite(s) or effector(s) are concatenated with the corresponding stoichiometric coefficient(s) and reaction side (L or R) into a single string (called an n-ary relation). If an operator equals a non-operator as, for example, in a receptor-ligand interaction (see Section 3.5.2), an empty substring is appended to the n-ary relation. This prevents the creation of an extra edge in the graph by a receptor forming a binary with itself. A delimiter separates each component in the string. For example, the following input reaction string:

1.13.11.21 beta-carotene 15,15'-dioxygenase beta-carotene + O2 = 2 retinal

is converted to the following output string of n-ary relations:

beta-carotene15,15'-dioxygenase$beta-carotene|L/1;beta-carotene15,15'-dioxygenase$O2|L/1;beta-carotene 15,15'-dioxygenase$retinal|R/2;

A semi-colon separates each n-ary relation, while a different delimiter separates each component of an n-ary relation. The enzyme is separated from a metabolite by '$' and the metabolite is separated from the reaction side indicator (L or R) by '|', while the latter is separated from the stoichiometric coefficient by '/'. This sequence of n-ary relations constitutes the computational representation of the input data. If more than one reaction string is input, the additional n-ary relations are appended to the output string.

These strings are then passed to the class *Relation* for further processing. They are also passed to the class *HTML*, where they are embedded into HTML code. An example of the output from the *ReactionParser* class is given in 4.3.1. The schematic for the reaction string decomposition process in the same class is shown in Fig. 4.14 below. The *ReactionParser* class does not deal with mass and charge balance because this information is not present in the data source.

4.3.19 Class *HTML*

The function of this class is to generate an HTML document containing the n-ary relations generated by the *ReactionParser()* method in the *ReactionParser* class. These n-ary relations are received as a parameter by the *getCode()* method, which embeds them in the HTML code that is contained within the method. For example, the Source for the reaction 'beta-carotene + O2 = 2 retinal' catalysed by the enzyme beta-carotene 15,15'-dioxygenase (EC 1.13.11.21) is as follows:

```
<title>TrivialApplet</title>
 <hr>
 <applet codebase="Java Classes" code="TrivialApplet.class" width=1150 height=700>
 <param name=GET_EVENT(S) value="beta-carotene15,15'-dioxygenase$beta-
carotene|L/1;beta-carotene15,15'-dioxygenase$O2|L/1;beta-carotene15,15'-
dioxygenase$retinal|R/2;">
 </applet>
 <hr>
```

The 'Show' menu allows the user to access the HTML file, which is automatically generated when the reactions/regulatory actions are displayed. As stated before, this file may be manually copied and entered into the Java applet so that the output from the application can be viewed and interacted with over the Internet.

4.3.20 Class *Relation*

The class *Relation* inherits the characteristics of class *Frame*. The object named panel is created as an instance of the *GraphPanel* class. The *getRelations()* method creates a *StringTokenizer* to parse the n-ary relations that are passed to it by the *ReactionParser()* method in the *ReactionParser* class. A *StringTokenizer* extracts information encoded in an input string that is regularly formatted. In this case, the string contains repeated substrings (n-ary relations) separated by a semi-colon, which it uses as a delimiter. Methods from the class *String*, which is part of the 'java.lang' package, are then used to separate the components of each n-ary relation. These components, including the operator, are then passed to the *addEdge()* method in the drawing class *GraphPanel*. The operator is also sent to the *fixedNode()* method in the same class. The *getRelations()* method also creates the display window in which the reactions are to be drawn.

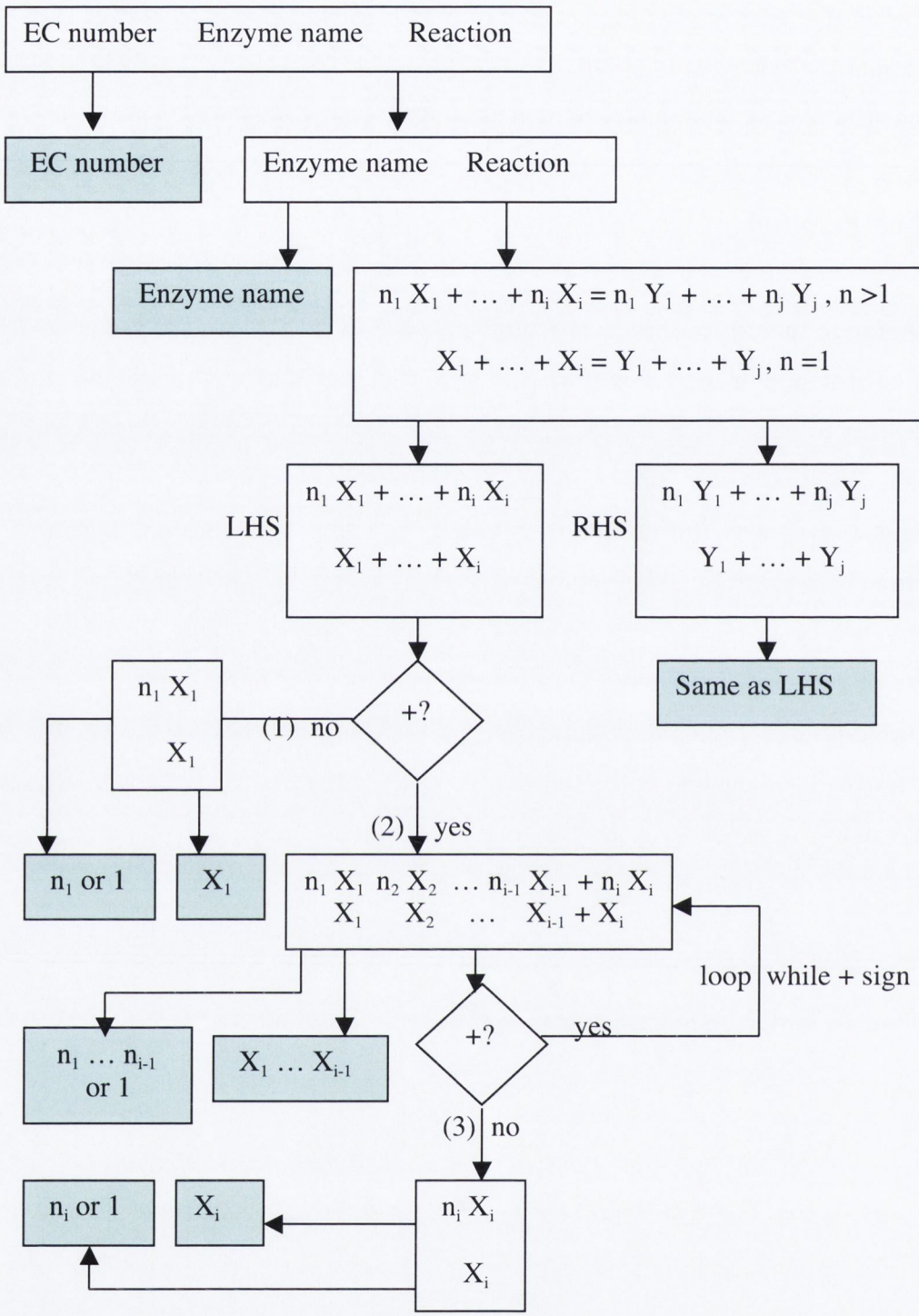


Fig.4.14. Schematic of part of the class *ReactionParser* showing the decomposition of a tab-delimited reaction string into EC number, enzyme name and reaction. The left-hand side (LHS) of the reaction is broken down into parts (n_iX_i), compounds (X_i) and stoichiometric coefficients (n_i). Three cases are considered: (1) no plus signs, (2) at least one plus sign, and (3) no more plus signs remaining, at which stage the program exits the plus-sign checking loop. The same process is applied to the right-hand side (RHS) of the reaction.

4.3.21 Drawing classes

The drawing classes consist of two small classes, class *Node* and class *Edge*, and a large class, class *GraphPanel*. These were based on the Graph Java applet distributed in the Java development distribution of Sun Microsystems Inc. First, the applet was converted into an application by:

- Changing the name of the *init()* method to the name of the class. This was now the constructor method for the class.
- Deleting the word void in the header for this method, since a constructor method has no return type.
- Altering the class header so that it extended *Frame* rather than *Applet*.
- Creating a new method called *main*, with the header public static void main(String[ ] args). This method created a frame object as an instance of the class.
- Deleting the import for the class *Applet*, since it was now redundant.
- Adding a method *actionPerformed()* to handle the event "the user clicking on the close window button". This involved adding "implements *ActionListener*" and "addActionListener(this)" in order to register the event handler.
- Getting rid of *getCodeBase*, and the *start()* and *stop()* methods, since these were special to the *Applet* class.

These would be regarded, perhaps, as standard changes. However, additional changes were made on a more empirical basis in order to facilitate the incorporation of the converted applet into the present application. For example, the thread that was instantiated in the *start()* method is now created and started within a new constructor method for the *GraphPanel* class. This was necessary to restore the motion of non-fixed nodes, which was lost in the conversion. A *thread* is an independently executing section of code. To include the thread, the *Runnable* interface is implemented. This handles the *run()* method, which holds the thread's code. The new constructor calls a constructor of Frame that takes a string argument, determines the layout pattern for the components in the frame, and adds labels, check boxes and buttons to it. The original layout manager *BorderLayout* has been changed to *FlowLayout*. The *StringTokenizer* is now in the class *Relation* (4.3.20) while the *main()* method is in the class *TrivialApplication* (4.3.2).

Other less critical changes involved deleting certain code while adding some extra functionality. For example, some changes have been made in the layout of the graphs. In

the original applet all the nodes moved freely across the display window. A new method, *fixedNode(),* has been added in order to fix the positions of the operator nodes relative to the non-operator nodes. Furthermore, the movement of the non-operator nodes is now controlled by the checkbox "Allow NOPs to move", which uses a Boolean variable to asses whether it has been checked or not. The variable named "relaxed" in the *run()* method has the value 'false' if this checkbox has not been checked, 'true' if it has been checked and false if it has been cleared. If the value is 'true', then the *relax()* method is activated. This method contains an algorithm, which has the effect of causing the nodes to move apart from each other. Since the operator nodes are fixed, it effectively separates the non-operator nodes. An additional simple algorithm for the initial placement of the nodes has been added to the method *addNode()*. This is termed a *linear vertical placement.* It arranges the nodes in columns starting at the upper left of the window. When a column reaches the bottom of the window, a new column is started to the right of the first one, and so on across the window. This arrangement can accommodate a lot of nodes in an organised manner and facilitates manual layout by the user.

The "Scramble" button, when pressed, scrambled all the nodes, and the "Random" checkbox, when clicked, randomly displaces them. These have been removed. An original "Stress" checkbox had associated functionality that displayed the result of a mathematical calculation, performed *inside* the *GraphPanel* class, on the edges of the graph. This has been replaced with the "Side indicator (L or R)" and "Reaction stoichiometry" check boxes, where the letters L and R indicate that non-operators belong to the LHS or RHS of an event (e.g., reaction), respectively, as represented in the files. These check boxes allow the corresponding data, which are imported from *outside* the *GraphPanel* class and the program, respectively, to be displayed on the edges of the graph as required. Additional check boxes have also been added. These include the "Side indicator (Colour)" check box, which, when checked, activates new code to colour the edges and the edge labels blue or green according to 'L' or 'R', respectively; the "Thick edge" check box, which, when checked, activates a second *drawLine()* to thicken an edge; and the "Show node borders" check box, which, when checked, activates code to draw a rectangle around the text. The new code to perform these operations when the check boxes are clicked has been added to the *update()* method, which draws the edges and their labels. The *update()* method calls the *paintNode()* method. The *paintNode()* method is responsible for drawing the nodes and contains new code to colour the names of effectors (blue for activators and red for inhibitors) if they are present in the data.

The method *addEdge()* receives the names of the operators and non-operators, the edge lengths, and the reaction-side information and stoichiometry, as parameters from the method *getRelations()* in the class *Relation*, and creates new edges with labels and sends the node names to the method *findNode()*. The method *addNode()* creates a new node with the name that is passed to it from *findNode()*. The method *findNode()* searches to see if the node that is required is already created (and returns it). If it is not created, *addNode()* is called to create it. As stated before, the *update()* method draws the edges and their labels and calls *paintNode()*, which draws the nodes. The *action()* method is called when one of the check boxes or buttons is clicked. This method examines the label of the object that invoked *action()* and uses that to decide on what message to send to the *update()* method, the *paintNode()* method or the *run()* method as the case may be. Finally, mouse event methods allow the user to interact with the nodes. For example, if the mouse is dragged (i.e., a mouse button is down and the mouse has moved) the *mouseDrag()* method is called to move the node that was picked.

4.3.22 Information flow

The flow of information between classes and between classes and files is shown in Fig. 4.15 below. The classes in the Java packages are not included in the diagram. As stated before, the computer starts the execution of the program at the first statement in *main()* in the class *TrivialApplication*. When *TrivialApplication* requests the *DataFile* class to select a data file, the selected file is first read by *DataFile* and then written to the "Reactions.txt" file. However, *TrivialApplication* can select the general data file or the user data file directly and write it to the "Reactions.txt" file. To view a selected file, *TrivialApplication* requests the *FileView* class to read the "Reactions.txt" file and send it to the *ReactionString* class (or display it on the Java console). All the information that is sent to *ReactionString* is displayed in the text area of a scrolling window other than the Java console.

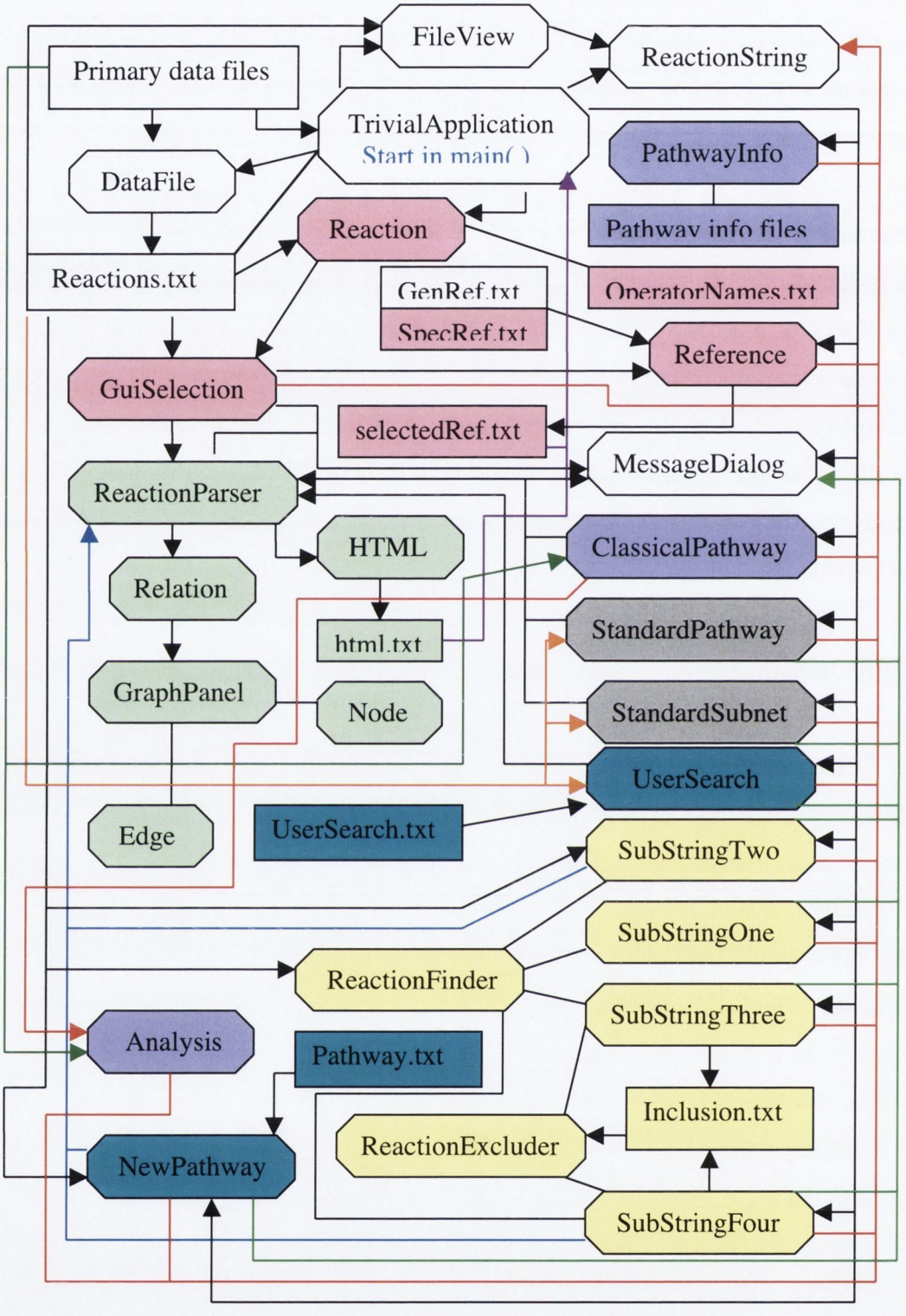


Fig.4.15. The flow of information between classes (octagons) and between classes and files (rectangles). Colour (non-white) is used to group classes and files that are more closely related. Lines of different colours do not intersect whereas those of the same colour intersect if they contact each other. Arrows show that the flow of information is in the direction indicated while lines without arrowheads indicate that the flow may be in either direction.

When operator codes or names, or reactions/regulatory actions are selected, *TrivialApplication* sends messages to the *MessageDialog* class, and also sends the string literal "Reactions.txt" and the number of records in the selected file to the *Reaction* class. The *Reaction* class reads the "Reactions.txt" file, isolates the operator codes or names, or reactions/regulatory actions, and sends them to the *GuiSelection* class, which displays them in lists in addition to sending messages to the *MessageDialog* class (and to the Java console). However, the *Reaction* class sorts the operator names in lexical order and writes them to the "OperatorNames.txt" file, from which it accesses them again, before sending them to the *GuiSelection* class. When selections are made from the lists, *GuiSelection* reads the "Reactions.txt" file and sends the corresponding reaction strings to the *ReactionParser* class and to the *ReactionString* class, and the operator codes to the *Reference* class, which compares them to the "SpecRef.txt" file. If the codes match, the corresponding references are written to the "selectedRef.txt" file. When requested, *TrivialApplication* reads this file and sends it to the *ReactionString* class. When the 'All regulatory actions' item in the Events menu is selected, *TrivialApplication* reads the "Effectors.dat" file and sends the reaction strings to the *ReactionString* class and the *ReactionParser* class.

When the user clicks 'Search for pathways' (or 'Search for subnets') in the Systems Search menu, *TrivialApplication* creates a window and requests the *StandardPathway* (or *StandardSubnet*) class to add buttons and text to it. Then when he presses a button with a pathway (or subnet) name on it, *StandardPathway* (or *StandardSubnet*) reads the "Reactions.txt" file and extracts the corresponding reaction strings from it and sends them to the *ReactionParser* class and to the *ReactionString* class, and sends a message to the *MessageDialog* class. When the user clicks 'User defined search' in the Systems Search menu, *TrivialApplication* creates a window and requests the *UserSearch* class to add buttons and text to it. Then when he selects an option and submits the search, *UserSearch* reads the user search file and the "Reactions.txt" file and looks for a match between the two. If a match is found, the corresponding reaction strings are sent to the *ReactionParser* class and the *ReactionString* class, and a message is sent to the *MessageDialog* class. The flow of information for the *NewPathway* class is similar to that for the *UserSearch* class, except that it reads the"Pathway.txt" file instead of the "UserSearch.txt" file.

When the user clicks 'Classical pathways' in the Classical Pathways menu, *TrivialApplication* creates a window and requests the *ClassicalPathway* class to add a

check box, a pathway list and text to it. When he double-clicks a pathway name, the corresponding pathway file is read. If, in addition, he clicks the check box (to include effectors) the "Effectors.dat" file is also read. In both cases, messages are sent to the *MessageDialog* class (and the Java console). The program extracts the corresponding reaction strings from the files and sends them to the *ReactionParser* class and to the *ReactionString* class. Furthermore, the *ClassicalPathway* class sends the name of the pathway file and the name of the pathway itself to the *Analysis* class, which imports the relevant reaction-strings and prepares a histogram of the frequency of occurrence of the operator classes for the selected pathway. Then the *Analysis* class sends the histogram data to the *ReactionString* class for display. When the user clicks 'Pathway information' in the Classical Pathways menu, *TrivialApplication* creates a window and requests the *PathwayInfo* class to add buttons with pathway names and associated information to it. Then when he presses a button, he gets the relevant information from a pathway description file. This is then passed to the *ReactionString* class.

When the user clicks 'Find' in the Substrings menu, *TrivialApplication* creates a window and requests the *SubStringOne* class to create a text field and labels. When he types a substring and presses enter, *SubStringOne* passes a FileReader object (which includes the file name "Reactions.txt") as a parameter to the *ReactionFinder()* method in the *ReactionFinder* class and also passes the substring as a parameter to the same method, which filters out all reaction strings that do not contain the given substring and only returns those that contain the substring. Then *SubStringOne* sends the corresponding reaction strings to the *ReactionString* class (and the Java console). When the user clicks 'Find and display' in the same menu, a similar sequence of events occurs in the *SubStringTwo* class except, in addition, *SubStringTwo* sends the filtered reaction strings to the *ReactionParser* class.

When the user clicks 'Find by inclusion/exclusion' in the Substrings menu, *TrivialApplication* creates a window and requests the *SubStringThree* class to create text fields and labels. When he types substrings into the 'inclusion' and 'exclusion' text fields and presses enter, *SubStringThree* passes a FileReader object (which includes the file name "Reactions.txt") as a parameter to the method *ReactionFinder()* in the *ReactionFinder* class and also passes the 'inclusion' substring as a parameter to the same method, which filters out all reaction strings that do not contain the given substring and only returns those that contain the substring. The 'inclusion' substring search results are written to the file

"Inclusion.txt" and then this file is used for the exclusion search, in which case *SubStringThree* passes a FileReader object as a parameter to the method *ReactionExcluder()* in the *ReactionExcluder* class and also passes the 'exclusion' substring as a parameter to the same method, which filters out all reaction strings that contain the given substring and only returns those that do not contain the substring. Then *SubStringThree* sends the corresponding reaction strings to the *ReactionString* class (and the Java console). When the user clicks 'Find by inclusion/exclusion & display' in the same menu, a similar sequence of events occurs in the *SubStringFour* class except, in addition, *SubStringFour* sends the filtered reaction strings to the *ReactionParser* class.

The *ReactionParser* class generates n-ary relations and sends them to the *HTML* class and the *Relation* class. The former generates an HTML document containing the n-ary relations and writes it to the "html.txt" file, while the latter sends the names of the operators and non-operators, the edge lengths, and the reaction-side information and stoichiometry to the *GraphPanel* class. When the user clicks 'HTML code' in the "Show" menu, *TrivialApplication* reads the "html.txt" file and sends the contents to the *ReactionString* class.

When the user clicks 'General' in the "References" menu, *TrivialApplication* requests the *Reference* class to read the "GenRef.txt" file. The *Reference* class sends the general literature references to the *ReactionString* class. When 'Reaction-specific' in the "References" menu is clicked, *TrivialApplication* requests the *Reference* class to read the "SpecRef.txt" file. The *Reference* class sends the reaction-specific references to the *ReactionString* class. When a user clicks "Search-specific" in the "References" menu, *TrivialApplication* reads the "selectedRef.txt" file and sends the selected reference(s) to the *ReactionString* class. As stated before, all information that is sent to *ReactionString* is displayed in the text area of a scrolling window

§4.4 Output from the application

4.4.1 Output from *ReactionParser* class

The output from the *ReactionParser* class is displayed on the Java console. Shown in table 4.1 below is the output for the following example reaction string:

1.13.11.21 beta-carotene 15,15'-dioxygenase beta-carotene + O2 = 2 retinal

If more than one reaction string is input, a similar output is given for each one.

This output is useful if the user needs to trouble-shoot the file. For example, if there is no space between the compound 'beta-carotene' and the '+' sign, the following will be output:

Compound 1 of left side: beta-carotene+ O2

The process will work and a reaction will be displayed, but it is obviously not correct. The expression "beta-carotene+ O2" may look all right to the human eye, but the parser is indicating that it is one compound. In the drawing, it will be displayed as a single node. This would be problematic for the graph connectivity of a sequence of linked reactions. The solution to the problem is to edit the relevant primary data file by adding a space between the compound name and the '+' sign.

Shown in table 4.2 below is the output for a reaction string in which the reaction is only available as a natural language description. Since the reaction does not contain the substring " = " (i.e., the equals sign with a space on each side), it cannot be parsed and hence cannot be displayed. Although it is unsuitable for use in a dataset designed for computational use, nevertheless, the output does give some useful information to the user in terms of the EC number, the name of enzyme and the nature of the reaction.

Original reaction string:

1.13.11.21 beta-carotene 15,15'-dioxygenase beta-carotene + O2 = 2 retinal

Operator code: 1.13.11.21 (stored in variable numberOC)

Operator reaction/regulatory action:

beta-carotene 15,15'-dioxygenase beta-carotene + O2 = 2 retinal (stored in variable opReaction)

Operator: beta-carotene 15,15'-dioxygenase (stored in variable Operator)

Reaction/regulatory action: beta-carotene + O2 = 2 retinal (stored in variable reaction)

Left side of reaction/regulatory action: beta-carotene + O2 (stored in variable left)

Part 1 of left side: beta-carotene (stored at index 1 of array partLeftSide)

Compound 1 of left side: beta-carotene (stored at index 1 of array cmpdLeftSide)

Coefficient 1 of left side: 1 (stored at index 1 of array stoichLeftSide)

Remainder: O2

Part 2 of left side: O2 (stored at index 2 of array partLeftSide)

Compound 2 of left side: O2 (stored at index 2 of array cmpdLeftSide)

Coefficient 2 of left side: 1 (stored at index 2 of array stoichLeftSide)

Number of compounds on LHS = 2

Right side of reaction/regulatory action: 2 retinal (stored in variable right)

Part 1 of right side: 2 retinal (stored at index 1 of array partRightSide)

Compound 1 of right side: retinal (stored at index 1 of array cmpdRightSide)

Coefficient 1 of right side: 2 (stored at index 1 of array stoichRightSide)

Number of compounds on RHS = 1

N-ary relation(s):

beta-carotene15,15'-dioxygenase$beta-carotene|L/1;beta-carotene15,15'-dioxygenase$O2|L/1;beta-carotene 15,15'-dioxygenase$retinal|R/2;

Table 4.1. The output from the *ReactionParser* class for the biochemical reaction beta-carotene + O_2 = 2 retinal catalysed by the enzyme beta-carotene 15,15'-dioxygenase (EC 1.13.11.21). The enzyme reaction is input as a tab-delimited reaction string, which is decomposed into EC number, enzyme and reaction. The reaction is further decomposed into parts, compounds and coefficients. A *part* is a compound and its stoichiometric coefficient. The relations of the enzyme with its metabolites are concatenated with the corresponding stoichiometric coefficients and reaction side (L or R) into a single string (referred to as an n-ary relation). A delimiter separates each component in the string. The n-ary relation constitutes the computational representation of the reaction information.

Original reaction string: 3.2.1.48 sucrose alpha-glucosidase Hydrolysis of sucrose and maltose by an alpha-D-glucosidase-type action Operator code: 3.2.1.48 (stored in variable numberOC) Operator reaction/regulatory action: sucrose alpha-glucosidase Hydrolysis of sucrose and maltose by an alpha-D-glucosidase-type action (stored in variable opReaction) Operator: sucrose alpha-glucosidase (stored in variable Operator) Reaction/regulatory action: Hydrolysis of sucrose and maltose by an alpha-D-glucosidase-type action (stored in variable reaction) This reaction is only available as a natural language description! It cannot be displayed. String index out of range: -1

Table 4.2. The output from the *ReactionParser* class for the biochemical reaction 'Hydrolysis of sucrose and maltose by an alpha-D-glucosidase-type action' catalysed by the enzyme sucrose alpha-glucosidase (EC 3.2.1.48). The enzyme reaction is input as a tab-delimited reaction string, which is decomposed into EC number, enzyme and reaction. The message 'String index out of range: -1' from the Java compiler indicates that the substring " = " was not found in the reaction.

4.4.2 Output from the drawing classes

The output from the drawing classes is displayed in a window called the 'Drawing Area'. There is an array of six checkboxes and two buttons at the top of this window. The check boxes have the following labels: "Side indicator (L or R)", "Side indicator (Colour)", "Reaction stoichiometry", "Thick edge", "Show node borders", and "Allow NOPs to move". The buttons have the labels "Shake NOPs" (which displaces the non-operators) and "Close window" (Fig. 4.16 below). The check boxes allow the visualisation of certain parameters as required (e.g., stoichiometry) and control particular aspects of the display (e.g., motion of non-operators).

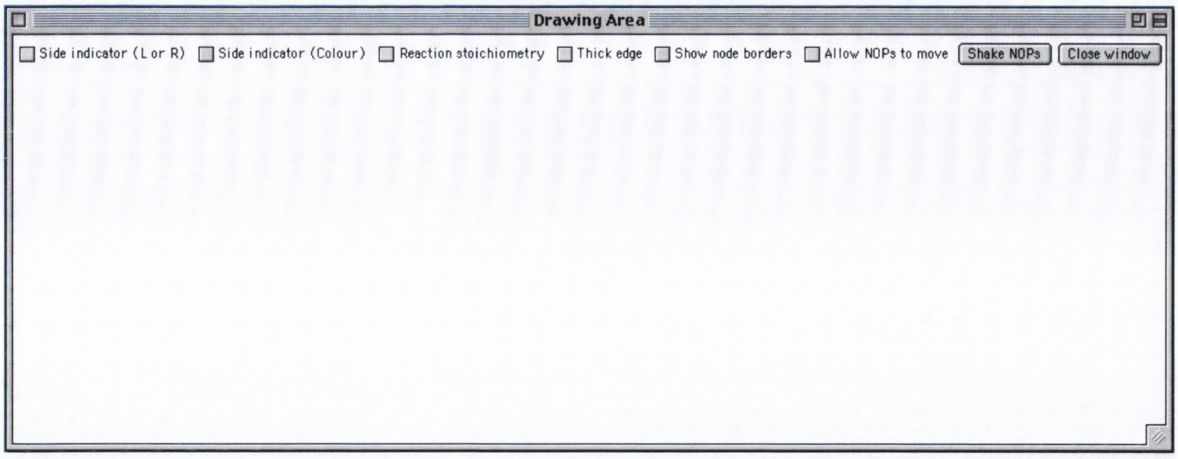


Fig.4.16. The drawing area for the display of metabolic reactions, pathways and subnets has an array of six checkboxes and two buttons at the top.

A display consists of a map of reactions, pathways or subnets, with or without modulators. The data to be displayed are transformed into vertices of a graph using varying vertex sizes. The size of a vertex (also referred to as a node) provides the space needed to display the information of an object such as the name or process (e.g., tautomerisation), i.e., the visualisation algorithm draws different vertex sizes to accommodate text of variable length. In this application objects are generally drawn with their names.

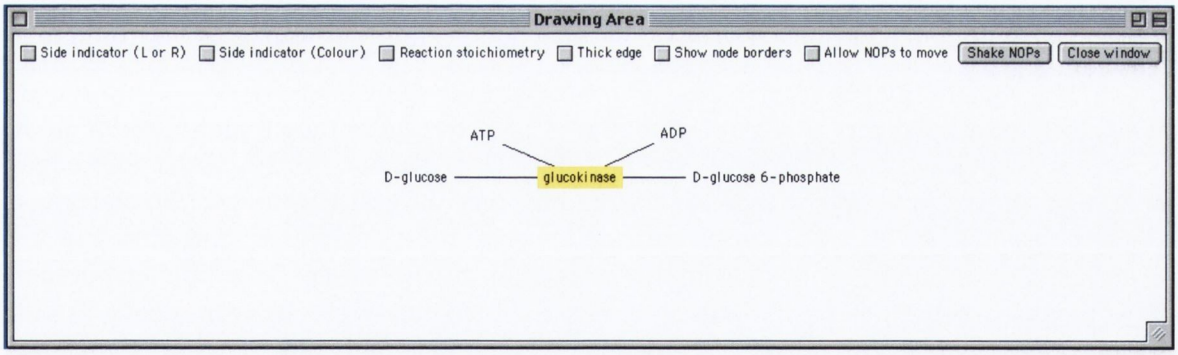


Fig.4.17. Diagram of the reaction ATP + D-glucose = ADP + D-glucose 6-phosphate, catalysed by the enzyme glucokinase (coloured yellow), as generated by selection of the E.C. number 2.7.1.2 from the operator codes GUI (Fig. 4.4).

A diagram of a reaction is a bipartite graph with the vertices corresponding to material objects and the edges representing relationships between the objects (Fig. 4.17 above). The objects are divided into an operator and non-operators (see 3.3). Colour (yellow) distinguishes the operator (OP) from the non-operators (NOPs). This makes it more intuitive and easy to navigate for the user. The edges are coloured black, which is their default colour.



Fig.4.18. The above representations of Fig. 4.17 show edge labels and edge or node properties, which are visualised by clicking the relevant check boxes. Top: blue edges with label 'L' of same colour indicate metabolites are on the LHS of reaction and green edges with label 'R' of same colour indicate they are on the RHS, as represented in the database. Centre: integers on edges indicate reaction stoichiometry (1 for all metabolites), with colour as before. Bottom: node borders are shown and edges are made thicker, with colour as before. It would be possible also to add charge balance, but the source material (Enzyme List) does not balance charge.

Certain information can be displayed as edge labels. The letters 'L' and 'R' are displayed on the edges when the "Side indicator (L or R)" check box is clicked. These letters indicate the side of the reaction (LHS or RHS, respectively) to which the corresponding compounds belong, as represented in the database. The 'L' is coloured blue, while the 'R' is coloured green, and similarly for their respective edges. This facilitates manual layout of the diagram. If the compounds on the left are substrates, then those on the right are products, and *vice versa*. Alternatively, reaction stoichiometry is displayed on the edges

when the "Reaction stoichiometry" check box is clicked. If the "Side indicator (Colour)" check box is clicked at the same time, the stoichiometry and the edges will be coloured, as before, according to the side of the reaction they are on. Certain properties of the edges and the nodes are also visualised when required. When the "Thick edge" check box is clicked, the edges are made thicker with colour as before. The node borders become visible when the "Show node borders" check box is clicked. This reveals that the non-operator nodes are white compared to the operator node, which is yellow. This functionality is shown in Fig. 4.18 above. It would be possible also to add charge balance, but the source material (Enzyme List) does not balance charge.

When the " Allow NOPs to move " checkbox is clicked, the non-operator nodes move towards the operator nodes. The former also move apart from each other by virtue of a "repulsive force' generated by the layout algorithm of the *relax()* method in the drawing class *GraphPanel*. However, if there are many nodes with common connections, clumping will occur in this mode. In the static mode, which is recommended as the preferred option for layout and visualisation, the nodes will remain apart. Clicking the "Shake NOPs" button displaces the non-operator nodes in the relaxed mode. Clicking the "Close window" button closes the drawing area. However, under certain circumstances the graph is not lost, and indeed, may be augmented and redisplayed. For example, different selections made from multiple-choice scrolling lists (Figs. 4.4 – 4.6 above) will be added to the graph and the graph redisplayed, even though the drawing area may be closed between selections.

When a user selects a particular reaction or pathway, the graphs are drawn dynamically by retrieving the relevant information from the files. The visualisation window then displays them. As stated before, a simple node placement algorithm arranges the nodes in columns starting at the upper left of the window. When a column reaches the bottom of the window, a new column is started to the right of the first one, and so on across the window. This arrangement accommodates a lot of nodes in an organised manner, and thus facilitates the user in enhancing it according to his requirements, by moving the nodes around. Manually moving a node automatically repositions the edges that are connected to that node and this in turn automatically repositions the labels associated with each edge. Fig. 4.19 below shows the initial layout of the reactions of the glycolysis pathway in a vertical column. The operators are coloured yellow.

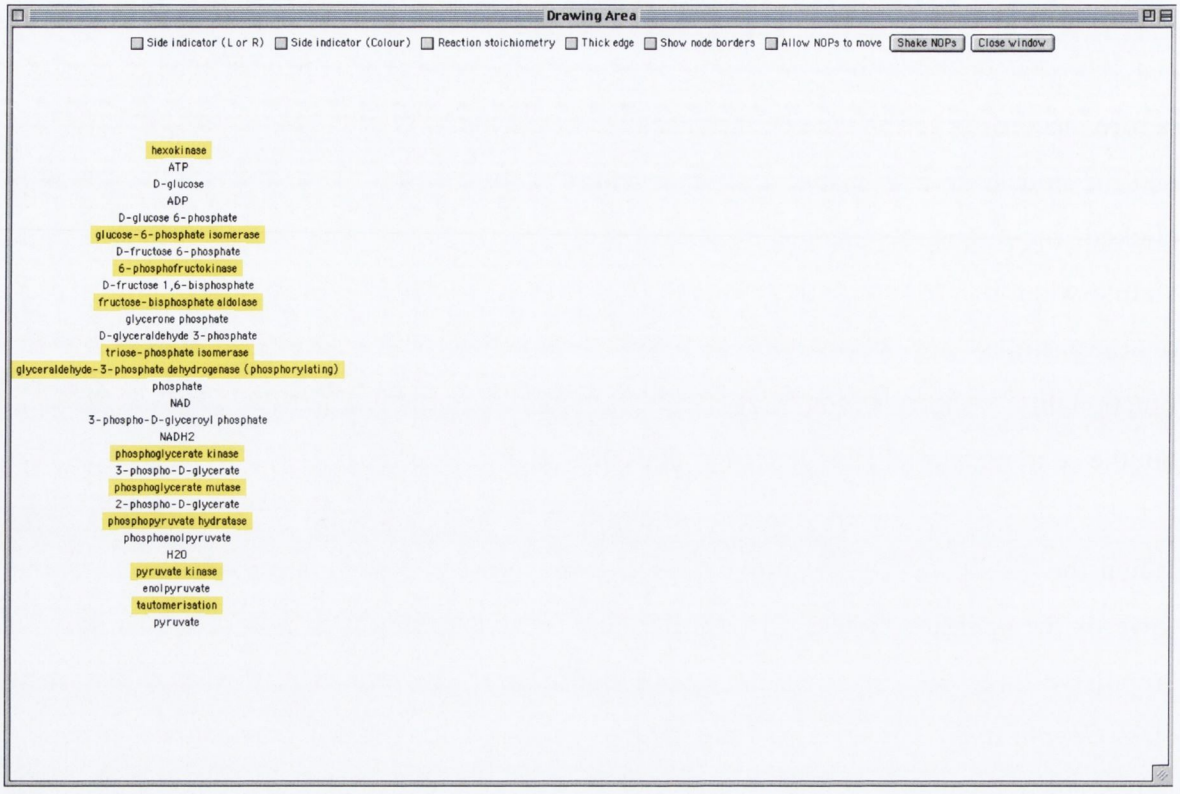


Fig.4.19. The diagram shows the initial arrangement of the glycolysis pathway reactions in the form of a column. The operators are coloured yellow. The pathway was accessed via the 'Classical pathways' menu item from the menu of the same name.

The pathway was accessed from the "Glycolysis.dat" file via the 'Classical pathways' menu item in the menu of the same name.

In Fig. 4.20 the display of Fig. 4.19 has been manually rearranged in the form a map. This was achieved by repositioning the operator nodes (yellow) in a snake layout and then, clicking the " Allow NOPs to move " checkbox, allowing the non-operator nodes to move towards them. In the static mode, the positions of all the nodes were fine-tuned. The operators, with one exception (tautomerisation), are enzymes, and the non-operators are metabolites. The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. This arises from the fact that in the "Glycolysis.dat" file the compounds on the LHS are substrates and those on the RHS are products in the reactions in order to conform to the direction of the pathway *in vivo*, which starts at D-glucose. The reaction stoichiometry is displayed as edge labels showing a value of 1 for all metabolites. The pathway stoichiometry (see Section 3.4 above), which is not displayed, is 2 from D-glyceraldehyde 3-phosphate to the end of the pathway (ketopyruvate). That is, a molecule of glucose is converted to two molecules of D-

glyceraldehyde 3-phosphate and in the rest of the pathway two molecules of each of the compounds take part in every reaction. The pathway generated, shows certain properties of the bipartite structure, namely, no pair of vertices joined by an edge has the same colour and the cycle at aldolase in the underlying undirected graph is even. Also, it incorporates an uncatalysed reaction, in which the operator is (the process of) tautomerisation.

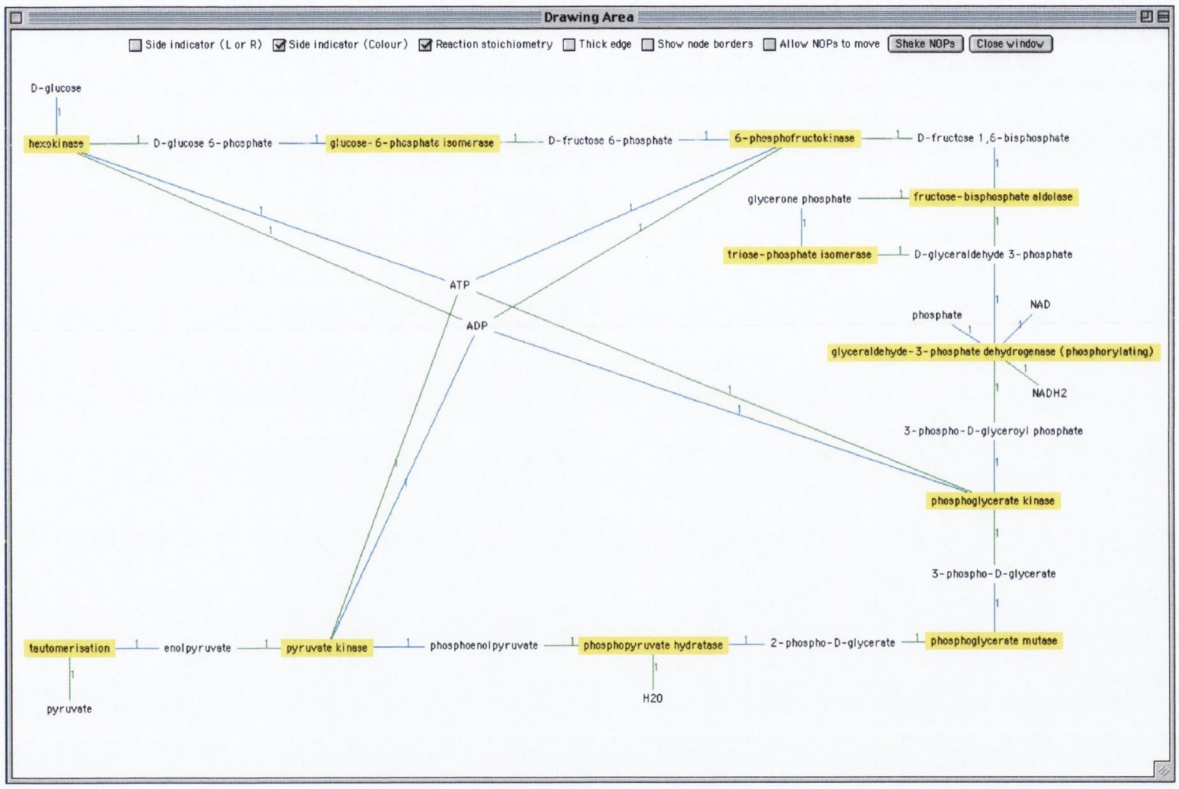


Fig.4.20. The diagram shows the nodes of Fig. 4.19 manually repositioned as described in the text. Yellow nodes are operators (all enzymes except 'tautomerisation') and the rest are non-operators (metabolites). The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. The stoichiometry is displayed as edge labels. The compounds ATP and ADP are represented as pooled metabolites. The pathway starts at D-glucose (indicated by the mouse pointer).

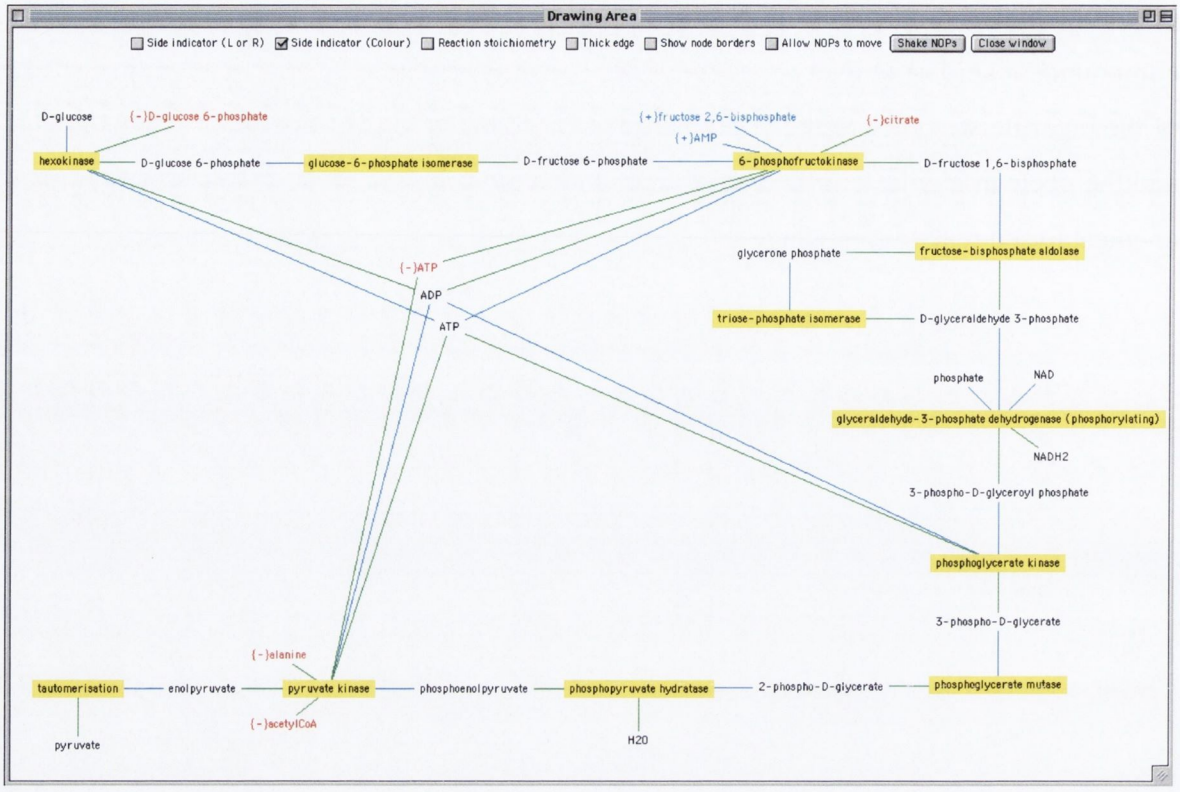


Fig.4.21. Map of glycolysis with effectors. Operators are yellow. Substrates and activators are connected to operators by blue edges, while products and inhibitors are connected to operators by green edges. Activators (blue font) are preceded by {+} and inhibitors (red font) are preceded by {-} to distinguish them from metabolites in the reactions.

Fig. 4.21 shows a map of glycolysis with effectors (modulators). It was accessed from the "Glycolysis.dat" and the "Effectors.dat" files via the 'Classical pathways' menu item in the menu of the same name with the check box for effectors clicked. All the nodes were manually placed. Alternatively, the effectors can be overlaid on Fig. 4.20 with a smaller amount of manual adjustment. As described previously, the activators (in blue font) are preceded by {+} and inhibitors (in red font) are preceded by {-}. The metabolites and effectors are both non-operators. Thus the graph remains bipartite. The first reaction is catalysed by hexokinase, which is inhibited by D-glucose 6-phosphate. Since D-glucose 6-phosphate can also feed into glycogen synthesis or the pentose phosphate pathway, the first irreversible step that is unique to glycolysis is that catalysed by phosphofructokinase and hence this is the main control step.

A user may manipulate the data using substring searches, and thus display reactions based on selected criteria. For example, the user may select any compound (enzyme or metabolite) and view all the reactions it is involved in. Using the 'Find and display' option from the "Substrings" menu with 'D-glucose 6-phosphate' entered in the text field

generated the diagram shown in Fig. 4.22, which displays all the reactions (8) in the user file for D-glucose 6-phosphate. It was laid out as described for Fig. 4.20. This may be described as a *metabolite-centric* approach. The first attempt at generating this map revealed an inconsistency in the "User.dat" file, i.e., the compound 'D-glucose 6-phosphate' was given as 'alpha-D-glucose 6-phosphate' in the following reaction:

alpha-D-glucose 1-phosphate = alpha-D-glucose 6-phosphate

This reaction is catalysed by the enzyme phosphoglucomutase (EC 5.4.2.2). The inconsistency resulted in a connectivity problem in the map. This has been rectified in the "User.dat" file.

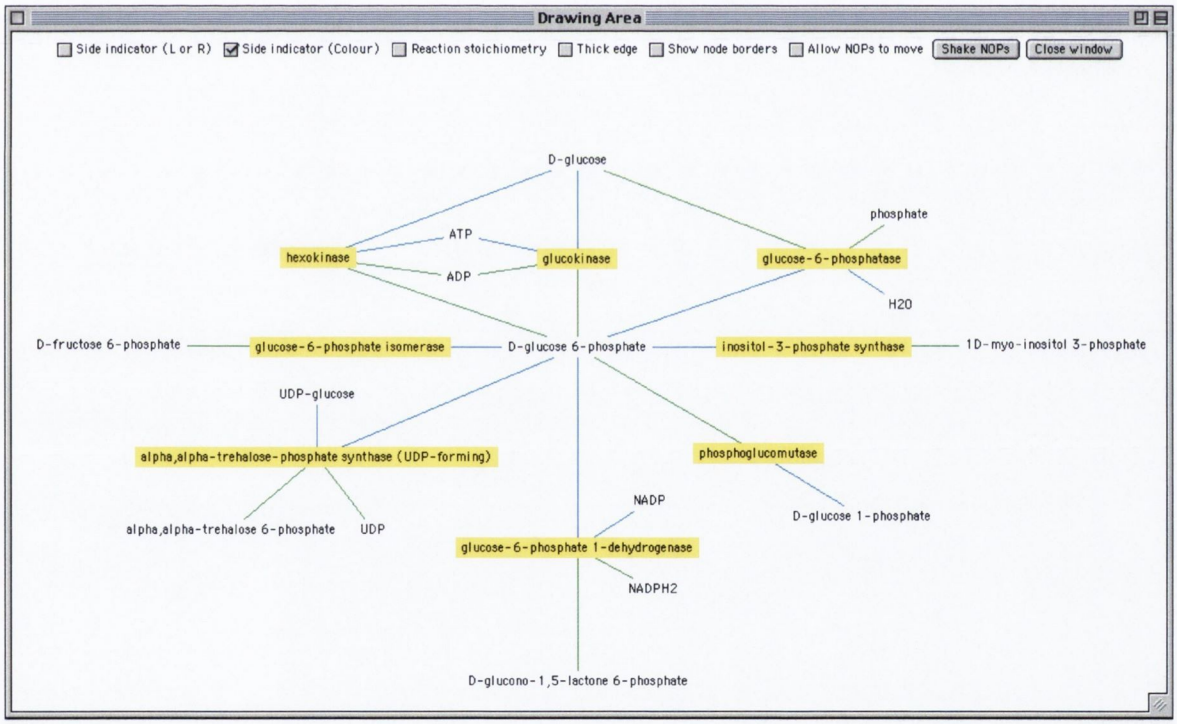


Fig.4.22. This diagram shows all the reactions in the user file in which D-glucose 6-phosphate (centre of diagram) is involved. Using the 'Find and display' option from the "Substrings" menu with 'D-glucose 6-phosphate' entered in the text field generated it. The operators (enzymes) are coloured yellow, while the rest are non-operators (metabolites). Edge colour, as in Fig. 4.18, indicates the side of the reaction to which the corresponding compounds belong as represented in the database.

The glycolysis pathway is essentially linear, whereas the topology of some pathways is much more complex, e.g., the pentose phosphate pathway. The latter is shown in Fig. 4.23 below. The operators are enzymes, and the non-operators are metabolites. In this case, the node borders are shown. The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. This arises from the fact that in the "PPP.dat" file the compounds on the LHS are substrates and those on the RHS are products in the reactions in order to conform to the direction of the pathway *in vivo*, which starts at D-glucose 6-phosphate. Using an *enzyme-centric* approach, it is not possible to distinguish the reactions of a given enzyme. The enzyme transketolase, for example, is involved in two different reactions. Appending a different number to the enzyme (transketolase1 and transketolase2) for each reaction separates them (see also Section 3.8 above). The pathway generated, shows certain properties of the bipartite structure, namely, no pair of vertices joined by an edge has the same colour and all the cycles in the underlying undirected graph are even.

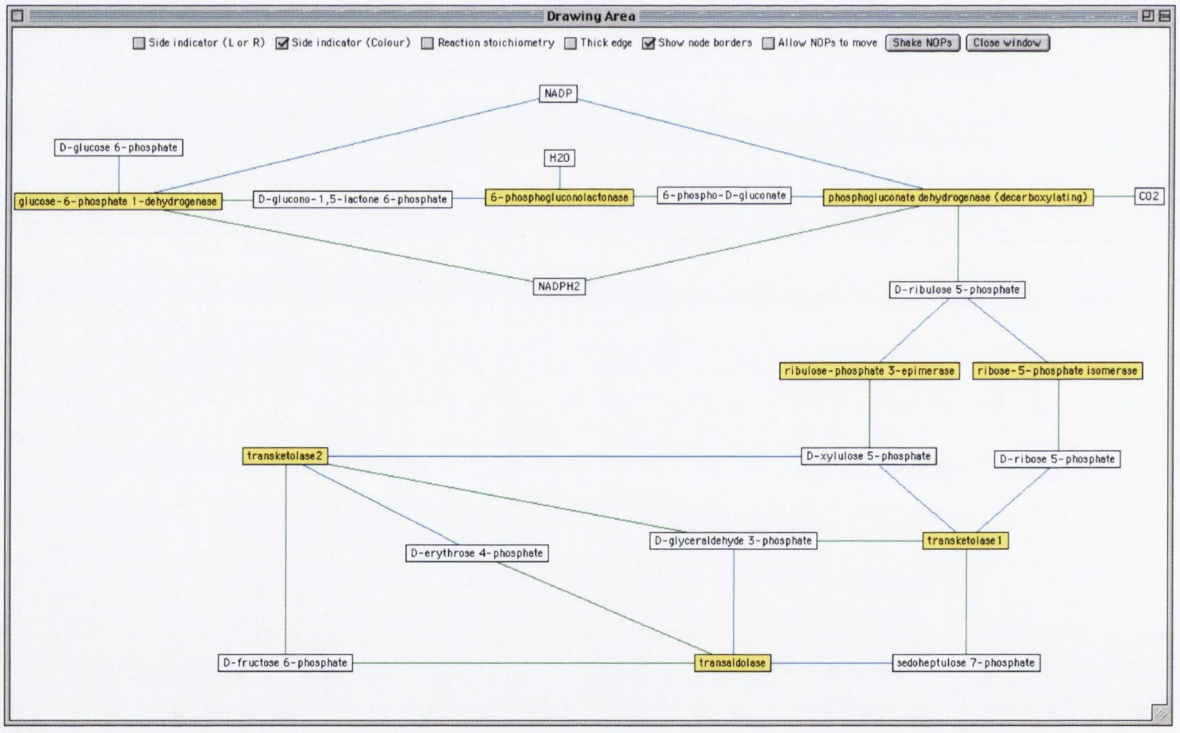


Fig.4.23. Map of the pentose phosphate pathway, which was manually laid out as described previously. Yellow nodes are operators (enzymes) and the rest are non-operators (metabolites). Node borders are shown. The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. The compounds NADP and $NADPH_2$ are represented as pooled metabolites. The pathway starts at D-glucose 6-phosphate (top left corner).

The trehalose pathway in *Magnaporthe grisea* (Wang et al., 2005) is shown in Fig. 4.24 below. This was generated from the user file by searching it with the following data contained in the"Pathway.txt" file:

3.2.1.28	alpha,alpha-trehalase
2.7.1.1	hexokinase
5.4.2.2	phosphoglucomutase
2.7.7.9	UTP-glucose-1-phosphate uridylyltransferase
2.4.1.15	alpha,alpha-trehalose-phosphate synthase (UDP-forming)
3.1.3.12	trehalose-phosphatase

In addition to the EC numbers and the enzyme names above, the search returned the following reactions, respectively, as text and in the form of a map:

alpha,alpha-trehalose + H2O = 2 D-glucose
ATP + D-glucose = ADP + D-glucose 6-phosphate
D-glucose 1-phosphate = D-glucose 6-phosphate*
UTP + alpha-D-glucose 1-phosphate = diphosphate + **UDPglucose**
UDP-glucose + D-glucose 6-phosphate = UDP + alpha,alpha-trehalose 6-phosphate
alpha,alpha-trehalose 6-phosphate + H2O = alpha,alpha-trehalose + phosphate

The diagram showed a lack of connectivity between UDPglucose and UDP-glucose (in bold), and between D-glucose 1-phosphate and alpha-D-glucose 1-phosphate (underlined) because of inconsistent representation in the user file. These were then rectified as follows: UDPglucose and UDP-glucose were replaced with UDP-D-glucose and alpha-D-glucose 1-phosphate was changed to D-glucose 1-phosphate. A subsequent search showed the correct linkage between the compounds in question. The resulting reaction-strings were saved to the file "Trehalose.dat". However, the reactions were filtered (and displayed) in the order in which they occurred in the user file. So, they were manually rearranged in pathway order (see Wang et al., 2005) in the "Trehalose.dat" file. In addition, the compounds on the LHS and the RHS in the phosphoglucomutase (EC 5.4.2.2) reaction were interchanged in order to conform to the direction of the pathway *in vivo* (*).

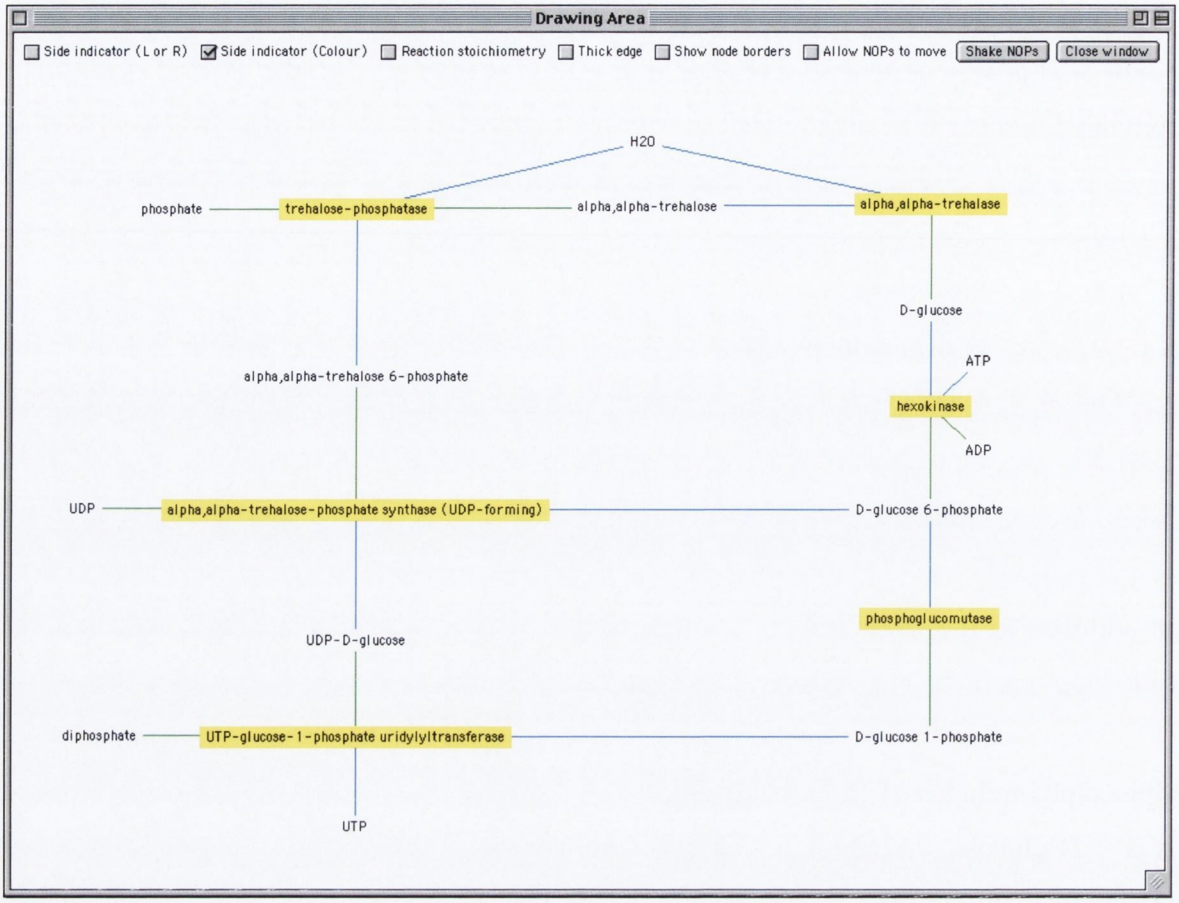


Fig.4.24. Map of the trehalose pathway in *Magnaporthe grisea*. This was manually laid out as described previously. Yellow nodes are operators (enzymes) and the rest are non-operators (metabolites). The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. The compound H_2O is represented as a pooled metabolite.

The "Trehalose.dat" file was accessed via the 'Any data file' menu item in the "Files" menu and automatically copied to the "Reactions.txt" file, which, as stated before, is used by the program. However, on running the program, the following message was output to the java console: "Input format error in tab(s) in line 7 of Reactions.txt file!". This was actually due to an extra blank line at the end of the "Trehalose.dat" file, which was removed. The pathway shows certain properties of the bipartite structure, namely, no pair of vertices joined by an edge has the same colour and all the cycles in the underlying undirected graph are even. In the pathway illustrated by Wang et al. (2005), Glucose-6-P appeared to be converted to UTP + Glucose-1-P. The bipartite graph model above avoids such ambiguities.

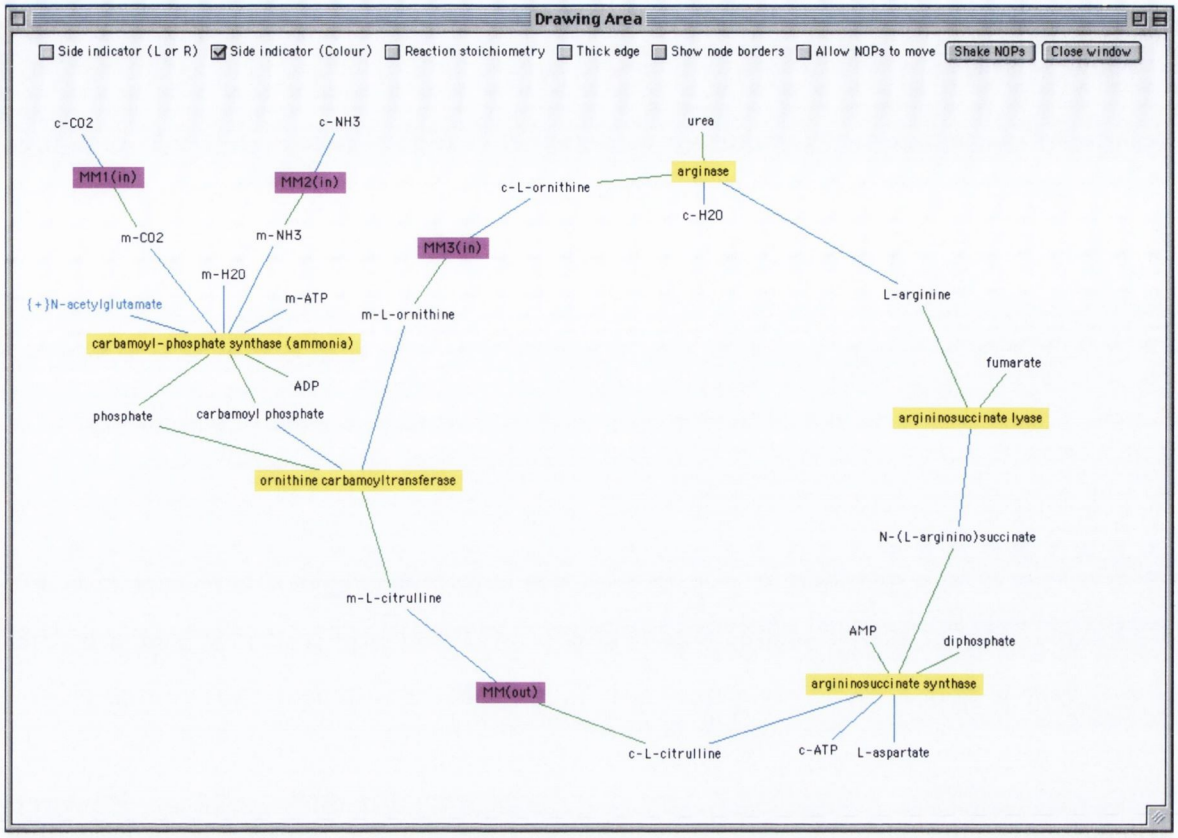


Fig.4.25. Map of the urea cycle. This was manually laid out as described previously. Coloured nodes are operators, with yellow nodes denoting enzymes and purple ones representing membranes enclosing a mitochondrion. The rest are non-operators (metabolites). The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. A metabolite connected to a membrane by a blue edge is moving towards it, while a metabolite connected by a green edge is moving away from it. The compound phosphate is represented as a pooled metabolite. The activator N-acetylglutamate (blue font) is preceded by {+} to distinguish it from a metabolite in the reaction.

The urea cycle is shown in Fig. 4.25 above. Coloured nodes are operators, with yellow ones representing enzymes and purple ones denoting membranes enclosing a mitochondrion. The rest are non-operators (metabolites). The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. This arises from the fact that in the "UreaCycle.dat" file the compounds on the LHS are substrates and those on the RHS are products in the reactions in order to conform to the direction of the pathway *in vivo*. A metabolite connected to a membrane by a blue edge is moving towards it, while a metabolite connected by a green edge is moving away from it. The pathway generated, shows certain properties of the bipartite structure, namely, edges join coloured vertices to non-coloured ones and all the cycles in the underlying undirected graph are even. The activity of carbamoyl-phosphate

synthase (ammonia) is determined (activated) by N-acetylglutamate, which is shown in blue font and preceded by {+}

Two further examples illustrate how pathway building may be used to reveal inconsistencies and errors in the database data. The spelling of a metabolite name (underlined below) in two consecutive reactions on the pentose phosphate pathway was found to be inconsistent:

D-glucose 6-phosphate + NADP = D-glucono-1,5-lactone 6-phosphate + $NADPH_2$

6-phospho-D-glucono-1,5-lactone + H_2O = 6-phospho-D-gluconate

These reactions are catalysed by the enzymes glucose-6-phosphate 1-dehydrogenase (EC 1.1.1.49) and 6-phosphogluconolactonase (EC 3.1.1.31), respectively. As stated before, typographical errors can also produce problems, e.g., the following was found: N -(L-arginino)succinate, i.e., the metabolite name contains a space between the leading N and the hyphen following it. These deficiencies resulted in a connectivity problem. However, the latter also resulted in the 'N' being treated as stoichiometry. They have been corrected in the "User.dat" file.

4.4.3 Additional output

Reactions that are displayed in diagrams are also shown as reaction equations in a separate scrolling window, together with EC numbers and enzymes. These data for the glycolysis pathway of Fig. 4.19 and Fig. 4.20 are shown in Fig. 4.26 below. A simple frequency histogram of operator code classes for classical pathways selected via the 'Classical pathways' menu item from the menu of the same name is automatically displayed in another scrolling window. The class frequency histogram for the glycolysis pathway of Fig. 4.19 and Fig. 4.20 is shown in Fig. 4.27 below. Information on named textbook pathways accessed via the 'Pathway information' menu item from the "Classical pathways" menu is displayed in the text area of a scrolling window. The information on glycolysis is shown in Fig. 4.28 below. Results for substring searches are output as strings of text to the Java Console. As a consequence, primary substring search results generated by an inclusion/exclusion filter may be viewed as text on this console.

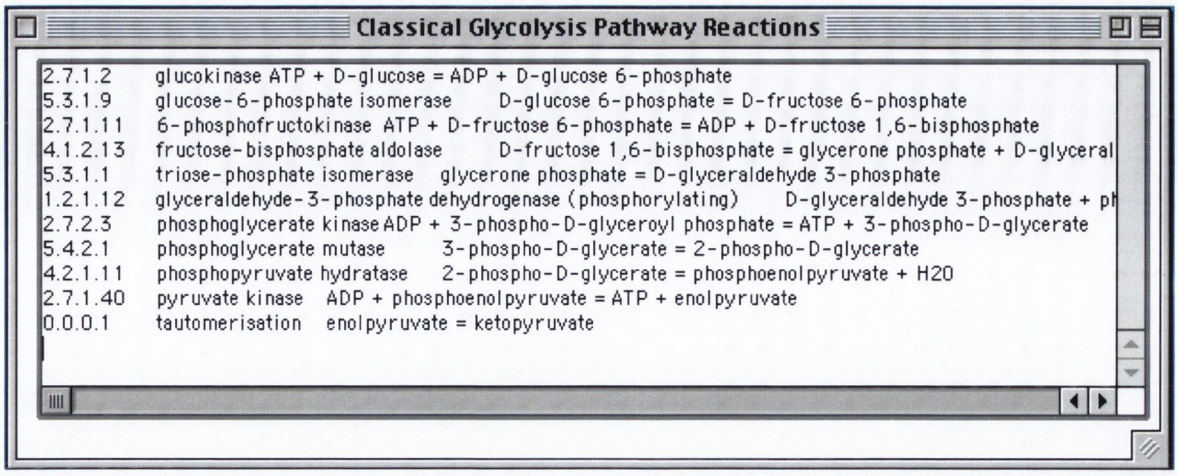


Fig.4.26. Shown in the window above are the reaction equations for the glycolysis pathway of Fig. 4.19 and Fig. 4.20, together with EC numbers and enzymes.

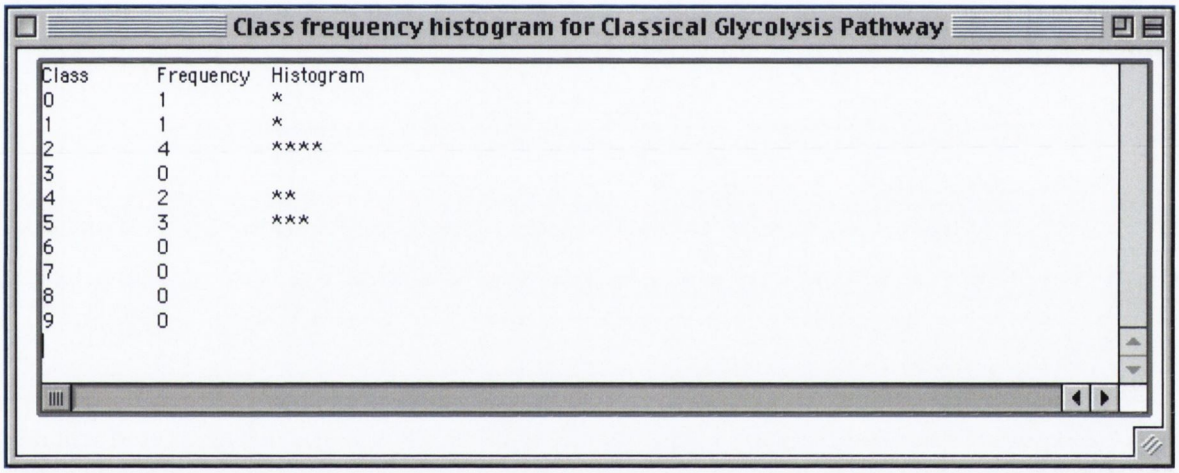


Fig.4.27. The display shows a simple frequency histogram of operator code classes for the glycolysis pathways selected via the 'Classical pathway' menu item from the menu of the same name. The numeric data are displayed graphically with a histogram that simply shows each class frequency value as a bar consisting of that many asterisks (*).

There is also output to secondary files. Going to the "Show" menu and clicking on the 'HTML code' menu item will access the HTML reaction code, which is generated when reactions are displayed. Going to the "References" menu and clicking on the 'Search-specific' menu item accesses search-specific references, which are generated when a user selects operators by code or name, or 'Reactions or regulatory actions' from the "Events" menu (Fig. 4.29 below).

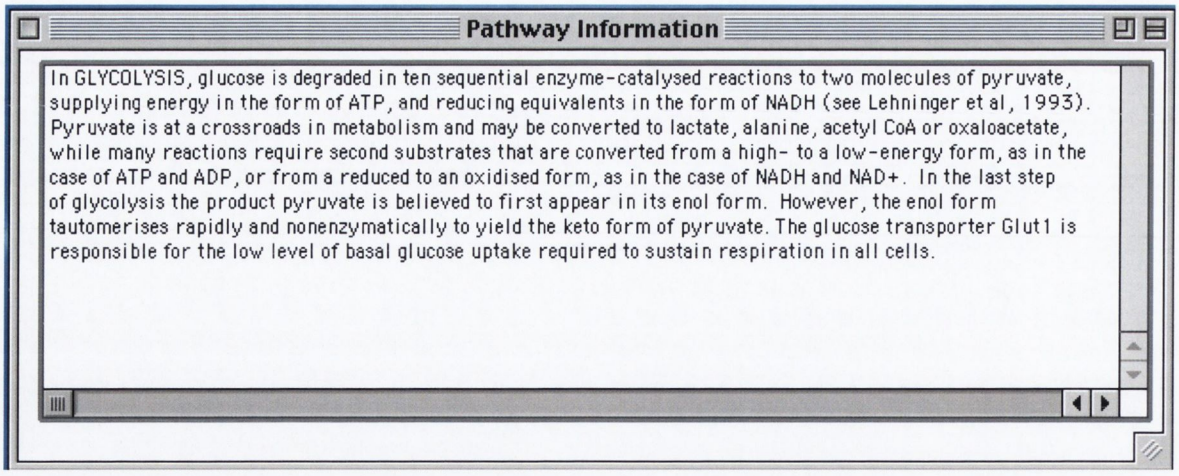


Fig.4.28. This is a description of the glycolysis pathway. It was accessed from the glycolysis pathway information file.

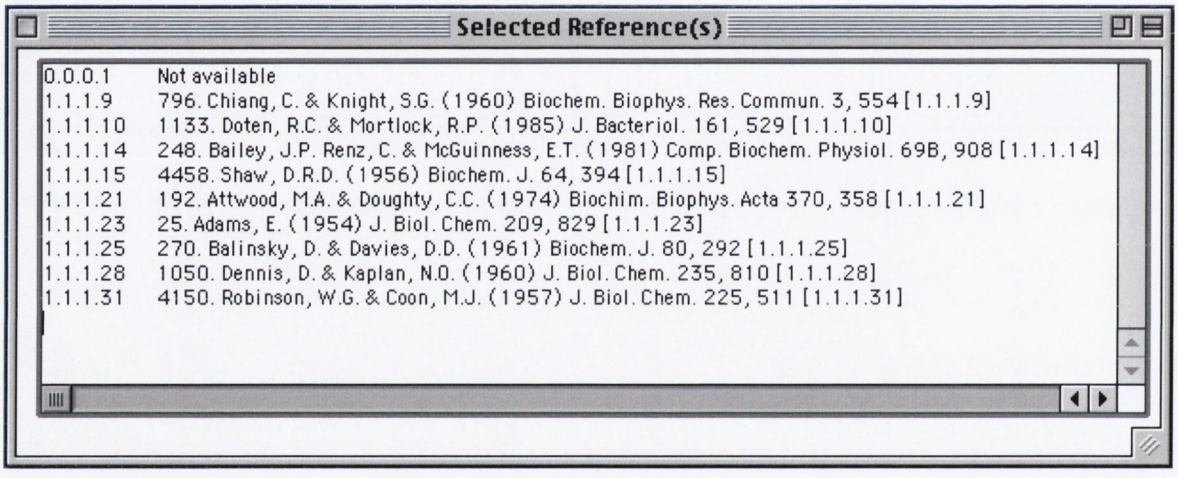


Fig.4.29. These are search-specific references, which were automatically generated when the relevant operators were selected by code. It would not be difficult to add titles, via the PMID numbers, at a later stage.

Finally, there is static output. Going to the "Classical pathways" menu and clicking on the 'Pathway information' menu item displays a small window containing buttons with the names of the pathways and pathway EC numbers. Clicking a button accesses the relevant information. Going to the "Information" menu and clicking the 'About using the system' or the 'About the application' menu items, accesses information on using the system or a simple note on the application itself, respectively. Going to the 'References' menu and clicking the 'General' or 'Reaction specific' menu items, accesses the files on general or reaction-specific references, respectively. From the "View file" menu, the user can access the contents of the "Reactions.txt" file via the Java Console or in the text area of another scrolling window.

§4.5 The WP applet

4.5.1 General description

An interactive Java applet (WebPathway) was devised. It allows visualisation and exploration of reactions, pathways and subnets over the Internet. The application program above converts reaction string data into an HTML format suitable for pathway display using this applet. The automatically generated HTML file is copied and entered into the applet. The data are fed to the applet via the applet arguments in the html source. The present applet is based on the Graph Java applet distributed in the Java development distribution of Sun Microsystems Inc. The modified applet was tested with reactions from the "Reactions" file. This applet also has links to external databases. It has been designed and implemented using an object-oriented approach. The source code for the applet may be viewed in Appendix A5.

4.5.2 Classses

There are four classes in this applet. They are class *TrivialApplet* and the drawing classes, *Node*, *Edge* and *GraphPane*l. The function of the *TrivialApplet* class is to set up a GUI for the applet. The drawing classes are similar to those in the RSD Java application above (4.3), with differences. These include additional simple node placement algorithms (described below) and a function to adjust a double value to two decimal places with rounding. The thread is back in the *start()* method, and the method *action( )*, to change a Boolean value when a check box or button is clicked, is back in *TrivialApplet*, as in the original applet. Further details are given below (4.5.3).

4.5.3 Modifications to the original applet

In the original applet there was no control over the import of data to the display window. In *TrivialApplet*, in the present applet, an array with the name(s) of the pathway(s) is initialised and an array of buttons, denoting the different pathways, is created. This allows the pathways to be called from the display window itself. However, in the present case, only one pathway, which is created by the application, is available, but see chapter 5 below. Furthermore, this applet has links to external databases via clickable buttons that

are hard-wired in Java. Initialising an array with the names of the Web sites and an array with the URLS of the Web sites achieves this. An array of buttons denotes the different Web sites and the method *init()* is overridden to display them. An implementation of the *actionPerformed()* method taken from the *ActionListener* Abstract class is used to find the name of the button that was pressed. Each web site name in the array of web site names is inspected and the relevant page from the Web site is displayed. As stated before, the border layout is used in the applet (see 4.2.4 above). It contains different zones. A component is added to a zone using the *add()* method and two parameters (the zone and the component) are passed to the method. In the original applet there were two zones ("Center" and "South"). An additional zone ("North") has been added to the present applet.

In *GraphPanel*, additional simple node placement algorithms have been added. These are termed *linear diagonal*, *circular* and *random*. The linear diagonal algorithm arranges the nodes diagonally across the window, starting at the upper left and proceeding towards the lower right of the window. It accommodates fewer nodes than the linear vertical one. The circular algorithm arranges the nodes in a circle and offers three options with respect to its size: small, medium or large. The random layout algorithm distributes the nodes around the window and accommodates very many nodes, but the window becomes cluttered as a result of overlapping nodes in the diagram.

The *paintNode()* method contains some extra items. For example, it has some mathematical features, which give the number of nodes and the number of edges in the graph, as well as the edge/node ratio. This ratio is proposed as a measure of the interconnectivity of the nodes that are displayed. The higher the ratio, the more connected the nodes are. The *paintNode()* method has a function to adjust a double value to two decimal places with rounding, which is used in the ratio calculation. It also displays messages in the display window for user attention. Finally, this method contains code to show the node borders and to deal with effectors if they are present in the data. The *update()* method has new code to deal with stoichiometry and reaction side indicators in addition to thickening an edge, as described above (see 4.2.15).

4.5.4 Running the applet

The necessary delimiters are implemented in the *StringTokenizer* to deal with the incoming information. For example, with respect to the EC 1.13.11.21, they will be of the form:

beta-carotene15,15'-dioxygenase$beta-carotene|L/1;
beta-carotene15,15'-dioxygenase$O2|L/1;
beta-carotene15,15'-dioxygenase$retinal|R/2;

In HTML format, it looks as follows:

```
<title>TrivialApplet</title>
 <hr>
 <applet codebase="Java Classes" code="TrivialApplet.class" width=1150 height=700>
 <param name=GET_REACTION(S) value="beta-carotene 15,15'-dioxygenase$beta-
carotene|L/1;beta-carotene 15,15'-dioxygenase$O2|L/1;beta-carotene 15,15'-
dioxygenase$retinal|R/2;">
 </applet>
 <hr>
```

The TrivialApplet.html icon is opened and the HTML code is pasted into it. The program is run, by clicking the arrowhead button, either on the floating toolbar or the GUI in CodeWarrior. To view the HTML code in the applet, the user may press the command and T keys at the same time, or go to the menubar, click Applet, and then click Show Applet Tag (or the equivalent PC actions). The data can also be entered manually into the code (see Chapter 5 below). Right clicking on a two-button mouse has no effect in the applet.

4.5.5 Output from the WP applet

When the program is run, by clicking the arrowhead button, either on the floating toolbar or the GUI in CodeWarrior, a large display window is generated (Fig. 4.30). This has been shown previously (Fig. 4.14 above) but is reproduced here for convenience and with the difference that it now displays the options in the drop down list at centre top. As stated before, this list allows selection of the graph layout algorithms. The window includes a display area for the selected reaction(s) and/or regulatory action(s), which are accessed via the 'GET_EVENT(S)' button inside top left. The check boxes and the button at the bottom of the window are connected with certain aspects of the display and have the same functionality as those with the equivalent labels in the application. However, the applet

also has a "Maths" option. Clicking the "Maths" check box reveals the number of nodes, the number of edges and the edge to node ratio. The 12 buttons (links) at the extreme left allow access to external databases.

The instruction "Select a LAYOUT and/or click GET_EVENT(S) button." is given across the top of the window. When the user follows this instruction, the following message is output:

NOW CLICK WINDOW OR CLICK 'Allow NOPs to move' CHECK BOX!

Either clicking in the window or clicking the "Allow motion" checkbox visualises the display. The former gives a static view, while the latter results in motion of the non-operator nodes. If a user clicks in the display area without having first selected a layout or clicked the "GET_EVENT(S) button", the following message is output:

Select a LAYOUT and/or click GET_EVENT(S) button!

When in use, the applet displays an instruction on how to change the initial simple layout.

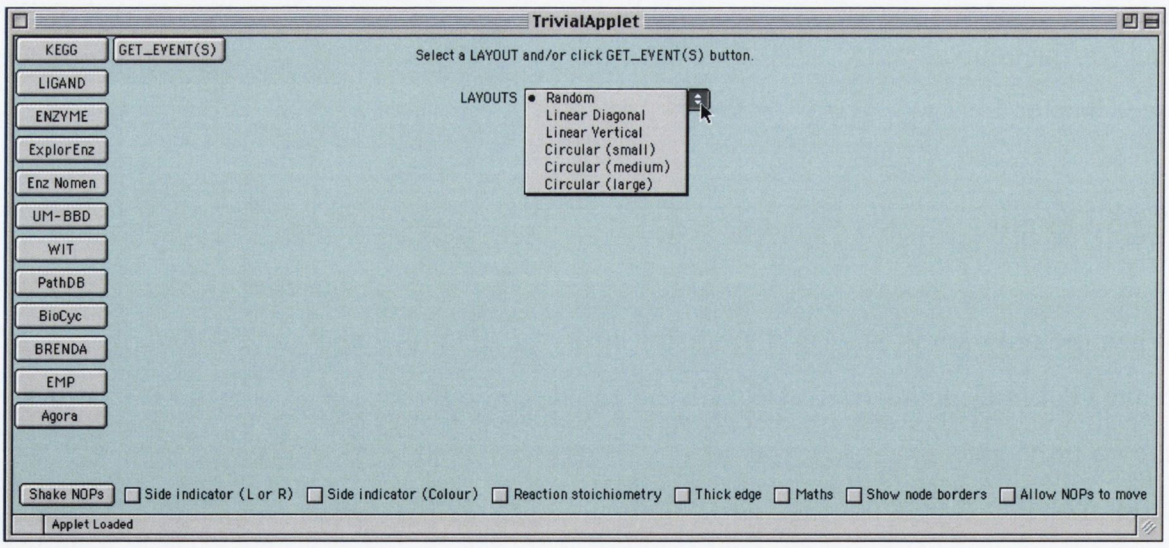


Fig.4.30. Graphical user interface for interaction with the applet. Instructions are given across the top. A drop down list at centre top allows selection of simple layout algorithms. The reactions and/or regulatory actions generated by the application and copied to the applet are accessed via the 'GET_EVENT(S)' button inside top left. The button and check boxes at the bottom are connected with certain aspects of the display. The buttons at the extreme left allow access to external databases.

Fig. 4.31 below shows the initial random layout of the pentose phosphate pathway. Operator nodes (enzymes) are yellow and non-operator nodes (metabolites) are white. A user may interact with the nodes by moving them around. Clicking the "Maths" check box revealed the number of nodes (22), the number of edges (28) and the edge to node ratio (1.27). The applet may be viewed in a browser by opening Internet Explorer and dragging TrivialApplet.html into the browser. This may be accessed over the Internet. From the menu "View", one can refresh the Applet, which is necessary in order to change the initial simple layout, or one can simply click the refresh icon in a browser. One can view the HTML source code by clicking the menu item 'Source' from the menu "View".

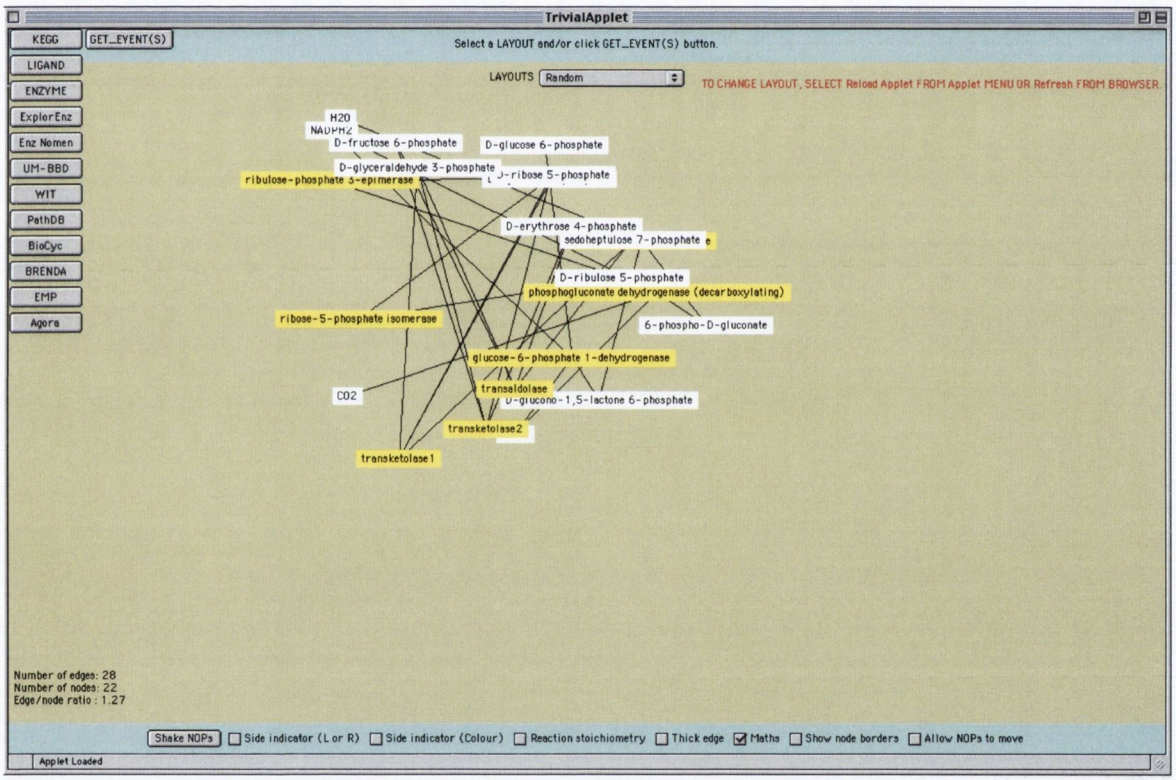


Fig.4.31. Initial random layout for the pentose phosphate pathway. The operator nodes (enzymes) are yellow and the non-operator nodes (metabolites) are white. The three zones ("North", "Center" and "South") of the border layout are clearly visible. Information on the number of nodes (22) and edges (28), and the edge/node ratio (1.27) is displayed in the bottom left corner.

Fig. 4.32 below shows the improved, manual layout of the pentose phosphate pathway. This was achieved as described previously for the application. Yellow nodes are operators (enzymes) and the rest are non-operators (metabolites). Node borders are shown. The metabolites connected to the enzymes by blue edges are substrates, while those connected

by green edges are products in the reactions. As stated before, this arises from the fact that in the "PPP.dat" file the compounds on the LHS are substrates and those on the RHS are products in the reactions in order to conform to the direction of the pathway *in vivo*, which starts at D-glucose 6-phosphate. The number of nodes (22) and edges (28), and the edge/node ratio (1.27) is displayed in the bottom left corner of the window.

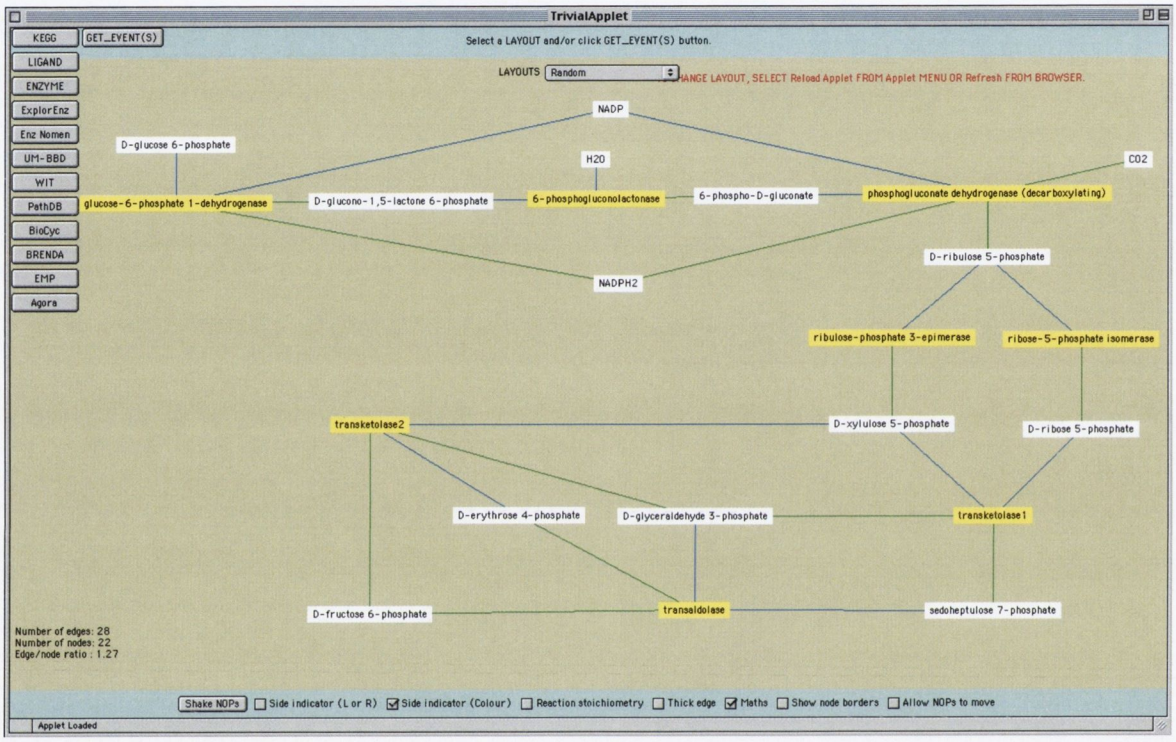


Fig.4.32. Map of the pentose phosphate pathway. The yellow nodes are operators (enzymes) and the rest are non-operators (metabolites). Node borders are shown. The metabolites connected to the enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. The compounds NADP and $NADPH_2$ are represented as pooled metabolites. The pathway starts at D-glucose 6-phosphate (top left corner). Information on the number of nodes (22) and edges (28), and the edge/node ratio (1.27) is displayed in the bottom left corner.

Fig. 4.33 below shows the graphs for the following three reactions:

adrenaline + adrenergic receptor = adrenaline-(adrenergic receptor)

glucagon + glucagon receptor = glucagon-(glucagon receptor)

e-D-glucose = D-glucose

The operators for the receptor reactions are the adrenergic receptor and the glucagon receptor, respectively. That for the transporter reaction is hs_Glut1 (Glut1 of Homo sapiens). This is an extended operator (see 3.7.2 above), where 'hs' represents Homo sapiens. The glucose transporter Glut1 is responsible for the low level of basal glucose uptake required to sustain respiration in all cells. The prefix 'e' on the left hand side of the reaction denotes the extracellular form of D-glucose and prevents D-glucose from joining with itself in the same reaction in the graph representation.

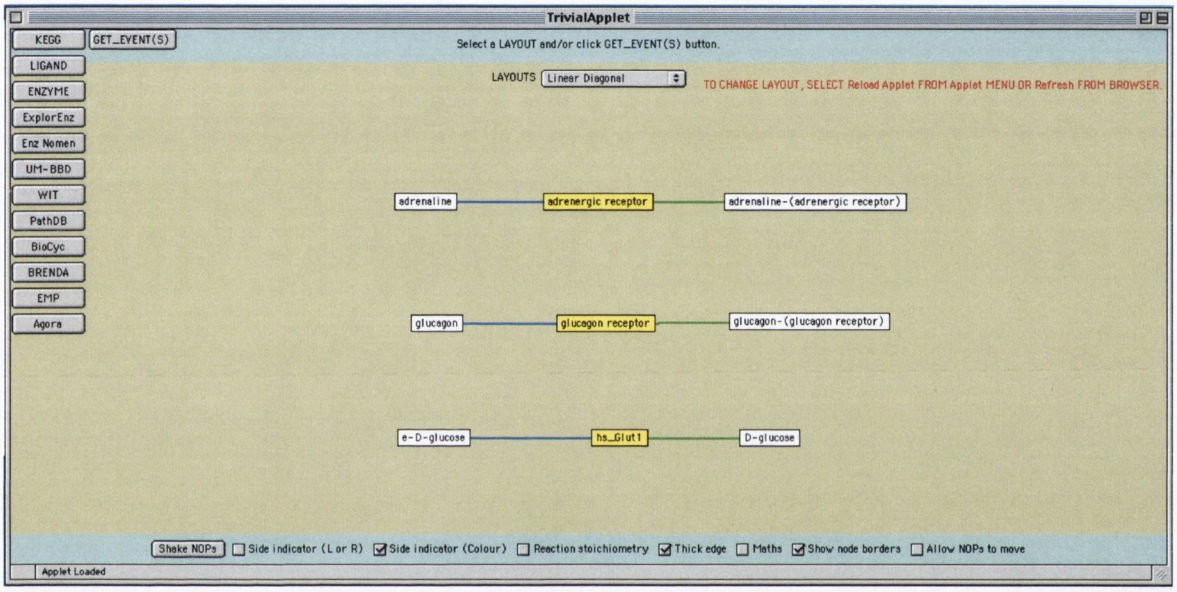


Fig.4.33. Graphs of two receptor reactions and one transporter reaction (see text). The yellow nodes are operators and the white ones are non-operators. Node borders and thick edges are shown. In the receptor reactions, a blue edge represents binding of the ligand (to the receptor) and a green one represents production of the ligand-receptor complex. In the transporter reaction, a blue edge represents movement of the non-operator towards the operator and a green one represents movement away from the operator.

§4.6 Conclusion

The application software and system, using the simple object bipartite graph model, was successfully able to produce graphs of biochemical reactions, which could be manually laid out as diagrams of metabolic pathways of varying complexity and style. It displayed certain information and performed certain functions as required. In addition, the system was able to generate and display useful information in the form of numeric data and text (Figs. 4.26 - 4.29 above). The interface was able to search the system in response to

mouse clicks, selections from lists, and textual input. The system provides an easy-to-implement and platform independent solution to the display and manipulation of metabolic pathways and provision of interactive access over the Internet.

Inconsistencies in the user file resulted in a number of connectivity problems. Greater standardisation is necessary, both within databases and between databases. The need to include spontaneous (uncatalysed) reactions and the fact that beta-D-glucose 1-phosphate spontaneously mutarotates to form alpha-D-glucose 1-phosphate, implies that the prefixes beta- and alpha- should be included where appropriate. Manual layout has the advantage that the user may arrange the pathway as he pleases, but the process could be tedious. Automated layout of large pathway graphs remains a problem to be solved. The manual transfer of data to the applet is a deficiency that needs to be addressed.

CHAPTER 5

EXTENSIONS

§5.1 Independent applets

The information in the "User.dat" and "General.dat" files is limited. For example, there is no information on reversibility *in vivo*, since such information is not given in the IUBMB Enzyme List, or on molecular formulae, etc. Even the named pathway files have limited information. Furthermore, the previous applet was limited to displaying only one set of reactions generated by the application at any one time. To address these limitations, two independent, self-contained applets are presented. One is a single pathway (glycolysis) applet, while the other deals with multiple reactions and pathways. The glycolytic pathway is used to describe and illustrate the general principles and approaches involved in the provision of detailed information in a known situation, i.e., how to represent different aspects of a well-known textbook pathway and optimise the representation (5.2). The multi-pathway applet (5.3) allows selection of different reactions and pathways within the same applet.

§5.2 Glycolysis pathway applet

5.2.1 General description

An interactive Java applet for visualising and exploring the glycolytic pathway over the Internet was devised. This is a specialised applet with a lot of functionality. It is independent of the application, since the input data (n-ary relations) were entered manually and the program is designed for a different input from that generated by the application. Some parameters are computed from the input data, while others are contained within the program. It shows new ways of visualising control and displaying information. In addition, it has some extra graph properties. The program displays the pathway with predefined coordinates for the layout of the operator (enzyme) nodes. The basic structure of the applet is as described previously (see Section 4.5), but has many additional features, which will be described. Furthermore, the present applet can display a lot of information in a manner that avoids clutter. The new functionality will be the main focus of this section. The applet has been designed and implemented using an object-oriented approach. The source code may be viewed in Appendix A6.

5.2.2 Data input

Data are input to the applet via the applet argument in the html source. However, some data are contained within the code. The following example shows the input for the hexokinase (HK) reaction:

<param name = glycolysis value = "HK$glucose/6|0&s#n!-1;HK$ATP1/0|3&c#n!-1;HK$ADP1/0|2&c#n!1;HK$G6P/6|1&p#n!1;HK$H+1/0|0&p#n!1">

A semi-colon separates each n-ary relation, and a different delimiter separates each component of an n-ary relation. The enzyme is separated from a metabolite by the dollar sign '$' and a metabolite is separated from the number of carbon atoms by a forward slash '/', while the latter is separated from the number of phosphorous atoms by '|', which in turn, is separated from metabolite role by the ampersand '&'. Next comes reversibility (where n = no) separated by '#', and finally, signed stoichiometry separated by the exclamation mark '!'. A similar format is used for the input of the other reactions. Numbers are appended to the names of the side compounds (e.g., ATP1) to allow the Java source code to display them as separate compounds (e.g., ATP) in the map. Within the applet, the value of the parameter is obtained by the *getParameter()* applet class method. The HTML source code is in Appendix A8 and may be viewed in the applet or in a browser as described for the WebPathway applet (see 4.5 above).

5.2.3 Classes

There are seven classes in this applet. Class *TrivialApplet* sets up a GUI with check boxes for the applet. Three new classes, *Information*, *InfoOptions* and *InfoText*, deal with different aspects of information. The drawing classes, *Node*, *Edge*, and *GraphPanel*, are similar to those in the "WebPathway" applet described in the previous Chapter (see Section 4.5), but the *GraphPanel* class has significant differences, which will be described. The packages used are the same as in the previous applet. Details of the classes are given below.

5.2.4 Class *TrivialApplet*

The class *TrivialApplet* inherits variables and methods from the class *Applet* in the package 'java.applet.Applet'. The objects named 'panel' and 'documentation' are created as instances of the *GraphPanel* and the *Information* classes, respectively. The *init()* method attaches a *BorderLayout* to the applet window and adds the 'panel' and 'documentation' objects to the Center and North locations, respectively. It then creates a new panel 'p', adds check boxes to it, and adds the panel to the South location. Some of the check boxes have similar functions to those in the previous applet (see Section 4.5) while others have different functions, which will be described. This method also assigns coordinates to the fixed (operator) nodes, so that the program can display the pathway with predefined coordinates for the layout of these nodes. Since the *getParameter()* method only returns the value of the parameter as a string, the substrings for carbon, phosphorous and stoichiometry are converted to integers during processing of tokens from the string tokenizer.

5.2.5 Class *Information*

The class *Information* inherits variables and methods from the class *Panel*. The objects named 'infopt' and 'infotext' are created as instances of the *InfoOption* and the *InfoText* classes, respectively. The function of this class is to add information components to the applet window. The method *Information()* determines the layout pattern for these components and adds 'infopt' and 'infotext' to the West and East locations, respectively, of the North region of the applet window.

5.2.6 Class *InfoOption*

The class *InfoOption* inherits variables and methods from the class *Panel*. The function of this class is to allow the user to select textual information as required. The method *InfoOption()* creates a dropdown list of choices for this purpose. The choices include the names of substances and parameters. An item is selected by clicking on it. The method *action()* gets the selected item and sends it to the *showline()* method in the *InfoText* class.

5.2.7 Class *InfoText*

The class *InfoText* inherits variables and methods from the class *TextArea*. The function of this class is to display the selected information in a text area. Variables of type String are declared and initialised with the information to be displayed, which is decided on by the *showline()* method. The *InfoText()* method sets the size of the text area and places the information into it. Instructions on how to use the applet are included. The information on compounds is not extensive, but sufficient to show how the system works. The contents may be viewed directly as part of the source code in Appendix A6.

5.2.8 Class *GraphPanel*

The *GraphPanel* class contains new features. For example, the method *GraphPanel()* sets up five drop down lists for selection of edge and node attributes. Event handlers are attached to the drop down lists, each of which generates an *ItemEvent* whenever its state is changed. Since the *ItemListener* contains only a single method, an anonymous class is used to implement the interface and handle the *ItemEvent*. These lists provide information in the pathway map as required.

The choices for *edge label* are "None", "Stoichiometry", "Metabolite role", "C value", and "P value", where C and P stand for carbon and phosphorous, respectively. As stated before, *stoichiometry* indicates the proportions of molecularities with which reactants and products are involved in reactions. This is displayed as signed integers on the edges. Thus, if 'n' represents the number of molecules, then '-n' indicates the consumption of n molecules, and 'n' the production of n molecules. Furthermore, the stoichiometry is given as *pathway stoichiometry* (indicated on the RHS of the graph), i.e., the reaction stoichiometry is doubled after aldolase converts D-fructose 1,6-bisphosphate to D-glyceraldehyde 3-phosphate and glycerone phosphate, with the subsequent conversion of the latter to the former by triose-phosphate isomerase, to reflect this situation (see Section 3.4). The *role* of a metabolite in a biochemical reaction represents a relationship with its enzyme and is, therefore, a property of the corresponding edge. The edge labels for the role of a metabolite as substrate, product, currency metabolite or cofactor are given as S, P, C, or F, respectively. Carbon and phosphorous values, displayed separately on the edges, represent the numbers of these atoms in the relevant compounds. All these data are input to the applet (see Section 5.2.2). The *update()* method handles the edge data (see below).

The choices for *edge function* are "None", "Metabolite flow", "Carbon flow", "Phosphorous flow", and "Reversibility". With respect to metabolite flow, a coloured edge represents the *direction* of a molecule in towards an enzyme (consumption of substrate), while an edge of a different colour represents the direction of a molecule out from an enzyme (formation of product). Carbon and phosphorous flow indicate by edge colour the 'flow' of carbon atoms and phosphorous atoms along the edges. Reversibility also uses edge colour to differentiate flow in one direction (essentially irreversible) from that in either direction (reversible). The reversibility data (n = no, y = yes) are also input to the applet (see Section 5.2.2). The *update()* method handles the colouring of the edges and their labels (see below).

The choices for NOP label are "None", "Vector", and "Formula Wt", where 'Wt' represents weight. The molecular formula of a compound indicates the proportions of atoms combined to make up the molecule. It is proposed to represent the molecular formula of a compound on a metabolic pathway, subnet or network by a *composition vector*, whose components give the number of each atom type on the network that is present in a link molecule. For glycolysis, the composition vector is (C,H,O,P), where C, H, O, and P are the usual symbols for hydrogen, oxygen and phosphorous, respectively. For example, the molecular formula for glucose is $C_6H_{12}O_6$ whereas the composition vector is (6,12,6,0). The number of components of a composition vector is equal to the number of atom types on the network. As a molecule moves through glycolysis, the values of the components of the composition vector change, but the number of components remains the same. The vector representation is considered more convenient when working with metabolic pathways and networks. The formula weight (or relative molecular mass, M_r) of a molecule is given as the sum of the atomic weights. The *paintNode()* method handles these data and displays them as labels above the relevant metabolite nodes (see below).

The choices for OP attribute are "None", "Ion", "TD", "EC number", "Gene name", "Isomerases", "Transferases", "Lyases", and "Irreversible", where 'TD' represents thermodynamics. Nearly all the glycolytic enzymes require Mg^{2+} for activity, including pyruvate kinase, which also requires K^+. Where required, these ions are displayed as labels above the relevant enzyme nodes. According to Voet and Voet (1995), *thermodynamics* in biochemistry is usually concerned with the conditions under which processes occur spontaneously (without outside intervention). The *free energy change* (ΔG) of a reaction determines whether it is spontaneous at a given temperature and

pressure. The ΔG value of a reaction under standard conditions of concentration (1M), temperature (25^{O} C) and pH (7.0) is defined as the *standard free energy change* of a reaction, denoted by $\Delta G^{0\prime}$ (delta G zero prime). The $\Delta G^{0\prime}$ values are displayed as labels above the relevant enzyme nodes, while the units (kJ/mol) are shown to the right of the graph. However, the reactions within the pathway cannot be considered in isolation. The EC numbers and the gene names are also displayed as labels above the corresponding enzyme nodes. Different node colours are used to highlight isomerases, transferases and lyases, and the (essentially) irreversible reactions.

The choices for effector are "None", "G6P", "ATP", "Citrate", "AcetylCoA", "Alanine", "Fluoride", "AMP", and "F2,6BP". Effectors are small molecules that regulate the activity of allosteric enzymes by activation or inhibition (see 1.3.1.). In glycolysis, AMP and F2,6BP are activators, while the rest are inhibitors. Fluoride is used to inhibit glycolysis *in vitro*, while the rest act *in vivo*. Selecting an effector from the list highlights any enzyme(s) regulated by it by assigning a new colour to the relevant node(s). Different colours are used for inhibitors and activators. The *paintNode()* method handles the display of the OP attributes and the information on effectors (see below).

The *update()* method assigns the colours to the various edge labels. The colours for stoichiometry and metabolite role are computed in the program from the sign of the stoichiometric coefficient. If the coefficient is negative, the colour is set to blue, and if it is positive, the colour is set to green. Magenta is assigned to carbon values and red to phosphorous values. The colours for metabolite flow are also computed from the sign of the stoichiometric coefficient, as already described, and interpreted below the diagram. The *update()* method also assigns the edge colours for carbon flow (magenta) and phosphorous flow (red), and computes those for reversibility. If reversibility is “y”, the corresponding edge is black, otherwise, if it is ‘n’, the corresponding edge is coloured according to the sign of the stoichiometric coefficient, as already described. The reversibility colours are explained below the map.

The *paintNode()* method assigns the NOP data to each metabolite and evaluates the formula weights. The data include the composition vector and the full, common name of the metabolite. The values for the components of the composition vectors are added to an array in the correct sequence for a given metabolite and the formula weights are computed

by adding the atomic weights in the composition vectors. For example, glucose (6, 12, 6, 0) has a formula weight of

$$(6 \times 12.01) + (12 \times 1.01) + (6 \times 16.00) + (0 \times 30.97) = 180.18$$

The values for the formula weights are corrected to two decimal places by the method *twoDecimalPlaces()*. The *paintNode()* method also assigns the OP data to each enzyme. These data include the ion name, the $\Delta G^{0'}$ value, the EC number, the gene name, the effector(s) and the full, common name of the enzyme. The ions and the composition vectors (with information on components) are coloured blue, the formula weights are coloured red, and the thermodynamic values, EC numbers and gene names are given a magenta colour. This method colours the isomerases, transferases and lyases with cyan, dark pink or magenta, respectively, and indicates at the RHS of the map, in a similar colour, the name of the class that is chosen. It also colours the inhibited enzymes red and the activated enzymes green, and indicates below the map, in a corresponding colour, the nature of the event, e.g., "Enzyme(s) inhibited by ATP: red node(s)". Finally, assigning a dark orange colour to the enzyme nodes highlights the essentially irreversible reactions, with a message below the map indicating this.

In general, the data from different lists may be viewed at the same time, with the exception of node colours, which are mutually exclusive. The information of any one type appears simultaneously for each node and/or edge in question. Data on ions, thermodynamics and irreversible reactions, and the information for the composition vectors, were found in Lehninger et al. (1993) and thermodynamic data are also available at the National Institute of Standards and Technology database "Thermodynamics of Enzyme-Catalyzed Reactions":
<http://xpdb.nist.gov/enzyme_thermodynamics/enzyme_thermodynamics_data.html>.
However, it must be remembered that the thermodynamic parameters of an individual reaction does not necessarily indicate the direction in which it may operate in an overall pathway, where the additive properties of the overall system must be applied (see Alberty, 2006). Data on effectors were taken from Lehninger et al. (1993), Hames et al. (1997), and Baynes and Dominiczak (2003). Tennent (1976) supplied the atomic weights and the gene names were obtained from Altenberg and Greulich (2004).

Clicking a check box at the bottom of the applet window activates the corresponding graph property. The non-operator nodes are white initially, but can be changed to light grey by clicking the "NOP fill" check box. The default colour for node font is black, but can be changed to red or blue by clicking the "Red font" or "Blue font" check box, respectively. The functionality for these properties is contained within the *paintNode()* method. Clicking a node, displays information in the centre of the graph about the compound it represents. The state of the "Pick" check box dictates whether this information remains permanently on screen (not clicked) or is only visible while the mouse is pressed (clicked). A Boolean variable in the *mouseUp()* method controls this.

The use of abbreviations for compounds makes the display more compact. However, clicking on a node shows the full name of the compound, the node label and the node name (see Section 1.4.2). Clicking on an enzyme node also shows a list of its activators and inhibitors, while clicking on a metabolite node shows what enzyme(s), if any, it regulates. In addition, clicking on glucose informs the user that the pathway starts at that compound, and likewise for pyruvate, that it ends there. The *paintNode()* method handles the content and display of this information.

The application above does not decide what is 'sides' (e.g., ATP) and what is not. For known pathways this is possible and is implemented in this applet. The code for the repetition of side compounds in the graph is contained within the *addNode()* method. Numbers are appended to the names of the side compounds (e.g., ATP1) in the html source. If the modified name of the compound is greater in length than the actual name, and if a substring, equal in length to the latter, has the same name, then the node label is assigned that name. This has the effect of separating these compounds in the map. However, it is not necessary to append a different number (or letter) to each side compound. Nevertheless, it is better to distinguish them, because, in the absence of the code, the compounds will still separate, but appear as, for example, ATP1, ATP2, etc. in the map. Otherwise, they will appear as pooled metabolites.

5.2.9 Output from the glycolysis pathway applet

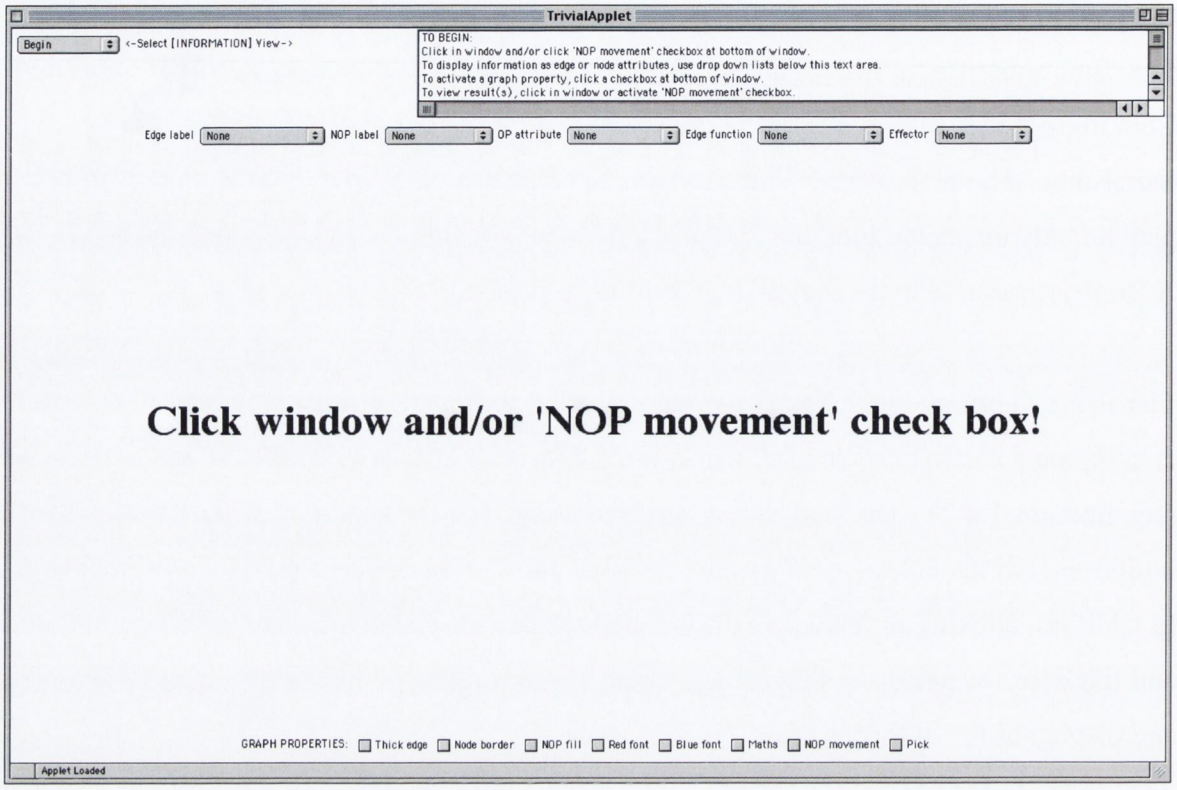


Fig.5.1. Graphical user interface for the glycolysis pathway applet. Information is accessed and displayed at the top of the window. Drop down lists below the information area allow selection of edge and node attributes. Clicking the check boxes accesses the graph properties at the bottom of the window. Following the instructions, given in the text box or the window, visualises the graph of the glycolysis pathway.

When the program is run, as described previously (4.5.5), a large display window is generated (Fig. 5.1). This GUI allows the user to interact with the applet. Clicking the 'NOP movement' check box visualises the initial layout of the glycolysis pathway graph shown in Fig. 5.2 below. The program displays the graph with predefined coordinates for the layout of the enzyme nodes (yellow). The 'side' compounds are repeated in the map.

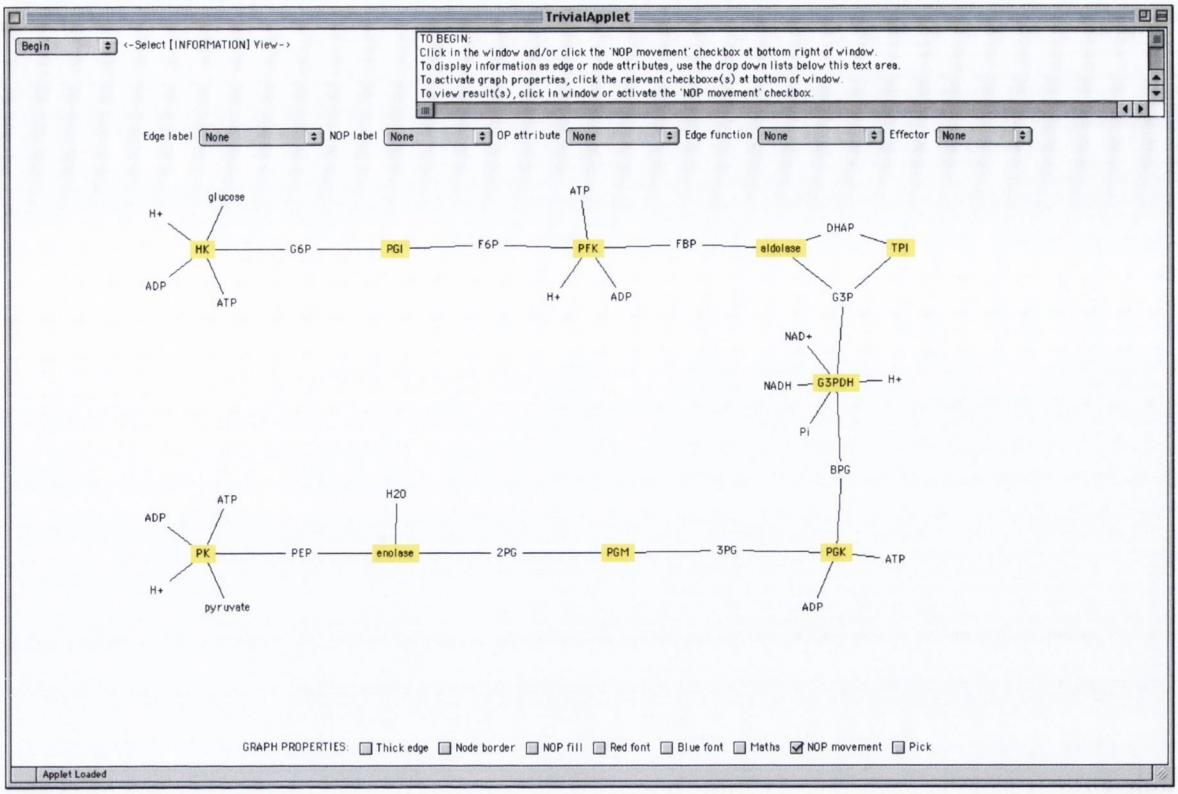


Fig.5.2. Initial layout of the glycolysis pathway graph. The enzyme nodes (yellow) are automatically laid out according to predefined coordinates assigned to them in the *init()* method of the *TrivialApplet* class. The rest are metabolite nodes. The 'side' compounds (e.g., ATP) are repeated in the graph.

Using the computer mouse, the metabolite nodes may be repositioned and the edges aligned to streamline the display as shown in Fig. 5.3 below. The graph displays the pathway stoichiometry as coloured edge labels, while information on stoichiometry is given in a text area at the top of the window. Blue edges (and blue negative integers) represent consumption of substrate, while green edges (and green positive integers) represent formation of product. Selecting a node with the mouse colours it pink and displays information about it at a fixed position (not relative to the nodes) in the centre of the graph. This information remains permanently on display while the "Pick" check box remains in the off position. For example, clicking 'ATP', as shown in Fig. 5.3, gives the following information:

Node label: ATP

Compound: adenosine triphosphate

Node name: 2

ATP inhibits PFK and PK

Node names (usually integers) must be unique in a graph, but two or more nodes can have the same label (see 1.4.2 above). However, since name of the node (2) is not important, its label (ATP) represents it.

The metabolite nodes are changed to light grey by clicking the "NOP fill" check box (Fig. 5.4). The graph shows the composition vectors as blue node labels for the main metabolites, while information on them is given in the text box. Other information is given in the graph. For example, the phosphoryl group ($-PO_3^{2-}$) is transferred from ATP to acceptor molecules, etc. From this information and the composition vectors, the charge and mass balance may be assessed. Clicking on an enzyme node shows what compounds, if any, regulate it. For example, clicking PFK shows it is inhibited by ATP and citrate, and activated by AMP and fructose 2,6-bisphosphate. Similarly, clicking on a metabolite (e.g., ATP) gives the enzyme(s) regulated by it, as shown in Fig. 5.3 below.

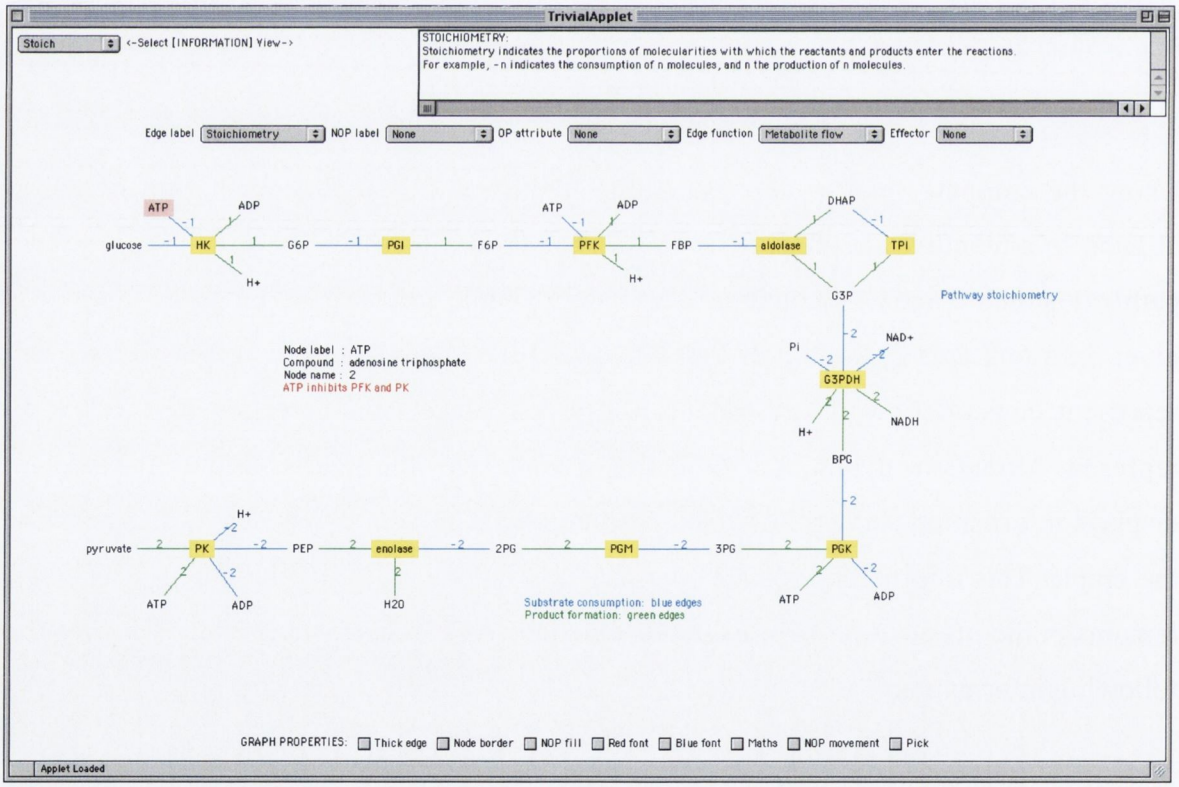


Fig.5.3. The graph of glycolysis shows the pathway stoichiometry as coloured edge labels. Information on stoichiometry is given in the text area at the top of the window. The enzymes are yellow and the rest are metabolites. An interpretation of the edge colours is given below the graph. Clicking 'ATP' (pink node) with the "Pick" check box in the off mode, permanently displays information about it in the centre of the graph.

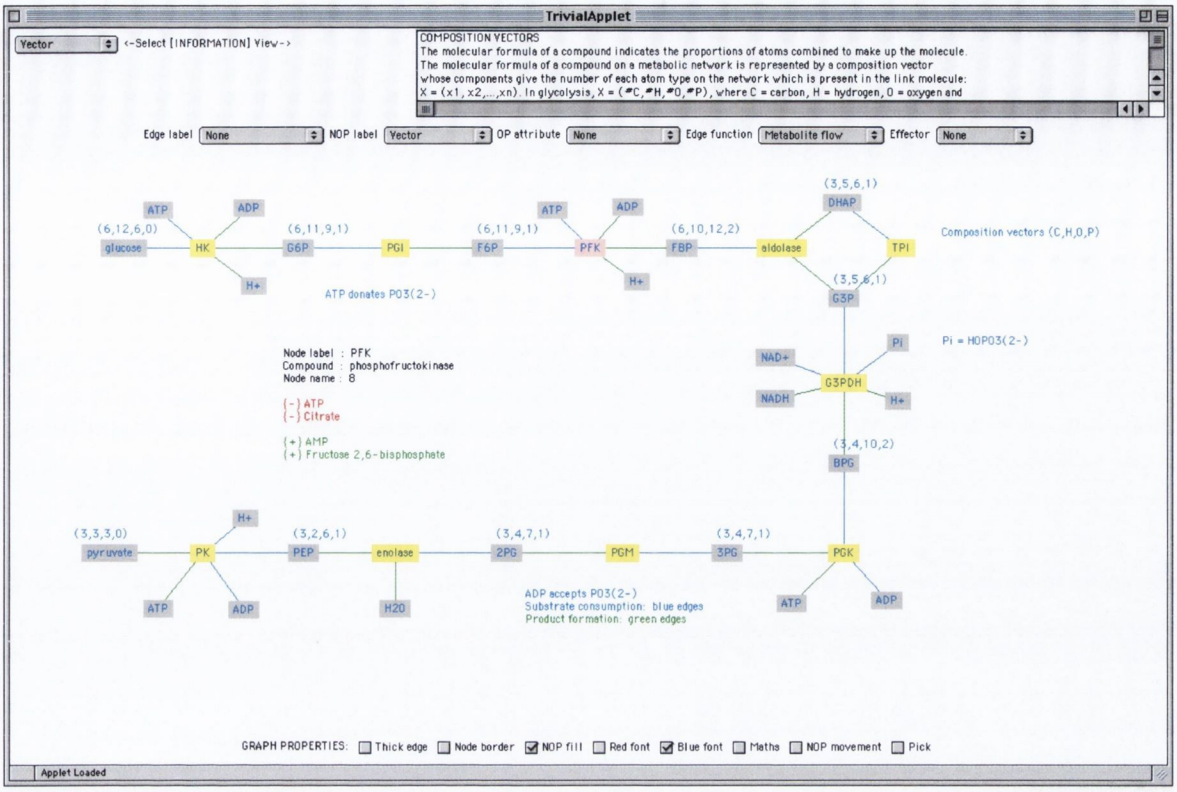


Fig.5.4. The graph of the glycolysis pathway shows the composition vectors as blue node labels for the main metabolites. Information on them is given in the text box. The enzymes are yellow and the metabolites are light grey. An interpretation of the edge colours is given below the graph. The text in the nodes is blue. Clicking 'PFK' (pink node) displays information about it in the centre of the graph. The {-} sign denotes inhibition (by the compound after it), and similarly, {+} denotes activation.

Clicking the "Blue font" (Fig. 5.4) or the "Red font" (Fig. 5.5) check box changes the font colour of the text in the nodes to blue or red, respectively. Fig. 5.5 below shows the formula weights as node labels for the main metabolites on the pathway, in addition to information on them. Clicking 'glucose' indicates that the pathway starts there.

Fig.5.6 and Fig.5.7 below show how the regulation of enzymes can be demonstrated using colour coding for the nodes. The enzymes PFK and PK are shown in red to indicate that ATP (the selected effector) inhibits them (Fig.5.6). The enzyme PFK is shown in green to indicate that F2,6BP (the selected effector) activates it (Fig.5.7). In Fig.5.6, the EC numbers (in magenta) are shown as node labels for the enzymes. Thermodynamic data and gene names can also be shown as enzyme node labels in a similar way. Red edges represent phosphorous flow, while edge labels indicate the number of these atoms as components of the relevant compounds. Carbon flow (not shown) can be visualised in a

similar fashion. These atoms allow some assessment of mass balance, i.e., the number of phosphorous (or carbon) atoms must be the same for input to, and output from, the enzyme nodes.

The "Pick" check box is shown in the off position, so the usual node information (e.g., name, etc.) is not given. Information on the number of edges (37) and nodes (37), and the edge/node ratio (1.0) is displayed in the centre by clicking the 'Maths' check box. By comparison, the glycolysis pathway with pooled ATP nodes has 35 edges, 29 nodes and an edge/node ratio of 1.21. This supports the use of the ratio as a measure of pathway interconnectivity. While pooled compounds may be more suitable for analysis, the repetition of side compounds appears more intuitive for display.

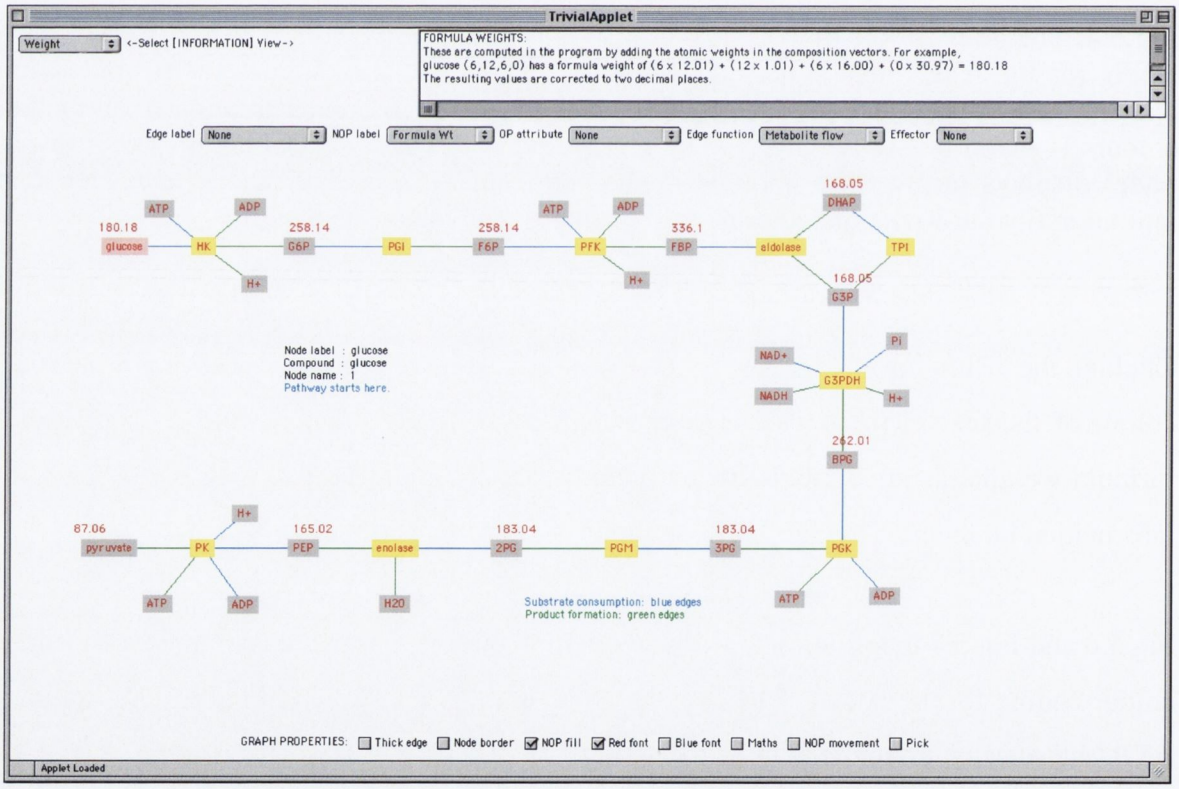


Fig.5.5. The graph of the glycolysis pathway shows the formula weights as red node labels for the main metabolites. Information on them is given in the text box. The enzymes are yellow and the metabolites are light grey. An interpretation of the edge colours is given below the graph. The text in the nodes is red. Clicking 'glucose' (pink node) displays information about it in the centre of the graph, including the fact that the pathway starts there.

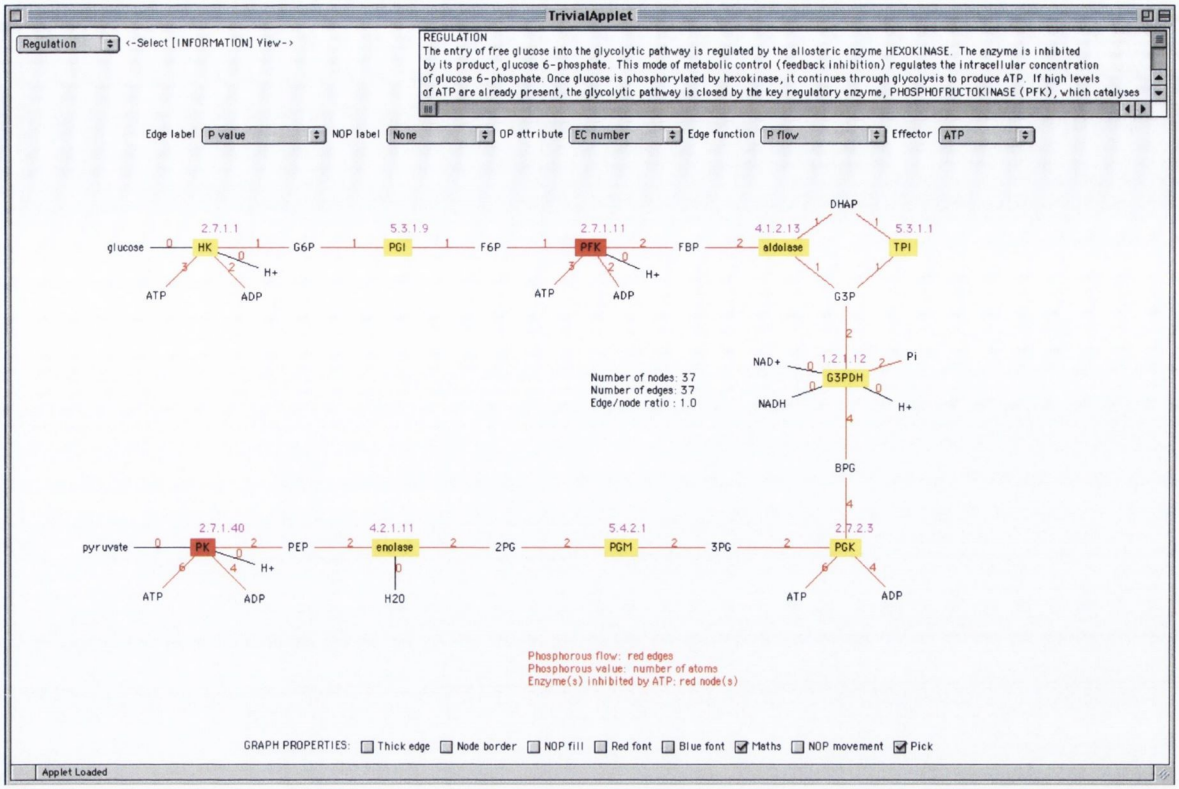


Fig.5.6. In this graph of the glycolysis pathway, the enzymes are coloured and the rest are metabolites. The enzymes PFK and PK are shown in red to indicate that ATP inhibits them (as indicated below the graph). Information on regulation is given in the text box. The EC numbers are shown as magenta node labels for the enzymes. Red edges represent phosphorous flow, while the red edge labels indicate the number of phosphorous atoms in the relevant compounds. Information on phosphorous is shown below the graph. The "Pick" check box is in the off position, so that the usual node information (e.g., name) is not given. Information on the number of nodes (37) and edges (37), and the edge/node ratio (1.0) is displayed in the centre of the graph as a result of the "Maths" check box being in the active mode.

In Fig.5.7, the magnesium cations are shown in blue as node labels for the enzymes. This is a convenient way to incorporate them in the representation. Information on magnesium is given in the text box. Edge colour is used to represent substrate-product relationships with the enzymes, as described previously.

In Fig.5.8, the dark orange nodes highlight the essentially irreversible reactions. The coloured edges adjacent to them denote flow in one direction (from blue to green) only, while the black edges denote flow in either direction. Minimal information on G6P is shown in the text box, with additional information on its node properties and regulation shown in the centre of the graph. Clicking the "Node border" check box renders the node borders visible.

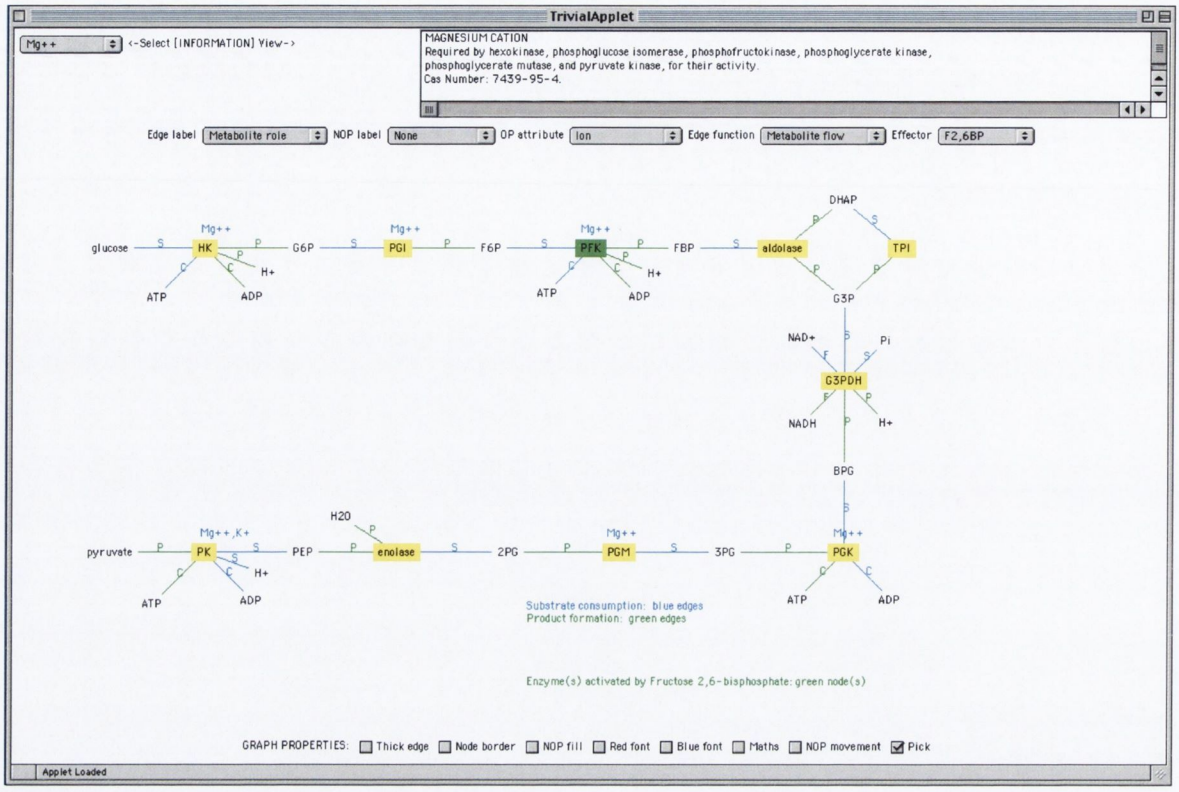


Fig.5.7. In this graph of the glycolysis pathway, the enzymes are coloured and the rest are metabolites. The enzyme PFK is shown in green to indicate that F2,6BP activates it (as indicated below the graph). The magnesium cations are shown as blue node labels for the enzymes. Information on magnesium is given in the text box. An interpretation of the edge colours is given below the graph.

In Fig.5.9, the cyan nodes highlight the isomerase reactions. The edge labels give the roles of the metabolites in the various reactions. These are S, P, C, and F, which denote substrate, product, currency metabolite and cofactor, respectively. Information on the node properties of pyruvate includes the fact that it is the end of the glycolysis pathway.

If the planar coordinates of the nodes and edges are changed, either in the relaxed state (with the "NOP movement" check box clicked) or by repositioning them with the mouse, their data move with them. The drop down lists are used to accommodate the amount of functionality available, since otherwise, too many check boxes would be required to achieve this.

It is relatively easy to extend the system to any other pathway or subnet (e.g., the citric acid cycle). The main differences for subnets would be a larger graph, a longer list of effectors and information items (mostly compounds), and an increase in the number of components in the composition vectors to include, for example, nitrogen and sulphur in order to deal with amino acid metabolism.

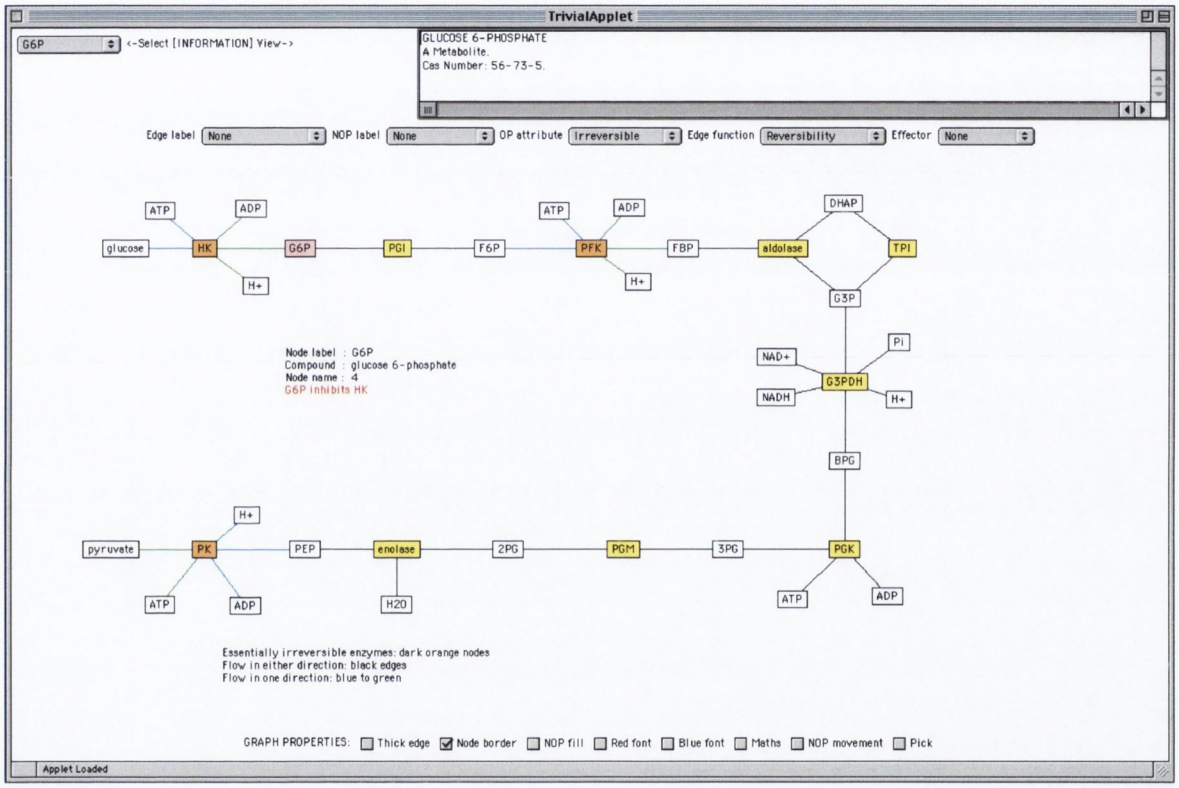


Fig.5.8. In this graph of the glycolysis pathway, the yellow and dark orange nodes are enzymes, while the rest are metabolites. The dark orange nodes represent the essentially irreversible reactions. The coloured edges adjacent to the orange nodes denote flow in one direction only (from blue to green), while the black edges denote flow in either direction. An interpretation of the node and edge colours is given below the graph. Minimal information on G6P is shown in the text box, with additional information in the centre of the graph. The "Node border" check box in the on position renders the node borders visible.

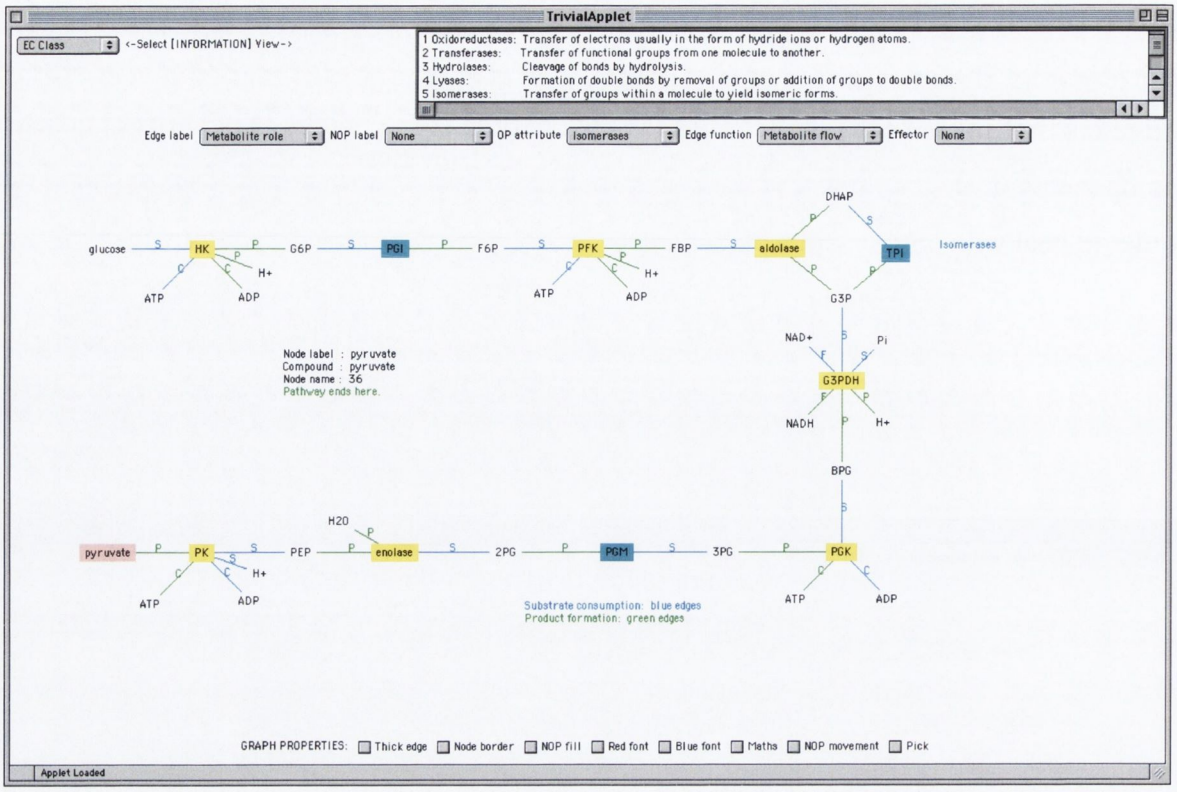


Fig.5.9. In this graph of glycolysis, the yellow and cyan nodes are enzymes, while the rest are metabolites. The cyan nodes represent the isomerases. Information on the EC classes is shown in the text box. The edge labels give the roles of the metabolites in the various reactions. These are S, P, C, and F, which denote substrate, product, currency metabolite and cofactor, respectively. Node properties for pyruvate (pink node) are displayed in the centre of the graph. An interpretation of the edge colours is given below the graph.

§5.3 Multi-pathway applet

5.3.1 General description

The multi-pathway applet allows selection of different reactions and pathways in the same applet. The program displays glycolysis, galactose metabolism, the pentose phosphate pathway, the urea cycle, the regulatory cascade for the activation of glycogen degradation in liver, and some single reactions. These are used to describe and illustrate the general principles and approaches involved in the provision of multiple items within a single applet. Furthermore, there is a facility for the user to add other reactions to construct additional pathways under a "User defined" option. In addition, the applet can partition reactions of a pathway on the basis of cell compartment. This is achieved in such a way so

as not to restrict the usefulness of the "User defined" option. The source code is in Appendix A7.

5.3.2 Data input

Data are input to the applet as in 5.2.2 but with some differences in the delimited components of the n-ary relations. This is illustrated by the following example:

glucose-6-phosphate isomerase$D-glucose 6-phosphate|C/-1;

The enzyme is separated from a metabolite by the dollar sign '$' (as before) and a metabolite is separated from the cell compartment (C for cytosol) by '|', and finally, signed stoichiometry is separated from the latter by a forward slash '/'. A generic example for the user is included in the html source for the "User defined" option. The HTML source code is in Appendix A9 and may be viewed in the applet or in a browser as described for the WebPathway applet (see Section 4.5).

5.3.3 Classes

There are four classes in the applet. Class *TrivialApplet* sets up a GUI with buttons, check boxes and a drop down list for the applet. The drawing classes, *Node*, *Edge*, and *GraphPanel*, are similar to those in the "WebPathway" applet above (4.5), but *GraphPanel* has some differences, which will be described. The packages used are the same as in the previous applets. Details of the classes follow.

5.3.4 Class *TrivialApplet*

In *TrivialApplet*, an array with the names of the pathways and reactions is initialised and an array of buttons, denoting the different pathways, is created. The *init()* method organises components in the window as in the "Glycolysis pathway applet" above, except that labels, instead of 'documentation' objects, are added to the North zone. The check boxes for the graph properties are similar to those in the "previous applet, except the "Pick" option is omitted and the "Shake" option is reintroduced. This method also sets up a drop down list for the selection of cell compartment and adds it to the Center zone (see

zones in Fig. 4.31 above). The code that processes tokens from the string tokenizer has been modified to select n-ary relations containing the compartment symbols 'C' (for cytosol) or 'M' (for mitochondrion) or both. Boolean variables linked to the compartment drop down list control the selection.

5.3.5 Class *GraphPanel*

The method *GraphPanel()* sets up three drop down lists for selection of simple node placement algorithms and edge attributes. The choices for the former are the same as in the "WebPathway" applet. The choices for *edge label* are "None", "Stoichiometry", and "Cell space". The *reaction* stoichiometry is displayed in colour as signed integers on the appropriate coloured edges. The signed stoichiometry facilitates laying out the graph, because a negative sign indicates a substrate, while a positive (or unsigned) coefficient indicates a product. Clicking "Cell space" indicates 'C' or 'M' on the edges. The choices for *edge function* are "None" and "Edge direction". Edge direction performs the same function as metabolite flow in the previous applet. This was changed to accommodate the notion of enzyme as substrate (see Fig. 5.11). The stoichiometry and cell space data are input to the applet (see Section 5.3.2).

The *update()* method assigns the colours to the edges and their labels, as in the "Glycolysis pathway applet" above. The coloured edges facilitate laying out the graph. This method also assigns the cell space letters 'C' and 'M' to the edges. The *paintNode()* method assigns the colours to the different types of operators. These are cyan for nodes involving receptors, dark orange for cAMP and yellow for enzymes. The *addNode()* method deals with a more extensive range of side compounds for repetition in the graphs, but the principle is the same.

If a user does not select a pathway or reaction, or chooses a compartment in which a selected item has no reactions, a 'NullPointerException' will occur when the window is clicked. In either case, the *mouseDown()* method, not detecting any nodes, will call the *message1()* method to give the following information to the user:

> Ensure some reactions are present in selected compartment.
> Then select a layout and/or click a button.
> If no response, reload or refresh applet, or check html source.

When the user clicks a button, the *actionPerformed()* method in class *TrivialApplet* calls the *message2()* method in the *GraphPanel* class to clear the previous text and give the following message to the user:

NOW CLICK WINDOW OR CLICK 'NOP movement' CHECK BOX!

However, the program does not indicate which action, not clicking a button or invalid compartment selection, caused the 'exception'.

5.3.6 Output from the multi-pathway applet

When the program is run, as described previously (4.5.5), the large display window shown in Fig. 5.10 is generated. The user may select a layout and a compartment, or use the default settings. Clicking a button selects the relevant pathway or reaction. Then clicking the window and/or 'NOP movement' checkbox visualises the selected item.

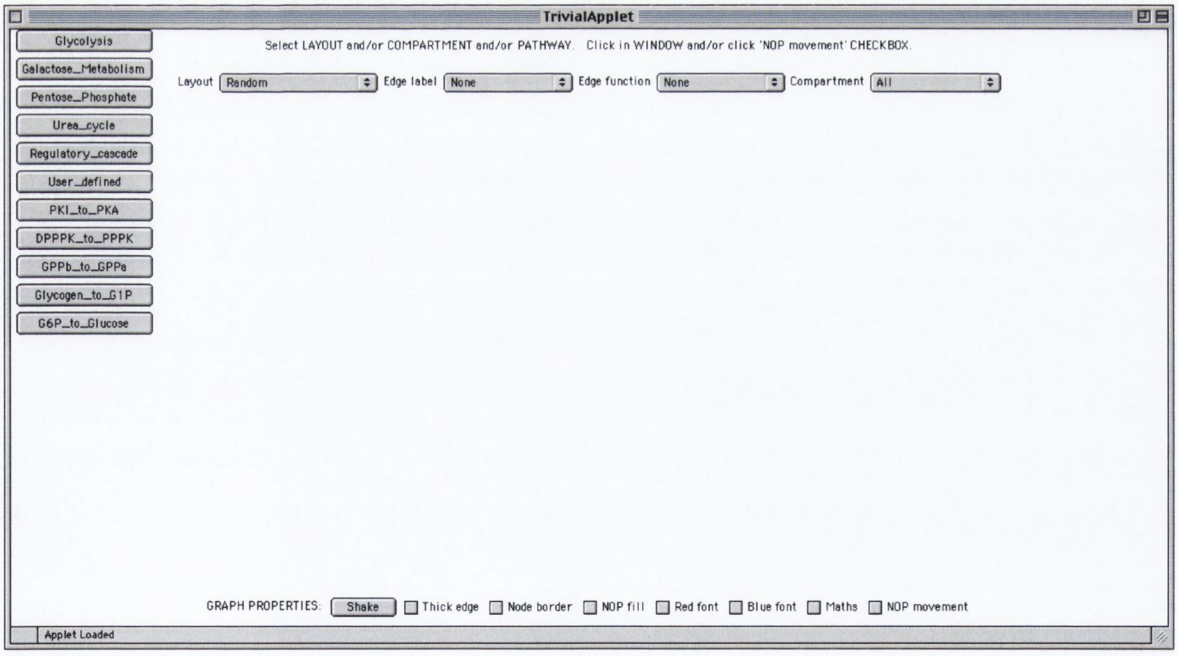


Fig.5.10. Graphical user interface for the multi-pathway applet. Brief instructions are given across the top of the window. Drop down lists below these instructions allow selection of simple layout algorithms, cell compartments and edge attributes. Clicking the check boxes accesses the graph properties at the bottom of the window. Following the instructions visualises the graph of the selected pathway or reaction.

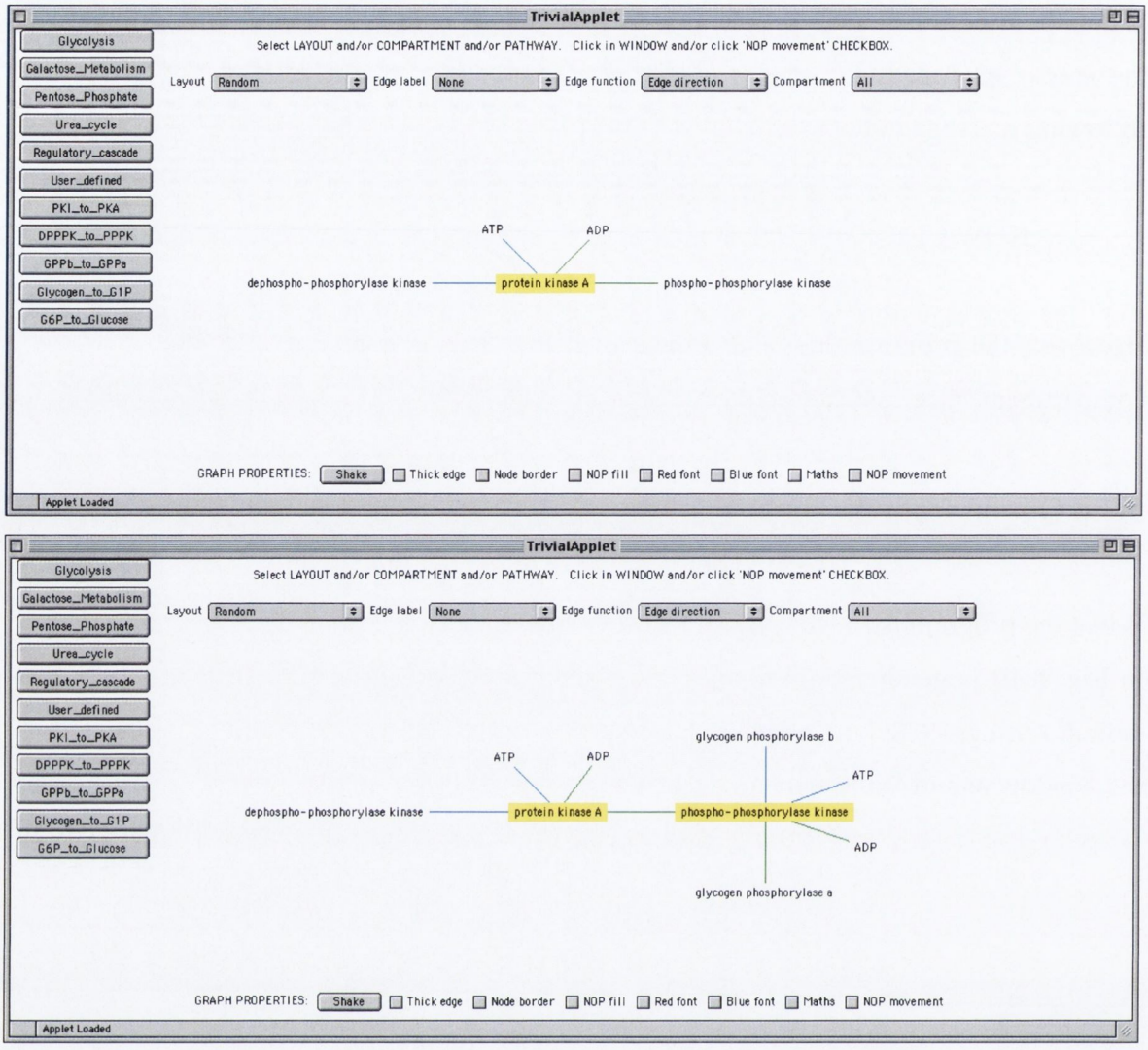


Fig. 5.11. Two reactions of the regulatory cascade for the activation of glycogen degradation in liver are shown above. The single reaction at top shows the enzyme phospho-phosphorylase kinase as substrate (LHS) and product RHS). Adding the next reaction in the cascade (bottom diagram) shows the product of the first reaction as enzyme for the second reaction.

Certain enzymes assume the roles of substrates when other enzymes are activating them in regulatory cascades. Since a biochemical object is only an operator (Def. 3.3) while it is in the act of mediating a particular biochemical event, an enzyme is a non-operator in a reaction in which it is being activated. The upper diagram of Fig. 5.11 shows the phosphorylation of the inactive dephospho-phosphorylase kinase (substrate) to active phospho-phosphophorylase kinase (product) by protein kinase A. However, the product of this reaction serves as a catalyst for the next reaction (lower diagram of Fig. 5.11). This is reflected in its change from a non-operator to an operator (yellow node).

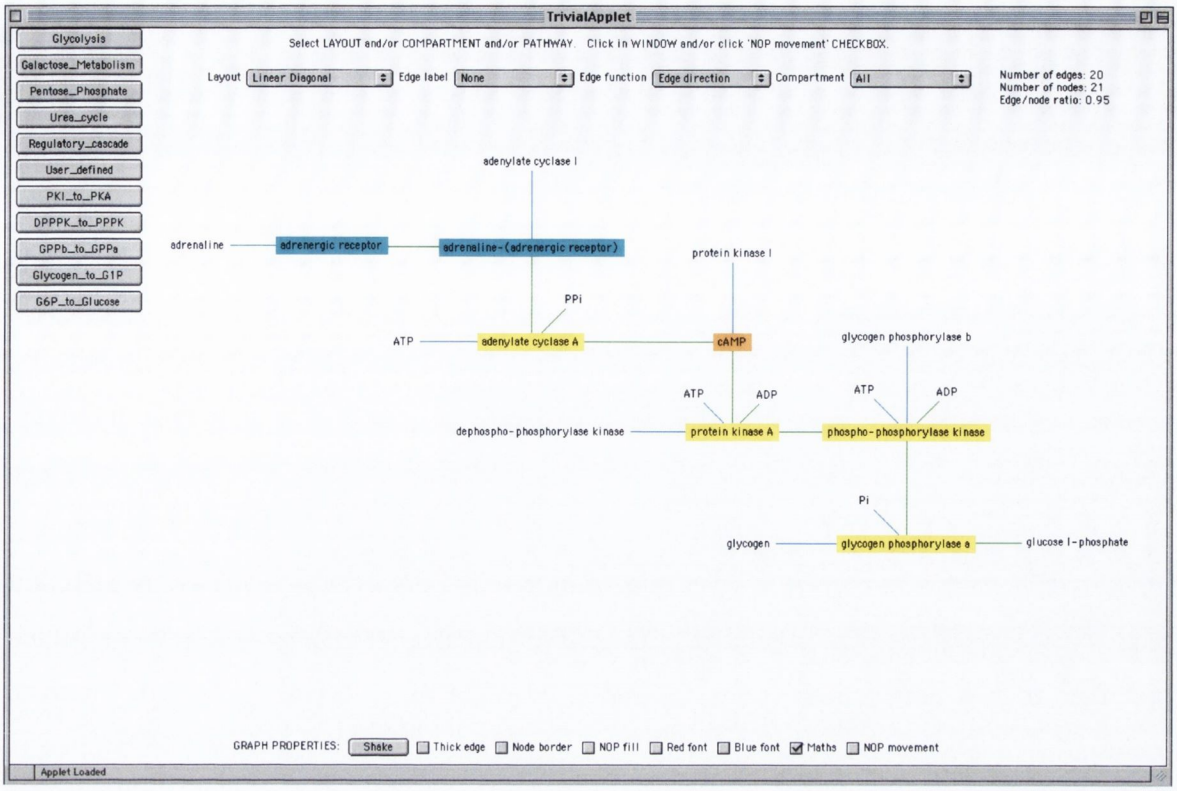


Fig. 5.12. Graph of a regulatory cascade for the activation of glycogen degradation in liver. Coloured nodes are operators while the rest are non-operators. Cyan nodes involve receptors, while the yellow nodes are activated enzymes. The dark orange node is cAMP. The 'side' compounds (e.g., ATP) are repeated in the graph. Blue edges represent binding, transformation or consumption of substrate, while green edges represent formation of product and/or activation. Information on the number of edges (20) and nodes (21), and the edge/node ratio (0.95) is displayed in the top right corner.

Fig. 5.12 above shows a graph of a regulatory cascade for the activation of glycogen degradation in liver (see Section 3.6 for description). A linear diagonal algorithm was used for the initial placement of the nodes. Nodes involving receptors and cAMP were automatically coloured cyan and dark orange, respectively, by the program. The other operators (activated enzymes) are yellow, while the rest are non-operators. The latter include metabolites and enzymes as substrates. The blue edges represent binding, transformation or consumption of substrate, while the green edges represent formation of product and/or activation. Clicking the "Maths" check box revealed the number of edges (20) and nodes (21), and the edge/node ratio (0.95).

In Section 3.6, intermodular links were formed by the product of one module acting as catalyst for the next module. Dashed edges connected the reactions by linking shared objects (Fig. 3.15 above). The output from the program shows something different (Fig. 5.12). For each *separate* reaction, a blue edge links an operator with its substrate, while a green edge connects it to its product. However, green edges link the operators to each other. In other words, one of the incoming edges is the same colour as one of the outgoing edges for each operator node in the cascade, whereas in a metabolic pathway no incoming edge is the same colour as an outgoing one. Thus, the representation allows a graph of a regulatory cascade to be clearly differentiated from the graph of a metabolic pathway. There is a direct line of action from operator to operator as evidenced by the graph. Thus, the sequence of edges of the same colour (e.g., green) could be interpreted as meaning signal transfer. Therefore, in the cascade, compounds linked to an operator by different standard coloured edges may indicate substrate-product relationships, while those linked by edges of the same colour may signify signal transfer. The graph is still bipartite because it is a tree, i.e., it has n-1 edges (as evidenced from the maths) and every tree is bipartite (see Section 1.4.3), but the nature of the partition has changed. The earlier representation of the cascade (Fig. 3.15) was helpful in deciding the input format that produced the present display (Fig. 5.12).

Shown below in Fig. 5.13 is a graph of the pathway for the conversion of galactose to glucose 6-phosphate, which then enters glycolysis. Overall, UDP-glucose is not consumed in the pathway. Also shown is the last reaction on the regulatory cascade of Fig. 5.12 joining the pathway at glucose 1-phosphate. This example illustrates how the graph of the regulatory cascade integrates seamlessly with the graph of a metabolic pathway.

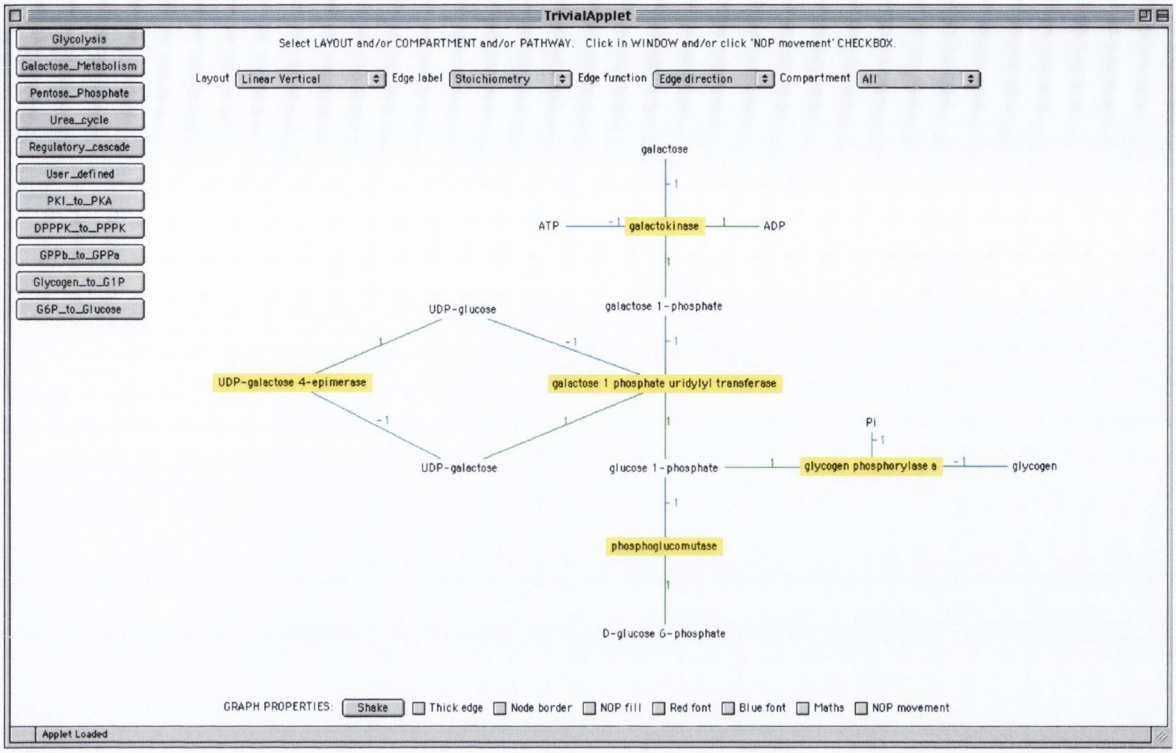


Fig. 5.13. The graph of galactose to glucose 6-phosphate joins with the last reaction on the regulatory cascade of Fig. 5.12 at glucose 1-phosphate. The yellow nodes are enzymes, while the rest are metabolites. The metabolites connected to enzymes by blue edges are substrates, while those connected by green edges are products in the reactions. Signed stoichiometry is displayed as edge labels.

Fig. 5.14 below shows the graph of the urea cycle reactions that occur within the mitochondrion. The magenta nodes represent mitochondrial membrane operators. A blue edge represents movement towards an operator, while a green edge indicates movement away from the operator. The cell space symbol 'M' (for mitochondrion) is shown on the edges. The code to select the n-ary relations for the reactions or half-reactions belonging to the mitochondrion used this information. The half-reactions are connected to the mitochondrial membrane nodes. The layout of these membrane nodes defines the boundary of the mitochondrion. This has not been done before with graph models. It allows the membranes to be displayed as integral components of the pathway. .

Fig. 5.15 below shows the graph of the urea cycle reactions that are exclusive to the cytosol. Node and edge colours are the same as in Fig. 5.14 above. In this case, the cell space symbol 'C' (for cytosol) is shown on the edges. The code to select the n-ary relations for the reactions or half-reactions belonging to the cytosol used this information. Assigning the cell space symbols to the edges is justified on the basis that it allows the half-reactions on either side of the membrane operators to be selected and thus represented in the graph. Apart from studying compartments, the facility to select pathway reactions by cell space allows the user to make good use of screen space.

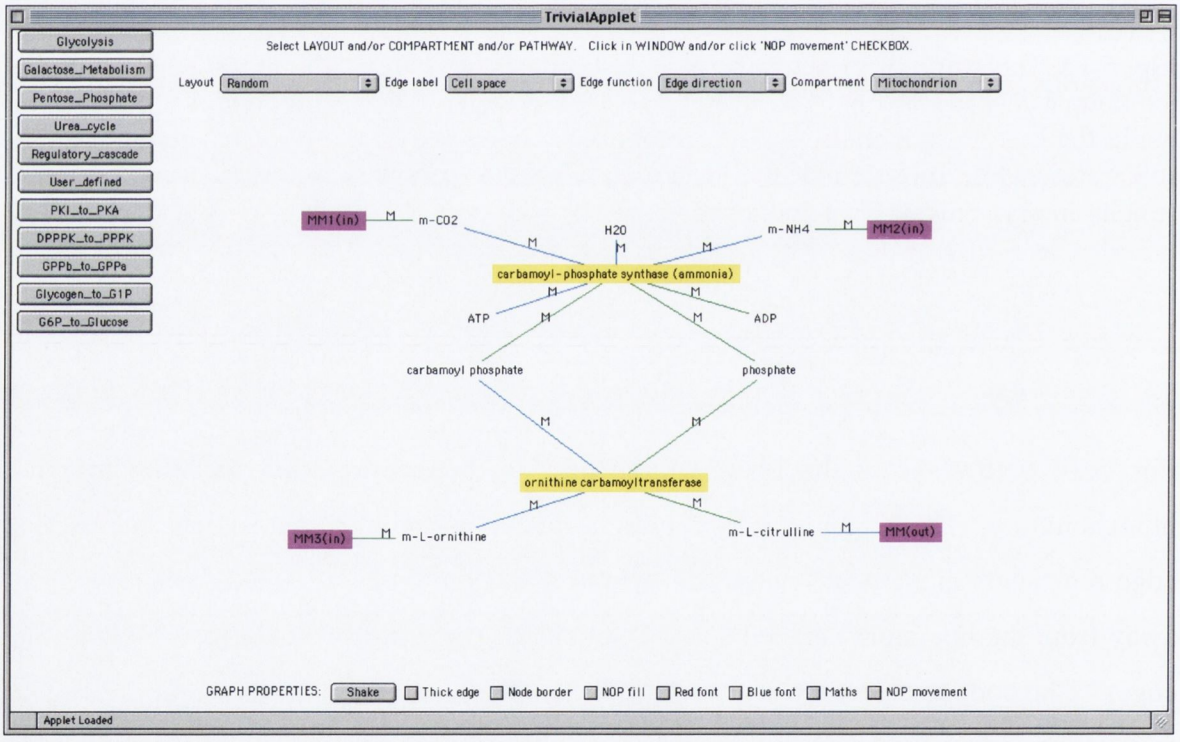


Fig. 5.14. This graph of the urea cycle shows the reactions that are exclusive to the mitochondrion. Coloured nodes are operators, while the rest are metabolites. Yellow nodes are enzymes, while the magenta ones represent mitochondrial membrane operators. A blue edge represents movement towards an operator and a green edge movement away from an operator. The cell space symbol 'M' (for mitochondrion) is shown on the edges.

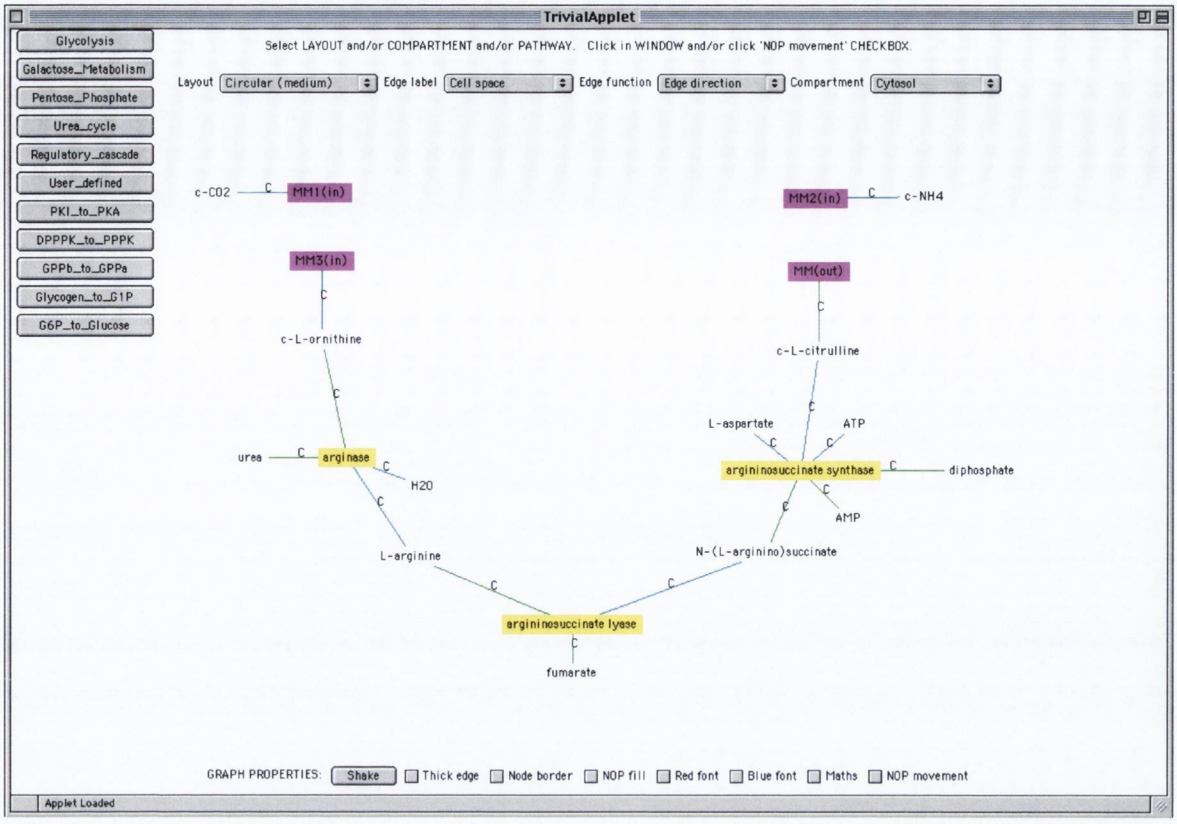


Fig. 5.15. This graph of the urea cycle shows the reactions that are exclusive to the cytosol. Coloured nodes are operators, while the rest are metabolites. Yellow nodes are enzymes, while the magenta ones represent mitochondrial membrane operators. A blue edge represents movement towards an operator and a green edge movement away from an operator. The cell space symbol 'C' (for cytosol) is shown on the edges.

§5.4 Conclusion

In the *glycolysis* applet, predefined coordinates for the operator (enzyme) nodes semi-automated the layout of the pathway. The use of abbreviations for compounds made the display more compact. Some information was contained within the program. This may be seen as a disadvantage. However, since most of this information is unlikely to change, this seemed justified, especially since the data are used here to demonstrate how the system works. The information on compounds is sparse, but sufficient to show the system in action. Repetition of side compounds enhanced the display.

A substantial amount of information was accessed and visualised without clutter. This was facilitated by the use of drop down lists and check boxes. Colour coding was used effectively to show inhibition and activation of enzymes, thus demonstrating an alternative

method of visualising control. Showing magnesium cations as node labels for enzymes was a convenient way of incorporating them in the representation. This was also true for EC numbers. Labels for metabolite nodes gave useful information on composition and formula weights. It would be useful to include charge balance as edge labels and to extend the system to other pathways and subnets.

In the *multi-pathway* applet, different selections of reactions and pathways, and combinations of them, were possible. The graph of a regulatory cascade was found to be different from the graph of a metabolic pathway. This is reasonable, since one would not expect a regulatory pathway to look exactly like a metabolic pathway. In the regulatory cascade, a sequence of edges of the same colour connecting the operators was interpreted as meaning signal transfer. The graph was still bipartite, but the nature of the partition was changed. While the number of items for display is limited, the "User defined" option makes it extendable. The provision of this facility enhances the power of the applet. Indeed, its strength lies in this. The assignment of cell compartment data to edges facilitated the partitioning of pathways on the basis of cell space and permitted the display of half reactions inside and outside the compartment. The representation of a compartment with this type of model is new. Finally, the applets clearly demonstrate the advantages of the computer over the textbook in the display and manipulation of metabolic pathways.

CHAPTER 6

DISCUSSION

§6.1 Discussion

Since metabolic pathways constitute a complex system, a mathematical description, even a structural one, must be simplified. Yet, the model must be sufficiently general so that it will encompass the system that is to be represented. An approach is presented for representing metabolic reaction systems. It uses simple object bipartite graphs, with *event operators* (the mediators of events) representing one species and *non-operators* (the non-mediators in events) the other. Edge colours describe relations between the two species and shared objects connect events together. The operators (e.g., enzymes) are integral components of the system interacting with the non-operators (e.g., substrates). This facilitates inclusion of receptors and transporters (operators), and effectors (non-operators) in the same model. A membrane operator separates two compartments and mediates the transport of compounds between them. Such pieces of information must be specified for each biochemical event, which represents a link in a complex network. Enclosing a sequence of events between two membrane operators represents compartmentalisation of that sequence. *Extended operators* can include additional information such as the species and chromosome of origin of the operator. Non-enzymatic reactions are included in the model by treating the activation energy as the operator. An operator in one event may be a non-operator in another, and *vice versa*. This is useful for addressing conceptual issues such as representing an enzyme as substrate (e.g., phosphorylation) or a metabolite (e.g., cyclic AMP) as mediator in a signal transduction pathway. Interestingly, hormones and effectors are non-operators in this model.

Certain authors also use the term 'event'. For example, Sirava et al. (2002) consider all occurring processes to be some kind of event, e.g., reaction. All entities taking part in that event are compounds and each participating compound plays a certain role. Their definition of a pathway (see Section 1.2.5) does not give an explicit criterion for connecting the reactions. In Reactome (Joshi-Tope et al., 2005), a reaction is defined as any event that converts inputs to outputs, where these are physical entities such as small or large molecules, or complexes of these. Pathways are considered as groups of reactions. In the present system, the definition of an event (see Definition 3.2) is similar to that of both authors except, it also includes the notion of control, while a pathway is well defined (see Definition 3.5). Some authors have also used the term 'operator'. For example, according to Seressiotis and Bailey (1988), the metabolic network can be viewed as a collection of operators, i.e., enzymes. In Li et al. (2004) and Hatzimanikatis et al. (2005),

pathways are generated automatically through repetitive application of a set of reaction operators to the reactants. Molecules are represented as matrices and enzyme-catalysed reactions are implemented through matrix addition, i.e., when a matrix representing an enzyme-catalysed reaction (reaction operator) is added to a matrix for the substrate, the matrix formed specifies the products of the reaction. In the present system, the concept of an operator is generalised (see Definition 3.3) and subsumes the others. Furthermore, an operator is event specific, i.e., an object is only an operator while it is mediating a particular event. Associated with every operator, and related to it, is a corresponding set of non-operators. An operator changes the states of the non-operators in an event.

A one-to-one correspondence was established between a simple object bipartite graph and the stoichiometry matrix for a system of biochemical reactions. An element of the matrix is a property of the system represented as an edge label in the graph, and *vice versa.* The concept is generalised so that any edge property of a simple object bipartite graph of a metabolic reaction system can be expressed in matrix form. This leads to a unified concept of object bipartite graphs and *associated matrices*, which can contain information on receptors, transporters, effectors, membranes and spontaneous reactions. The representation in terms of both matrices and graphs gives considerable power to the model. The advantage of the matrix approach is its underlying mathematical representation in terms of matrix equations, which are supported by a multitude of theorems and methods (see Strang, 1988). On the other hand, well-established techniques from graph theory can be applied.

Unlike classical pathway drawings, in which enzymes are displayed as labels on links between metabolites and not as integral components of a system interacting with substrates, the present model emphasises the structural aspect of the reaction system and the interactive nature of its parts. An advantage of the bipartite display is that if an enzyme is removed or missing, its links are also removed. However, in classical drawings, a link can still exist between metabolites even though the enzyme is missing. Removing a label does not affect the structure of the graph, but removing a vertex does. Furthermore, the links in the bipartite structure allow the display of properties, such as stoichiometry, metabolite roles, etc. as edge labels, which is not possible with classical drawings. The representation of enzymes by vertices can transform the representation to a gene-centric one, through the substitution of enzymes for their associate gene(s). Because enzymes may generally be regarded as being equivalent to the genes coding them in each organism,

this graph representation is most useful in superimposing genomic information on metabolic pathways, which helps further to elucidate factors that may affect an organism's metabolism. In addition, the model facilitates a metabolite- or enzyme-centric view of metabolism, which can show all reactions and/or all pathways using a particular metabolite or enzyme.

Substrate and reaction graphs are ambiguous and therefore unsuitable for the display of metabolic pathways (see Deville et al., 2003). In a protein interaction graph, an edge connects two proteins if they interact with each other (Westhead et al., 2002). However, since many metabolic reactions have more than one substrate and/or product, they violate the condition that an edge is a connection between two nodes. The use of hypergraphs (see Section1.4.5) can solve this problem (Krishnamurthy et al., 2003), but common methods for the analysis and visualisation of graphs are not immediately applicable to them. The simple object bipartite graph overcomes the above deficiencies.

The present model differs from other related models in that there are no "artefacts" in the form of reaction nodes (Gagneur et al., 2003; Kazic et al., 1999, Reddy et al., 1996; Zeigarnik and Temkin, 1994). The reaction nodes, which have no counterparts in a biological system, may make the display non-intuitive and could interfere with analysis, since algorithms may treat them as biochemical objects (3.1). In particular, the present model differs from that of Zeigarnik and Temkin (1994) in that an edge label is used to indicate the stoichiometric coefficients instead of edge multiplicity. The use of multiple edges to represent stoichiometry creates computational problems for the representation. For example, the metabolites in the reaction catalysed by the enzyme ferredoxin-nitrite reductase (EC 1.7.7.1) have stoichiometric coefficients of 1, 2, 6, 1, 6 and 7 from left to right, respectively. Therefore, the computer would have to draw different numbers of edges between the nodes. Simple graphs are less difficult computationally to draw and aesthetically more pleasing than multiple edges. Kohn and Letzkus (1983) and Kohn and Lemieux (1991) used vertices of different types to represent compounds. This allowed them to distinguish between metabolites and enzymes. Thus, the biochemical meaning of the vertices was more intuitive, but the graphs were not bipartite. Object-oriented models should permit a more detailed description, but the model presented in Deville et al. (2003) had multiple nodes associated with an enzyme. These would make a pathway display complex and difficult to follow.

Therefore, by comparison with other models, simple object bipartite graphs, in which one type of vertex is assigned to operators and a second type to non-operators, were found

most suitable for representing metabolic reaction systems because they (1) are unambiguous, (2) make it possible to display complex reactions, (3) allow the use of graph algorithms and standard drawing techniques, (4) can represent reaction stoichiometry and other properties of reactions as edge labels, (5) do not contain multiple edges, (6) do not contain artefacts, and (7) can model various biochemical processes. The simple object bipartite graph establishes a basis for a unified model of metabolic pathways and regulatory cascades, and facilitates their display and manipulation. However, it should not be considered a definite description of cellular events but rather one approach that may help in the understanding of such events. It gives a lot of power to methods of representation, display and manipulation of metabolic reaction systems. This may not necessarily completely solve the situation with respect to a *standard* model, but it is at least a way forward.

The binary relation (binary) of an operator with its non-operator, defined as a connected operator_ non-operator pair, dropped naturally out of the adjacency list representation of the simple object bipartite graph of a reaction. This permitted biochemical reactions and pathways to be represented computationally as specific sets of binaries. The representation was extended to higher order relations (n-ary relations), which included stoichiometry. However, with these relations it was not possible to distinguish different reactions catalysed by the same enzyme. Appending a different number to the name of the enzyme (e.g., transketolase) for each reaction solved this. This proved useful, since it is more intuitive to distinguish such reactions when displaying a pathway. However, the problem still persists for an enzyme-centric view of metabolism, in which an enzyme is represented once only in a map. Alternatively, different enzymes may catalyse the same reaction. This did not pose a problem for the representation. The n-ary relations are of fundamental importance to the representation and manipulation of the data because they allow an event to be described in terms of objects and binary interactions between them in a qualitative (e.g., substrate) and quantitative (e.g., stoichiometry) manner. They connect the input data (converted to n-ary relations) of the system to its graphical output (the graph constructed from the n-ary relations) via the Java code,

A Java-based system for display and manipulation of reaction systems realised the aims of the project. The system consists of an application and applet. The applet extensions will be discussed later. Using the simple object bipartite graph model, the application can produce pathway maps of varying complexity and style from stored files of reaction

information. Reactions that are displayed in diagrams are also shown as reaction equations. It can also display information in the maps and perform certain functions on them as required. The use of colour codes for nodes and edges makes it more intuitive and easy to navigate for the user. In addition, the system can generate and display useful information in the form of numeric data and text. The system is user-friendly – just click on a menu and select an item. The interface can search it in response to mouse clicks, selections from lists, and textual input. Standard pathways may be generated and classical ones, with or without modulators, may be accessed from pathway files. Any sequence of reactions, including a pathway, may be searched for on submission of a file containing them. Pathways may be created from a list of enzymes (or EC numbers). The system is not restricted to named pathways, but can generate reactions in any combination from lists of codes, operators or reactions, (Figs. 4.4 – 4.6) or based on selected criteria using substring searches (Fig. 4.15). Since the applet can operate within a browser and appear in a web page, it realises the goal of a Web-based system. However, since the applet cannot access files on the computer for reasons of security, the application generates a file in HTML format for viewing and manipulation online. The format has to be copied and entered manually into the applet, which limits its use.

The size of the database was not important in the present project. Its purpose is only to contain the relevant information used in the project and show how this information may be utilised, i.e., to demonstrate the system. As such, its comparison to other databases is not an issue. Although flat files are used to store data locally, they do have some relational properties, but not the facilities of a structured query language (SQL). For example, the EC numbers relate files for reactions, reaction-specific references and effectors. Furthermore, the parsing of the reaction strings lessens the need for a relational facility. Reaction information may be stored long-term in an Excel spreadsheet defined as a simple database. The format for input to the database was written in conventional biochemical notation that corresponds to the standard notation of the IUBMB and is thus intuitive to biochemists. Furthermore, it describes the reactions and associated information in a user-friendly way and, since these are tab delimited, can easily be reproduced from a database of reaction information. For example, the EC number, the enzyme name and the reaction can have corresponding fields in Excel. This obviates the need for a large database, since data from any other databases that can be imported into Excel can then be fed as tab- delimited files to the present system. This enables any set of reactions to be selected and incorporated as an additional pathway.

Automated layout was not a goal of this work, but a longer-term objective. Layout is relatively easy if one knows the pathway in advance. However, it must have the flexibility that users may not want to lay it out as the programmer dictates. Also, flexibility for people to add additional reactions and pathways to a graph may not be readily practicable in terms of the confines of a computer display. Layout of an unknown pathway *de novo* cannot be predicted in advance. An advantage of manual layout is that a pathway can be chosen to look as the user wants it to look and not the way the textbooks present it. In the present case users must be allowed the freedom to decide how a pathway should be laid out for their own specific needs. Automated layout is far from being the answer to everything.

The layout of large metabolic networks is difficult and generating these networks can result in clutter and entanglement. Indeed, the usefulness of such an undertaking is doubtful, because it is limited by the size of the screen. Furthermore, automated pathway synthesis can lead to a combinatorial explosion of pathways. To deal with these problems, an alternative approach has been developed as part of this project. Automatic pre-processing of the input file by the program allows all the reactions to be passed to a scrolling list from which they can be selected for visualisation in any combination or sequence, and similarly for enzymes and EC numbers. Furthermore, the presentation of these items in lists eliminates input error, facilitates searching and permits ease of selection. On the other hand, as the lists get longer, it will take more time to add the items to them. However, this should only be a problem for a file containing all known reactions, and not for one containing the reactions of a single organism. The scrolling lists make browsing of large data sets convenient by supporting comfortable navigation through them. Furthermore, they facilitate the user in controlling the number of reactions displayed. He only selects the reactions of interest and thus can build up particular sequences of reactions and increment them piecemeal. However, it was considered useful to include a simple algorithm that would arrange the nodes in an orderly fashion initially. For example, a linear vertical algorithm arranges the nodes in columns starting at the upper left of the window. When a column reaches the bottom of the window, a new column is started to the right of the first one, and so on across the window. This arrangement can accommodate a lot of nodes in an organised manner and facilitates manual layout by the user. Manual layout has the advantage that the user may arrange the pathway as he pleases, but the process can be tedious. A complete solution to layouts for metabolic pathways will probably take a lot of concerted effort and time.

An application of the substring searches allows the user to select any compound (enzyme or metabolite) and view all the reactions it is involved in (see Fig. 4.22). If a database contains all reactions catalysed by a particular enzyme, the substring search will find all of these, and display them in the form of a map. This will be a rather powerful application of the process. An advantage of this is that the user will be able to examine all pathways and/or reactions that produce and consume a given compound, and/or asses the overall metabolic effect of an enzyme inhibitor or an inherited deficiency of an enzyme. Even if numbers distinguish enzymes, a substring search can access all of them together. If there are bypasses in a metabolic network, suppressing an enzyme by a drug may have a limited effect. The substring searches can be used to find alternative paths in the network. The number of possible paths between two given compounds may be very large due to a high degree of substrate connectivity (Kuffner et al., 2000). This might be addressed by specifying the number of reaction steps allowed, but not all major pathways necessarily use the minimum number of steps. Furthermore, imposing constraints is not considered a satisfactory solution for reducing the number of computed paths when the aim is to predict new pathways or reconstruct the pathways of an organism whose metabolism has not been fully elucidated. It is useful to allow the user to make selections at each stage of the synthesis. The present system, by means of the reaction scrolling list and/or the substring searches, allows this.

Many of today's tools are for the expert user. An independent system is required that allows the user to supply information in a particular format, which could be generated by a database from stored information, from a genome or by the user himself. There is a strong need for personal systems allowing a user to handle and combine large data sets from different sources and possibly generate new insights from the data. The present system meets the needs of the individual biologist who wishes to use it for his personal research. The user can create personal data sets by selecting from public databases the interactions he finds *reliable* and wishes to use, and combine them with his own observations. There is a need to compile one's own data sets for the sake of completeness. For example, the second transketolase reaction (transketolose2) was not found in BRENDA at the time of the search. So, if one inherits a dataset from someone else, it may not contain certain reactions. But more importantly, if one is studying an organism, the question arises as to whether the reaction is missing from the organism or has not been found in a database that has been searched. Furthermore, a portion of a pathway not clearly understood yet may be omitted. User-specific references for each enzyme can be added to the files. This is a

useful feature. Flat file versions of all the databases in EcoCyc, MetaCyc and BioCyc (Karp et al., 1998; Karp et al., 2002; Karp et al., 2005) may be downloaded. A user has permission to do whatever he likes with them. This implies the need for separate, independent computer programmes and systems to display and manipulate these data. The present system could be used for this purpose, although this might require a parser for each new data resource and reformatting of the data to the precise format expected here. A user can add new information. For example, the bench biologist can input his data and display them step by step as he performs his research, and then can look for links to his reaction(s) in the database.

The concept of a standard pathway (and subnet) as defined in section 4.1.4 above may be somewhat naïve because in some cases there are significant differences in the biosynthetic pathways for certain compounds in different organisms (see Section 1.2.1). However, it might be more appropriate in the context of reference networks for key model organisms (Srinivasan et al., 2007), though simpler in principle. The basic idea is to automatically check an organism's reaction network for 'standard' pathways appropriate to that organism. Thus, genome-wide explorations of reactions could be conducted with this system, and departures from the 'standard' pathways ascertained. It would be easy to check an organism for missing enzymes on a pathway, but checking for alternative reactions would be more difficult, and would probably involve searching for alternative routes. As already mentioned, substring searches would facilitate this. However, if the definition of a pathway changes, it would be necessary to make some changes to the program. An alternative option allows the user to search a file for defined reaction sequences as entered into a search-file. This gives a user more control over a search. Another option in this category allows the user to create a 'new' pathway from a data file by submitting the operator codes and/or names only. This is a useful feature, which makes the creation of pathways more convenient by automating their retrieval from a file.

Classical pathways are accessed directly, with or without effectors, from specific pathway files. This has the disadvantage that they are not part of the main database. On the other hand, it allows the inclusion of non-standard components. For example, the urea cycle includes mitochondrial membrane operators (see Fig. 4.25). A sequence of reactions between two mitochondrial membrane operators indicates they are in the mitochondrion. This representation of a compartment as an emergent property of the model is new.

Furthermore, as part of the model, the membrane operators will be included in the class frequency histogram as follows:

Class	Frequency	Histogram
0	0	
1	0	
2	1	*
3	1	*
4	1	*
5	0	
6	2	**
7	0	
8	0	
9	4	****

The membrane operators belong to class 9 of the operator code, and as can be seen, the class is represented four times on the pathway. These data may be easily obtained by examining the map, but in large data sets they would be difficult to ascertain by inspection. Showing the activator {+}N-acetylglutamate at the regulation site makes good use of screen space. Furthermore, it is easy to distinguish from a metabolite.

In the *glycolysis* applet, used here as an example, clutter is avoided by accessing and visualising a lot of information as required, while colour coding is effective in showing the influence of effectors on enzymes, and highlighting enzyme classes and essentially irreversible reactions. Although magnesium cations are important in this pathway, they are frequently not included. Showing them as node labels for enzymes is a convenient way of incorporating them in the display. It is clear that repetition of side compounds makes the display more readable. In the *multi-pathway* applet, the reactions and pathways available for display are minimal examples to illustrate the approach, but the "User defined" option makes it extendable. The graph of the regulatory cascade, in which a sequence of edges of the same colour connecting operators indicated signal transfer, has not been seen before. Restricting edges to only those whose attributes satisfied certain criteria limited the number of nodes on screen. For example, the assignment of cell compartment data to edges facilitated the partitioning of pathways on the basis of cell space and permitted the display of half reactions inside and outside the compartment (e.g., urea cycle). The application and

applets clearly demonstrate the advantages of the computer over textbook and static drawings in the display and manipulation of metabolic pathways.

In the applets, information on the number of edges and nodes, and the edge/node ratio is displayed on screen by clicking the 'Maths' check box. By definition (see Section 1.4.2) a graph has at least one vertex and possibly no edges. Thus, the applets use the edge/node ratio rather than a node/edge ratio, because division by zero is not defined. This information has proved useful in trouble-shooting the system. For example, an older version of the Enzyme List represented the adenylate kinase reaction (EC 2.7.4.3) as:

$$ATP + AMP = ADP + ADP$$

The corresponding graph generated by the system appeared correct, with three metabolite nodes connected to one enzyme node by one edge each, resulting in three edges. However, the maths indicated 4 edges. Furthermore, the edge connecting ADP to the enzyme showed a stoichiometry of one instead of two. In a more recent version of the Enzyme List, this reaction is now represented as:

$$ATP + AMP = 2\ ADP$$

This generates the same graph, with the maths indicating 3 edges, one of which shows a stoichiometry of two for ADP. This solves the problem and gives the correct result. However, it highlights the importance of using the correct data format and monitoring the system.

It is difficult to automatically draw metabolic pathways from entries in the Enzyme Nomenclature Database (http://www.chem.qmul.ac.uk/iubmb/enzyme/), since some metabolites have not been named consistently. This became evident when pathway building in the present system revealed inconsistencies in the user data file (see Section 4.4.2 above), resulting in a lack of connectivity between certain compounds. Since these data were taken from the Enzyme Nomenclature Database, this has implications for network analysis based directly on this database, or indirectly, since other metabolic databases are based on it. Furthermore, it contains several generic or representative reactions, which are unsuitable for drawing metabolic pathways (see Section 4.1.3 above). Thus major efforts are needed to produce datasets of reaction information that are suitable

for computational purposes, but this is not the aim of the present project. However, the 'unsuitable reactions' were removed from or manually replaced in the user data file and any inconsistencies that were discovered in the names of compounds were rectified (see Appendix A1). Static diagrams of pathways, in which certain enzymes participate, are available in the Enzyme Nomenclature Database, but it does not have facilities for the automated generation of pathway graphs, as in the present system, even though these have to be manually laid out.

Other systems that display pathways include, among the larger ones, the ENZYME database (http://www.expasy.ch/enzyme/), which has access to Boehringer Mannheim's Biochemical Pathways on the ExPASy server. This is an electronic version of the Biochemical Pathways Wallchart, provided as a series of linked images. An electronic index allows localization of any metabolite or enzyme on the chart. Each enzyme name in the chart is linked to its corresponding entry in ENZYME, and vice versa. The database does not have facilities for the automated generation and manipulation of pathway graphs, as in the present system. On the other hand, the present system does not have facilities for finding compounds on the maps or linking them to entries in the files.

In KEGG (http://www.genome.ad.jp/kegg/kegg.html), pathway maps and overviews are manually drawn pictures, and reactions in the pathway maps do not include side compounds (e.g., ATP). In the present system, pathway graphs are dynamically produced and manually laid out, and all pathway maps display side compounds. A set of static and separate diagrams of metabolic maps, as in KEGG, does not facilitate the exploration of a metabolic network because it is difficult to check all the separate diagrams. It is tedious to draw diagrams by hand, especially large ones. Furthermore, one may not connect up all the main metabolites, which makes the derivation of balance equations more difficult and prone to error. An advantage of the present approach is automatic graph updating when new data are obtained. However, in 2007, Klukas and Schreiber presented a new approach for dynamic visualisation and interactive navigation especially tailored to KEGG pathway diagrams. These diagrams are given as KEGG Markup Language (KGML) files. Their methods use a combination of semi-static and dynamic visualisation.

KEGG retrieves pathways using an EC number, a compound ID, or a gene name. The pathway maps contain names for the compounds and EC numbers for the enzymes. Clicking on a metabolite name or EC number in a pathway map finds information on it. In

the present system, reactions are retrieved using an EC number, an enzyme name, or a reaction. So the system is not restricted to named pathways. The pathway maps contain names for the compounds and enzymes. Clicking on a metabolite or enzyme name finds information on it only in the glycolysis pathway applet. However, the application, web applet and multi-applet contain information as edge labels and edge functions in addition to graph enhancements. The glycolysis applet contains extensive information in the form of node functions and labels, and edge functions and labels in addition to graph enhancements. The KEGG PATHWAY Database Webpage currently (2008) contains an extensive list of pathways. Clicking a pathway name accesses the corresponding pathway.

Searching LIGAND (which is part of KEGG) for a compound by name returns a list of names containing that compound name as a substring with links to the corresponding pathway maps. In the present system, searching for a compound by name is done via a substring search, which retrieves the reactions it is involved in. However, substring searches can also retrieve reactions based on selected criteria, such as enzyme type (e.g., kinases). Clicking on Metabolic pathways in the Pathway Map Ortholog Table in KEGG, returns links to pathway maps by name, and clicking on a specific name returns the reference pathway. Matching enzyme genes in a gene table with enzymes on the reference pathway automatically generates organism-specific pathways, and following the connections of enzymes that are coloured green allows the organism-specific pathway to be visualised. In the present system, pathways can be found by name from specific pathway files. Pathways can also be found by submitting a file containing a list of the corresponding EC numbers or enzyme names to the user database, but no organism-specific ones are available.

The EcoCyc database (Karp et al., 1998) and *MetaCyc* (Karp et al., 2002) are accessible through BioCyc (http://BioCyc.org/). BioCyc pathways correspond to single biological processes that take place in a single organism. The pathways are drawn automatically using complicated graph-drawing algorithms (see Section 1.6.2), which produce good quality results. In the present system, the graphs are also drawn automatically, but the layout is manual, leaving most of the work to the user. Their system fulfils the special requirements and graphical conventions of classical drawings of metabolic pathways, whereas the present system favours the simple object bipartite graph approach for reasons outlined in Sections 3.2 and 3.9. In their system, pathways can be displayed at different levels of detail. An intermediate level includes enzymes, EC numbers, genes and the

chemical structures of the main compounds. A lower level gives the same information without chemical structures. With greater detail, the edges become cluttered with information. However, the user can customise the diagrams with respect to enzyme names, gene names, EC numbers, layouts, regulation, etc., but this has to be done in a different page beforehand. In the present system, the user has a great deal of control over details of the display while viewing the pathway. This is achieved with the use of drop down lists and check boxes on the same page as the map.

Within the graphical representation of pathways in their system, inhibitors and activators of reactions are represented, which in some cases involve very long arrows from source to destination. In the present system, effectors are placed close to the regulation site, or colouring the affected enzyme represents their action. In BioCyc, enzyme information consists of names, enzyme classification, catalysed reactions and pathways in which the enzyme occurs. Compound properties are represented with names, chemical formulae, molecular weights, CAS (Chemical Abstracts Service) number and structural formulae. In the present system, structural formulae are not available. Their system cannot find alternative routes in a metabolic network, whereas such routes can be found, piecemeal (not automatically), in the present system.

The *University of Minnesota Biocatalysis/Biodegradation Database* (UM-BBD) provides information on reactions related to biodegradation pathways of xenobiotic compounds in microorganisms. (Ellis et al., 2003). The UM-BBD contains two different types of pathway representations: a text format representation and a static drawing. The enzymes pages contain a list of reactions that are catalysed by the enzyme and links to each of the static pathway maps in which one or more reactions catalysed by this enzyme appear. The compound page also contains a list of all reactions in the database which use or produce this compound and links to the pathway maps in which the compound appears. The present system does not contain any links to compounds in the maps. In their system, a dynamically generated pathway map starting from a defined reaction can be generated and is shown only in text format representation. As stated before, the present system generates graphs dynamically, which are laid out manually. These have node and edge attributes that can be displayed as required.

In contrast to static databases, which provide predefined reactions and fixed metabolic maps, other tools allow dynamic retrieval and visualisation of interaction networks (Suderman and Hallett, 2007). Although it is claimed that Cytoscape (Shannon et al, 2003)

is applicable to any system of molecular components, it is considered to be most powerful when used in conjunction with large databases of protein-protein, protein-DNA and genetic interactions. It supports standard and customisable network display styles, ability to import a large variety of interaction files, and zoomable network views. Many plugins are available for importing networks from various data formats, analysing networks, and generating networks from literature searches. Barsky et al. (2007) created a Cytoscape plugin (Cerebral) that uses subcellular localisation annotation to automatically position nodes in layers. In the present system a compartment boundary is automatically incorporated into the operators of the object bipartite graph model. This obviates the need to have compartment lines in the map, although it is noted that their method is useful for laying out nodes. No convincing demonstration of the suitability of Cytoscape (or plugin) to display metabolic pathways intuitively is observed. For example, a plugin for metabolic pathways, which was demonstrated at a Cytoscape workshop during the 16th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB) in Toronto, 2008, produced a display that was hard to follow and difficult to interpret.

VisANT 3.0 (Hu et al., 2007) includes multi-scale visualisation of multiple pathways and editing and annotating pathways using a KEGG compatible visual notation. Pathways are represented as contracted linked metanodes, which can be expanded. This is not available in the present system. VisANT uses layout information contained in KGML (KEGG Markup Language) rather than an automatic layout algorithm. Nodes and edges can be added or deleted. In the present system, nodes and edges can be added but not deleted. However, the effect of deleting these in a pathway can be achieved by omitting to select them via the scrolling selection lists (see Section 4.1.4). Their system does not show the side compounds or the names of the metabolites (just KEGG ID numbers), whereas in the present system these are displayed.

It is hoped that the present work will be of value in the field of systems biology. While this attempt is a small contribution to the field, it may point towards more sophisticated tools that incorporate both qualitative and quantitative aspects into an integrated system. Perhaps some will be inspired by the direction taken here. The present work, together with that of others, is a continuing effort in a relatively new endeavour. This system might be of use in the study of metabolic networks, metabolic pathway modelling and computer-aided instruction in Biochemistry.

§6.2 Future developments

"The modern physicist may rightfully be proud of his spectacular achievements in science and technology. However, he should always be aware that the foundations of his imposing edifice, the basic notions of his discipline, such as the concept of mass, are entangled with serious uncertainties and perplexing difficulties that have as yet not been resolved" (Jammer, 1997).

Future developments will involve: (1) improvements to the existing systems and (2) extensions to the systems to include quantitative data and new functionality. With regard to (1), the file system in the application needs to be changed. While selection of the general and user files is satisfactory, having to type the other file names is a limitation. This will become more evident as the number of files increases. The solution will probably be a scrolling list of files that is automatically updated with new ones.

Improvements to visualisation should include colouring the area occupied by a cell space based on the coordinates of the membrane operators. Anything that binds to an enzyme or other protein is a ligand, regardless of whether or not it is a substrate and undergoes a subsequent reaction (Palmer, 1995). For this reason a blue edge for the binding of a ligand may be used, but arrows for the green product edges could be used to give a better indication of direction. It is also important to make the representation more extensible by considering polymerisation and autocatalysis, etc. Furthermore, it would be useful to employ other measures of graph topology, e.g., degree, degree distribution, shortest path length, etc. (see Barabási and Oltvai, 2004; Klamt and Stelling, 2003).

The present system uses digits to distinguish between the two different reactions catalysed by the same enzyme, e.g., transketolase1 and transketolase2 in the pentose phosphate pathway, for the purpose of display (Fig. 4.23). However, it is not standard practice to represent enzymes in this way in a database. On the other hand, the enzyme-centric view of metabolism requires the enzyme transketolase to be represented once only in the map, in which case it would be difficult to distinguish the different reactions. A temporary solution to the latter could involve using numbers, which correspond to the numbers appended to the enzymes, on the edges of the graph. However, preserving temporal relationships for reactions of the same enzyme or merging vertices with identical labels for an enzyme-centric view are problems for the representation that require general solutions in the longer term.

The system should be made available for personal use. For this purpose it would be necessary to enhance it with database connectivity. This could be done using Java Database Connectivity (JDBC), which allows access to SQL-compatible databases. An additional useful feature would be the generation of different formats for input to other systems to avail of their functionality. For example, it would be useful to generate a stoichiometry matrix using Mathematica (Mulquiney and Kuchel, 2003). The functions *StoichiometryMatrix* and *NMatrix* automate this process, thus generating the required matrix from a list of reactions. To do this, it is first necessary to input a list that contains the reactions in a format the *MetabolicControlAnalysis* package commands can recognise. This format closely resembles the input format to the java application. Therefore, it would not be difficult to modify the program to do this. Alternatively, it might be better to alter the program to generate an annotated form of the stoichiometry matrix.

On the other hand, application-specific formats force end-users to build parsers for each new data resource and reformat the data to the format expected by their system. However, certain data format standards are available. Two well-known examples, SBML (Systems Biology Markup Language) and BioPAX (Biological Pathways Exchange Language), are open, XML-based (http://www.w3.org/XML/) standards that use a levelled approach, i.e., each standard is described at various levels of complexity and specificity. Users may choose the simplest level sufficient to represent the information in their dataset. Level one of BioPAX (http://www.biopax.org/) is used for representing metabolic pathways, while level 2 is for representing molecular interactions. According to StrömbÄck and Lambrix (2005), SBML (http://www.sbml.org/) is best suited for simulation models of molecular pathways. Another popular example is CellML (Lloyd et al., 2004). This is a free, open-source, XML-based standard for defining mathematical models of cellular function. Its applications include biological pathway models. It is desirable that the present system should support these standard file formats.

An important task for the future is the development of automatic layouts for the pathway graphs. Manual layout has been a stepping-stone to the larger goal of automatic layout that may be tweaked by the user. However, it is generally considered that the known graph drawing algorithms (see Di Battista et al., 1999) are inadequate for visualising biochemical reaction networks according to established conventions of biochemistry. Although good results have been obtained by using labelling techniques, where the enzymes and side compounds are treated as edge labels and placed separately (Karp and Paley, 1994b), this

would not be appropriate for the simple object bipartite graph model, where enzymes are nodes. Some solutions concentrate on the placement of the main substrates and products only (Becker and Rojas, 2001), but this is not a valid representation of metabolism. The layout should be automated in ways biochemists will find aesthetically and scientifically pleasing. The glycolysis applet uses predefined coordinates for the placement of the enzyme nodes in a snake layout. It may be possible to build on this. Since there are at least two non-operators to every operator, this would reduce the problem by two-thirds. Of course, the coordinates would have to be computed on the fly. In this case, they could be input to the applet via the applet argument in the html source. The snake layout could be used as the nucleus for optimising the layout of the ancillary information related to each node and edge. It might also be useful to have adjustable layout parameters. It is of interest to note that Gauges et al. (2006) developed an extension to the SBML file format that makes it possible to store layout information. Another related problem requires a general solution for the layout with repetition of side nodes with the same label. Automated layout of large pathway graphs remains a recognised problem to be solved.

The display of large networks continues to present challenges. It is useful to view the whole network, while at the same time being able to limit the view to specific subgraphs, hiding or collapsing the rest of the network when not required. A pathway may be considered as an event in which source metabolites are converted into target metabolites via a specific sequence of reactions. The extended operator for this event would be the set of individual operators specific to each reaction (see Section 3.7.2). This would allow a single node to represent a pathway. The intersection between pathways would be achieved by the net inputs and outputs. This would allow users to represent any set of related nodes as a single node and to display the relations between these nodes while preserving network topology. Subnet nodes would contain related pathways and so on up the hierarchy. This hierarchy of events would allow the user to toggle the display of nodes and edges, and expansions and contractions of the graph.

It may be useful to consider an alternative IDE that might be more robust and helpful, such as Eclipse, which is free and widely used for Java. For example, Sorokin et al. (2006) developed the Edinburgh Pathway Editor (EPE) as an Eclipse standalone application. Eclipse (http://www.eclipse.org/) is an open-source community whose projects provide a vendor-neutral open development platform and application frameworks for building software. The EPE uses the Eclipse Graphical Editing Framework (GEF) as the basis for

its drawing functionality. The GEF allows developers to create a rich graphical editor from an existing application model. Eclipse allows developers to maintain a convenient balance between software extensibility and maintainability through the use of extension points that allow researchers to develop specialised plugin software for scientific computing.

It is important to identify and eliminate inefficiencies. For example, every query requires re-parsing the flat files. This repeated parsing is inefficient. It might be better to parse them once and store the results. A commonly used method is to pre-compute and store intermediate data for faster query execution. Benchmarking CPU time and memory used as a function of number of reactions and some network topological parameters would give some idea of performance. It should uncover bottlenecks and reveal the limits of the system. Furthermore, the manual transfer of data to the WebPathway applet is a deficiency that needs to be addressed. It is important to be able to save pathway maps that are produced by the system. This is not possible at present, but should be implemented as soon as possible. The code hard-wires some pathway-specific data. This needs to be investigated.

A number of miscellaneous items need attending to. These include listing all metabolites and assigning compound numbers to them, finding all pathways from one compound to another, displaying EC numbers or gene names as operator labels, adding PubMed links to references, displaying compound structures on maps, showing charge balance as edge labels, and developing the application further to contain some of the functionality of the glycolysis applet. The use of synonyms is important for finding compounds because different names and abbreviations are used for the same compound. They are not included in the present system, but should be added to it. With respect to this task, ChemFinder (http://chemfinder.cambridgesoft.com/), ChEBI (http://www.ebi.ac.uk/chebi/), InCHI (http://www.iupac.org/inchi/) and BND (http://www.biocheminfo.org/) are reliable sources of information. Furthermore, it might be useful to change the input format to indicate the pathway(s) to which the reaction belongs, so that parsing the reaction strings could access the pathway(s). This would involve extending the reaction string and modifying the corresponding software to accommodate the changes.

Future developments should deal with quantitative aspects of the model. To start with, techniques from graph theory and matrix algebra may be applied. For example, Sen (1991) used a graph-theoretic approach for calculating the Control Coefficients of

metabolic pathways. Furthermore, according to the same author, a technique using matrix inversion is the most convenient to use. It would be interesting to check the performance of the simple object bipartite graph in calculating the same Control Coefficients and to compare it with the matrix approach, considering the fact that an edge property of the graph is an element of its corresponding matrix.

It would be useful to automatically generate the dynamical material balance equations of a process under the form of a mixed algebraic-differential system, for example, by taking advantage of the knowledge involved in the diagram. This would save the user the trouble of writing down the equations describing the process. He would not need prior knowledge in computer science or mathematics to construct mathematical models of biochemical processes. Thus, given a set of reactions, and ignoring the external metabolites, the rate at which any metabolite concentration changes can be expressed as the sum of the rates of the reactions that produce it minus the sum of the rates of those that consume it, multiplied by the relevant stoichiometry. For metabolite X with concentration x (produced in reaction 1 and consumed in reaction 2), for example, the rate of change of its concentration is $dx/dt = v_1 - v_2$, where v_j denotes the rate of reaction j. There may be many reactions. Some of the equations may be more complicated than others because some metabolites are involved in more processes than others. The idea is to automatically generate these equations, or to see if they can be represented on the simple object bipartite graph and/or in its corresponding matrix. Of course, the equations can be assembled into a single stoichiometric matrix for analysis.

Future developments would also involve extending the system to include mass balance, kinetics and other quantitative data, by making use of the properties of the simple object bipartite graph. From a brief consideration, it would appear that mass balance (using composition vectors) should be possible, but kinetics, especially for complex (non linear) reactions, will probably be more challenging. The ultimate goal is to model the whole cell and to perform functional computations and simulations over this model. Although this project is far from fulfilling such an expectation, its concepts and implementation may be helpful for future efforts in this direction. In the long term, it would be useful to address the issue of the complex inter-relationship between different tissues and between different organs. Finally, all the uses to which this system can be put are probably unknown. It may point in new directions and lead to things not yet envisaged.

REFERENCES

Aho, A.V., Ullman, J.D. (1995). *Foundations of Computer Science* (C Edition). Computer Science Press, New York.

Aldous, J.M., Wilson, R.J. (2000). *Graphs and Applications: An Introductory Approach.* Springer, London.

Alberty, R.A. (2006) Calculation of Equilibrium Compositions of Systems of Enzyme-Catalyzed Reactions. *J. Phys. Chem. B* **110**, 24775 – 24779.

Altenberg, B., Greulich, K.O. (2004). Genes of glycolysis are ubiquitously overexpressed in 24 cancer classes. *Genomics* **6**, 1014 – 1020.

Asratian, A.S., Denley, T.M.J., Haggkvist, R. (1998). *Bipartite Graphs and their Applications.* Cambridge University Press, Cambridge.

Bairoch, A. (2000). The ENZYME database in 2000. . *Nucleic Acids Res.* **28**, 304 – 305.

Balaban, A.T. (1994). Reaction Graphs. In: Bonchev, D., Mekenyan, O. (Eds), *Graph Theoretical Approaches to Chemical Reactivity.* Kluwer, Dordrecht, NL.

Barabási, A.-L., Oltvai, Z.N. (2004). Network Biology: Understanding the Cell's Functional Organization. *Nature Rev. Genet.* **5**,101 – 113.

Barnett, S. (1995). *Some Modern Applications of Mathematics.* Prentice Hall/Ellis Horwood, London.

Barsky, A., Gardy, J.L., Hancock, R.E.W., Munzner, T. (2007). Cerebral: a Cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation. *Bioinformatics* **23**, 1040 – 1042.

Baynes, J., Dominiczak, M.H. (2003). *Medical Biochemistry.* Mosby, Barcelona, Spain.

Beasley, J.E., Planes, F.J. (2007). Recovering metabolic pathways via optimization. *Bioinformatics* **23**, 92 – 98.

Becker, M.Y., Rojas, I. (2001). A graph layout algorithm for drawing metabolic pathways. *Bioinformatics* **17**, 461 – 467.

Beineke, L.W., Wilson, R.J., Cameron, P.J. (2004). Topics in Algebraic Graph Theory. Cambridge University Press, New York, USA.

Benkö, G., Flamm, C., Stadler, F. (2003). Generic Properties of Chemical Networks: Artificial Chemistry Based on Graph Rewriting. In: Banzhaf, W., Christaller, T., Dittrich, T.P., Kim, J.T., Ziegler, J. (eds.), Advances in Artificial Life. *Lecture Notes in Computer Science* **2801**, 10 – 20, Springer-Verlag, Heidelberg, Germany.

Bishop, J. (1998). *Java Gently: Programming Principles Explained*, second edition. Addison-Wesley, Harlow, England.

Bonchev, D., Mekenyan, O. (Eds). (1994). *Graph Theoretical Approaches to Chemical Reactivity*. Kluwer, Dordrecht, NL.

Boyer, R. (1999). *Concepts in Biochemistry*. Brooks/Cole, California.

Busch, W., Saier, M.H. (2004). The IUBMB-endorsed transporter classification system. *Molecular Biotechnology* **27**, 253 – 262.

Campbell, M.K. (1995). *Biochemistry*. Saunders College Publishing, Orlando.

Campbell, N.A. (1996). *Biology*. Benjamin/Cummings, Menlo Park, California.

Card, S.K., Mackinlay, J.D., Shneiderman, B. (1999). *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, San Francisco.

Carey, F.A. (1996). *Organic Chemistry*. McGraw-Hill, New York.

Coffee, C.J. (1998). *Metabolism*. Fence Creek Publishing, Madison, Connecticut.

Connolly, T., Begg, C. (1999). *Database Systems: A Practical Approach to Design, Implementation, and Management*, second edition. Addison-Wesley, London.

Degtyarenko, K., de Matos, P., Ennis, M., Hastings, J., Zbinden, M., McNaught, A., Alcántara, R., Darsow, M., Guedj, M., Ashburner, M. (2008). ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Res.* **36**, D344 – D350.

Demir, E., Babur, O., Dogrusoz, U., Gursoy, A., Nisanci, G., Cetin-Atalay, R., Ozturk, M. (2002). Patika: an integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics* **18**, 996 – 1003.

Deville, Y., Gilbert, D., van Helden, J.,Wodak, S.J. (2003). An overview of data models for the analysis of biochemical pathways. *Brief. Bioinform* **4**, 246 – 259.

Diamond, D., Hanratty, V.C.A. (1997). *Spreadsheet Applications in Chemistry using Microsoft Excel.* Wiley-Interscience, New York.

Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G. (1999). Graph Drawing: *Algorithms for the Visualization of Graphs.* Prentice Hall, New Jersey.

Dickerson, J.A., Berleant, D., Cox, Z., Qi, W., Wurtele, E. (2001). Creating metabolic network models using text mining and expert knowledge. *Atlantic Symposium on Molecular Biology and Genome Information Systems and Technology.* Durham, North Carolina, pp 26 – 30.

Eades, P. (1984). A heuristic for graph drawing. *Congr. Numer.* **42**, 149 – 160.

Ellis, L.B.M., Speedie, S.M., McLeish, R. (1998). Representing metabolic pathway information: an object-oriented approach. *Bioinformatics* **14**, 803 – 806.

Ellis, L.B.M. (2001). Metabolic Pathway Databases. *Encyclopaedia of life Sciences*, www.els.net, Macmillan, pp 589 – 594.

Ellis, L.B.M., Hou, B.K., Kang, W., Wackett, L.P. (2003). The University of Minnesota Biocatalysis/Biodegradation Database: post-genomic data mining. *Nucleic Acids Res.* **31**, 262 – 265.

Elmasri, R., Navathe, S.B. (1994). *Fundamentals of Database Systems*, second edition. The Benjamin/Cummings Publishing Company, Inc., CA.

Fraleigh, J.B. (1982). *A First Course in Abstract Algebra*, third edition. Addison-Wesley, USA.

Fruchterman, T.M.J., Reingold, E.M. (1991). Graph Drawing by Force-directed Placement. *Soft. Pract. Exper.* **21**, 1129 – 1164.

Gagneur, J., Jackson, D.B., Casari, G. (2003). Hierarchical analysis of dependency in metabolic networks. *Bioinformatics* **19**, 1027 – 1034.

Gauges, R., Rost, U., Sahle, S., Wegner, K. (2006). A model diagram layout for SBML. *Bioinformatics* **22**, 1879 – 1885.

Givan, C.V. (1999). Evolving concepts in plant glycolysis: two centuries of progress. *Biol. Rev.* **74**, 277 – 309.

Goesmann, A., Haubrock, M., Meyer, F., Kalinowski, J., Giegerich, R. (2002). PathFinder: reconstruction and dynamic visualisation of metabolic pathways. *Bioinformatics* **18**, 124 – 129.

Goto, S., Bono, H., Ogata, H., Fujibuchi, W., Nishioka, T., Sato, K., Kanehisa, M. (1996). Organizing and computing metabolic pathway data in terms of binary relations. In: Altman, R.B., Dunker, K., Hunter, L., Klein, T.E. (eds.), *Pacific Symposium on Biocomputing '97.* World Scientific, Singapore, pp 175 – 186.

Goto, S., Nishioka, T., Kanehisa, M. (1998). LIGAND: chemical database for enzyme reactions. *Bioinformatics* **14**, 591 – 599.

Grossman, P. (1995). Discrete Mathematics for Computing. Macmillan Press Ltd., London.

Hames, B.D., Hooper, N.M., Houghton, J.D. (1997). *Instant Notes in Biochemistry.* BIOS, Oxford.

Hammond, M. (2002). *Programming for Linguists.* Blackwell, Oxford.

Hatzimanikatis, V., Li, C., Ionita, J.A., Henry, C.S., Jankowski, M.D., Broadbelt, L.J. (2005). Exploring the diversity of complex metabolic networks. *Bioinformatics* **21**, 1603 – 1609.

Hein, P. (1966). *Grooks.* MIT Press.

Heinrich, R., Schuster, S. (1996). *The Regulation of Cellular Systems.* Chapman & Hall, New York.

Heinrich, R., Schuster, S. (1998). The modelling of metabolic systems. Structure, control and optimality. *BioSystems* **47**, 61 – 77.

Holford. M., Li, N., Nadkarni, P., Zhao, H. (2005). VitaPad: visualisation tools for the analysis of pathway data. *Bioinformatics* **21**, 1596 – 1602.

Holme, P., Huss, M., Jeong, H. (2003). Subnetwork hierarchies of biochemical pathways. *Bioinformatics* **19**, 532 – 538.

Holmes, B. (1998). *Programming with Java.* Jones and Bartlett Publishers, Sudbury, Massachusetts.

Home, A.B., Hodgman, T.C., Spence, H.D., Dalby, A.R. (2004). Constructing an enzyme-centric view of metabolism. *Bioinformatics* **20**, 2050 – 2055.

Hu, Z., Mellor, J., Wu, J., Yamada, T., Holloway, D., DeLisi, C. (2005). VisANT: data-integrating visual framework for biological networks and modules. *Nucleic Acids Res.* **33**, W352 – W357.

Hu, Z., Ng, D.M., Yamada, T., Chen, C., Kawashima, S., Mellor, J., Linghu, B., Kanehisa, M., Stuart, J.M., DeLisi, C. (2007). VisANT 3.0: new modules for pathway visualisation, editing prediction and construction. *Nucleic Acids Res.* **35**, W625 – W632.

Hubbard, J.R., Huray, A. (2004). *Data Structures with Java.* Pearson Prentice Hall, USA.

Jammer, M. (1997). Concepts of Mass in Classical and Modern Physics. Dover, U.S.A.

Joshi-Tope, G., Gillespie, M., Vastrik, I., D'Eustachio, P., Schmidt, E., de Bono, B., Jassal, B., Gopinath, G.R., Wu, G.R., Matthews, L., Lewis, S., Birney, E., Stein, L. (2005). Reactome: a knowledgebase of biological pathways. *Nucleic Acids Res.* **33**, D428 – D432.

Junker, B.H., Koschützki, D., Schreiber, F. (2006). Kinetic Modelling with the Systems Biology Modelling Environment SyBME. *J. Integrative Bioinformatics*

Kahn, D., Westerhoff, H.V. (1991). Control Theory of Regulatory Cascades. *J. Theor. Biol.* **153**, 255 – 285.

Kamada, T., Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Inform. Process. Lett.* **31**, 7 – 15.

Kanehisa, M., Goto, S. (2000). KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res.* **28**, 27 – 30.

Kanehisa, M., Goto, S., Kawashima, S., Nakaya, A. (2002). The KEGG databases at GenomeNet. *Nucleic Acids Res.* **30**, 42 – 46.

Kanehisa, M., Goto, S., Kawashima, S., Okuno, Y., Hattori, M. (2004). The KEGG resource for deciphering the genome. *Nucleic Acids Res.* **32**, D277 – D280.

Karp, P., Paley, S. (1994a). Representations of Metabolic Knowledge: Pathways. In: Altman, R., Brutlag, D., Karp, P., Lathrop, R., Searls, D. (eds), *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology.* AAAI Press, Menlo Park, CA., pp 203 – 211.

Karp, P., Paley, S. (1994b). Automated drawing of metabolic pathways. In: Lim, H., Cantor, C., Robins, R. (Eds), *Third International Conference on Bioinformatics and Genome Research*. Tallahassee, Florida, pp. 225 – 238.

Karp, P., Lowrance, J., Strat, T., Wilkins, D. (1994). The Grasper-CL Management System. *LISP and Symbolic Computation* **7**, 245 – 282.

Karp, P.D., Riley, M., Paley, S.M., Pellegrini-Toole, A., Krummenacker, M. (1997). EcoCyc: Encyclopaedia of *Escherichia coli* genes and metabolism. *Nucleic Acids Res.* **25**, 43 – 50.

Karp, P.D. (1998). Metabolic databases. *TIBS*. 114 – 116.

Karp, P.D., Riley, M., Paley, S.M., Pellegrini-Toole, A., Krummenacker, M. (1998). EcoCyc: Encyclopaedia of *Escherichia coli* genes and metabolism. *Nucleic Acids Res.* **26**, 50 – 53.

Karp, P.D., Riley, M., Paley, S.M., Pellegrini-Toole, A. (2002). The MetaCyc Database. *Nucleic Acids Res.* **30**, 59 – 61.

Karp, P.D., Ouzounis, C.A., Moore-Kochlacs, C., Goldovsky, L., Kalpa, P., Ahren, D., Tsoka, S., Darzentas, N., Kunin, V., López-Bigas, N. (2005). Expansion of the BioCyc collection of pathway/genome databases to 160 genomes. *Nucleic Acids Res.* **33**, 6083 – 6089.

Kazic, T., Dunford-Shore, B., Holcomb, J., Fabrizio, F., Wise, W. (2000). *Moirai*: toward the Simulation of Cellular Systems. IBC Technical Report.

Képès, F. (2007). *Biological Networks*. World Scientific.

Kitano, H. (2003). A graphical notation for biochemical networks. *Biosilico* **1**, 169 – 176.

Klamt, S., Stelling, J. (2003). Stoichiometric analysis of metabolic networks. *Tutorial at the 4th International Conference on Systems Biology*, St. Louis, Missouri, USA.

Klipp, E., Herwig, R., Kowald, A., Wierling, C., Lehrach, H. (2005). *Systems Biology in Practice*. WILEY-VCH, Weinheim.

Klukas, C., Schreiber, F. (2007). Dynamic exploration and editing of KEGG pathway diagrams. *Bioinformatics* **23**, 344 – 350.

Kohn, M., Letzkus, W. (1983). A Graph-theoretical Analysis of Metabolic Regulation. *J. Theor. Biol.* **100**, 293 – 304.

Kohn, M.C., Lemieux, D.R. (1991). Identification of Regulatory Properties of Metabolic Networks by Graph Theoretical Modelling. *J. Theor. Biol.* **150**, 3 – 25.

Kohn, M.C., Tohmaz, A.S., Giroux, K.J. (2002). Robustness of MetaNet graph models: Predicting control of urea production in humans. *Biosystems* **65**, 61 – 78.

König, D. (1916). Uber Graphen und ihre Anwendungen, *Math. Annalen* **77**, 453 – 465.

Koyutürk, M., Grama, A., Szpankowski, W. (2004). An efficient algorithm for detecting frequent subgraphs in biological networks. *Bioinformatics* **20**, i200 – i207.

Krishnamurthy, L., Nadeau, J., Ozsoyoglu, G., Ozsoyoglu, M., Schaeffer, M., Tasan, M., Xu, W. (2003). Pathways database system: an integrated system for biological pathways. *Bioinformatics* **19**, 930 – 937.

Kuffner, R., Zimmer, R., Lengaur, T. (2000). Pathway analysis in metabolic databases via differential metabolic display (DMD). *Bioinformatics* **16**, 825 – 836.

Lee, T.J., Paulsen, I., Karp, P. (2008). Annotation-based inference of transporter function. *Bioinformatics* **24**, i259 – i267.

Lehninger, A.L., Nelson, D.L., Cox, M.M. (1993). *Principles of Biochemistry*, second edition. Worth, New York.

Li, C., Henry, C.S., Jankowski, M.D., Ionita, J.A., Hatzimanikatis, V., Broadbelt, L.J. (2004). Computational discovery of biochemical routes to specialty chemicals. *Chem. Eng.Sci.* **59**, 5051 – 5060.

Lloyd, C.M., Halstead, M.D.B., Nielsen, P.F. (2004). CellML: its future, present and past. *Prog. Biophys. Mol. Biol.* **85**, 433 – 450.

Luenberger, D.G. (1989). *Linear and Nonlinear Programming*, second edition. Addison Wesley, USA.

Luyf, A.C.M., de Gast, J., van Kampen, A.H.C. (2002). Visualizing metabolic activity on a genome-wide scale. *Bioinformatics* **18**, 813 – 818.

Ma, H., Zeng, A. (2003). Reconstruction of metabolic networks from genome data and analysis of their global structure for various organisms. *Bioinformatics* **19**, 270 – 277.

Ma, H., Zhao, X., Yuan, Y., Zeng, A. (2004). Decomposition of metabolic network into functional modules based on the global connectivity structure of reaction graph. *Bioinformatics* **20**, 1870 – 1876.

Mavrovouniotis, M. L., Stephanopoulos, G., Stephanopoulos, G. (1990). Computer-aided synthesis of biochemical pathways. *Biotechnol. Bioeng.* **36**, 1119 – 1132.

McDonald, A.G., Boyce, S., Moss, G.P., Dixon, H.B.F., Tipton, K.F. (2007). ExplorEnz: a MySQL database of the IUBMB enzyme nomenclature. *BMC Biochemistry* **8**, 14 – 20.

Michal, G. (1998). On representation of metabolic pathways. *Biosystems* **47**, 1 – 7.

Michal, G. (Ed.), (1999). *Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology*. Wiley, New York.

Mittenthal, J.E. (1996). An algorithm to assemble pathways from processes. In: Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E. (Eds.), *Pacific Symposium on Biocomputing '97*. World Scientific, Singapore, pp. 292 – 303.

Mittenthal, J.E., Clarke, B., Waddell, T.G., Fawcett, G. (2001). A New Method for Assembling Metabolic Networks, with Application to the Krebs Citric Acid Cycle. *J. Theor. Biol.* **208**, 361 – 382.

Muir, J. (1997). *Excel 97 Basic Skills: A progressive course for new users.* Letts Educational, London.

Mullan, L.J. (2003). Biological sequence databases. *Brief. Bioinform* **4** (1), 75 – 77.

Mulquiney, P.J., Kuchel, P.W. (2003). Modelling Metabolism with *Mathematica.* CRC Press, Boca Raton, Florida.

Munnelly, B. (1999). *ECDL The Complete Coursebook.* Gill & Macmillan, Dublin.

Murray, R.K., Granner, D.K., Mayes, P.A., Rodwell, V.W. (2003). *Harper's Illustrated Biochemistry*, twenty-sixth edition. McGraw-Hill, New York.

Nicholson, D.E. (2000). *Metabolic pathways chart*, 21st edition. International Union of Biochemistry and Molecular Biology, www.iubmb.unibe.ch/.

Ochs, R.S., Khadikar, S., Ozsoyoslu, G., Khwaja, A.A. (1994). Relational Database Model for Metabolic Information. *Proc. 3rd. Int. Conf. on Bioinformatics and Genome Research*, pp. 251 – 263.

Ogata, H., Goto, S., Fujibuchi, W., Kanehisa, M. (1998). Computation with the KEGG pathway database. *BioSystems* **47**, 119 – 128.

Ogata, H., Fujibuchi, W., Goto, S., Kanehisa, M. (2000). A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Res.*, **28**, 4021 – 4028.

Opperdoes, F.R., Borst, P. (1977). Localization of nine glycolytic enzymes in a microbody-like organelle in *Trypanosoma Brucei: the glycosome. FEBS Lett.* **80**, 360 – 364.

Orengo, C.A., Jones, D.T., Thornton, J.M. (2003). *Bioinformatics: Genes, Proteins and Computers*. Bios, Oxford.

Overbeek, R., Larsen, N., Pusch, G.D., D'Souza, M., Selkov Jr., E., Kyrpides, N., Fonstein, M., Maltsev, N., Selkov, E. (2000). WIT: integrated system for high-throughput genome sequence analysis and metabolic reconstruction. *Nucleic Acids Res.*, **28**, 123 – 125.

Paley, S.M., Karp, P.D. (1996). Adapting EcoCyc for use on the World Wide Web. Gene **172**, GC43 – GC50.

Palmer, T. (1995). *Understanding Enzymes*, fourth edition. Prentice Hall/Ellis Horwood, London.

Palsson, B.Ø. (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press, New York, USA.

Papin J.A., Price N.D., Wilback S.J., Fell D.A., Palsson B. Ø. (2003). Metabolic pathways in the post-genome era. *Trends Biochem. Sci.* **28**, 250 – 258.

Peterson, J.L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Inc., Englewood Cliffs, N.J.

Pfeiffer, T., Sanchez-Valdenebro, I., Nuño, J.C., Montero, F., Schuster, S. (1999). METATOOL: for studying metabolic networks. *Bioinformatics* **15**, 251 – 257.

Planes, F.J., Beasley, J.E. (2008). A critical examination of stoichiometric and path-finding approaches to metabolic pathways. *Brief. Bioinform* **9**, 422 – 436.

Pollack, J.D., Williams, M.V., McElhaney, R.N. (1997). The comparative metabolism of the mollicutes (Mycoplasmas): the utility for taxonomic classification and the relationship of putative gene annotation and phylogeny to enzymatic function in the smallest free-living cells. *Crit. Rev. Microbiol.* **23**, 269 – 354.

Rahman, S.A., Advani, P., Schunk, R., Schrader, R., Schomberg, D. (2005). Metabolic pathway analysis web service (Pathway Hunter Tool at CUBIC). *Bioinformatics* **21**, 1189 – 1193.

Rashidi, H.H., Buehler, L.K. (2000). Bioinformatics Basics: Applications in Biological Science and Medicine. CRC Press LLC, Boca Raton, Florida.

Ravasz, E., Somera, A.L., Mongru, D.A., Oltvai, Z.N., Barabasi, A.-L. (2002). Hierarchical Organisation of Modularity in Metabolic Networks. *Science* **297**, 1551 – 1555.

Reddy, V.N., Liebman, M.N., Mavrovouniotis, M. L. (1996). Qualitative analysis of biochemical reaction systems. *Comput. Biol. Med.* **26**, 9 – 24.

Render, B., Stair, Jr., R.M. (2000). *Quantitative Analysis for Management*, seventh edition. Prentice Hall, New Jersey.

Rojdestvenski, I. (2003) Metabolic pathways in three dimensions. *Bioinformatics* **19**, 2436 – 2441.

Rolland, F.D. (1998). *The Essence of Databases*. Prentice Hall, London.

Rothwell, D., M. (1993). *Databases: An Introduction.* McGraw-Hill.

Shannon, P., Markiel, A., Ozier, O., Baliga, N.S., Wang, J.T., Ramage, D., Amin, N., Schwikowski, B., Ideker, T. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res. **13**, 2498 – 2504.

Schilling, C.H., Letscher, D., Palsson, B. Ø. (2000). Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. *J. Theor. Biol.* **203**, 229 – 248.

Schmidt, S., Dandekar, T. (2002). Metabolic Pathways: Postgenomic Computational Approaches. In: Collado-Vides, J., Hofestädt, R. (Eds), *Gene Regulation and Metabolism.* MIT Press, Massachusetts.

Schomberg, I., Chang, A., Schomberg, D. (2002). BRENDA, enzyme data and metabolic information. *Nucleic Acids Res.* **30**, 47 – 49.

Schomberg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., Schomberg, D. (2004). BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Res.* **32**, D431 – D433.

Schreiber, F. (2002). High quality visualisation of biochemical pathways in BioPath. *In Silico Biology* **2** (2), 59 – 73.

Schuster, S., Hilgetag, C. (1994). On elementary flux modes in biochemical reaction systems at steady state. *J. Biol. Syst.* **2**, 165 – 182.

Schuster, S., Dandekar, T., Fell, D.A. (1999). Detection of elementary flux modes in biochemical networks: a promising tool for pathway analysis and metabolic engineering. *Trends Biotechnol.* **17**, 53 – 60.

Schuster, S., Fell, D.A., Dandekar, T.A. (2000). A general definition of metabolic pathways useful for systematic organisation and analysis of complex metabolic networks. *Nature. Biotechnol.* **18**, 326 – 332.

Schuster, S., Pfeiffer, T., Moldenhauer, F., Koch, I., Dandekar, T. (2002). Exploring the pathway structure of metabolism: decomposition into subnetworks and application to *Mycoplasma pneumoniae*. *Bioinformatics* **18**, 351 – 357.

Selkov, E., Maltsev, N., Olsen, G.J., Overbeek, R., Whitman, W.B. (1997). A reconstruction of the metabolism of *Methanococcus jannaschii* from sequence data. *Gene* **197**, GC11 – 26.

Sen, A.K. (1991). Calculation of Control Coefficients of metabolic pathways: A flux-oriented graph-theoretic approach. *Biochem. J.* **279**, 55 – 65.

Seressiotis, A., Bailey, J.E. (1986). MPS: An algorithm and data base for metabolic pathway synthesis. *Biotechnol. Lett.* **8**, 837 – 842.

Seressiotis, A., Bailey, J.E. (1988). MPS: An artificially intelligent software system for the analysis and synthesis of metabolic pathways. *Biotechnol. Bioeng* **31**, 587 – 602.

Sirava, M., Schäfer, T. Eiglsperger, M., Kaufmann, M., Kohlbacher, O., Bornberg-Bauer, E., Lenhof, H. (2002). BioMiner – modelling, analyzing, and visualizing biochemical pathways and networks. *Bioinformatics* **18**, S219 – S230.

Sorokin, A., Paliy, K., Selkov, A., Demin, O.V., Dronov, S., Ghazal, P., Goryanin, I. (2006). The Pathway Editor: A tool for managing complex biological networks. *IBM J. Res. Dev.* **50**, 561 – 573.

Sprott, G.D., Ekiel, I., Patel, G.B. (1993). Metabolic pathways in *Methanococcus jannaschii* and other methanogenic bacteria. *Appl. Environ. Microbiol.* **59**, 1092 – 1098.

Srinivasan, B.S., Shah, N.H., Flannick, J.A., Abeliuk, E., Novak, A.F., Batzoglou, S. (2007). Current progress in network research: toward reference networks for key model organisms. *Brief. Bioinform* **8**, 318 – 332.

Stephanopulous, G.N., Aristidou, A.A., Nielsen, J. (1998). *Metabolic Engineering*. Academic Press, San Diego.

Stryer, L., Berg, J.M., Tymoczko, J.L. (2007). *Biochemistry*, sixth edition. Freeman, New York.

StrömbÄck, L., Lambrix, P. (2005). Representations of molecular pathways: an evaluation of SBML, PSI MI and BioPAX. *Bioinformatics* **24**, 4401 – 4407.

Suderman, M., Hallett, M. (2007). Tools for visually exploring biological networks. *Bioinformatics* **23**, 2651 – 2659.

Sugiayama, K., Tagawa, S., Toda, M. (1981). Methods for Visual Understanding of Hierarchical Systems. *IEEE Trans. Syst. Cybern.* **11**, 109 – 125.

Temkin, O.N., Zeigarnik, A.V, Bonchev, D. (1994). Graph-Theoretical Models of Complex Reaction Mechanisms and their Elementary Steps. In: Bonchev, D., Mekenyan, O. (Eds), *Graph Theoretical Approaches to Chemical Reactivity*. Kluwer, Dordrecht, NL.

Tennent, R.M. (1976). *Science Data Book*. Oliver & Boyd, Edinburgh.

Terzer, M., Stelling, J. (2008). Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics* **24**, 2229 – 2235.

Tipton, K.F., O'Sullivan, M.I., Davey, G.P., O'Sullivan, J. (2003). It can be a complicated life being an enzyme. *Biochem. Soc. Trans.* **31**, 711 – 715.

Tucker, A. (2002). *Applied Combinatorics, fourth edition*. Wiley, New York, USA.

Urbanczik, R., Wagner, C. (2005). An improved algorithm for stoichiometric network analysis: theory and applications. *Bioinformatics* **21**, 1203 – 1210.

van Helden, J., Gilbert, D., Wernisch, L., Schroeder, M., Wodak, S. (2001). Applications of regulatory sequence analysis and metabolic network analysis to the interpretation of gene Expression data. *Lecture Notes in Computer Sciences* **2066**, 155 – 172.

Voet, D., Voet, J.G. (1995). *Biochemistry*, second edition. Wiley, New York.

Voit, E.O. (2000). *Computational Analysis of Biochemical Systems*. Cambridge University Press, New York.

von Kamp, A., Schuster, S. (2006). Metatool 5.0: fast and flexible elementary modes analysis. *Bioinformatics* **22**, 1930 – 1931.

Wagner, A., Fell, D. (2001). The small world inside metabolic networks. *Proc. R. Soc. Lond. Ser. B* **268**, 1803 – 810.

Wang, Z.-Y., Jenkinson, J.M., Holcombe, L.J., Soanes, D.M., Veneault-Fourrey, C., Bhambra, G.K., Talbot, N.J. (2005). The molecular biology of appressorium turgor

generation by the rice blast fungus *Magneporthe grisea*. *Biochem. Soc. Trans.* **33**, 384 – 388.

Webb, E.C. (1992). *Enzyme Nomenclature: Recommendations of the nomenclature committee of the International Union of Biochemistry and Molecular Biology on the nomenclature and classification of enzymes*. Academic Press Limited, London.

Wegner, K., Kummer, U. (2005). A new dynamical layout algorithm for complex biochemical reaction networks. BMC Bioinformatics 6, 212 – 223.

Weininger, D. (1988). SMILES 1. Introduction and encoding rules. *J. Chem. Inf. Comput. Sci.* **28**, 31 – 36.

Westhead, D.R., Parish, J.H., Twyman, R.M. (2002). *Instant Notes: Bioinformatics*. BIOS Scientific Publishers Limited, Oxford.

Wiechert W. (2002). Modeling and simulation: tools for metabolic engineering. *J. Biotechnol.* **94**, 37 – 63.

Wittig, U., De Beuckelaer, A. (2001). Analysis and comparison of metabolic pathway databases. *Brief. Bioinform* **2**, 126 – 142.

Ye, S.Q. (2008). *Bioinformatics: A Practical Approach*. Chapman & Hall/CRC, London.

Yoon, J., Si, Y., Nolan, R., Lee, K. (2007). Modular decomposition of metabolic reaction networks based on flux analysis and pathway projection. *Bioinformatics* **23**, 2433 – 2440.

Zeigarnik, A.V., Temkin, O.N. (1994). A graph-theoretic model of complex reaction mechanisms: bipartite graphs and the stoichiometry of complex reactions. *Kinet. Katal.* (Engl. Transl.) **35**, 691 – 701.

APPENDICES

Appendix A1

The data in the "General.dat" and the "User.dat" data files are shown below. Each record contains the following information: the operator code, the operator name and the reaction or regulatory action. These files are tab delimited, so they can be input automatically to the Java application. The "General.dat" file includes the reactions of the Nicholson Metabolic Pathways wall chart (Nicholson, 2000). These were downloaded from the *Enzyme Nomenclature* web site (http://www.chem.qmul.ac.uk/iubmb/enzyme/) using the EC numbers from the chart. The "User.dat" file is a simpler version of the "General.dat" file, i.e., unsuitable reaction equations (for pathway construction) have been removed or replaced manually. Furthermore, two reactions of trehalose metabolism (2.4.1.15 and 3.2.1.28), an additional reaction for transketolase (2.2.1.1), two ligand-receptor binding reactions (8.0.0.1 and 8.0.0.2), one transporter reaction (2.A.1.1.28) and two spontaneous (uncatalysed) reactions (0.0.0.1 and 0.0.0.2) have been manually added to the "User.dat" file. These items are highlighted in blue.

Spontaneous (uncatalysed) reactions and receptors are assigned to arbitrary classes 0 and 8, respectively. These are not internationally accepted codes but are defined here for the purpose of the project. A transporter classification (TC) number consists of five elements separated by points (Busch and Saier, 2004). The TC number 2.A.1.1.28 represents the erythrocyte/brain hexose facilitator, GLUT1 of Homo sapiens. However, the information on transporters as represented in the Membrane Transport Protein Classification Database (www.tcdb.org) is unsuitable for drawing pathways, and GLUT1 is represented as Glut1. Therefore, reaction 2.A.1.1.28 was converted into a suitable format as follows:

2.A.1.1.28 hs_Glut1 e-D-glucose = D-glucose

The 'Example' field in the transporter database includes a specific reference to species. It was possible to represent this information here in the extended operator (see Section 3.7.2) hs_Glut1, where 'hs' represents Homo sapiens. Furthermore, the transporter database does not have a field for specific transport reactions but only contains generalised transport reactions as comments. However, it was necessary to represent this reaction explicitly here for the purpose of pathway drawing. The prefix 'e' on the left hand side of the reaction

denotes the extracellular form of D-glucose and prevents D-glucose from joining with itself in the same reaction in the graph representation. On the other hand, a corresponding prefix to denote intracellular D-glucose was not included in order to link the transporter reaction with the first reaction (2.7.1.1) on the glycolysis pathway.

Digits tare used to distinguish between the two different reactions catalysed by the enzyme transketolase (transketolase1 and transketolase2) in the pentose phosphate pathway, for the purpose of display (see Fig. 4.23).

General.dat

```
1.1.1.1   alcohol dehydrogenase    an alcohol + NAD = an
aldehyde or ketone + NADH2
1.1.1.8   glycerol-3-phosphate dehydrogenase (NAD)     sn-
glycerol 3-phosphate + NAD = glycerone phosphate + NADH2
1.1.1.9   D-xylulose reductase     xylitol + NAD = D-
xylulose + NADH2
1.1.1.10  L-xylulose reductase     xylitol + NADP = L-
xylulose + NADPH2
1.1.1.14  L-iditol 2-dehydrogenase L-iditol + NAD = L-
sorbose + NADH2
1.1.1.15  D-iditol 2-dehydrogenase D-iditol + NAD = D-
sorbose + NADH2
1.1.1.19  L-glucuronate reductase  L-gulonate + NADP = D-
glucuronate + NADPH2
1.1.1.21  aldehyde reductase  alditol + NAD(P) = aldose +
NAD(P)H
1.1.1.22  UDPglucose 6-dehydrogenase    UDPglucose + 2 NAD +
H2O = UDPglucuronate + 2 NADH2
1.1.1.23  histidinol dehydrogenase L-histidinol + 2 NAD = L-
histidine + 2 NADH2
1.1.1.25  shikimate 5-dehydrogenase     shikimate + NADP =
5-dehydroshikimate + NADPH2
1.1.1.27  L-lactate dehydrogenase  (S)-lactate + NAD =
pyruvate + NADH2
1.1.1.28  D-lactate dehydrogenase  (R)-lactate + NAD =
pyruvate + NADH2
1.1.1.29  glycerate dehydrogenase  (R)-glycerate + NAD =
hydroxypyruvate + NADH2
1.1.1.30  3-hydroxybutyrate dehydrogenase    (R)-3-
hydroxybutanoate + NAD = acetoacetate + NADH2
1.1.1.31  3-hydroxyisobutyrate dehydrogenase 3-hydroxy-2-
methylpropanoate + NAD = 2-methyl-3-oxopropanoate + NADH2
```

1.1.1.32 mevaldate reductase (R)-mevalonate + NAD = mevaldate + NADH2

1.1.1.34 hydroxymethylglutaryl-CoA reductase (NADPH2) (R)-mevalonate + CoA + 2 NADP = (S)-3-hydroxy-3-methylglutaryl-CoA + 2 NADPH2

1.1.1.35 3-hydroxyacyl-CoA dehydrogenase (S)-3-hydroxyacyl-CoA + NAD = 3-oxoacyl-CoA + NADH2

1.1.1.37 malate dehydrogenase (S)-malate + NAD = oxaloacetate + NADH2

1.1.1.38 malate dehydrogenase (oxaloacetate-decarboxylating) (S)-malate + NAD = pyruvate + CO2 + NADH2

1.1.1.39 malate dehydrogenase (decarboxylating) (S)-malate + NAD = pyruvate + CO2 + NADH2

1.1.1.41 isocitrate dehydrogenase (NAD) isocitrate + NAD = 2-oxoglutarate + CO2 + NADH2

1.1.1.43 phosphogluconate 2-dehydrogenase 6-phospho-D-gluconate + NAD(P) = 6-phospho-2-dehydro-D-gluconate + NAD(P)H

1.1.1.44 phosphogluconate dehydrogenase (decarboxylating) 6-phospho-D-gluconate + NADP = D-ribulose 5-phosphate + CO2 + NADPH2

1.1.1.45 L-gulonate 3-dehydrogenase L-gulonate + NAD = 3-dehydro-L-gulonate + NADH2

1.1.1.49 glucose-6-phosphate 1-dehydrogenase D-glucose 6-phosphate + NADP = D-glucono-1,5-lactone 6-phosphate + NADPH2

1.1.1.56 ribitol 2-dehydrogenase ribitol + NAD = D-ribulose + NADH2

1.1.1.59 3-hydroxypropionate dehydrogenase 3-hydroxypropanoate + NAD = 3-oxopropanoate + NADH2

1.1.1.60 2-hydroxy-3-oxopropionate reductase (R)-glycerate + NAD(P) = 2-hydroxy-3-oxopropanoate + NAD(P)H

1.1.1.61 4-hydroxybutyrate dehydrogenase 4-hydroxybutanoate + NAD = succinate semialdehyde + NADH2

1.1.1.79 glyoxylate reductase (NADP) glycolate + NADP = glyoxylate + NADPH2

1.1.1.81 hydroxypyruvate reductase D-glycerate + NAD(P) = hydroxypyruvate + NAD(P)H

1.1.1.82 malate dehydrogenase (NADP) (S)-malate + NADP = oxaloacetate + NADPH2

1.1.1.85 3-isopropylmalate dehydrogenase 3-carboxy-2-hydroxy-4-methylpentanoate + NAD = 3-carboxy-4-methyl-2-oxopentanoate + NADH2

1.1.1.86 ketol-acid reductoisomerase (R)-2,3-dihydroxy-3-methylbutanoate + NADP = (S)-2-hydroxy-2-methyl-3-oxobutanoate + NADPH2

1.1.1.95 phosphoglycerate dehydrogenase 3-phosphoglycerate + NAD = 3-phosphohydroxypyruvate + NADH2

1.1.1.100 3-oxoacyl-[acyl-carrier-protein] reductase (3R)-3-hydroxyacyl-[acyl-carrier protein] + NADP = 3-oxoacyl-[acyl-carrier protein] + NADPH2

1.1.1.102 3-dehydrosphinganine reductase sphinganine + NADP = 3-dehydrosphinganine + NADPH2

1.1.1.105 retinol dehydrogenase retinol + NAD = retinal + NADH2

1.1.1.155 homoisocitrate dehydrogenase (-)-1-hydroxy-1,2,4-butanetricarboxylate + NAD = 2-oxoadipate + CO2 + NADH2

1.1.1.157 3-hydroxybutyryl-CoA dehydrogenase (S)-3-hydroxybutanoyl-CoA + NADP = 3-acetoacetyl-CoA + NADPH2

1.1.1.158 UDP-N-acetylmuramate dehydrogenase UDP-N-acetylmuramate + NADP = UDP-N-acetyl-3-O-(1-carboxyvinyl)-D-glucosamine + NADPH2

1.1.1.169 2-dehydropantoate 2-reductase (R)-pantoate + NADP = 2-dehydropantoate + NADPH2

1.1.1.204 xanthine dehydrogenase xanthine + NAD + H2O = urate + NADH2

1.1.1.205 IMP dehydrogenase Inosine 5'-phosphate + NAD + H2O = xanthosine 5'-phosphate + NADH2

1.1.3.8 L-gulonolactone oxidase L-gulono-1,4-lactone + O2 = L-xylo-hexulonolactone + H2O

1.1.3.15 (S)-2-hydroxy-acid oxidase (S)-2-hydroxy acid + O2 = 2-oxo acid + H2O2

1.1.3.22 xanthine oxidase xanthine + H2O + O2 = urate + H2O2

1.1.99.1 choline dehydrogenase choline + acceptor = betaine aldehyde + reduced acceptor

1.1.99.5 glycerol-3-phosphate dehydrogenase sn-glycerol 3-phosphate + acceptor = glycerone phosphate + reduced acceptor

1.2.1.3 aldehyde dehydrogenase (NAD) an aldehyde + NAD + H2O = an acid + NADH2

1.2.1.7 benzaldehyde dehydrogenase (NADP) benzaldehyde + NADP + H2O = benzoate + NADPH2

1.2.1.8 betaine-aldehyde dehydrogenase betaine aldehyde + NAD + H2O = betaine + NADH2

1.2.1.11 aspartate-semialdehyde dehydrogenase L-aspartate 4-semialdehyde + phosphate + NADP = L-4-aspartyl phosphate + NADPH2

1.2.1.12 glyceraldehyde-3-phosphate dehydrogenase (phosphorylating) D-glyceraldehyde 3-phosphate + phosphate + NAD = 3-phospho-D-glyceroyl phosphate + NADH2

1.2.1.13 glyceraldehyde-3-phosphate dehydrogenase (NADP) (phosphorylating) D-glyceraldehyde 3-phosphate + phosphate + NADP = 3-phospho-D-glyceroyl phosphate + NADPH2

1.2.1.16 succinate-semialdehyde dehydrogenase [NAD(P)] succinate semialdehyde + NAD(P) + H2O = succinate + NAD(P)H2

1.2.1.18 malonate-semialdehyde dehydrogenase (acetylating) 3-oxopropanoate + CoA + NAD(P) = acetyl-CoA + CO2 + NAD(P)H2

1.2.1.19 aminobutyraldehyde dehydrogenase 4-aminobutanal + NAD + H2O = 4-aminobutanoate + NADH2

1.2.1.21 glycolaldehyde dehydrogenase glycolaldehyde + NAD + H2O = glycolate + NADH2

1.2.1.23 2-oxoaldehyde dehydrogenase (NAD) a 2-oxoaldehyde + NAD + H2O = a 2-oxo acid + NADH2

1.2.1.24 succinate-semialdehyde dehydrogenase succinate semialdehyde + NAD + H2O = succinate + NADH2

1.2.1.25 2-oxoisovalerate dehydrogenase (acylating) 3-methyl-2-oxobutanoate + CoA + NAD = 2-methylpropanoyl-CoA + CO2 + NADH2

1.2.1.26 2,5-dioxovalerate dehydrogenase 2,5-dioxopentanoate + NADP + H2O = 2-oxoglutarate + NADPH2

1.2.1.27 methylmalonate-semialdehyde dehydrogenase (acylating) 2-methyl-3-oxopropanoate + CoA + NAD = propanoyl-CoA + CO2 + NADH2

1.2.1.28 benzaldehyde dehydrogenase (NAD) benzaldehyde + NAD + H2O = benzoate + NADH2

1.2.1.31 L-aminoadipate-semialdehyde dehydrogenase L-2-aminoadipate 6-semialdehyde + NAD(P) + H2O = L-2-aminoadipate + NAD(P)H2

1.2.1.32 aminomuconate-semialdehyde dehydrogenase 2-aminomuconate 6-semialdehyde + NAD + H2O = 2-aminomuconate + NADH2

1.2.1.36 retinal dehydrogenase retinal + NAD + H2O = retinoate + NADH2

1.2.1.39 phenylacetaldehyde dehydrogenase phenylacetaldehyde + NAD + H2O = phenylacetate + NADH2

1.2.1.41 glutamate-5-semialdehyde dehydrogenase L-glutamate 5-semialdehyde + phosphate + NADP = L-glutamyl 5-phosphate + NADPH2

1.2.1.52 oxoglutarate dehydrogenase (NADP) 2-oxoglutarate + CoA + NADP = succinyl-CoA + CO2 + NADPH2

1.2.3.5 glyoxylate oxidase glyoxylate + H2O + O2 = oxalate + H2O2

1.2.4.1 pyruvate dehydrogenase (lipoamide) pyruvate + lipoamide = S-acetyldihydrolipoamide + CO2

1.2.4.2 oxoglutarate dehydrogenase (lipoamide) 2-oxoglutarate + lipoamide = S-succinyldihydrolipoamide + CO2

1.2.7.1 pyruvate synthase pyruvate + CoA + oxidized ferredoxin = acetyl-CoA + CO2 + reduced ferredoxin

1.2.7.2 2-oxobutyrate synthase 2-oxobutanoate + CoA + oxidized ferredoxin = propanoyl-CoA + CO2 + reduced ferredoxin

1.3.1.1 dihydrouracil dehydrogenase (NAD) 5,6-dihydrouracil + NAD = uracil + NADH2

1.3.1.2 dihydropyrimidine dehydrogenase (NADP) 5,6-dihydrouracil + NADP = uracil + NADPH2

1.3.1.6 fumarate reductase (NADH2) succinate + NAD = fumarate + NADH2

1.3.1.8 acyl-CoA dehydrogenase (NADP) acyl-CoA + NADP = 2,3-dehydroacyl-CoA + NADPH2

1.3.1.9 enoyl-[acyl-carrier-protein] reductase (NADH2) acyl-[acyl-carrier protein] + NAD = trans-2,3-dehydroacyl-[acyl-carrier protein] + NADH2

1.3.1.10 enoyl-[acyl-carrier-protein] reductase (NADPH2, B-specific) acyl-[acyl-carrier protein] + NADP = trans-2,3-dehydroacyl-[acyl-carrier protein] + NADPH2

1.3.1.13 prephenate dehydrogenase (NADP) prephenate + NADP = 4-hydroxyphenylpyruvate + CO2 + NADPH2

1.3.1.14 orotate reductase (NADH2) (S)-dihydroorotate + NAD = orotate + NADH2

1.3.1.26 dihydrodipicolinate reductase 2,3,4,5-tetrahydrodipicolinate + NAD(P) = 2,3-dihydrodipicolinate + NAD(P)H

1.3.1.35 phosphatidylcholine desaturase 1-acyl-2-oleoyl-sn-glycero-3-phosphocholine + NAD = 1-acyl-2-linoleoyl-sn-glycero-3-phosphocholine + NADH2

1.3.3.3 coproporphyrinogen oxidase coproporphyrinogen-III + O2 = protoporphyrinogen-IX + 2 CO2

1.3.3.4 protoporphyrinogen oxidase protoporphyrinogen-IX + O2 = protoporphyrin-IX + H2O

1.3.5.1 succinate dehydrogenase (ubiquinone) succinate + ubiquinone = fumarate + ubiquinol

1.3.99.1 succinate dehydrogenase succinate + acceptor = fumarate + reduced acceptor

1.3.99.2 butyryl-CoA dehydrogenase butanoyl-CoA + acceptor = 2-butenoyl-CoA + reduced acceptor

1.3.99.3 acyl-CoA dehydrogenase acyl-CoA + acceptor = 2,3-dehydroacyl-CoA + reduced acceptor

1.3.99.7 glutaryl-CoA dehydrogenase glutaryl-CoA + acceptor = crotonoyl-CoA + CO2 + reduced acceptor

1.3.99.10 isovaleryl-CoA dehydrogenase 3-methylbutanoyl-CoA + acceptor = 3-methylbut-2-enoyl-CoA + reduced acceptor

1.4.1.1 alanine dehydrogenase L-alanine + H2O + NAD = pyruvate + NH3 + NADH2

1.4.1.2 glutamate dehydrogenase L-glutamate + H2O + NAD = 2-oxoglutarate + NH3 + NADH2

1.4.1.7 serine dehydrogenase L-serine + H2O + NAD = 3-hydroxypyruvate + NH3 + NADH2

1.4.1.8 valine dehydrogenase (NADP) L-valine + H2O + NADP = 3-methyl-2-oxobutanoate + NH3 + NADPH2

1.4.1.9 leucine dehydrogenase L-leucine + H2O + NAD = 4-methyl-2-oxopentanoate + NH3 + NADH2

1.4.1.10 glycine dehydrogenase glycine + H2O + NAD = glyoxylate + NH3 + NADH2

1.4.1.13 glutamate synthase (NADPH2) 2 L-glutamate + NADP = L-glutamate + 2-oxoglutarate + NADPH2

1.4.1.14 glutamate synthase (NADH2) 2 L-glutamate + NAD = L-glutamine + 2-oxoglutarate + NADH2

1.4.1.19 tryptophan dehydrogenase L-tryptophan + NAD(P) = (indol-3-yl)pyruvate + NH3 + NAD(P)H2

1.4.3.1 D-aspartate oxidase D-aspartate + H2O + O2 = oxaloacetate + NH3 + H2O2

1.4.3.2 L-amino-acid oxidase an L-amino acid + H2O + O2 = a 2-oxo acid + NH3 + H2O2

1.4.3.4 amine oxidase (flavin-containing) RCH2NH2 + H2O + O2 = RCHO + NH3 + H2O2

1.4.3.6 amine oxidase (copper-containing) RCH2NH2 + H2O + O2 = RCHO + NH3 + H2O2

1.4.3.8 ethanolamine oxidase ethanolamine + H2O + O2 = glycolaldehyde + NH3 + H2O2

1.4.4.2 glycine dehydrogenase (decarboxylating) glycine + lipoylprotein = S-aminomethyldihydrolipoylprotein + CO2

1.5.1.1 pyrroline-2-carboxylate reductase L-proline + NAD(P) = 1-pyrroline-2-carboxylate + NAD(P)H

1.5.1.2 pyrroline-5-carboxylate reductase L-proline + NAD(P) = 1-pyrroline-5-carboxylate + NAD(P)H

1.5.1.3 dihydrofolate reductase 5,6,7,8-tetrahydrofolate + NADP = 7,8-dihydrofolate + NADPH

1.5.1.5 methylenetetrahydrofolate dehydrogenase (NADP) 5,10-methylenetetrahydrofolate + NADP = 5,10-methenyltetrahydrofolate + NADPH

1.5.1.6 formyltetrahydrofolate dehydrogenase 10-formyltetrahydrofolate + NADP + H2O = tetrahydrofolate + CO2 + NADPH

1.5.1.7 saccharopine dehydrogenase (NAD, L-lysine-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NAD + H2O = L-lysine + 2-oxoglutarate + NADH2

1.5.1.8 saccharopine dehydrogenase (NADP, L-lysine-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NADP + H2O = L-lysine + 2-oxoglutarate + NADPH

1.5.1.9 saccharopine dehydrogenase (NAD, L-glutamate-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NAD + H2O = L-glutamate + 2-aminoadipate 6-semialdehyde + NADH2

1.5.1.10 saccharopine dehydrogenase (NADP, L-glutamate-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NADP + H2O = L-glutamate + 2-aminoadipate 6-semialdehyde + NADPH

1.5.1.12 1-pyrroline-5-carboxylate dehydrogenase 1-pyrroline-5-carboxylate + NAD + H2O = L-glutamate + NADH2

1.5.1.21 Delta1-piperideine-2-carboxylate reductase L-pipecolate + NADP = Delta1-piperideine-2-carboxylate + NADPH2

1.5.3.1 sarcosine oxidase sarcosine + H2O + O2 = glycine + formaldehyde + H2O2

1.5.99.1 sarcosine dehydrogenase sarcosine + acceptor + H2O = glycine + formaldehyde + reduced acceptor

1.5.99.2 dimethylglycine dehydrogenase N,N-dimethylglycine + acceptor + H2O = sarcosine + formaldehyde + reduced acceptor

1.5.99.3 L-pipecolate dehydrogenase L-pipecolate + acceptor = 2,3,4,5-tetrahydropyridine-2-carboxylate + reduced acceptor

1.5.99.8 proline dehydrogenase L-proline + acceptor + H2O = (S)-1-pyrroline-5-carboxylate + reduced acceptor

1.6.4.1 cystine reductase (NADH2) NADH2 + L-cystine = NAD + 2 L-cysteine

1.6.5.3 NADH2 dehydrogenase (ubiquinone) NADH2 + ubiquinone = NAD + ubiquinol

1.6.6.1 nitrate reductase (NADH2) NADH2 + nitrate = NAD + nitrite + H2O

1.6.6.2 nitrate reductase [NAD(P)H2] NAD(P)H2 + nitrate = NAD(P) + nitrite + H2O

1.6.6.3 nitrate reductase (NADPH2) NADPH2 + nitrate = NADP + nitrite + H2O

1.6.6.4 nitrite reductase [NAD(P)H2] 3 NAD(P)H2 + nitrite = 3 NAD(P) + NH4OH + H2O

1.6.6.8 GMP reductase NADPH2 + guanosine 5'-phosphate = NADP + inosine 5'-phosphate + NH3

1.7.3.3 urate oxidase urate + O2 = unidentified products

1.7.7.1 ferredoxin-nitrite reductase ammonia + 2 H2O + 6 oxidized ferredoxin = nitrite + 6 reduced ferredoxin + 7 H+

1.7.99.4 nitrate reductase nitrite + acceptor = nitrate + reduced acceptor

1.8.1.3 hypotaurine dehydrogenase hypotaurine + H2O + NAD = taurine + NADH2

1.8.1.4 dihydrolipoamide dehydrogenase dihydrolipoamide + NAD = lipoamide + NADH2

1.8.2.1 sulfite dehydrogenase sulfite + 2 ferricytochrome c + H2O = sulfate + 2 ferrocytochrome c

1.8.3.1 sulfite oxidase sulfite + O2 + H2O = sulfate + H2O2

1.8.7.1 sulfite reductase (ferredoxin) hydrogen sulfide + 3 oxidized ferredoxin + 3 H2O = sulfite + 3 reduced ferredoxin

1.8.99.1 sulfite reductase hydrogen sulfide + acceptor + 3 H2O = sulfite + reduced acceptor

1.9.3.1 cytochrome-c oxidase 4 ferrocytochrome c + O2 = 4 ferricytochrome c + 2 H2O

1.10.2.1 L-ascorbate-cytochrome-b5 reductase L-ascorbate + ferricytochrome-b5 = monodehydroascorbate + ferrocytochrome b5

1.10.2.2 ubiquinol-cytochrome-c reductase QH2 + 2 ferricytochrome c = Q + 2 ferrocytochrome c

1.10.3.3 L-ascorbate oxidase 2 L-ascorbate + O2 = 2 dehydroascorbate + 2 H2O

1.10.99.1 plastoquinol-plastocyanin reductase plastoquinol-1 + 2 oxidized plastocyanin = plastoquinone + 2 reduced plastocyanin

1.13.11.1 catechol 1,2-dioxygenase catechol + O2 = cis,cis-muconate

1.13.11.2 catechol 2,3-dioxygenase catechol + O2 = 2-hydroxymuconate semialdehyde

1.13.11.5 homogentisate 1,2-dioxygenase homogentisate + O2 = 4-maleylacetoacetate

1.13.11.6 3-hydroxyanthranilate 3,4-dioxygenase 3-hydroxyanthranilate + O2 = 2-amino-3-carboxymuconate semialdehyde

1.13.11.11 tryptophan 2,3-dioxygenase L-tryptophan + O2 = L-formylkynurenine

1.13.11.20 cysteine dioxygenase L-cysteine + O2 = 3-sulfinoalanine

1.13.11.21 beta-carotene 15,15'-dioxygenase beta-carotene + O2 = 2 retinal

1.13.11.27 4-hydroxyphenylpyruvate dioxygenase 4-hydroxyphenylpyruvate + O2 = homogentisate + CO2

1.13.99.1 myo-inositol oxygenase myo-inositol + O2 = D-glucuronate + H2O

1.14.11.1 gamma-butyrobetaine dioxygenase 4-trimethylammoniobutanoate + 2-oxoglutarate + O2 = 3-hydroxy-4-trimethylammoniobutanoate + succinate + CO2

1.14.11.2 procollagen-proline dioxygenase procollagen L-proline + 2-oxoglutarate + O2 = procollagen trans-4-hydroxy-L-proline + succinate + CO2

1.14.11.8 trimethyllysine dioxygenase N6,N6,N6-trimethyl-L-lysine + 2-oxoglutarate + O2 = 3-hydroxy-N6,N6,N6-trimethyl-L-lysine + succinate + CO2

1.14.12.1 anthranilate 1,2-dioxygenase (deaminating, decarboxylating) anthranilate + NAD(P)H2 + O2 + 2 H2O = catechol + CO2 + NAD(P) + NH3

1.14.12.10 benzoate 1,2-dioxygenase benzoate + NADH2 + O2 = catechol + CO2 + NAD

1.14.13.5 imidazoleacetate 4-monooxygenase 4-imidazoleacetate + NADH2 + O2 = 5-hydroxy-4-imidazoleacetate + NAD + H2O

1.14.13.9 kynurenine 3-monooxygenase L-kynurenine + NADPH2 + O2 = 3-hydroxy-L-kynurenine + NADP + H2O

1.14.13.12 benzoate 4-monooxygenase benzoate + NADPH2 + O2 = 4-hydroxybenzoate + NADP + H2O

1.14.13.39 nitric-oxide synthase L-arginine + n NADPH2 + m O2 = citrulline + nitric oxide + n NADP

1.14.16.1 phenylalanine 4-monooxygenase L-phenylalanine + tetrahydrobiopterin + O2 = L-tyrosine + dihydrobiopterin + H2O

1.14.16.2 tyrosine 3-monooxygenase L-tyrosine + tetrahydrobiopterin + O2 = 3,4-dihydroxy-L-phenylalanine + dihydrobiopterin + H2O

1.14.16.4 tryptophan 5-monooxygenase L-tryptophan + tetrahydrobiopterin + O2 = 5-hydroxy-L-tryptophan + dihydrobiopterin + H2O

1.14.17.1 dopamine beta-monooxygenase 3,4-dihydroxyphenethylamine + ascorbate + O2 = noradrenaline + dehydroascorbate + H2O

1.14.18.1 monophenol monooxygenase L-tyrosine + L-dopa + O2 = L-dopa + dopaquinone + H2O

1.14.19.1 stearoyl-CoA 9-desaturase stearoyl-CoA + AH2 + O2 = oleoyl-CoA + A + 2 H2O

1.14.19.3 linoleoyl-CoA desaturase linoleoyl-CoA + AH2 + O2 = gamma-linolenoyl-CoA + A + 2 H2O

1.14.99.1 prostaglandin-endoperoxide synthase arachidonate + AH2 + 2 O2 = prostaglandin H2 + A + H2O

1.14.99.7 squalene monooxygenase squalene + AH2 + O2 = (S)-squalene-2,3-epoxide + A + H2O

1.17.4.1 ribonucleoside-diphosphate reductase 2'-deoxyribonucleoside diphosphate + oxidized thioredoxin + H2O = ribonucleoside diphosphate + reduced thioredoxin

1.18.6.1 nitrogenase 3 reduced ferredoxin + 6 H+ + N2 + n ATP = 3 oxidized ferredoxin + 2 NH3 + n ADP + n phosphate

2.1.1.1 nicotinamide N-methyltransferase S-adenosyl-L-methionine + nicotinamide = S-adenosyl-L-homocysteine + 1-methylnicotinamide

2.1.1.2 guanidinoacetate N-methyltransferase S-adenosyl-L-methionine + guanidinoacetate = S-adenosyl-L-homocysteine + creatine

2.1.1.3 thetin-homocysteine S-methyltransferase dimethylsulfonioacetate + L-homocysteine = S-methylthioglycolate + L-methionine

2.1.1.4 acetylserotonin O-methyltransferase S-adenosyl-L-methionine + N-acetylserotonin = S-adenosyl-L-homocysteine + melatonin

2.1.1.5 betaine-homocysteine S-methyltransferase trimethylammonioacetate + L-homocysteine = dimethylglycine + L-methionine

2.1.1.6 catechol O-methyltransferase S-adenosyl-L-methionine + a catechol = S-adenosyl-L-homocysteine + a guaiacol

2.1.1.10 homocysteine S-methyltransferase S-adenosyl-L-methionine + L-homocysteine = S-adenosyl-L-homocysteine + L-methionine

2.1.1.13 5-methyltetrahydrofolate-homocysteine S-methyltransferase 5-methyltetrahydrofolate + L-homocysteine = tetrahydrofolate + L-methionine

2.1.1.14 5-methyltetrahydropteroyltriglutamate-homocysteine S-methyltransferase 5-methyltetrahydropteroyltri-L-glutamate + L-homocysteine = tetrahydropteroyltri-L-glutamate + L-methionine

2.1.1.17 phosphatidylethanolamine N-methyltransferase S-adenosyl-L-methionine + phosphatidylethanolamine = S-adenosyl-L-homocysteine + phosphatidyl-N-methylethanolamine

2.1.1.20 glycine N-methyltransferase S-adenosyl-L-methionine + glycine = S-adenosyl-L-homocysteine + sarcosine

2.1.1.28 phenylethanolamine N-methyltransferase S-adenosyl-L-methionine + phenylethanolamine = S-adenosyl-L-homocysteine + N-methylphenylethanolamine

2.1.1.43 histone-lysine N-methyltransferase S-adenosyl-L-methionine + histone L-lysine = S-adenosyl-L-homocysteine + histone N6-methyl-L-lysine

2.1.1.45 thymidylate synthase 5,10-methylenetetrahydrofolate + dUMP = dihydrofolate + dTMP

2.1.2.1 glycine hydroxymethyltransferase 5,10-methylenetetrahydrofolate + glycine + H2O = tetrahydrofolate + L-serine

2.1.2.2 phosphoribosylglycinamide formyltransferase 10-formyltetrahydrofolate + N1-(5-phospho-D-ribosyl)glycinamide = tetrahydrofolate + N2-formyl-N1-(5-phospho-D-ribosyl)glycinamide

2.1.2.3 phosphoribosylaminoimidazolecarboxamide formyltransferase 10-formyltetrahydrofolate + 5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxamide = tetrahydrofolate + 5-formamido-1-(5-phospho-D-ribosyl)imidazole-4-carboxamide

2.1.2.5 glutamate formimidoyltransferase 5-formimidoyltetrahydrofolate + L-glutamate = tetrahydrofolate + N-formimidoyl-L-glutamate

2.1.2.10 aminomethyltransferase S-aminomethyldihydrolipoylprotein + (6S)-tetrahydrofolate = dihydrolipoylprotein + (6R)-5,10-methylenetetrahydrofolate + NH3

2.1.3.1 methylmalonyl-CoA carboxyltransferase (S)-2-methyl-3-oxopropanoyl-CoA + pyruvate = propanoyl-CoA + oxaloacetate

2.1.3.2 aspartate carbamoyltransferase carbamoyl phosphate + L-aspartate = phosphate + N-carbamoyl-L-aspartate

2.1.3.3 ornithine carbamoyltransferase carbamoyl phosphate + L-ornithine = phosphate + L-citrulline

2.1.4.1 glycine amidinotransferase L-arginine + glycine = L-ornithine + guanidinoacetate

2.2.1.1 transketolase sedoheptulose 7-phosphate + D-glyceraldehyde 3-phosphate = D-ribose 5-phosphate + D-xylulose 5-phosphate

2.2.1.2 transaldolase sedoheptulose 7-phosphate + D-glyceraldehyde 3-phosphate = D-erythrose 4-phosphate + D-fructose 6-phosphate

2.3.1.1 amino-acid N-acetyltransferase acetyl-CoA + L-glutamate = CoA + N-acetyl-L-glutamate

2.3.1.4 glucosamine-phosphate N-acetyltransferase acetyl-CoA + D-glucosamine 6-phosphate = CoA + N-acetyl-D-glucosamine 6-phosphate

2.3.1.5 arylamine N-acetyltransferase acetyl-CoA + an arylamine = CoA + an N-acetylarylamine

2.3.1.6 choline O-acetyltransferase acetyl-CoA + choline = CoA + O-acetylcholine

2.3.1.7 carnitine O-acetyltransferase acetyl-CoA + carnitine = CoA + O-acetylcarnitine

2.3.1.8 phosphate acetyltransferase acetyl-CoA + phosphate = CoA + acetyl phosphate

2.3.1.9 acetyl-CoA C-acetyltransferase acetyl-CoA + acetyl-CoA = CoA + acetoacetyl-CoA

2.3.1.12 dihydrolipoamide S-acetyltransferase acetyl-CoA + dihydrolipoamide = CoA + S-acetyldihydrolipoamide

2.3.1.15 glycerol-3-phosphate O-acyltransferase acyl-CoA + sn-glycerol 3-phosphate = CoA + 1-acyl-sn-glycerol 3-phosphate

2.3.1.16 acetyl-CoA C-acyltransferase acyl-CoA + acetyl-CoA = CoA + 3-oxoacyl-CoA

2.3.1.20 diacylglycerol O-acyltransferase acyl-CoA + 1,2-diacylglycerol = CoA + triacylglycerol

2.3.1.21 carnitine O-palmitoyltransferase palmitoyl-CoA + L-carnitine = CoA + L-palmitoylcarnitine

2.3.1.23 1-acylglycerophosphocholine O-acyltransferase acyl-CoA + 1-acyl-sn-glycero-3-phosphocholine = CoA + 1,2-diacyl-sn-glycero-3-phosphocholine

2.3.1.24 sphingosine N-acyltransferase acyl-CoA + sphingosine = CoA + N-acylsphingosine

2.3.1.30 serine O-acetyltransferase acetyl-CoA + L-serine = CoA + O-acetyl-L-serine

2.3.1.32 lysine N-acetyltransferase acetyl phosphate + L-lysine = phosphate + N6-acetyl-L-lysine

2.3.1.35 glutamate N-acetyltransferase N2-acetyl-L-ornithine + L-glutamate = L-ornithine + N-acetyl-L-glutamate

2.3.1.37 5-aminolevulinate synthase succinyl-CoA + glycine = 5-aminolevulinate + CoA + CO2

2.3.1.38 [acyl-carrier-protein] S-acetyltransferase acetyl-CoA + [acyl-carrier protein] = CoA + acetyl-[acyl-carrier protein]

2.3.1.39 [acyl-carrier-protein] S-malonyltransferase malonyl-CoA + [acyl-carrier protein] = CoA + malonyl-[acyl-carrier protein]

2.3.1.41 3-oxoacyl-[acyl-carrier-protein] synthase acyl-[acyl-carrier protein] + malonyl-[acyl-carrier protein] = 3-oxoacyl-[acyl-carrier protein] + CO2 + [acyl-carrier protein]

2.3.1.46 homoserine O-succinyltransferase succinyl-CoA + L-homoserine = CoA + O-succinyl-L-homoserine

2.3.1.50 serine C-palmitoyltransferase palmitoyl-CoA + L-serine = CoA + 3-dehydro-D-sphinganine + CO2

2.3.1.51 1-acylglycerol-3-phosphate O-acyltransferase acyl-CoA + 1-acyl-sn-glycerol 3-phosphate = CoA + 1,2-diacyl-sn-glycerol 3-phosphate

2.3.1.76 retinol O-fatty-acyltransferase acyl-CoA + retinol = CoA + retinyl ester

2.4.1.1 phosphorylase (1,4-alpha-D-glucosyl)n + phosphate = (1,4-alpha-D-glucosyl)n-1 + alpha-D-glucose 1-phosphate

2.4.1.4 amylosucrase sucrose + (1,4-alpha-D-glucosyl)n = D-fructose + (1,4-alpha-D-glucosyl)n+1

2.4.1.9 inulosucrase sucrose + (2,1-beta-D-fructosyl)n = glucose + (2,1-beta-D-fructosyl)n+1

2.4.1.11 glycogen(starch) synthase UDPglucose + (1,4-alpha-D-glucosyl)n = UDP + (1,4-alpha-D-glucosyl)n+1

2.4.1.13 sucrose synthase UDPglucose + D-fructose = UDP + sucrose

2.4.1.17 glucuronosyltransferase UDPglucuronate + acceptor = UDP + acceptor beta-D-glucuronoside

2.4.1.21 starch synthase ADPglucose + (1,4-alpha-D-glucosyl)n = ADP + (1,4-alpha-D-glucosyl)n+1

2.4.1.22 lactose synthase UDPgalactose + D-glucose = UDP + lactose

2.4.1.23 sphingosine beta-galactosyltransferase UDPgalactose + sphingosine = UDP + psychosine

2.4.1.29 cellulose synthase (GDP-forming) GDPglucose + (1,4-beta-D-glucosyl)n = GDP + (1,4-beta-D-glucosyl)n+1

2.4.1.32 glucomannan 4-beta-mannosyltransferase GDPmannose + (glucomannan)n = GDP + (glucomannan)n+1

2.4.1.41 polypeptide N-acetylgalactosaminyltransferase UDP-N-acetyl-D-galactosamine + polypeptide = UDP + N-acetyl-D-galactosaminyl-polypeptide

2.4.1.62 ganglioside galactosyltransferase UDPgalactose + N-acetyl-D-galactosaminyl-(N-acetylneuraminyl)-D-galactosyl-1,4-beta-D-glucosyl-N-acylsphingosine = UDP + D-galactosyl-1,3-beta-N-acetyl-D-galactosaminyl-(N-acetylneuraminyl)-D-galactosyl-D-glucosyl-N-acylsphingosine

2.4.1.68 glycoprotein 6-alpha-L-fucosyltransferase GDP-L-fucose + N4-(N-acetyl-beta-D-glucosaminyl-1,2-alpha-D-mannosyl-1,3-(R1-alpha-1,6)-beta-D-mannosyl-beta-N-acetyl-1,4-D-glucosaminyl-1,4-N-acetyl-D-glucosaminyl)asparagine = GDP + N4-(N-acetyl-beta-D-glucosaminyl-1,2-alpha-D-mannosyl-1,3-(R1-alpha-1,6)-beta-D-mannosyl-1,4-beta-N-acetyl-D-glucosaminyl-1,4-(alpha-L-fucosyl-1,6)-N-acetyl-D-glucosaminyl)-asparagine

2.4.1.69 galactoside 2-alpha-L-fucosyltransferase GDP-L-fucose + beta-D-galactosyl-R = GDP + alpha-L-fucosyl-1,2-beta-D-galactosyl-R

2.4.2.1 purine-nucleoside phosphorylase purine nucleoside + phosphate = purine + alpha-D-ribose 1-phosphate

2.4.2.2 pyrimidine-nucleoside phosphorylase pyrimidine nucleoside + phosphate = pyrimidine + alpha-D-ribose 1-phosphate

2.4.2.4 thymidine phosphorylase thymidine + phosphate = thymine + 2-deoxy-D-ribose 1-phosphate

2.4.2.8 hypoxanthine phosphoribosyltransferase IMP + diphosphate = hypoxanthine + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.9 uracil phosphoribosyltransferase UMP + diphosphate = uracil + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.10 orotate phosphoribosyltransferase orotidine 5'-phosphate + diphosphate = orotate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.11 nicotinate phosphoribosyltransferase nicotinate D-ribonucleotide + diphosphate = nicotinate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.14 amidophosphoribosyltransferase 5-phospho-beta-D-ribosylamine + diphosphate + L-glutamate = L-glutamine + 5-phospho-alpha-D-ribose 1-diphosphate + H2O

2.4.2.15 guanosine phosphorylase guanosine + phosphate = guanine + D-ribose 1-phosphate

2.4.2.17 ATP phosphoribosyltransferase 1-(5-phospho-D-ribosyl)-ATP + diphosphate = ATP + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.18 anthranilate phosphoribosyltransferase N-(5-phospho-D-ribosyl)-anthranilate + diphosphate = anthranilate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.19 nicotinate-nucleotide diphosphorylase (carboxylating) nicotinate D-ribonucleotide + diphosphate + CO2 = pyridine-2,3-dicarboxylate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.99.1 beta-galactoside alpha-2,6-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosamine = CMP + alpha-N-acetylneuraminyl-2,6-beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosamine

2.4.99.2 monosialoganglioside sialyltransferase CMP-N-acetylneuraminate + D-galactosyl-N-acetyl-D-galactosaminyl-

(N-acetylneuraminyl)-D-galactosyl-D-glucosylceramide = CMP + N-acetylneuraminyl-D-galactosyl-N-acetyl-D-galactosaminyl-(N-acetylneuraminyl)-D-galactosyl-D-glucosylceramide

2.4.99.3 alpha-N-acetylgalactosaminide alpha-2,6-sialyltransferase CMP-N-acetylneuraminate + glycano-1,3-(N-acetyl-alpha-D-galactosaminyl)-glycoprotein = CMP + glycano-(2,6-alpha-N-acetylneuraminyl)-(N-acetyl-D-galactosaminyl)-glycoprotein

2.4.99.4 beta-galactoside alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,3-N-acetyl-alpha-D-galactosaminyl-R = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,3-N-acetyl-alpha-D-galactosaminyl-R

2.4.99.5 galactosyldiacylglycerol alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + 1,2-diacyl-3-beta-D-galactosyl-sn-glycerol = CMP + 1,2-diacyl-3-[3-(alpha-D-N-acetylneuraminyl)-beta-D-galactosyl]-sn-glycerol

2.4.99.6 N-acetyllactosaminide alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-N-acetyl-D-glucosaminyl-glycoprotein = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,4-N-acetyl-D-glucosaminyl-glycoprotein

2.4.99.7 (alpha-N-acetylneuraminyl-2,3-alpha-galactosyl-1,3)-N-acetyl-galactosaminide alpha-2,6-sialyltransferase CMP-N-acetylneuraminate + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,3-N-acetyl-D-galactosaminyl-R = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactose-1,3-(alpha-N-acetylneuraminyl-2,6)-N-acetyl-D-galactosaminyl-R

2.4.99.8 alpha-N-acetylneuraminate alpha-2,8-sialyltransferase CMP-N-acetylneuraminate + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-R = CMP + alpha-N-acetylneuraminyl-2,8-alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-R

2.4.99.9 lactosylceramide alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-beta-D-

glucosylceramide = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,4-beta-D-glucosyl-ceramide

2.4.99.10 neolactotetraosylceramide alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosaminyl-1,3-beta-D-galactosyl-1,4-D-glucosylceramide = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosaminyl-1,3-beta-D-galactosyl-1,4-D-glucosylceramide

2.4.99.11 lactosylceramide alpha-2,6-N-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-beta-D-glucosylceramide = CMP + alpha-N-acetylneuraminyl-2,6-beta-galactosyl-1,4-beta-D-glucosylceramide

2.5.1.1 dimethylallyltranstransferase dimethylallyl diphosphate + isopentenyl diphosphate = diphosphate + geranyl diphosphate

2.5.1.6 methionine adenosyltransferase ATP + L-methionine + H2O = phosphate + diphosphate + S-adenosyl-L-methionine

2.5.1.10 geranyltranstransferase geranyl diphosphate + isopentenyl diphosphate = diphosphate + trans,trans-farnesyl diphosphate

2.5.1.16 spermidine synthase S-adenosylmethioninamine + putrescine = 5'-methylthioadenosine + spermidine

2.5.1.19 3-phosphoshikimate 1-carboxyvinyltransferase phosphoenolpyruvate + 3-phosphoshikimate = phosphate + 5-O-(1-carboxyvinyl)-3-phosphoshikimate

2.5.1.21 farnesyl-diphosphate farnesyltransferase 2 farnesyl diphosphate = diphosphate + presqualene diphosphate

2.5.1.22 spermine synthase S-adenosylmethioninamine + spermidine = 5'-methylthioadenosine + spermine

2.5.1.29 farnesyltranstransferase trans,trans-farnesyl diphosphate + isopentenyl diphosphate = diphosphate + geranylgeranyl diphosphate

2.5.1.32 geranylgeranyl-diphosphate geranylgeranyltransferase 2 geranylgeranyl diphosphate = diphosphate + prephytoene diphosphate

2.6.1.1 aspartate transaminase L-aspartate + 2-oxoglutarate = oxaloacetate + L-glutamate

2.6.1.2 alanine transaminase L-alanine + 2-oxoglutarate = pyruvate + L-glutamate

2.6.1.4 glycine transaminase glycine + 2-oxoglutarate = glyoxylate + L-glutamate

2.6.1.5 tyrosine transaminase L-tyrosine + 2-oxoglutarate = 4-hydroxyphenylpyruvate + L-glutamate

2.6.1.6 leucine transaminase L-leucine + 2-oxoglutarate = 4-methyl-2-oxopentanoate + L-glutamate

2.6.1.9 histidinol-phosphate transaminase L-histidinol phosphate + 2-oxoglutarate = 3-(imidazol-4-yl)-2-oxopropyl phosphate + L-glutamate

2.6.1.11 acetylornithine transaminase N2-acetyl-L-ornithine + 2-oxoglutarate = N-acetyl-L-glutamate 5-semialdehyde + L-glutamate

2.6.1.13 ornithine-oxo-acid transaminase L-ornithine + a 2-oxo acid = L-glutamate 5-semialdehyde + an L-amino acid

2.6.1.16 glutamine-fructose-6-phosphate transaminase (isomerizing) L-glutamine + D-fructose 6-phosphate = L-glutamate + D-glucosamine 6-phosphate

2.6.1.17 succinyldiaminopimelate transaminase N-succinyl-L-2,6-diaminoheptanedioate + 2-oxoglutarate = N-succinyl-2-L-amino-6-oxoheptanedioate + L-glutamate

2.6.1.18 beta-alanine-pyruvate transaminase L-alanine + 3-oxopropanoate = pyruvate + beta-alanine

2.6.1.19 4-aminobutyrate transaminase 4-aminobutanoate + 2-oxoglutarate = succinate semialdehyde + L-glutamate

2.6.1.22 (S)-3-amino-2-methylpropionate transaminase (S)-3-amino-2-methylpropanoate + 2-oxoglutarate = 2-methyl-3-oxopropanoate + L-glutamate

2.6.1.23 4-hydroxyglutamate transaminase 4-hydroxy-L-glutamate + 2-oxoglutarate = 4-hydroxy-2-oxoglutarate + L-glutamate

2.6.1.27 tryptophan transaminase L-tryptophan + 2-oxoglutarate = indolepyruvate + L-glutamate

2.6.1.32 valine-3-methyl-2-oxovalerate transaminase L-valine + (S)-3-methyl-2-oxopentanoate = 3-methyl-2-oxobutanoate + L-isoleucine

2.6.1.36 L-lysine 6-transaminase L-lysine + 2-oxoglutarate = 2-aminoadipate 6-semialdehyde + L-glutamate

2.6.1.39 2-aminoadipate transaminase L-2-aminoadipate + 2-oxoglutarate = 2-oxoadipate + L-glutamate

2.6.1.42 branched-chain-amino-acid transaminase L-leucine + 2-oxoglutarate = 4-methyl-2-oxopentanoate + L-glutamate

2.6.1.51 serine-pyruvate transaminase L-serine + pyruvate = 3-hydroxypyruvate + L-alanine

2.6.1.52 phosphoserine transaminase O-phospho-L-serine + 2-oxoglutarate = 3-phosphonooxypyruvate + L-glutamate

2.6.1.65 N6-acetyl-beta-lysine transaminase 6-acetamido-3-aminohexanoate + 2-oxoglutarate = 6-acetamido-3-oxohexanoate + L-glutamate

2.6.1.66 valine-pyruvate transaminase L-valine + pyruvate = 3-methyl-2-oxobutanoate + L-alanine

2.7.1.1 hexokinase ATP + D-hexose = ADP + D-hexose 6-phosphate

2.7.1.2 glucokinase ATP + D-glucose = ADP + D-glucose 6-phosphate

2.7.1.3 ketohexokinase ATP + D-fructose = ADP + D-fructose 1-phosphate

2.7.1.4 fructokinase ATP + D-fructose = ADP + D-fructose 6-phosphate

2.7.1.6 galactokinase ATP + D-galactose = ADP + alpha-D-galactose 1-phosphate

2.7.1.7 mannokinase ATP + D-mannose = ADP + D-mannose 6-phosphate

2.7.1.11 6-phosphofructokinase ATP + D-fructose 6-phosphate = ADP + D-fructose 1,6-bisphosphate

2.7.1.15 ribokinase ATP + D-ribose = ADP + D-ribose 5-phosphate

2.7.1.16 ribulokinase ATP + L(or D)-ribulose = ADP + L(or D)-ribulose 5-phosphate

2.7.1.17 xylulokinase ATP + D-xylulose = ADP + D-xylulose 5-phosphate

2.7.1.19 phosphoribulokinase ATP + D-ribulose 5-phosphate = ADP + D-ribulose 1,5-bisphosphate

2.7.1.23 NAD kinase ATP + NAD = ADP + NADPH2

2.7.1.24 dephospho-CoA kinase ATP + dephospho-CoA = ADP + CoA

2.7.1.25 adenylyl-sulfate kinase ATP + adenylyl sulfate = ADP + 3'-phosphoadenylyl sulfate

2.7.1.28 triokinase ATP + D-glyceraldehyde = ADP + D-glyceraldehyde 3-phosphate

2.7.1.30 glycerol kinase ATP + glycerol = ADP + sn-glycerol 3-phosphate

2.7.1.31 glycerate kinase ATP + (R)-glycerate = ADP + 3-phospho-(R)-glycerate

2.7.1.32 choline kinase ATP + choline = ADP + O-phosphocholine

2.7.1.33 pantothenate kinase ATP + pantothenate = ADP + D-4'-phosphopantothenate

2.7.1.34 pantetheine kinase ATP + pantetheine = ADP + pantetheine 4'-phosphate

2.7.1.36 mevalonate kinase ATP + (R)-mevalonate = ADP + (R)-5-phosphomevalonate

2.7.1.39 homoserine kinase ATP + L-homoserine = ADP + O-phospho-L-homoserine

2.7.1.40 pyruvate kinase ATP + pyruvate = ADP + phosphoenolpyruvate

2.7.1.47 D-ribulokinase ATP + D-ribulose = ADP + D-ribulose 5-phosphate

2.7.1.53 L-xylulokinase ATP + L-xylulose = ADP + L-xylulose 5-phosphate

2.7.1.60 N-acylmannosamine kinase ATP + N-acyl-D-mannosamine = ADP + N-acyl-D-mannosamine 6-phosphate

2.7.1.64 inositol 1-kinase ATP + myo-inositol = ADP + 1L-myo-inositol 1-phosphate

2.7.1.71 shikimate kinase ATP + shikimate = ADP + shikimate 3-phosphate

2.7.1.80 diphosphate-serine phosphotransferase diphosphate + L-serine = phosphate + O-phospho-L-serine

2.7.1.107 diacylglycerol kinase ATP + 1,2-diacylglycerol = ADP + 1,2-diacyl-sn-glycerol 3-phosphate

2.7.2.1 acetate kinase ATP + acetate = ADP + acetyl phosphate

2.7.2.3 phosphoglycerate kinase ATP + 3-phospho-D-glycerate = ADP + 3-phospho-D-glyceroyl phosphate

2.7.2.4 aspartate kinase ATP + L-aspartate = ADP + 4-phospho-L-aspartate

2.7.2.6 formate kinase ATP + formate = ADP + formyl phosphate

2.7.2.11 glutamate 5-kinase ATP + L-glutamate = ADP + L-glutamate 5-phosphate

2.7.3.2 creatine kinase ATP + creatine = ADP + phosphocreatine

2.7.4.2 phosphomevalonate kinase ATP + (R)-5-phosphomevalonate = ADP + (R)-5-diphosphomevalonate

2.7.4.3 adenylate kinase ATP + AMP = ADP + ADP

2.7.4.4 nucleoside-phosphate kinase ATP + nucleoside phosphate = ADP + nucleoside diphosphate

2.7.4.6 nucleoside-diphosphate kinase ATP + nucleoside diphosphate = ADP + nucleoside triphosphate

2.7.4.9 dTMP kinase ATP + dTMP = ADP + dTDP

2.7.4.14 cytidylate kinase ATP + (d)CMP = ADP + (d)CDP

2.7.6.1 ribose-phosphate diphosphokinase ATP + D-ribose 5-phosphate = AMP + 5-phospho-alpha-D-ribose 1-diphosphate

2.7.7.3 pantetheine-phosphate adenylyltransferase ATP + pantetheine 4'-phosphate = diphosphate + dephospho-CoA

2.7.7.4 sulfate adenylyltransferase ATP + sulfate = diphosphate + adenylylsulfate

2.7.7.6 DNA-directed RNA polymerase nucleoside triphosphate + RNAn = diphosphate + RNAn+1

2.7.7.7 DNA-directed DNA polymerase deoxynucleoside triphosphate + DNAn = diphosphate + DNAn+1

2.7.7.9 UTP-glucose-1-phosphate uridylyltransferase UTP + alpha-D-glucose 1-phosphate = diphosphate + UDPglucose

2.7.7.10 UTP-hexose-1-phosphate uridylyltransferase UTP + alpha-D-galactose 1-phosphate = diphosphate + UDP galactose

2.7.7.12 UDPglucose-hexose-1-phosphate uridylyltransferase UDPglucose + alpha-D-galactose 1-phosphate = alpha-D-glucose 1-phosphate + UDPgalactose

2.7.7.13 mannose-1-phosphate guanylyltransferase GTP + alpha-D-mannose 1-phosphate = diphosphate + GDPmannose

2.7.7.14 ethanolamine-phosphate cytidylyltransferase CTP + ethanolamine phosphate = diphosphate + CDPethanolamine

2.7.7.15 choline-phosphate cytidylyltransferase CTP + choline phosphate = diphosphate + CDPcholine

2.7.7.18 nicotinate-nucleotide adenylyltransferase ATP + nicotinate ribonucleotide = diphosphate + deamido-NAD

2.7.7.22 mannose-1-phosphate guanylyltransferase (GDP) GDP + D-mannose 1-phosphate = phosphate + GDP mannose

2.7.7.23 UDP-N-acetylglucosamine diphosphorylase UTP + N-acetyl-alpha-D-glucosamine 1-phosphate = diphosphate + UDP-N-acetyl-D-glucosamine

2.7.7.24 glucose-1-phosphate thymidylyltransferase dTTP + alpha-D-glucose 1-phosphate = diphosphate + dTDP glucose

2.7.7.27 glucose-1-phosphate adenylyltransferase ATP + alpha-D-glucose 1-phosphate = diphosphate + ADP glucose

2.7.7.34 glucose-1-phosphate guanylyltransferase GTP + alpha-D-glucose 1-phosphate = diphosphate + GDP glucose

2.7.7.41 phosphatidate cytidylyltransferase CTP + phosphatidate = diphosphate + CDP diacylglycerol

2.7.7.43 N-acylneuraminate cytidylyltransferase CTP + N-acylneuraminate = diphosphate + CMP-N-acylneuraminate

2.7.8.1 ethanolaminephosphotransferase CDP ethanolamine + 1,2-diacylglycerol = CMP + a phosphatidyl-ethanolamine

2.7.8.2 diacylglycerol cholinephosphotransferase CDP choline + 1,2-diacylglycerol = CMP + a phosphatidylcholine

2.7.8.3 ceramide cholinephosphotransferase CDP choline + N-acylsphingosine = CMP + sphingomyelin

2.7.8.5 CDPdiacylglycerol-glycerol-3-phosphate 3-phosphatidyltransferase CDP diacylglycerol + sn-glycerol 3-phosphate = CMP + 3(3-sn-phosphatidyl)-sn-glycerol 1-phosphate

2.7.8.8 CDPdiacylglycerol-serine O-phosphatidyltransferase CDP diacylglycerol + L-serine = CMP + 3-O-sn-phosphatidyl-L-serine

2.7.8.11 CDP diacylglycerol-inositol 3-phosphatidyltransferase CDP diacylglycerol + myo-inositol = CMP + phosphatidyl-1D-myo-inositol

2.7.8.13 phospho-N-acetylmuramoyl-pentapeptide-transferase UDPMurAc(oyl-L-Ala-D-gamma-Glu-L-Lys-D-Ala-D-Ala) + undecaprenyl phosphate = UMP + MurAc(oyl-L-Ala-D-gamma-Glu-L-Lys-D-Ala-D-Ala)-diphospho-undecaprenol

2.8.3.1 propionate CoA-transferase acetyl-CoA + propanoate = acetate + propanoyl-CoA

2.8.3.5 3-oxoacid CoA-transferase succinyl-CoA + a 3-oxo acid = succinate + a 3-oxoacyl-CoA

2.8.3.6 3-oxoadipate CoA-transferase succinyl-CoA + 3-oxoadipate = succinate + 3-oxoadipyl-CoA

3.1.1.3 triacylglycerol lipase triacylglycerol + H2O = diacylglycerol + a carboxylate

3.1.1.4 phospholipase A2 phosphatidylcholine + H2O = 1-acylglycerophosphocholine + a carboxylate

3.1.1.5 lysophospholipase 2-lysophosphatidylcholine + H2O = glycerophosphocholine + a carboxylate

3.1.1.7 acetylcholinesterase acetylcholine + H2O = choline + acetate

3.1.1.17 gluconolactonase D-glucono-1,5-lactone + H2O = D-gluconate

3.1.1.21 retinyl-palmitate esterase retinyl palmitate + H2O = retinol + palmitate

3.1.1.24 3-oxoadipate enol-lactonase 3-oxoadipate enol-lactone + H2O = 3-oxoadipate

3.1.1.28 acylcarnitine hydrolase O-acylcarnitine + H2O = a fatty acid + L-carnitine

3.1.1.31 6-phosphogluconolactonase 6-phospho-D-glucono-1,5-lactone + H2O = 6-phospho-D-gluconate

3.1.1.32 phospholipase A1 phosphatidylcholine + H2O = 2-acylglycerophosphocholine + a carboxylate

3.1.2.1 acetyl-CoA hydrolase acetyl-CoA + H2O = CoA + acetate

3.1.2.3 succinyl-CoA hydrolase succinyl-CoA + H2O = CoA + succinate

3.1.2.4 3-hydroxyisobutyryl-CoA hydrolase 3-hydroxy-2-methylpropanoyl-CoA + H2O = CoA + 3-hydroxy-2-methylpropanoate

3.1.2.11 acetoacetyl-CoA hydrolase acetoacetyl-CoA + H2O = CoA + acetoacetate

3.1.3.2 acid phosphatase An orthophosphoric monoester + H2O = an alcohol + phosphate

3.1.3.3 phosphoserine phosphatase L(or D)-O-phosphoserine + H2O = L(or D)-serine + phosphate

3.1.3.4 phosphatidate phosphatase A 3-sn-phosphatidate + H2O = a 1,2-diacyl-sn-glycerol + phosphate

3.1.3.5 5'-nucleotidase A 5'-ribonucleotide + H2O = a ribonucleoside + phosphate

3.1.3.9 glucose-6-phosphatase D-glucose 6-phosphate + H2O = D-glucose + phosphate

3.1.3.10 glucose-1-phosphatase D-glucose 1-phosphate + H2O = D-glucose + phosphate

3.1.3.11 fructose-bisphosphatase D-fructose 1,6-bisphosphate + H2O = D-fructose 6-phosphate + orthophosphate

3.1.3.15 histidinol-phosphatase L-histidinol phosphate + H2O = L-histidinol + phosphate

3.1.3.25 inositol-1(or 4)-monophosphatase myo-inositol 1-phosphate + H2O = myo-inositol + phosphate

3.1.3.27 phosphatidylglycerophosphatase phosphatidylglycerophosphate + H2O = phosphatidylglycerol + phosphate

3.1.3.29 N-acylneuraminate-9-phosphatase N-acylneuraminate 9-phosphate + H2O = N-acylneuraminate + phosphate

3.1.3.31 nucleotidase A nucleotide + H2O = a nucleoside + phosphate

3.1.4.2 glycerophosphocholine phosphodiesterase sn-glycero-3-phosphocholine + H2O = choline + sn-glycerol 3-phosphate

3.1.4.3 phospholipase C A phosphatidylcholine + H2O = 1,2-diacylglycerol + choline phosphate

3.1.4.4 phospholipase D A phosphatidylcholine + H2O = choline + a phosphatidate

3.1.4.10 1-phosphatidylinositol phosphodiesterase 1-phosphatidyl-D-myo-inositol = D-myo-inositol 1,2-cyclic phosphate + diacylglycerol

3.1.4.12 sphingomyelin phosphodiesterase sphingomyelin + H2O = N-acylsphingosine + choline phosphate

3.2.1.3 glucan 1,4-alpha-glucosidase Hydrolysis of terminal 1,4-linked alpha-D-glucose residues successively from non-reducing ends of the chains with release of beta-D-glucose

3.2.1.21 beta-glucosidase Hydrolysis of terminal, non-reducing beta-D-glucose residues with release of beta-D-glucose

3.2.1.23 beta-galactosidase Hydrolysis of terminal non-reducing beta-D-galactose residues in beta-D-galactosides

3.2.1.26 beta-fructofuranosidase Hydrolysis of terminal non-reducing beta-D-fructofuranoside residues in beta-D-fructofuranosides

3.2.1.45 glucosylceramidase D-glucosyl-N-acylspingosine + H2O = D-glucose + N-acylsphingosine

3.2.1.46 galactosylceramidase D-galactosyl-N-acylsphingosine + H2O = D-galactose + N-acylsphingosine

3.2.1.48 sucrose alpha-glucosidase Hydrolysis of sucrose and maltose by an alpha-D-glucosidase-type action

3.3.1.1 adenosylhomocysteinase H2O + S-adenosyl-L-homocysteine = L-homocysteine + adenosine

3.5.1.1 asparaginase L-asparagine + H2O = L-aspartate + NH3

3.5.1.2 glutaminase L-glutamine + H2O = L-glutamate + NH3

3.5.1.6 beta-ureidopropionase N-carbamoyl-beta-alanine + H2O = beta-alanine + CO2 + NH3

3.5.1.7 ureidosuccinase N-carbamoyl-L-aspartate + H2O = L-aspartate + CO2 + NH3

3.5.1.8 formylaspartate deformylase N-formyl-L-aspartate + H2O = formate + L-aspartate

3.5.1.9 arylformamidase N-formyl-L-kynurenine + H2O = formate + L-kynurenine

3.5.1.16 acetylornithine deacetylase N2-acetyl-L-ornithine + H2O = acetate + L-ornithine

3.5.1.18 succinyl-diaminopimelate desuccinylase N-succinyl-LL2,6-diaminoheptanedioate + H2O = succinate + LL2,6-diaminoheptanedioate

3.5.1.19 nicotinamidase nicotinamide + H2O = nicotinate + NH3

3.5.1.20 citrullinase L-citrulline + 2 H2O = L-ornithine + CO2 + NH3

3.5.1.22 pantothenase pantothenate + H2O = pantoate + beta-alanine

3.5.1.23 ceramidase N-acylsphingosine + H2O = a carboxylate + sphingosine

3.5.1.32 hippurate hydrolase hippurate + H2O = benzoate + glycine

3.5.2.2 dihydropyrimidinase 5,6-dihydrouracil + H2O = 3-ureidopropanoate

3.5.2.3 dihydroorotase (S)-dihydroorotate + H2O = N-carbamoyl-L-aspartate

3.5.2.5 allantoinase allantoin + H2O = allantoate

3.5.2.7 imidazolonepropionase 4-imidazolone-5-propanoate + H2O = N-formimino-L-glutamate

3.5.2.10 creatininase creatinine + H2O = creatine

3.5.3.1 arginase L-arginine + H2O = L-ornithine + urea

3.5.3.4 allantoicase allantoate + H2O = (-)-ureidoglycolate + urea

3.5.3.5 formimidoylaspartate deiminase N-formimidoyl-L-aspartate + H2O = N-formyl-L-aspartate + NH3

3.5.3.6 arginine deiminase L-arginine + H2O = L-citrulline + NH3

3.5.4.1 cytosine deaminase cytosine + H2O = uracil + NH3

3.5.4.3 guanine deaminase guanine + H2O = xanthine + NH3

3.5.4.6 AMP deaminase AMP + H2O = IMP + NH3

3.5.4.9 methenyltetrahydrofolate cyclohydrolase 5,10-methenyltetrahydrofolate + H2O = 10-formyltetrahydrofolate

3.5.4.10 IMP cyclohydrolase IMP + H2O = 5-formamido-1-(5-phospho-D-ribosyl)imidazole-4-carboxamide

3.5.4.12 dCMP deaminase dCMP + H2O = dUMP + NH3

3.5.4.19 phosphoribosyl-AMP cyclohydrolase 1-(5-phosphoribosyl)-AMP + H2O = 1-(5-phosphoribosyl)-5-[(5-phosphoribosylamino)methylideneamino]imidazole-4-carboxamide

3.5.4.22 1-pyrroline-4-hydroxy-2-carboxylate deaminase 1-pyrroline-4-hydroxy-2-carboxylate + H2O = 2,5-dioxopentanoate + NH3

3.6.1.3 adenosinetriphosphatase ATP + H2O = ADP + phosphate

3.6.1.15 nucleoside-triphosphatase NTP + H2O = NDP + phosphate

3.6.1.31 phosphoribosyl-ATP diphosphatase 1-(5-phosphoribosyl)-ATP + H2O = 1-(5-phosphoribosyl)-AMP + diphosphate

3.7.1.2 fumarylacetoacetase 4-fumarylacetoacetate + H2O = acetoacetate + fumarate

3.7.1.3 kynureninase L-kynurenine + H2O = anthranilate + L-alanine

3.7.1.9 2-hydroxymuconate-semialdehyde hydrolase 2-hydroxymuconate semialdehyde + H2O = formate + 2-oxopent-4-enoate

3.9.1.1 phosphoamidase N-phosphocreatine + H2O = creatine + phosphate

4.1.1.1 pyruvate decarboxylase A 2-oxo acid = an aldehyde + CO2

4.1.1.3 oxaloacetate decarboxylase oxaloacetate = pyruvate + CO2

4.1.1.4 acetoacetate decarboxylase acetoacetate + H+ = acetone + CO2

4.1.1.9 malonyl-CoA decarboxylase malonyl-CoA = acetyl-CoA + CO2

4.1.1.11 aspartate 1-decarboxylase L-aspartate = beta-alanine + CO2

4.1.1.12 aspartate 4-decarboxylase L-aspartate = L-alanine + CO2

4.1.1.15 glutamate decarboxylase L-glutamate = 4-aminobutanoate + CO2

4.1.1.17 ornithine decarboxylase L-ornithine = putrescine + CO2

4.1.1.20 diaminopimelate decarboxylase meso-2,6-diaminoheptanedioate = L-lysine + CO2

4.1.1.21 phosphoribosylaminoimidazole carboxylase 5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxylate = 5-amino-1-(5-phospho-D-ribosyl)imidazole + CO2

4.1.1.22 histidine decarboxylase L-histidine = histamine + CO2

4.1.1.23 orotidine-5'-phosphate decarboxylase orotidine 5'-phosphate = UMP + CO2

4.1.1.25 tyrosine decarboxylase L-tyrosine = tyramine + CO2

4.1.1.28 aromatic-L-amino-acid decarboxylase L-tryptophan = tryptamine + CO2

4.1.1.29 sulfoalanine decarboxylase 3-sulfino-L-alanine = hypotaurine + CO2

4.1.1.31 phosphoenolpyruvate carboxylase phosphate + oxaloacetate = H2O + phosphoenolpyruvate + CO2

4.1.1.32 phosphoenolpyruvate carboxykinase (GTP) GTP + oxaloacetate = GDP + phosphoenolpyruvate + CO2

4.1.1.33 diphosphomevalonate decarboxylase ATP + (R)-5-diphosphomevalonate = ADP + phosphate + isopentenyl diphosphate + CO2

4.1.1.34 dehydro-L-gulonate decarboxylase 3-dehydro-L-gulonate = L-xylulose + CO2

4.1.1.36 phosphopantothenoylcysteine decarboxylase N-[(R)-4-phosphopantothenoyl]-L-cysteine = pantotheine 4'-phosphate + CO2

4.1.1.37 uroporphyrinogen decarboxylase
uroporphyrinogen III = coproporphyrinogen + 4 CO2

4.1.1.38 phosphoenolpyruvate carboxykinase (diphosphate)
diphosphate + oxaloacetate = phosphate + phosphoenolpyruvate + CO2

4.1.1.39 ribulose-bisphosphate carboxylase D-ribulose 1,5-bisphosphate + CO2 = 2 3-phospho-D-glycerate

4.1.1.41 methylmalonyl-CoA decarboxylase (S)-2-methyl-3-oxopropanoyl-CoA = propanoyl-CoA + CO2

4.1.1.43 phenylpyruvate decarboxylase phenylpyruvate = phenylacetaldehyde + CO2

4.1.1.45 aminocarboxymuconate-semialdehyde decarboxylase 2-amino-3-(3-oxoprop-2-enyl)-but-2-enedioate = 2-aminomuconate semialdehyde + CO2

4.1.1.47 tartronate-semialdehyde synthase 2 glyoxylate = tartronate semialdehyde + CO2

4.1.1.48 indole-3-glycerol-phosphate synthase 1-(2-carboxyphenylamino)-1-deoxy-D-ribulose 5-phosphate = C1-(3-indolyl)-glycerol 3-phosphate + CO2 + H2O

4.1.1.49 phosphoenolpyruvate carboxykinase (ATP) ATP + oxaloacetate = ADP + phosphoenolpyruvate + CO2

4.1.1.50 adenosylmethionine decarboxylase S-adenosyl-L-methionine = (5-deoxy-5-adenosyl)(3-aminopropyl)-methylsulfonium salt + CO2

4.1.1.53 phenylalanine decarboxylase L-phenylalanine = phenylethylamine + CO2

4.1.1.65 phosphatidylserine decarboxylase phosphatidyl-L-serine = phosphatidylethanolamine + CO2

4.1.1.71 2-oxoglutarate decarboxylase 2-oxoglutarate = succinate semialdehyde + CO2

4.1.2.5 threonine aldolase L-threonine = glycine + acetaldehyde

4.1.2.9 phosphoketolase D-xylulose 5-phosphate + phosphate = acetyl phosphate + D-glyceraldehyde 3-phosphate + H2O

4.1.2.12 ketopantoaldolase 2-hydroxy-2-isopropylbutanedioate = 3-methyl-2-oxobutanoate + formaldehyde

4.1.2.13 fructose-bisphosphate aldolase D-fructose 1,6-bisphosphate = glycerone phosphate + D-glyceraldehyde 3-phosphate

4.1.2.14 2-dehydro-3-deoxy-phosphogluconate aldolase 2-dehydro-3-deoxy-D-gluconate 6-phosphate = pyruvate + D-glyceraldehyde 3-phosphate

4.1.2.15 2-dehydro-3-deoxy-phosphoheptonate aldolase 2-dehydro-3-deoxy-D-arabino-heptonate 7-phosphate + phosphate = phosphoenolpyruvate + D-erythrose 4-phosphate + H2O

4.1.3.1 isocitrate lyase isocitrate = succinate + glyoxylate

4.1.3.2 malate synthase L-malate + CoA = acetyl-CoA + H2O + glyoxylate

4.1.3.4 hydroxymethylglutaryl-CoA lyase (S)-3-hydroxy-3-methylglutaryl-CoA = acetyl-CoA + acetoacetate

4.1.3.5 hydroxymethylglutaryl-CoA synthase (S)-3-hydroxy-3-methylglutaryl-CoA + CoA = acetyl-CoA + H2O + acetoacetyl-CoA

4.1.3.7 citrate (si)-synthase citrate + CoA = acetyl-CoA + H2O + oxaloacetate

4.1.3.8 ATP citrate (pro-S)-lyase ATP + citrate + CoA = ADP + phosphate + acetyl-CoA + oxaloacetate

4.1.3.16 4-hydroxy-2-oxoglutarate aldolase 4-hydroxy-2-oxoglutarate = pyruvate + glyoxylate

4.1.3.18 acetolactate synthase 2-acetolactate + CO2 = 2 pyruvate

4.1.3.20 N-acylneuraminate-9-phosphate synthase N-acylneuraminate 9-phosphate + phosphate = N-acyl-D-mannosamine 6-phosphate + phosphoenolpyruvate + H2O

4.1.3.21 homocitrate synthase 2-hydroxybutane-1,2,4-tricarboxylate + CoA = acetyl-CoA + H2O + 2-oxoglutarate

4.1.3.22 citramalate lyase (3S)-citramalate = acetate + pyruvate

4.1.3.27 anthranilate synthase chorismate + L-glutamine = anthranilate + pyruvate + L-glutamate

4.1.99.1 tryptophanase L-tryptophan + H2O = indole + pyruvate + NH3

4.2.1.2 fumarate hydratase (S)-malate = fumarate + H2O

4.2.1.3 aconitate hydratase citrate = cis-aconitate + H2O

4.2.1.4 citrate dehydratase citrate = cis-aconitate + H2O

4.2.1.9 dihydroxy-acid dehydratase 2,3-dihydroxy-3-methylbutanoate = 3-methyl-2-oxobutanoate + H2O

4.2.1.10 3-dehydroquinate dehydratase 3-dehydroquinate = 3-dehydroshikimate + H2O

4.2.1.11 phosphopyruvate hydratase 2-phospho-D-glycerate = phosphoenolpyruvate + H2O

4.2.1.12 phosphogluconate dehydratase 6-phospho-D-gluconate = 2-dehydro-3-deoxy-6-phospho-D-gluconate + H2O

4.2.1.13 L-serine dehydratase L-serine + H2O = pyruvate + NH3 + H2O

4.2.1.16 threonine dehydratase L-threonine + H2O = 2-oxobutanoate + NH3 + H2O

4.2.1.17 enoyl-CoA hydratase (3S)-3-hydroxyacyl-CoA = trans-2(or 3)-enoyl-CoA + H2O

4.2.1.18 methylglutaconyl-CoA hydratase (S)-3-hydroxy-3-methylglutaryl-CoA = trans-3-methylglutaconyl-CoA + H2O

4.2.1.19 imidazoleglycerol-phosphate dehydratase D-erythro-1-(imidazol-4-yl)glycerol 3-phosphate = 3-(imidazol-4-yl)-2-oxopropyl phosphate + H2O

4.2.1.20 tryptophan synthase L-serine + 1-(indol-3-yl)glycerol 3-phosphate = L-tryptophan + glyceraldehyde 3-phosphate

4.2.1.22 cystathionine beta-synthase L-serine + L-homocysteine = cystathionine + H2O

4.2.1.24 porphobilinogen synthase 2 5-aminolevulinate = porphobilinogen + 2 H2O

4.2.1.33 3-isopropylmalate dehydratase 3-isopropylmalate = 2-isopropylmaleate + H2O

4.2.1.34 (S)-2-methylmalate dehydratase (S)-2-methylmalate = 2-methylfumarate + H2O

4.2.1.36 homoaconitate hydratase 2-hydroxybutane-1,2,4-tricarboxylate = but-1-ene-1,2,4-tricarboxylate + H2O

4.2.1.46 dTDPglucose 4,6-dehydratase dTDPglucose = dTDP-4-dehydro-6-deoxy-D-glucose + H2O

4.2.1.47 GDPmannose 4,6-dehydratase GDPmannose = GDP-4-dehydro-6-deoxy-D-mannose + H2O

4.2.1.49 urocanate hydratase 4,5-dihydro-4-oxo-5-imidazolepropanoate = urocanate + H2O

4.2.1.51 prephenate dehydratase prephenate = phenylpyruvate + H2O + CO2

4.2.1.52 dihydrodipicolinate synthase L-aspartate 4-semialdehyde + pyruvate = dihydrodipicolinate + 2 H2O

4.2.1.55 3-hydroxybutyryl-CoA dehydratase (3R)-3-hydroxybutanoyl-CoA = crotonoyl-CoA + H2O

4.2.1.58 crotonoyl-[acyl-carrier-protein] hydratase (3R)-3-hydroxybutanoyl-[acyl-carrier protein] = but-2-enoyl-[acyl-carrier protein] + H2O

4.2.1.59 3-hydroxyoctanoyl-[acyl-carrier-protein] dehydratase (3R)-3-hydroxyoctanoyl-[acyl-carrier protein] = 2-octenoyl-[acyl-carrier protein] + H2O

4.2.1.60 3-hydroxydecanoyl-[acyl-carrier-protein] dehydratase (3R)-3-hydroxydecanoyl-[acyl-carrier protein] = 2-decenoyl-[acyl-carrier protein] (or 3-decenoyl-[acyl-carrier protein]) + H2O

4.2.1.61 3-hydroxypalmitoyl-[acyl-carrier-protein] dehydratase (3R)-3-hydroxypalmitoyl-[acyl-carrier protein] = 2-hexadecenoyl-[acyl-carrier protein] + H2O

4.2.1.75 uroporphyrinogen-III synthase hydroxymethylbilane = uroporphyrinogen III + H2O

4.2.1.80 2-oxopent-4-enoate hydratase 4-hydroxy-2-oxopentanoate = 2-oxopent-4-enoate + H2O

4.2.3.1 threonine synthase O-phospho-L-homoserine + H2O = L-threonine + phosphate

4.2.3.4 3-dehydroquinate synthase 3-deoxy-arabino-heptulosonate 7-phosphate = 3-dehydroquinate + phosphate

4.2.3.5 chorismate synthase 5-O-(1-carboxyvinyl)-3-phosphoshikimate = chorismate + phosphate

4.2.99.8 cysteine synthase O3-acetyl-L-serine + hydrogen sulfide = L-cysteine + acetate

4.2.99.9 O-succinylhomoserine (thiol)-lyase O-succinyl-L-homoserine + L-cysteine = cystathionine + succinate

4.3.1.1 aspartate ammonia-lyase L-aspartate = fumarate + NH3

4.3.1.2 methylaspartate ammonia-lyase L-threo-3-methylaspartate = mesaconate + NH3

4.3.1.3 histidine ammonia-lyase L-histidine = urocanate + NH3

4.3.1.4 formimidoyltetrahydrofolate cyclodeaminase 5-formimidoyltetrahydrofolate = 5,10-methenyltetrahydrofolate + NH3

4.3.1.8 hydroxymethylbilane synthase 4 porphobilinogen + H2O = hydroxymethylbilane + 4 NH3

4.3.2.1 argininosuccinate lyase N -(L-arginino)succinate = fumarate + L-arginine

4.3.2.2 adenylosuccinate lyase ***** multiline or wierd, edit by hand **

4.4.1.1 cystathionine gamma-lyase L-cystathionine + H2O = L-cysteine + NH3 + 2-oxobutanoate

4.4.1.8 cystathionine beta-lyase cystathionine + H2O = L-homocysteine + NH3 + pyruvate

4.6.1.1 adenylate cyclase ATP = 3',5'-cyclic AMP + diphosphate

4.99.1.1 ferrochelatase protoporphyrin + Fe2+ = protoheme + 2 H+

5.1.3.1 ribulose-phosphate 3-epimerase D-ribulose 5-phosphate = D-xylulose 5-phosphate

5.1.3.2 UDPglucose 4-epimerase UDPglucose = UDPgalactose

5.1.3.4 L-ribulose-phosphate 4-epimerase L-ribulose 5-phosphate = D-xylulose 5-phosphate

5.1.3.6 UDPglucuronate 4-epimerase UDPglucuronate = UDP-D-galacturonate

5.1.3.7 UDP-N-acetylglucosamine 4-epimerase UDP-N-acetyl-D-glucosamine = UDP-N-acetyl-D-galactosamine

5.1.3.12 UDPglucuronate 5'-epimerase UDPglucuronate = UDP-L-iduronate

5.1.3.13 dTDP-4-dehydrorhamnose 3,5-epimerase dTDP-4-dehydro-6-deoxy-D-glucose = dTDP-4-dehydro-6-deoxy-L-mannose

5.1.3.14 UDP-N-acetylglucosamine 2-epimerase UDP-N-acetyl-D-glucosamine = UDP-N-acetyl-D-mannosamine

5.1.99.1 methylmalonyl-CoA epimerase (R)-2-methyl-3-oxopropanoyl-CoA = (S)-2-methyl-3-oxopropanoyl-CoA

5.2.1.2 maleylacetoacetate isomerase 4-maleylacetoacetate = 4-fumarylacetoacetate

5.2.1.3 retinal isomerase all-trans-retinal = 11-cis-retinal

5.2.1.7 retinol isomerase all-trans-retinol = 11-cis-retinol

5.3.1.1 triose-phosphate isomerase D-glyceraldehyde 3-phosphate = glycerone phosphate

5.3.1.3 arabinose isomerase D-arabinose = D-ribulose

5.3.1.4 L-arabinose isomerase L-arabinose = L-ribulose

5.3.1.5 xylose isomerase D-xylose = D-xylulose

5.3.1.6 ribose-5-phosphate isomerase D-ribose 5-phosphate = D-ribulose 5-phosphate

5.3.1.8 mannose-6-phosphate isomerase D-mannose 6-phosphate = D-fructose 6-phosphate

5.3.1.9 glucose-6-phosphate isomerase D-glucose 6-phosphate = D-fructose 6-phosphate

5.3.1.15 D-lyxose ketol-isomerase D-lyxose = D-xylulose

5.3.1.16 1-(5-phosphoribosyl)-5-[(5-phosphoribosylamino)methylideneamino)imidazole-4-carboxamide isomerase 1-(5-phosphoribosyl)-5-[(5-phosphoribosylamino)methylideneamino)imidazole-4-carboxamide = 5-[(5-phospho-1-deoxyribulos-1-ylamino)methylideneamino]-1-(5-phosphoribosyl)imidazole-4-carboxamide

5.3.99.5 thromboxane-A synthase (5Z,13E)-(15S)-9alpha,11alpha-epidioxy-15-hydroxyprosta-5,13-dienoate =

(5Z,13E)-(15S)-9alpha,11alpha-epoxy-15-hydroxythromba-5,13-dienoate

5.4.2.1 phosphoglycerate mutase 2-phospho-D-glycerate = 3-phospho-D-glycerate

5.4.2.2 phosphoglucomutase alpha-D-glucose 1-phosphate = alpha-D-glucose 6-phosphate

5.4.2.3 phosphoacetylglucosamine mutase N-acetyl-D-glucosamine 1-phosphate = N-acetyl-D-glucosamine 6-phosphate

5.4.2.8 phosphomannomutase D-mannose 1-phosphate = D-mannose 6-phosphate

5.4.3.8 glutamate-1-semialdehyde 2,1-aminomutase (S)-4-amino-5-oxopentanoate = 5-aminolevulinate

5.4.99.1 methylaspartate mutase L-threo-3-methylaspartate = L-glutamate

5.4.99.2 methylmalonyl-CoA mutase (R)-2-methyl-3-oxopropanoyl-CoA = succinyl-CoA

5.4.99.5 chorismate mutase chorismate = prephenate

5.4.99.7 lanosterol synthase (S)-2,3-epoxysqualene = lanosterol

5.5.1.1 muconate cycloisomerase 2,5-dihydro-5-oxofuran-2-acetate = cis,cis-hexadienedioate

5.5.1.4 inositol-3-phosphate synthase D-glucose 6-phosphate = 1D-myo-inositol 3-phosphate

6.2.1.4 succinate-CoA ligase (GDP-forming) GTP + succinate + CoA = GDP + phosphate + succinyl-CoA

6.2.1.25 benzoate-CoA ligase ATP + benzoate + CoA = AMP + diphosphate + benzoyl-CoA

6.3.1.1 aspartate-ammonia ligase ATP + L-aspartate + NH3 = AMP + diphosphate + L-asparagine

6.3.1.2 glutamate-ammonia ligase ATP + L-glutamate + NH3 = ADP + phosphate + L-glutamine

6.3.1.4 aspartate-ammonia ligase (ADP-forming) ATP + L-aspartate + NH3 = ADP + phosphate + L-asparagine

6.3.1.5 NAD synthase ATP + deamido-NAD + NH3 = AMP + diphosphate + NAD

6.3.2.1 pantoate-beta-alanine ligase ATP + (R)-pantoate + beta-alanine = AMP + diphosphate + (R)-pantothenate

6.3.2.2 glutamate-cysteine ligase ATP + L-glutamate + L-cysteine = ADP + phosphate + gamma-L-glutamyl-L-cysteine

6.3.2.3 glutathione synthase ATP + gamma-L-glutamyl-L-cysteine + glycine = ADP + phosphate + glutathione

6.3.2.5 phosphopantothenate-cysteine ligase CTP + (R)-4'-phosphopantothenate + L-cysteine = unidentified products of CTP breakdown + (R)-4'-phosphopantothenoyl-L-cysteine

6.3.2.6 phosphoribosylaminoimidazolesuccinocarboxamide synthase ATP + 5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxylate + L-aspartate = ADP + phosphate + (S)-2-[5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxamido]succinate

6.3.2.7 UDP-N-acetylmuramoyl-L-alanyl-D-glutamate-lysine ligase ATP + UDP-N-acetylmuramoyl-L-alanyl-D-glutamate + L-lysine = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysine

6.3.2.8 UDP-N-acetylmuramate-alanine ligase ATP + UDP-N-acetylmuramate + L-alanine = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanine

6.3.2.9 UDP-N-acetylmuramoylalanine-D-glutamate ligase ATP + UDP-N-acetylmuramoyl-L-alanine + glutamate = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-glutamate

6.3.2.10 UDP-N-acetylmuramoylalanyl-D-glutamyl-lysine-D-alanyl-D-alanine ligase ATP + UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysine + D-alanyl-D-alanine = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysyl-D-alanyl-D-alanine

6.3.2.13 UDP-N-acetylmuramoylalanyl-D-glutamate-2,6-diamino-pimelate ligase ATP + UDP-N-acetylmuramoyl-L-alanyl-D-glutamate + meso-2,6-diaminoheptanedioate = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-gamma-glutamyl-meso-2,6-diamino-heptanedioate

6.3.3.1 phosphoribosylformylglycinamidine cyclo-ligase ATP + 2-(formamido)-N1-(5-phospho-D-ribosyl)acetamidine = ADP + phosphate + 5-amino-1-(5-phospho-D-ribosyl)imidazole

6.3.4.1 GMP synthase ATP + xanthosine 5'-phosphate + NH3 = AMP + diphosphate + GMP

6.3.4.2 CTP synthase ATP + UTP + NH3 = ADP + phosphate + CTP

6.3.4.3 formate-tetrahydrofolate ligase ATP + formate + tetrahydrofolate = ADP + phosphate + 10-formyltetrahydrofolate

6.3.4.4 adenylosuccinate synthase GTP + IMP + L-aspartate = GDP + phosphate + N6-(1,2-dicarboxyethyl)-AMP

6.3.4.5 argininosuccinate synthase ATP + L-citrulline + L-aspartate = AMP + diphosphate + N -(L-arginino)succinate

6.3.4.7 ribose-5-phosphate-ammonia ligase ATP + ribose 5-phosphate + NH3 = ADP + phosphate + 5-phosphoribosylamine

6.3.4.13 phosphoribosylamine-glycine ligase ATP + 5-phospho-D-ribosylamine + glycine = ADP + phosphate + N1-(5-phospho-D-ribosyl)glycinamide

6.3.4.16 carbamoyl-phosphate synthase (ammonia) 2 ATP + NH3 + CO2 + H2O = 2 ADP + phosphate + carbamoyl phosphate

6.3.4.17 formate-dihydrofolate ligase ATP + formate + dihydrofolate = ADP + phosphate + 10-formyldihydrofolate

6.3.5.1 NAD synthase (glutamine-hydrolysing) ATP + deamido-NAD + L-glutamine + H2O = AMP + diphosphate + NAD + L-glutamate

6.3.5.2 GMP synthase (glutamine-hydrolysing) ATP + xanthosine 5'-phosphate + L-glutamine + H2O = AMP + diphosphate + GMP + L-glutamate

6.3.5.3 phosphoribosylformylglycinamidine synthase ATP + N2-formyl-N1-(5-phospho-D-ribosyl)glycinamide + L-glutamine + H2O = ADP + phosphate + 2-(formamido)-N1-(5-phospho-D-ribosyl)acetamidine + L-glutamate

6.3.5.4 asparagine synthase (glutamine-hydrolysing) ATP + L-aspartate + L-glutamine = AMP + diphosphate + L-asparagine + L-glutamate

6.3.5.5 carbamoyl-phosphate synthase (glutamine-hydrolysing) 2 ATP + L-glutamine + CO2 + H2O = 2 ADP + phosphate + L-glutamate + carbamoyl phosphate

6.4.1.1 pyruvate carboxylase ATP + pyruvate + HCO3- = ADP + phosphate + oxaloacetate

6.4.1.2 acetyl-CoA carboxylase ATP + acetyl-CoA + HCO3- = ADP + phosphate + malonyl-CoA

6.4.1.3 propionyl-CoA carboxylase ATP + propanoyl-CoA + HCO3- = ADP + phosphate + (S)-methylmalonyl-CoA

6.4.1.4 methylcrotonoyl-CoA carboxylase ATP + 3-methylcrotonoyl-CoA + HCO3- = ADP + phosphate + 3-methylglutaconyl-CoA

User.dat

```
0.0.0.1   tautomerisation     enolpyruvate = ketopyruvate
0.0.0.2   mutarotation   beta-D-glucose 1-phosphate = alpha-
D-glucose 1-phosphate
1.1.1.1   alcohol dehydrogenase    ethanol + NAD+ =
acetaldehyde + NADH + H+
1.1.1.8   glycerol-3-phosphate dehydrogenase (NAD)     sn-
glycerol 3-phosphate + NAD = glycerone phosphate + NADH2
1.1.1.9   D-xylulose reductase     xylitol + NAD = D-
xylulose + NADH2
1.1.1.10  L-xylulose reductase     xylitol + NADP = L-
xylulose + NADPH2
1.1.1.14  L-iditol 2-dehydrogenase L-iditol + NAD = L-
sorbose + NADH2
1.1.1.15  D-iditol 2-dehydrogenase D-iditol + NAD = D-
sorbose + NADH2
1.1.1.19  L-glucuronate reductase  L-gulonate + NADP = D-
glucuronate + NADPH2
1.1.1.21  aldehyde reductase  alditol + NAD(P) = aldose +
NAD(P)H
1.1.1.22  UDPglucose 6-dehydrogenase    UDPglucose + 2 NAD +
H2O = UDPglucuronate + 2 NADH2
1.1.1.23  histidinol dehydrogenase L-histidinol + 2 NAD = L-
histidine + 2 NADH2
1.1.1.25  shikimate 5-dehydrogenase     shikimate + NADP =
5-dehydroshikimate + NADPH2
1.1.1.27  L-lactate dehydrogenase  (S)-lactate + NAD =
pyruvate + NADH2
1.1.1.28  D-lactate dehydrogenase  (R)-lactate + NAD =
pyruvate + NADH2
1.1.1.29  glycerate dehydrogenase  (R)-glycerate + NAD =
hydroxypyruvate + NADH2
1.1.1.30  3-hydroxybutyrate dehydrogenase    (R)-3-
hydroxybutanoate + NAD = acetoacetate + NADH2
```

1.1.1.31 3-hydroxyisobutyrate dehydrogenase 3-hydroxy-2-methylpropanoate + NAD = 2-methyl-3-oxopropanoate + NADH2

1.1.1.32 mevaldate reductase (R)-mevalonate + NAD = mevaldate + NADH2

1.1.1.34 hydroxymethylglutaryl-CoA reductase (NADPH2) (R)-mevalonate + CoA + 2 NADP = (S)-3-hydroxy-3-methylglutaryl-CoA + 2 NADPH2

1.1.1.35 3-hydroxyacyl-CoA dehydrogenase (S)-3-hydroxyacyl-CoA + NAD = 3-oxoacyl-CoA + NADH2

1.1.1.37 malate dehydrogenase (S)-malate + NAD = oxaloacetate + NADH2

1.1.1.38 malate dehydrogenase (oxaloacetate-decarboxylating) (S)-malate + NAD = pyruvate + CO2 + NADH2

1.1.1.39 malate dehydrogenase (decarboxylating) (S)-malate + NAD = pyruvate + CO2 + NADH2

1.1.1.41 isocitrate dehydrogenase (NAD) isocitrate + NAD = 2-oxoglutarate + CO2 + NADH2

1.1.1.43 phosphogluconate 2-dehydrogenase 6-phospho-D-gluconate + NAD(P) = 6-phospho-2-dehydro-D-gluconate + NAD(P)H

1.1.1.44 phosphogluconate dehydrogenase (decarboxylating) 6-phospho-D-gluconate + NADP = D-ribulose 5-phosphate + CO2 + NADPH2

1.1.1.45 L-gulonate 3-dehydrogenase L-gulonate + NAD = 3-dehydro-L-gulonate + NADH2

1.1.1.49 glucose-6-phosphate 1-dehydrogenase D-glucose 6-phosphate + NADP = D-glucono-1,5-lactone 6-phosphate + NADPH2

1.1.1.56 ribitol 2-dehydrogenase ribitol + NAD = D-ribulose + NADH2

1.1.1.59 3-hydroxypropionate dehydrogenase 3-hydroxypropanoate + NAD = 3-oxopropanoate + NADH2

1.1.1.60 2-hydroxy-3-oxopropionate reductase (R)-glycerate + NAD(P) = 2-hydroxy-3-oxopropanoate + NAD(P)H

1.1.1.61 4-hydroxybutyrate dehydrogenase 4-hydroxybutanoate + NAD = succinate semialdehyde + NADH2

1.1.1.79 glyoxylate reductase (NADP) glycolate + NADP = glyoxylate + NADPH2

1.1.1.81 hydroxypyruvate reductase D-glycerate + NAD(P) = hydroxypyruvate + NAD(P)H

1.1.1.82 malate dehydrogenase (NADP) (S)-malate + NADP = oxaloacetate + NADPH2

1.1.1.85 3-isopropylmalate dehydrogenase 3-carboxy-2-hydroxy-4-methylpentanoate + NAD = 3-carboxy-4-methyl-2-oxopentanoate + NADH2

1.1.1.86 ketol-acid reductoisomerase (R)-2,3-dihydroxy-3-methylbutanoate + NADP = (S)-2-hydroxy-2-methyl-3-oxobutanoate + NADPH2

1.1.1.95 phosphoglycerate dehydrogenase 3-phosphoglycerate + NAD = 3-phosphohydroxypyruvate + NADH2

1.1.1.100 3-oxoacyl-[acyl-carrier-protein] reductase (3R)-3-hydroxyacyl-[acyl-carrier protein] + NADP = 3-oxoacyl-[acyl-carrier protein] + NADPH2

1.1.1.102 3-dehydrosphinganine reductase sphinganine + NADP = 3-dehydrosphinganine + NADPH2

1.1.1.103 L-threonine 3-dehydrogenase L-threonine + NAD+ = L-2-amino-3-oxobutanoate + NADH + H+

1.1.1.105 retinol dehydrogenase retinol + NAD = retinal + NADH2

1.1.1.155 homoisocitrate dehydrogenase (-)-1-hydroxy-1,2,4-butanetricarboxylate + NAD = 2-oxoadipate + CO2 + NADH2

1.1.1.157 3-hydroxybutyryl-CoA dehydrogenase (S)-3-hydroxybutanoyl-CoA + NADP = 3-acetoacetyl-CoA + NADPH2

1.1.1.158 UDP-N-acetylmuramate dehydrogenase UDP-N-acetylmuramate + NADP = UDP-N-acetyl-3-O-(1-carboxyvinyl)-D-glucosamine + NADPH2

1.1.1.169 2-dehydropantoate 2-reductase (R)-pantoate + NADP = 2-dehydropantoate + NADPH2

1.1.1.204 xanthine dehydrogenase xanthine + NAD + H2O = urate + NADH2

1.1.1.205 IMP dehydrogenase Inosine 5'-phosphate + NAD + H2O = xanthosine 5'-phosphate + NADH2

1.1.3.8 L-gulonolactone oxidase L-gulono-1,4-lactone + O2 = L-xylo-hexulonolactone + H2O

1.1.3.15 (S)-2-hydroxy-acid oxidase (S)-2-hydroxy acid + O2 = 2-oxo acid + H2O2

1.1.3.22 xanthine oxidase xanthine + H2O + O2 = urate + H2O2

1.2.1.7 benzaldehyde dehydrogenase (NADP) benzaldehyde + NADP + H2O = benzoate + NADPH2

1.2.1.8 betaine-aldehyde dehydrogenase betaine aldehyde + NAD + H2O = betaine + NADH2

1.2.1.11 aspartate-semialdehyde dehydrogenase L-aspartate 4-semialdehyde + phosphate + NADP = L-4-aspartyl phosphate + NADPH2

1.2.1.12 glyceraldehyde-3-phosphate dehydrogenase (phosphorylating) D-glyceraldehyde 3-phosphate + phosphate + NAD = 3-phospho-D-glyceroyl phosphate + NADH2

1.2.1.13 glyceraldehyde-3-phosphate dehydrogenase (NADP) (phosphorylating) D-glyceraldehyde 3-phosphate + phosphate + NADP = 3-phospho-D-glyceroyl phosphate + NADPH2

1.2.1.16 succinate-semialdehyde dehydrogenase [NAD(P)] succinate semialdehyde + NAD(P) + H2O = succinate + NAD(P)H2

1.2.1.18 malonate-semialdehyde dehydrogenase (acetylating) 3-oxopropanoate + CoA + NAD(P) = acetyl-CoA + CO2 + NAD(P)H2

1.2.1.19 aminobutyraldehyde dehydrogenase 4-aminobutanal + NAD + H2O = 4-aminobutanoate + NADH2

1.2.1.21 glycolaldehyde dehydrogenase glycolaldehyde + NAD + H2O = glycolate + NADH2

1.2.1.24 succinate-semialdehyde dehydrogenase succinate semialdehyde + NAD + H2O = succinate + NADH2

1.2.1.25 2-oxoisovalerate dehydrogenase (acylating) 3-methyl-2-oxobutanoate + CoA + NAD = 2-methylpropanoyl-CoA + CO2 + NADH2

1.2.1.26 2,5-dioxovalerate dehydrogenase 2,5-dioxopentanoate + NADP + H2O = 2-oxoglutarate + NADPH2

1.2.1.27 methylmalonate-semialdehyde dehydrogenase (acylating) 2-methyl-3-oxopropanoate + CoA + NAD = propanoyl-CoA + CO2 + NADH2

1.2.1.28 benzaldehyde dehydrogenase (NAD) benzaldehyde + NAD + H2O = benzoate + NADH2

1.2.1.31 L-aminoadipate-semialdehyde dehydrogenase L-2-aminoadipate 6-semialdehyde + NAD(P) + H2O = L-2-aminoadipate + NAD(P)H2

1.2.1.32 aminomuconate-semialdehyde dehydrogenase 2-aminomuconate 6-semialdehyde + NAD + H2O = 2-aminomuconate + NADH2

1.2.1.36 retinal dehydrogenase retinal + NAD + H2O = retinoate + NADH2

1.2.1.39 phenylacetaldehyde dehydrogenase phenylacetaldehyde + NAD + H2O = phenylacetate + NADH2

1.2.1.41 glutamate-5-semialdehyde dehydrogenase L-glutamate 5-semialdehyde + phosphate + NADP = L-glutamyl 5-phosphate + NADPH2

1.2.1.52 oxoglutarate dehydrogenase (NADP) 2-oxoglutarate + CoA + NADP = succinyl-CoA + CO2 + NADPH2

1.2.3.5 glyoxylate oxidase glyoxylate + H2O + O2 = oxalate + H2O2

1.2.4.1 pyruvate dehydrogenase (lipoamide) pyruvate + lipoamide = S-acetyldihydrolipoamide + CO2

1.2.4.2 oxoglutarate dehydrogenase (lipoamide) 2-oxoglutarate + lipoamide = S-succinyldihydrolipoamide + CO2

1.2.7.1 pyruvate synthase pyruvate + CoA + oxidized ferredoxin = acetyl-CoA + CO2 + reduced ferredoxin

1.2.7.2 2-oxobutyrate synthase 2-oxobutanoate + CoA + oxidized ferredoxin = propanoyl-CoA + CO2 + reduced ferredoxin

1.3.1.1 dihydrouracil dehydrogenase (NAD) 5,6-dihydrouracil + NAD = uracil + NADH2

1.3.1.2 dihydropyrimidine dehydrogenase (NADP) 5,6-dihydrouracil + NADP = uracil + NADPH2

1.3.1.6 fumarate reductase (NADH2) succinate + NAD = fumarate + NADH2

1.3.1.8 acyl-CoA dehydrogenase (NADP) acyl-CoA + NADP = 2,3-dehydroacyl-CoA + NADPH2

1.3.1.9 enoyl-[acyl-carrier-protein] reductase (NADH2) acyl-[acyl-carrier protein] + NAD = trans-2,3-dehydroacyl-[acyl-carrier protein] + NADH2

1.3.1.10 enoyl-[acyl-carrier-protein] reductase (NADPH2, B-specific) acyl-[acyl-carrier protein] + NADP = trans-2,3-dehydroacyl-[acyl-carrier protein] + NADPH2

1.3.1.13 prephenate dehydrogenase (NADP) prephenate + NADP = 4-hydroxyphenylpyruvate + CO2 + NADPH2

1.3.1.14 orotate reductase (NADH2) (S)-dihydroorotate + NAD = orotate + NADH2

1.3.1.26 dihydrodipicolinate reductase 2,3,4,5-tetrahydrodipicolinate + NAD(P) = 2,3-dihydrodipicolinate + NAD(P)H

1.3.1.35 phosphatidylcholine desaturase 1-acyl-2-oleoyl-sn-glycero-3-phosphocholine + NAD = 1-acyl-2-linoleoyl-sn-glycero-3-phosphocholine + NADH2

1.3.3.3 coproporphyrinogen oxidase coproporphyrinogen-III + O2 = protoporphyrinogen-IX + 2 CO2

1.3.3.4 protoporphyrinogen oxidase protoporphyrinogen-IX + O2 = protoporphyrin-IX + H2O

1.3.5.1 succinate dehydrogenase (ubiquinone) succinate + ubiquinone = fumarate + ubiquinol

1.4.1.1 alanine dehydrogenase L-alanine + H2O + NAD = pyruvate + NH3 + NADH2

1.4.1.2 glutamate dehydrogenase L-glutamate + H2O + NAD = 2-oxoglutarate + NH3 + NADH2

1.4.1.7 serine dehydrogenase L-serine + H2O + NAD = 3-hydroxypyruvate + NH3 + NADH2

1.4.1.8 valine dehydrogenase (NADP) L-valine + H2O + NADP = 3-methyl-2-oxobutanoate + NH3 + NADPH2

1.4.1.9 leucine dehydrogenase L-leucine + H2O + NAD = 4-methyl-2-oxopentanoate + NH3 + NADH2

1.4.1.10 glycine dehydrogenase glycine + H2O + NAD = glyoxylate + NH3 + NADH2

1.4.1.13 glutamate synthase (NADPH2) 2 L-glutamate + NADP = L-glutamate + 2-oxoglutarate + NADPH2

1.4.1.14 glutamate synthase (NADH2) 2 L-glutamate + NAD = L-glutamine + 2-oxoglutarate + NADH2

1.4.1.19 tryptophan dehydrogenase L-tryptophan + NAD(P) = (indol-3-yl)pyruvate + NH3 + NAD(P)H2

1.4.3.1 D-aspartate oxidase D-aspartate + H2O + O2 = oxaloacetate + NH3 + H2O2

1.4.3.4 amine oxidase (flavin-containing) RCH2NH2 + H2O + O2 = RCHO + NH3 + H2O2

1.4.3.6 amine oxidase (copper-containing) RCH2NH2 + H2O + O2 = RCHO + NH3 + H2O2

1.4.3.8 ethanolamine oxidase ethanolamine + H2O + O2 = glycolaldehyde + NH3 + H2O2

1.4.4.2 glycine dehydrogenase (decarboxylating) glycine + lipoylprotein = S-aminomethyldihydrolipoylprotein + CO2

1.5.1.1 pyrroline-2-carboxylate reductase L-proline + NAD(P) = 1-pyrroline-2-carboxylate + NAD(P)H

1.5.1.2 pyrroline-5-carboxylate reductase L-proline + NAD(P) = 1-pyrroline-5-carboxylate + NAD(P)H

1.5.1.3 dihydrofolate reductase 5,6,7,8-tetrahydrofolate + NADP = 7,8-dihydrofolate + NADPH

1.5.1.5 methylenetetrahydrofolate dehydrogenase (NADP) 5,10-methylenetetrahydrofolate + NADP = 5,10-methenyltetrahydrofolate + NADPH

1.5.1.6 formyltetrahydrofolate dehydrogenase 10-formyltetrahydrofolate + NADP + H2O = tetrahydrofolate + CO2 + NADPH

1.5.1.7 saccharopine dehydrogenase (NAD, L-lysine-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NAD + H2O = L-lysine + 2-oxoglutarate + NADH2

1.5.1.8 saccharopine dehydrogenase (NADP, L-lysine-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NADP + H2O = L-lysine + 2-oxoglutarate + NADPH

1.5.1.9 saccharopine dehydrogenase (NAD, L-glutamate-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NAD + H2O = L-glutamate + 2-aminoadipate 6-semialdehyde + NADH2

1.5.1.10 saccharopine dehydrogenase (NADP, L-glutamate-forming) N6-(L-1,3-dicarboxypropyl)-L-lysine + NADP + H2O = L-glutamate + 2-aminoadipate 6-semialdehyde + NADPH

1.5.1.12 1-pyrroline-5-carboxylate dehydrogenase 1-pyrroline-5-carboxylate + NAD + H2O = L-glutamate + NADH2

1.5.1.21 Delta1-piperideine-2-carboxylate reductase L-pipecolate + NADP = Delta1-piperideine-2-carboxylate + NADPH2

1.5.3.1 sarcosine oxidase sarcosine + H2O + O2 = glycine + formaldehyde + H2O2

1.6.4.1 cystine reductase (NADH2) NADH2 + L-cystine = NAD + 2 L-cysteine

1.6.5.3 NADH2 dehydrogenase (ubiquinone) NADH2 + ubiquinone = NAD + ubiquinol

1.6.6.1 nitrate reductase (NADH2) NADH2 + nitrate = NAD + nitrite + H2O

1.6.6.2 nitrate reductase [NAD(P)H2] NAD(P)H2 + nitrate = NAD(P) + nitrite + H2O

1.6.6.3 nitrate reductase (NADPH2) NADPH2 + nitrate = NADP + nitrite + H2O

1.6.6.4 nitrite reductase [NAD(P)H2] 3 NAD(P)H2 + nitrite = 3 NAD(P) + NH4OH + H2O

1.6.6.8 GMP reductase NADPH2 + guanosine 5'-phosphate = NADP + inosine 5'-phosphate + NH3

1.7.7.1 ferredoxin-nitrite reductase ammonia + 2 H2O + 6 oxidized ferredoxin = nitrite + 6 reduced ferredoxin + 7 H+

1.8.1.3 hypotaurine dehydrogenase hypotaurine + H2O + NAD = taurine + NADH2

1.8.1.4 dihydrolipoamide dehydrogenase dihydrolipoamide + NAD = lipoamide + NADH2

1.8.2.1 sulfite dehydrogenase sulfite + 2 ferricytochrome c + H2O = sulfate + 2 ferrocytochrome c

1.8.3.1 sulfite oxidase sulfite + O2 + H2O = sulfate + H2O2

1.8.7.1 sulfite reductase (ferredoxin) hydrogen sulfide + 3 oxidized ferredoxin + 3 H2O = sulfite + 3 reduced ferredoxin

1.9.3.1 cytochrome-c oxidase 4 ferrocytochrome c + O2 = 4 ferricytochrome c + 2 H2O

1.10.2.1 L-ascorbate-cytochrome-b5 reductase L-ascorbate + ferricytochrome-b5 = monodehydroascorbate + ferrocytochrome b5

1.10.2.2 ubiquinol-cytochrome-c reductase QH2 + 2 ferricytochrome c = Q + 2 ferrocytochrome c

1.10.3.3 L-ascorbate oxidase 2 L-ascorbate + O2 = 2 dehydroascorbate + 2 H2O

1.10.99.1 plastoquinol-plastocyanin reductase plastoquinol-1 + 2 oxidized plastocyanin = plastoquinone + 2 reduced plastocyanin

1.13.11.1 catechol 1,2-dioxygenase catechol + O2 = cis,cis-muconate

1.13.11.2 catechol 2,3-dioxygenase catechol + O2 = 2-hydroxymuconate semialdehyde

1.13.11.5 homogentisate 1,2-dioxygenase homogentisate + O2 = 4-maleylacetoacetate

1.13.11.6 3-hydroxyanthranilate 3,4-dioxygenase 3-hydroxyanthranilate + O2 = 2-amino-3-carboxymuconate semialdehyde

1.13.11.11 tryptophan 2,3-dioxygenase L-tryptophan + O2 = L-formylkynurenine

1.13.11.20 cysteine dioxygenase L-cysteine + O2 = 3-sulfinoalanine

1.13.11.21 beta-carotene 15,15'-dioxygenase beta-carotene + O2 = 2 retinal

1.13.11.27 4-hydroxyphenylpyruvate dioxygenase 4-hydroxyphenylpyruvate + O2 = homogentisate + CO2

1.13.99.1 myo-inositol oxygenase myo-inositol + O2 = D-glucuronate + H2O

1.14.11.1 gamma-butyrobetaine dioxygenase 4-trimethylammoniobutanoate + 2-oxoglutarate + O2 = 3-hydroxy-4-trimethylammoniobutanoate + succinate + CO2

1.14.11.2 procollagen-proline dioxygenase procollagen L-proline + 2-oxoglutarate + O2 = procollagen trans-4-hydroxy-L-proline + succinate + CO2

1.14.11.8 trimethyllysine dioxygenase N6,N6,N6-trimethyl-L-lysine + 2-oxoglutarate + O2 = 3-hydroxy-N6,N6,N6-trimethyl-L-lysine + succinate + CO2

1.14.12.1 anthranilate 1,2-dioxygenase (deaminating, decarboxylating) anthranilate + NAD(P)H2 + O2 + 2 H2O = catechol + CO2 + NAD(P) + NH3

1.14.12.10 benzoate 1,2-dioxygenase benzoate + NADH2 + O2 = catechol + CO2 + NAD

1.14.13.5 imidazoleacetate 4-monooxygenase 4-imidazoleacetate + NADH2 + O2 = 5-hydroxy-4-imidazoleacetate + NAD + H2O

1.14.13.9 kynurenine 3-monooxygenase L-kynurenine + NADPH2 + O2 = 3-hydroxy-L-kynurenine + NADP + H2O

1.14.13.12 benzoate 4-monooxygenase benzoate + NADPH2 + O2 = 4-hydroxybenzoate + NADP + H2O

1.14.13.39 nitric-oxide synthase L-arginine + n NADPH2 + m O2 = citrulline + nitric oxide + n NADP

1.14.16.1 phenylalanine 4-monooxygenase L-phenylalanine + tetrahydrobiopterin + O2 = L-tyrosine + dihydrobiopterin + H2O

1.14.16.2 tyrosine 3-monooxygenase L-tyrosine + tetrahydrobiopterin + O2 = 3,4-dihydroxy-L-phenylalanine + dihydrobiopterin + H2O

1.14.16.4 tryptophan 5-monooxygenase L-tryptophan + tetrahydrobiopterin + O2 = 5-hydroxy-L-tryptophan + dihydrobiopterin + H2O

1.14.17.1 dopamine beta-monooxygenase 3,4-dihydroxyphenethylamine + ascorbate + O2 = noradrenaline + dehydroascorbate + H2O

1.14.18.1 monophenol monooxygenase L-tyrosine + L-dopa + O2 = L-dopa + dopaquinone + H2O

1.14.19.1 stearoyl-CoA 9-desaturase stearoyl-CoA + AH2 + O2 = oleoyl-CoA + A + 2 H2O

1.14.19.3 linoleoyl-CoA desaturase linoleoyl-CoA + AH2 + O2 = gamma-linolenoyl-CoA + A + 2 H2O

1.14.99.1 prostaglandin-endoperoxide synthase arachidonate + AH2 + 2 O2 = prostaglandin H2 + A + H2O

1.14.99.7 squalene monooxygenase squalene + AH2 + O2 = (S)-squalene-2,3-epoxide + A + H2O

1.17.4.1 ribonucleoside-diphosphate reductase 2'-deoxyribonucleoside diphosphate + oxidized thioredoxin + H2O = ribonucleoside diphosphate + reduced thioredoxin

1.18.6.1 nitrogenase 3 reduced ferredoxin + 6 H+ + N2 + n ATP = 3 oxidized ferredoxin + 2 NH3 + n ADP + n phosphate

2.1.1.1 nicotinamide N-methyltransferase S-adenosyl-L-methionine + nicotinamide = S-adenosyl-L-homocysteine + 1-methylnicotinamide

2.1.1.2 guanidinoacetate N-methyltransferase S-adenosyl-L-methionine + guanidinoacetate = S-adenosyl-L-homocysteine + creatine

2.1.1.3 thetin-homocysteine S-methyltransferase dimethylsulfonioacetate + L-homocysteine = S-methylthioglycolate + L-methionine

2.1.1.4 acetylserotonin O-methyltransferase S-adenosyl-L-methionine + N-acetylserotonin = S-adenosyl-L-homocysteine + melatonin

2.1.1.5 betaine-homocysteine S-methyltransferase trimethylammonioacetate + L-homocysteine = dimethylglycine + L-methionine

2.1.1.10 homocysteine S-methyltransferase S-adenosyl-L-methionine + L-homocysteine = S-adenosyl-L-homocysteine + L-methionine

2.1.1.13 5-methyltetrahydrofolate-homocysteine S-methyltransferase 5-methyltetrahydrofolate + L-homocysteine = tetrahydrofolate + L-methionine

2.1.1.14 5-methyltetrahydropteroyltriglutamate-homocysteine S-methyltransferase 5-methyltetrahydropteroyltri-L-glutamate + L-homocysteine = tetrahydropteroyltri-L-glutamate + L-methionine

2.1.1.17 phosphatidylethanolamine N-methyltransferase S-adenosyl-L-methionine + phosphatidylethanolamine = S-adenosyl-L-homocysteine + phosphatidyl-N-methylethanolamine

2.1.1.20 glycine N-methyltransferase S-adenosyl-L-methionine + glycine = S-adenosyl-L-homocysteine + sarcosine

2.1.1.28 phenylethanolamine N-methyltransferase S-adenosyl-L-methionine + phenylethanolamine = S-adenosyl-L-homocysteine + N-methylphenylethanolamine

2.A.1.1.28 hs_Glut1 e-D-glucose = D-glucose

2.1.1.43 histone-lysine N-methyltransferase S-adenosyl-L-methionine + histone L-lysine = S-adenosyl-L-homocysteine + histone N6-methyl-L-lysine

2.1.1.45 thymidylate synthase 5,10-methylenetetrahydrofolate + dUMP = dihydrofolate + dTMP

2.1.2.1 glycine hydroxymethyltransferase 5,10-methylenetetrahydrofolate + glycine + H2O = tetrahydrofolate + L-serine

2.1.2.2 phosphoribosylglycinamide formyltransferase 10-formyltetrahydrofolate + N1-(5-phospho-D-ribosyl)glycinamide = tetrahydrofolate + N2-formyl-N1-(5-phospho-D-ribosyl)glycinamide

2.1.2.3 phosphoribosylaminoimidazolecarboxamide formyltransferase 10-formyltetrahydrofolate + 5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxamide = tetrahydrofolate + 5-formamido-1-(5-phospho-D-ribosyl)imidazole-4-carboxamide

2.1.2.5 glutamate formimidoyltransferase 5-formimidoyltetrahydrofolate + L-glutamate = tetrahydrofolate + N-formimidoyl-L-glutamate

2.1.2.10 aminomethyltransferase S-aminomethyldihydrolipoylprotein + (6S)-tetrahydrofolate = dihydrolipoylprotein + (6R)-5,10-methylenetetrahydrofolate + NH3

2.1.3.1 methylmalonyl-CoA carboxyltransferase (S)-2-methyl-3-oxopropanoyl-CoA + pyruvate = propanoyl-CoA + oxaloacetate

2.1.3.2 aspartate carbamoyltransferase carbamoyl phosphate + L-aspartate = phosphate + N-carbamoyl-L-aspartate

2.1.3.3 ornithine carbamoyltransferase carbamoyl phosphate + L-ornithine = phosphate + L-citrulline

2.1.4.1 glycine amidinotransferase L-arginine + glycine = L-ornithine + guanidinoacetate

2.2.1.1 transketolase1 sedoheptulose 7-phosphate + D-glyceraldehyde 3-phosphate = D-ribose 5-phosphate + D-xylulose 5-phosphate

2.2.1.1 transketolase2 D-erythrose 4-phosphate + D-xylulose 5-phosphate = D-fructose 6-phosphate + D-glyceraldehyde 3-phosphate

2.2.1.2 transaldolase sedoheptulose 7-phosphate + D-glyceraldehyde 3-phosphate = D-erythrose 4-phosphate + D-fructose 6-phosphate

2.3.1.1 amino-acid N-acetyltransferase acetyl-CoA + L-glutamate = CoA + N-acetyl-L-glutamate

2.3.1.4 glucosamine-phosphate N-acetyltransferase acetyl-CoA + D-glucosamine 6-phosphate = CoA + N-acetyl-D-glucosamine 6-phosphate

2.3.1.6 choline O-acetyltransferase acetyl-CoA + choline = CoA + O-acetylcholine

2.3.1.7 carnitine O-acetyltransferase acetyl-CoA + carnitine = CoA + O-acetylcarnitine

2.3.1.8 phosphate acetyltransferase acetyl-CoA + phosphate = CoA + acetyl phosphate

2.3.1.9 acetyl-CoA C-acetyltransferase acetyl-CoA + acetyl-CoA = CoA + acetoacetyl-CoA

2.3.1.12 dihydrolipoamide S-acetyltransferase acetyl-CoA + dihydrolipoamide = CoA + S-acetyldihydrolipoamide

2.3.1.15 glycerol-3-phosphate O-acyltransferase acyl-CoA + sn-glycerol 3-phosphate = CoA + 1-acyl-sn-glycerol 3-phosphate

2.3.1.16 acetyl-CoA C-acyltransferase acyl-CoA + acetyl-CoA = CoA + 3-oxoacyl-CoA

2.3.1.20 diacylglycerol O-acyltransferase acyl-CoA + 1,2-diacylglycerol = CoA + triacylglycerol

2.3.1.21 carnitine O-palmitoyltransferase palmitoyl-CoA + L-carnitine = CoA + L-palmitoylcarnitine

2.3.1.23 1-acylglycerophosphocholine O-acyltransferase acyl-CoA + 1-acyl-sn-glycero-3-phosphocholine = CoA + 1,2-diacyl-sn-glycero-3-phosphocholine

2.3.1.24 sphingosine N-acyltransferase acyl-CoA + sphingosine = CoA + N-acylsphingosine

2.3.1.30 serine O-acetyltransferase acetyl-CoA + L-serine = CoA + O-acetyl-L-serine

2.3.1.32 lysine N-acetyltransferase acetyl phosphate + L-lysine = phosphate + N6-acetyl-L-lysine

2.3.1.35 glutamate N-acetyltransferase N2-acetyl-L-ornithine + L-glutamate = L-ornithine + N-acetyl-L-glutamate

2.3.1.37 5-aminolevulinate synthase succinyl-CoA + glycine = 5-aminolevulinate + CoA + CO2

2.3.1.38 [acyl-carrier-protein] S-acetyltransferase acetyl-CoA + [acyl-carrier protein] = CoA + acetyl-[acyl-carrier protein]

2.3.1.39 [acyl-carrier-protein] S-malonyltransferase malonyl-CoA + [acyl-carrier protein] = CoA + malonyl-[acyl-carrier protein]

2.3.1.41 3-oxoacyl-[acyl-carrier-protein] synthase acyl-[acyl-carrier protein] + malonyl-[acyl-carrier protein] = 3-oxoacyl-[acyl-carrier protein] + CO2 + [acyl-carrier protein]

2.3.1.46 homoserine O-succinyltransferase succinyl-CoA + L-homoserine = CoA + O-succinyl-L-homoserine

2.3.1.50 serine C-palmitoyltransferase palmitoyl-CoA + L-serine = CoA + 3-dehydro-D-sphinganine + CO2

2.3.1.51 1-acylglycerol-3-phosphate O-acyltransferase acyl-CoA + 1-acyl-sn-glycerol 3-phosphate = CoA + 1,2-diacyl-sn-glycerol 3-phosphate

2.3.1.76 retinol O-fatty-acyltransferase acyl-CoA + retinol = CoA + retinyl ester

2.4.1.1 phosphorylase (1,4-alpha-D-glucosyl)n + phosphate = (1,4-alpha-D-glucosyl)n-1 + alpha-D-glucose 1-phosphate

2.4.1.4 amylosucrase sucrose + (1,4-alpha-D-glucosyl)n = D-fructose + (1,4-alpha-D-glucosyl)n+1

2.4.1.8 maltose phosphorylase maltose + phosphate = D-glucose + beta-D-glucose 1-phosphate

2.4.1.9 inulosucrase sucrose + (2,1-beta-D-fructosyl)n = glucose + (2,1-beta-D-fructosyl)n+1

2.4.1.11 glycogen(starch) synthase UDPglucose + (1,4-alpha-D-glucosyl)n = UDP + (1,4-alpha-D-glucosyl)n+1

2.4.1.13 sucrose synthase UDPglucose + D-fructose = UDP + sucrose

2.4.1.15 alpha,alpha-trehalose-phosphate synthase (UDP-forming) UDP-D-glucose + D-glucose 6-phosphate = UDP + alpha,alpha-trehalose 6-phosphate

2.4.1.21 starch synthase ADPglucose + (1,4-alpha-D-glucosyl)n = ADP + (1,4-alpha-D-glucosyl)n+1

2.4.1.22 lactose synthase UDPgalactose + D-glucose = UDP + lactose

2.4.1.23 sphingosine beta-galactosyltransferase UDPgalactose + sphingosine = UDP + psychosine

2.4.1.29 cellulose synthase (GDP-forming) GDPglucose + (1,4-beta-D-glucosyl)n = GDP + (1,4-beta-D-glucosyl)n+1

2.4.1.32 glucomannan 4-beta-mannosyltransferase GDPmannose + (glucomannan)n = GDP + (glucomannan)n+1

2.4.1.41 polypeptide N-acetylgalactosaminyltransferase UDP-N-acetyl-D-galactosamine + polypeptide = UDP + N-acetyl-D-galactosaminyl-polypeptide

2.4.1.62 ganglioside galactosyltransferase UDPgalactose + N-acetyl-D-galactosaminyl-(N-acetylneuraminyl)-D-galactosyl-1,4-beta-D-glucosyl-N-acylsphingosine = UDP + D-galactosyl-1,3-beta-N-acetyl-D-galactosaminyl-(N-acetylneuraminyl)-D-galactosyl-D-glucosyl-N-acylsphingosine

2.4.1.68 glycoprotein 6-alpha-L-fucosyltransferase GDP-L-fucose + N4-(N-acetyl-beta-D-glucosaminyl-1,2-alpha-D-mannosyl-1,3-(R1-alpha-1,6)-beta-D-mannosyl-beta-N-acetyl-1,4-D-glucosaminyl-1,4-N-acetyl-D-glucosaminyl)asparagine = GDP + N4-(N-acetyl-beta-D-glucosaminyl-1,2-alpha-D-mannosyl-1,3-(R1-alpha-1,6)-beta-D-mannosyl-1,4-beta-N-acetyl-D-glucosaminyl-1,4-(alpha-L-fucosyl-1,6)-N-acetyl-D-glucosaminyl)-asparagine

2.4.1.69 galactoside 2-alpha-L-fucosyltransferase GDP-L-fucose + beta-D-galactosyl-R = GDP + alpha-L-fucosyl-1,2-beta-D-galactosyl-R

2.4.2.1 purine-nucleoside phosphorylase purine nucleoside + phosphate = purine + alpha-D-ribose 1-phosphate

2.4.2.2 pyrimidine-nucleoside phosphorylase pyrimidine nucleoside + phosphate = pyrimidine + alpha-D-ribose 1-phosphate

2.4.2.4 thymidine phosphorylase thymidine + phosphate = thymine + 2-deoxy-D-ribose 1-phosphate

2.4.2.8 hypoxanthine phosphoribosyltransferase IMP + diphosphate = hypoxanthine + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.9 uracil phosphoribosyltransferase UMP + diphosphate = uracil + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.10 orotate phosphoribosyltransferase orotidine 5'-phosphate + diphosphate = orotate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.11 nicotinate phosphoribosyltransferase nicotinate D-ribonucleotide + diphosphate = nicotinate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.14 amidophosphoribosyltransferase 5-phospho-beta-D-ribosylamine + diphosphate + L-glutamate = L-glutamine + 5-phospho-alpha-D-ribose 1-diphosphate + H2O

2.4.2.15 guanosine phosphorylase guanosine + phosphate = guanine + D-ribose 1-phosphate

2.4.2.17 ATP phosphoribosyltransferase 1-(5-phospho-D-ribosyl)-ATP + diphosphate = ATP + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.18 anthranilate phosphoribosyltransferase N-(5-phospho-D-ribosyl)-anthranilate + diphosphate = anthranilate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.2.19 nicotinate-nucleotide diphosphorylase (carboxylating) nicotinate D-ribonucleotide + diphosphate + CO2 = pyridine-2,3-dicarboxylate + 5-phospho-alpha-D-ribose 1-diphosphate

2.4.99.1 beta-galactoside alpha-2,6-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosamine = CMP + alpha-N-acetylneuraminyl-2,6-beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosamine

2.4.99.2 monosialoganglioside sialyltransferase CMP-N-acetylneuraminate + D-galactosyl-N-acetyl-D-galactosaminyl-(N-acetylneuraminyl)-D-galactosyl-D-glucosylceramide = CMP + N-acetylneuraminyl-D-galactosyl-N-acetyl-D-galactosaminyl-(N-acetylneuraminyl)-D-galactosyl-D-glucosylceramide

2.4.99.3 alpha-N-acetylgalactosaminide alpha-2,6-sialyltransferase CMP-N-acetylneuraminate + glycano-1,3-(N-acetyl-alpha-D-galactosaminyl)-glycoprotein = CMP + glycano-(2,6-alpha-N-acetylneuraminyl)-(N-acetyl-D-galactosaminyl)-glycoprotein

2.4.99.4 beta-galactoside alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,3-N-acetyl-alpha-D-galactosaminyl-R = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,3-N-acetyl-alpha-D-galactosaminyl-R

2.4.99.5 galactosyldiacylglycerol alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + 1,2-diacyl-3-beta-D-galactosyl-sn-glycerol = CMP + 1,2-diacyl-3-[3-(alpha-D-N-acetylneuraminyl)-beta-D-galactosyl]-sn-glycerol

2.4.99.6 N-acetyllactosaminide alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-N-acetyl-D-glucosaminyl-glycoprotein = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,4-N-acetyl-D-glucosaminyl-glycoprotein

2.4.99.7 (alpha-N-acetylneuraminyl-2,3-alpha-galactosyl-1,3)-N-acetyl-galactosaminide alpha-2,6-sialyltransferase CMP-N-acetylneuraminate + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,3-N-acetyl-D-galactosaminyl-R = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactose-1,3-(alpha-N-acetylneuraminyl-2,6)-N-acetyl-D-galactosaminyl-R

2.4.99.8 alpha-N-acetylneuraminate alpha-2,8-sialyltransferase CMP-N-acetylneuraminate + alpha-N-

acetylneuraminyl-2,3-beta-D-galactosyl-R = CMP + alpha-N-acetylneuraminyl-2,8-alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-R

2.4.99.9 lactosylceramide alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-beta-D-glucosylceramide = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,4-beta-D-glucosyl-ceramide

2.4.99.10 neolactotetraosylceramide alpha-2,3-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosaminyl-1,3-beta-D-galactosyl-1,4-D-glucosylceramide = CMP + alpha-N-acetylneuraminyl-2,3-beta-D-galactosyl-1,4-N-acetyl-beta-D-glucosaminyl-1,3-beta-D-galactosyl-1,4-D-glucosylceramide

2.4.99.11 lactosylceramide alpha-2,6-N-sialyltransferase CMP-N-acetylneuraminate + beta-D-galactosyl-1,4-beta-D-glucosylceramide = CMP + alpha-N-acetylneuraminyl-2,6-beta-galactosyl-1,4-beta-D-glucosylceramide

2.5.1.1 dimethylallyltranstransferase dimethylallyl diphosphate + isopentenyl diphosphate = diphosphate + geranyl diphosphate

2.5.1.6 methionine adenosyltransferase ATP + L-methionine + H2O = phosphate + diphosphate + S-adenosyl-L-methionine

2.5.1.10 geranyltranstransferase geranyl diphosphate + isopentenyl diphosphate = diphosphate + trans,trans-farnesyl diphosphate

2.5.1.16 spermidine synthase S-adenosylmethioninamine + putrescine = 5'-methylthioadenosine + spermidine

2.5.1.19 3-phosphoshikimate 1-carboxyvinyltransferase phosphoenolpyruvate + 3-phosphoshikimate = phosphate + 5-O-(1-carboxyvinyl)-3-phosphoshikimate

2.5.1.21 farnesyl-diphosphate farnesyltransferase 2 farnesyl diphosphate = diphosphate + presqualene diphosphate

2.5.1.22 spermine synthase S-adenosylmethioninamine + spermidine = 5'-methylthioadenosine + spermine

2.5.1.29 farnesyltranstransferase trans,trans-farnesyl diphosphate + isopentenyl diphosphate = diphosphate + geranylgeranyl diphosphate

2.5.1.32 geranylgeranyl-diphosphate geranylgeranyltransferase 2 geranylgeranyl diphosphate = diphosphate + prephytoene diphosphate

2.5.1.61 hydroxymethylbilane synthase 4 porphobilinogen + H2O = hydroxymethylbilane + 4 NH3

2.6.1.1 aspartate transaminase L-aspartate + 2-oxoglutarate = oxaloacetate + L-glutamate

2.6.1.2 alanine transaminase L-alanine + 2-oxoglutarate = pyruvate + L-glutamate

2.6.1.4 glycine transaminase glycine + 2-oxoglutarate = glyoxylate + L-glutamate

2.6.1.5 tyrosine transaminase L-tyrosine + 2-oxoglutarate = 4-hydroxyphenylpyruvate + L-glutamate

2.6.1.6 leucine transaminase L-leucine + 2-oxoglutarate = 4-methyl-2-oxopentanoate + L-glutamate

2.6.1.9 histidinol-phosphate transaminase L-histidinol phosphate + 2-oxoglutarate = 3-(imidazol-4-yl)-2-oxopropyl phosphate + L-glutamate

2.6.1.11 acetylornithine transaminase N2-acetyl-L-ornithine + 2-oxoglutarate = N-acetyl-L-glutamate 5-semialdehyde + L-glutamate

2.6.1.16 glutamine-fructose-6-phosphate transaminase (isomerizing) L-glutamine + D-fructose 6-phosphate = L-glutamate + D-glucosamine 6-phosphate

2.6.1.17 succinyldiaminopimelate transaminase N-succinyl-L-2,6-diaminoheptanedioate + 2-oxoglutarate = N-succinyl-2-L-amino-6-oxoheptanedioate + L-glutamate

2.6.1.18 beta-alanine-pyruvate transaminase L-alanine + 3-oxopropanoate = pyruvate + beta-alanine

2.6.1.19 4-aminobutyrate transaminase 4-aminobutanoate + 2-oxoglutarate = succinate semialdehyde + L-glutamate

2.6.1.22 (S)-3-amino-2-methylpropionate transaminase (S)-3-amino-2-methylpropanoate + 2-oxoglutarate = 2-methyl-3-oxopropanoate + L-glutamate

2.6.1.23 4-hydroxyglutamate transaminase 4-hydroxy-L-glutamate + 2-oxoglutarate = 4-hydroxy-2-oxoglutarate + L-glutamate

2.6.1.27 tryptophan transaminase L-tryptophan + 2-oxoglutarate = indolepyruvate + L-glutamate

2.6.1.32 valine-3-methyl-2-oxovalerate transaminase L-valine + (S)-3-methyl-2-oxopentanoate = 3-methyl-2-oxobutanoate + L-isoleucine

2.6.1.36 L-lysine 6-transaminase L-lysine + 2-oxoglutarate = 2-aminoadipate 6-semialdehyde + L-glutamate

2.6.1.39 2-aminoadipate transaminase L-2-aminoadipate + 2-oxoglutarate = 2-oxoadipate + L-glutamate

2.6.1.42 branched-chain-amino-acid transaminase L-leucine + 2-oxoglutarate = 4-methyl-2-oxopentanoate + L-glutamate

2.6.1.51 serine-pyruvate transaminase L-serine + pyruvate = 3-hydroxypyruvate + L-alanine

2.6.1.52 phosphoserine transaminase O-phospho-L-serine + 2-oxoglutarate = 3-phosphonooxypyruvate + L-glutamate

2.6.1.65 N6-acetyl-beta-lysine transaminase 6-acetamido-3-aminohexanoate + 2-oxoglutarate = 6-acetamido-3-oxohexanoate + L-glutamate

2.6.1.66 valine-pyruvate transaminase L-valine + pyruvate = 3-methyl-2-oxobutanoate + L-alanine

2.7.1.1 hexokinase ATP + D-glucose = ADP + D-glucose 6-phosphate

2.7.1.2 glucokinase ATP + D-glucose = ADP + D-glucose 6-phosphate

2.7.1.3 ketohexokinase ATP + D-fructose = ADP + D-fructose 1-phosphate

2.7.1.4 fructokinase ATP + D-fructose = ADP + D-fructose 6-phosphate

2.7.1.6 galactokinase ATP + D-galactose = ADP + alpha-D-galactose 1-phosphate

2.7.1.7 mannokinase ATP + D-mannose = ADP + D-mannose 6-phosphate

2.7.1.11 6-phosphofructokinase ATP + D-fructose 6-phosphate = ADP + D-fructose 1,6-bisphosphate

2.7.1.15 ribokinase ATP + D-ribose = ADP + D-ribose 5-phosphate

2.7.1.16 ribulokinase ATP + L(or D)-ribulose = ADP + L(or D)-ribulose 5-phosphate

2.7.1.17 xylulokinase ATP + D-xylulose = ADP + D-xylulose 5-phosphate

2.7.1.19 phosphoribulokinase ATP + D-ribulose 5-phosphate = ADP + D-ribulose 1,5-bisphosphate

2.7.1.23 NAD kinase ATP + NAD = ADP + NADPH2

2.7.1.24 dephospho-CoA kinase ATP + dephospho-CoA = ADP + CoA

2.7.1.25 adenylyl-sulfate kinase ATP + adenylyl sulfate = ADP + 3'-phosphoadenylyl sulfate

2.7.1.28 triokinase ATP + D-glyceraldehyde = ADP + D-glyceraldehyde 3-phosphate

2.7.1.30 glycerol kinase ATP + glycerol = ADP + sn-glycerol 3-phosphate

2.7.1.31 glycerate kinase ATP + (R)-glycerate = ADP + 3-phospho-(R)-glycerate

2.7.1.32 choline kinase ATP + choline = ADP + O-phosphocholine

2.7.1.33 pantothenate kinase ATP + pantothenate = ADP + D-4'-phosphopantothenate

2.7.1.34 pantetheine kinase ATP + pantetheine = ADP + pantetheine 4'-phosphate

2.7.1.36 mevalonate kinase ATP + (R)-mevalonate = ADP + (R)-5-phosphomevalonate

2.7.1.39 homoserine kinase ATP + L-homoserine = ADP + O-phospho-L-homoserine

2.7.1.40 pyruvate kinase ATP + pyruvate = ADP + phosphoenolpyruvate

2.7.1.47 D-ribulokinase ATP + D-ribulose = ADP + D-ribulose 5-phosphate

2.7.1.53 L-xylulokinase ATP + L-xylulose = ADP + L-xylulose 5-phosphate

2.7.1.60 N-acylmannosamine kinase ATP + N-acyl-D-mannosamine = ADP + N-acyl-D-mannosamine 6-phosphate

2.7.1.64 inositol 1-kinase ATP + myo-inositol = ADP + 1L-myo-inositol 1-phosphate

2.7.1.71 shikimate kinase ATP + shikimate = ADP + shikimate 3-phosphate

2.7.1.80 diphosphate-serine phosphotransferase diphosphate + L-serine = phosphate + O-phospho-L-serine

2.7.1.107 diacylglycerol kinase ATP + 1,2-diacylglycerol = ADP + 1,2-diacyl-sn-glycerol 3-phosphate

2.7.2.1 acetate kinase ATP + acetate = ADP + acetyl phosphate

2.7.2.3 phosphoglycerate kinase ATP + 3-phospho-D-glycerate = ADP + 3-phospho-D-glyceroyl phosphate

2.7.2.4 aspartate kinase ATP + L-aspartate = ADP + 4-phospho-L-aspartate

2.7.2.6 formate kinase ATP + formate = ADP + formyl phosphate

2.7.2.11 glutamate 5-kinase ATP + L-glutamate = ADP + L-glutamate 5-phosphate

2.7.3.2 creatine kinase ATP + creatine = ADP + phosphocreatine

2.7.4.2 phosphomevalonate kinase ATP + (R)-5-phosphomevalonate = ADP + (R)-5-diphosphomevalonate

2.7.4.3 adenylate kinase ATP + AMP = ADP + ADP

2.7.4.4 nucleoside-phosphate kinase ATP + nucleoside phosphate = ADP + nucleoside diphosphate

2.7.4.6 nucleoside-diphosphate kinase ATP + nucleoside diphosphate = ADP + nucleoside triphosphate

2.7.4.9 dTMP kinase ATP + dTMP = ADP + dTDP

2.7.4.14 cytidylate kinase ATP + (d)CMP = ADP + (d)CDP

2.7.6.1 ribose-phosphate diphosphokinase ATP + D-ribose 5-phosphate = AMP + 5-phospho-alpha-D-ribose 1-diphosphate

2.7.7.3 pantetheine-phosphate adenylyltransferase ATP + pantetheine 4'-phosphate = diphosphate + dephospho-CoA

2.7.7.4 sulfate adenylyltransferase ATP + sulfate = diphosphate + adenylylsulfate

2.7.7.6 DNA-directed RNA polymerase nucleoside triphosphate + RNAn = diphosphate + RNAn+1

2.7.7.7 DNA-directed DNA polymerase deoxynucleoside triphosphate + DNAn = diphosphate + DNAn+1

2.7.7.9 UTP-glucose-1-phosphate uridylyltransferase UTP + D-glucose 1-phosphate = diphosphate + UDP-D-glucose

2.7.7.10 UTP-hexose-1-phosphate uridylyltransferase UTP + alpha-D-galactose 1-phosphate = diphosphate + UDP galactose

2.7.7.12 UDPglucose-hexose-1-phosphate uridylyltransferase UDPglucose + alpha-D-galactose 1-phosphate = alpha-D-glucose 1-phosphate + UDPgalactose

2.7.7.13 mannose-1-phosphate guanylyltransferase GTP + alpha-D-mannose 1-phosphate = diphosphate + GDPmannose

2.7.7.14 ethanolamine-phosphate cytidylyltransferase CTP + ethanolamine phosphate = diphosphate + CDPethanolamine

2.7.7.15 choline-phosphate cytidylyltransferase CTP + choline phosphate = diphosphate + CDPcholine

2.7.7.18 nicotinate-nucleotide adenylyltransferase ATP + nicotinate ribonucleotide = diphosphate + deamido-NAD

2.7.7.22 mannose-1-phosphate guanylyltransferase (GDP) GDP + D-mannose 1-phosphate = phosphate + GDP mannose

2.7.7.23 UDP-N-acetylglucosamine diphosphorylase UTP + N-acetyl-alpha-D-glucosamine 1-phosphate = diphosphate + UDP-N-acetyl-D-glucosamine

2.7.7.24 glucose-1-phosphate thymidylyltransferase dTTP + alpha-D-glucose 1-phosphate = diphosphate + dTDP glucose

2.7.7.27 glucose-1-phosphate adenylyltransferase ATP + alpha-D-glucose 1-phosphate = diphosphate + ADP glucose

2.7.7.34 glucose-1-phosphate guanylyltransferase GTP + alpha-D-glucose 1-phosphate = diphosphate + GDP glucose

2.7.7.41 phosphatidate cytidylyltransferase CTP + phosphatidate = diphosphate + CDP diacylglycerol

2.7.7.43 N-acylneuraminate cytidylyltransferase CTP + N-acylneuraminate = diphosphate + CMP-N-acylneuraminate

2.7.8.3 ceramide cholinephosphotransferase CDP choline + N-acylsphingosine = CMP + sphingomyelin

2.7.8.5 CDPdiacylglycerol-glycerol-3-phosphate 3-phosphatidyltransferase CDP diacylglycerol + sn-glycerol 3-phosphate = CMP + 3(3-sn-phosphatidyl)-sn-glycerol 1-phosphate

2.7.8.8 CDPdiacylglycerol-serine O-phosphatidyltransferase CDP diacylglycerol + L-serine = CMP + 3-O-sn-phosphatidyl-L-serine

2.7.8.11 CDP diacylglycerol-inositol 3-phosphatidyltransferase CDP diacylglycerol + myo-inositol = CMP + phosphatidyl-1D-myo-inositol

2.7.8.13 phospho-N-acetylmuramoyl-pentapeptide-transferase UDPMurAc(oyl-L-Ala-D-gamma-Glu-L-Lys-D-Ala-D-Ala) + undecaprenyl phosphate = UMP + MurAc(oyl-L-Ala-D-gamma-Glu-L-Lys-D-Ala-D-Ala)-diphospho-undecaprenol

2.8.3.1 propionate CoA-transferase acetyl-CoA + propanoate = acetate + propanoyl-CoA

2.8.3.6 3-oxoadipate CoA-transferase succinyl-CoA + 3-oxoadipate = succinate + 3-oxoadipyl-CoA

3.1.1.7 acetylcholinesterase acetylcholine + H2O = choline + acetate

3.1.1.17 gluconolactonase D-glucono-1,5-lactone + H2O = D-gluconate

3.1.1.21 retinyl-palmitate esterase retinyl palmitate + H2O = retinol + palmitate

3.1.1.24 3-oxoadipate enol-lactonase 3-oxoadipate enol-lactone + H2O = 3-oxoadipate

3.1.1.31 6-phosphogluconolactonase D-glucono-1,5-lactone 6-phosphate + H2O = 6-phospho-D-gluconate

3.1.2.3 succinyl-CoA hydrolase succinyl-CoA + H2O = CoA + succinate

3.1.2.4 3-hydroxyisobutyryl-CoA hydrolase 3-hydroxy-2-methylpropanoyl-CoA + H2O = CoA + 3-hydroxy-2-methylpropanoate

3.1.2.11 acetoacetyl-CoA hydrolase acetoacetyl-CoA + H2O = CoA + acetoacetate

3.1.3.3 phosphoserine phosphatase L(or D)-O-phosphoserine + H2O = L(or D)-serine + phosphate

3.1.3.9 glucose-6-phosphatase D-glucose 6-phosphate + H2O = D-glucose + phosphate

3.1.3.10 glucose-1-phosphatase D-glucose 1-phosphate + H2O = D-glucose + phosphate

3.1.3.11 fructose-bisphosphatase D-fructose 1,6-bisphosphate + H2O = D-fructose 6-phosphate + orthophosphate

3.1.3.12 trehalose-phosphatase alpha,alpha-trehalose 6-phosphate + H2O = alpha,alpha-trehalose + phosphate

3.1.3.15 histidinol-phosphatase L-histidinol phosphate + H2O = L-histidinol + phosphate

3.1.3.25 inositol-1(or 4)-monophosphatase myo-inositol 1-phosphate + H2O = myo-inositol + phosphate

3.1.3.27 phosphatidylglycerophosphatase
phosphatidylglycerophosphate + H2O = phosphatidylglycerol + phosphate

3.1.3.29 N-acylneuraminate-9-phosphatase N-acylneuraminate 9-phosphate + H2O = N-acylneuraminate + phosphate

3.1.4.2 glycerophosphocholine phosphodiesterase sn-glycero-3-phosphocholine + H2O = choline + sn-glycerol 3-phosphate

3.1.4.3 phospholipase C A phosphatidylcholine + H2O = 1,2-diacylglycerol + choline phosphate

3.1.4.10 1-phosphatidylinositol phosphodiesterase 1-phosphatidyl-D-myo-inositol = D-myo-inositol 1,2-cyclic phosphate + diacylglycerol

3.1.4.12 sphingomyelin phosphodiesterase sphingomyelin + H2O = N-acylsphingosine + choline phosphate

3.2.1.28 alpha,alpha-trehalase alpha,alpha-trehalose + H2O = 2 D-glucose

3.2.1.45 glucosylceramidase D-glucosyl-N-acylspingosine + H2O = D-glucose + N-acylsphingosine

3.2.1.46 galactosylceramidase D-galactosyl-N-acylsphingosine + H2O = D-galactose + N-acylsphingosine

3.3.1.1 adenosylhomocysteinase H2O + S-adenosyl-L-homocysteine = L-homocysteine + adenosine

3.5.1.1 asparaginase L-asparagine + H2O = L-aspartate + NH3

3.5.1.2 glutaminase L-glutamine + H2O = L-glutamate + NH3

3.5.1.6 beta-ureidopropionase N-carbamoyl-beta-alanine + H2O = beta-alanine + CO2 + NH3

3.5.1.7 ureidosuccinase N-carbamoyl-L-aspartate + H2O = L-aspartate + CO2 + NH3

3.5.1.8 formylaspartate deformylase N-formyl-L-aspartate + H2O = formate + L-aspartate

3.5.1.9 arylformamidase N-formyl-L-kynurenine + H2O = formate + L-kynurenine

3.5.1.16 acetylornithine deacetylase N2-acetyl-L-ornithine + H2O = acetate + L-ornithine

3.5.1.18 succinyl-diaminopimelate desuccinylase N-succinyl-LL2,6-diaminoheptanedioate + H2O = succinate + LL2,6-diaminoheptanedioate

3.5.1.19 nicotinamidase nicotinamide + H2O = nicotinate + NH3

3.5.1.20 citrullinase L-citrulline + 2 H2O = L-ornithine + CO2 + NH3

3.5.1.22 pantothenase pantothenate + H2O = pantoate + beta-alanine

3.5.1.32 hippurate hydrolase hippurate + H2O = benzoate + glycine

3.5.2.2 dihydropyrimidinase 5,6-dihydrouracil + H2O = 3-ureidopropanoate

3.5.2.3 dihydroorotase (S)-dihydroorotate + H2O = N-carbamoyl-L-aspartate

3.5.2.5 allantoinase allantoin + H2O = allantoate

3.5.2.7 imidazolonepropionase 4-imidazolone-5-propanoate + H2O = N-formimino-L-glutamate

3.5.2.10 creatininase creatinine + H2O = creatine

3.5.3.1 arginase L-arginine + H2O = L-ornithine + urea

3.5.3.4 allantoicase allantoate + H2O = (-)-ureidoglycolate + urea

3.5.3.5 formimidoylaspartate deiminase N-formimidoyl-L-aspartate + H2O = N-formyl-L-aspartate + NH3

3.5.3.6 arginine deiminase L-arginine + H2O = L-citrulline + NH3

3.5.4.1 cytosine deaminase cytosine + H2O = uracil + NH3

3.5.4.3 guanine deaminase guanine + H2O = xanthine + NH3

3.5.4.6 AMP deaminase AMP + H2O = IMP + NH3

3.5.4.9 methenyltetrahydrofolate cyclohydrolase 5,10-methenyltetrahydrofolate + H2O = 10-formyltetrahydrofolate

3.5.4.10 IMP cyclohydrolase IMP + H2O = 5-formamido-1-(5-phospho-D-ribosyl)imidazole-4-carboxamide

3.5.4.12 dCMP deaminase dCMP + H2O = dUMP + NH3

3.5.4.19 phosphoribosyl-AMP cyclohydrolase 1-(5-phosphoribosyl)-AMP + H2O = 1-(5-phosphoribosyl)-5-[(5-phosphoribosylamino)methylideneamino]imidazole-4-carboxamide

3.5.4.22 1-pyrroline-4-hydroxy-2-carboxylate deaminase 1-pyrroline-4-hydroxy-2-carboxylate + H2O = 2,5-dioxopentanoate + NH3

3.6.1.3 adenosinetriphosphatase ATP + H2O = ADP + phosphate

3.6.1.15 nucleoside-triphosphatase NTP + H2O = NDP + phosphate

3.6.1.31 phosphoribosyl-ATP diphosphatase 1-(5-phosphoribosyl)-ATP + H2O = 1-(5-phosphoribosyl)-AMP + diphosphate

3.7.1.2 fumarylacetoacetase 4-fumarylacetoacetate + H2O = acetoacetate + fumarate

3.7.1.3 kynureninase L-kynurenine + H2O = anthranilate + L-alanine

3.7.1.9 2-hydroxymuconate-semialdehyde hydrolase 2-hydroxymuconate semialdehyde + H2O = formate + 2-oxopent-4-enoate

3.9.1.1 phosphoamidase N-phosphocreatine + H2O = creatine + phosphate

4.1.1.3 oxaloacetate decarboxylase oxaloacetate = pyruvate + CO2

4.1.1.4 acetoacetate decarboxylase acetoacetate + H+ = acetone + CO2

4.1.1.9 malonyl-CoA decarboxylase malonyl-CoA = acetyl-CoA + CO2

4.1.1.11 aspartate 1-decarboxylase L-aspartate = beta-alanine + CO2

4.1.1.12 aspartate 4-decarboxylase L-aspartate = L-alanine + CO2

4.1.1.15 glutamate decarboxylase L-glutamate = 4-aminobutanoate + CO2

4.1.1.17 ornithine decarboxylase L-ornithine = putrescine + CO2

4.1.1.20 diaminopimelate decarboxylase meso-2,6-diaminoheptanedioate = L-lysine + CO2

4.1.1.21 phosphoribosylaminoimidazole carboxylase 5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxylate = 5-amino-1-(5-phospho-D-ribosyl)imidazole + CO2

4.1.1.22 histidine decarboxylase L-histidine = histamine + CO2

4.1.1.23 orotidine-5'-phosphate decarboxylase orotidine 5'-phosphate = UMP + CO2

4.1.1.25 tyrosine decarboxylase L-tyrosine = tyramine + CO2

4.1.1.28 aromatic-L-amino-acid decarboxylase L-tryptophan = tryptamine + CO2

4.1.1.29 sulfoalanine decarboxylase 3-sulfino-L-alanine = hypotaurine + CO2

4.1.1.31 phosphoenolpyruvate carboxylase phosphate + oxaloacetate = H2O + phosphoenolpyruvate + CO2

4.1.1.32 phosphoenolpyruvate carboxykinase (GTP) GTP + oxaloacetate = GDP + phosphoenolpyruvate + CO2

4.1.1.33 diphosphomevalonate decarboxylase ATP + (R)-5-diphosphomevalonate = ADP + phosphate + isopentenyl diphosphate + CO2

4.1.1.34 dehydro-L-gulonate decarboxylase 3-dehydro-L-gulonate = L-xylulose + CO2

4.1.1.36 phosphopantothenoylcysteine decarboxylase N-[(R)-4-phosphopantothenoyl]-L-cysteine = pantotheine 4'-phosphate + CO2

4.1.1.37 uroporphyrinogen decarboxylase
uroporphyrinogen III = coproporphyrinogen + 4 CO2

4.1.1.38 phosphoenolpyruvate carboxykinase (diphosphate)
diphosphate + oxaloacetate = phosphate + phosphoenolpyruvate + CO2

4.1.1.39 ribulose-bisphosphate carboxylase D-ribulose 1,5-bisphosphate + CO2 = 2 3-phospho-D-glycerate

4.1.1.41 methylmalonyl-CoA decarboxylase (S)-2-methyl-3-oxopropanoyl-CoA = propanoyl-CoA + CO2

4.1.1.43 phenylpyruvate decarboxylase phenylpyruvate = phenylacetaldehyde + CO2

4.1.1.45 aminocarboxymuconate-semialdehyde decarboxylase 2-amino-3-(3-oxoprop-2-enyl)-but-2-enedioate = 2-aminomuconate semialdehyde + CO2

4.1.1.47 tartronate-semialdehyde synthase 2 glyoxylate = tartronate semialdehyde + CO2

4.1.1.48 indole-3-glycerol-phosphate synthase 1-(2-carboxyphenylamino)-1-deoxy-D-ribulose 5-phosphate = C1-(3-indolyl)-glycerol 3-phosphate + CO2 + H2O

4.1.1.49 phosphoenolpyruvate carboxykinase (ATP) ATP + oxaloacetate = ADP + phosphoenolpyruvate + CO2

4.1.1.50 adenosylmethionine decarboxylase S-adenosyl-L-methionine = (5-deoxy-5-adenosyl)(3-aminopropyl)-methylsulfonium salt + CO2

4.1.1.53 phenylalanine decarboxylase L-phenylalanine = phenylethylamine + CO2

4.1.1.65 phosphatidylserine decarboxylase phosphatidyl-L-serine = phosphatidylethanolamine + CO2

4.1.1.71 2-oxoglutarate decarboxylase 2-oxoglutarate = succinate semialdehyde + CO2

4.1.2.5 threonine aldolase L-threonine = glycine + acetaldehyde

4.1.2.9 phosphoketolase D-xylulose 5-phosphate + phosphate = acetyl phosphate + D-glyceraldehyde 3-phosphate + H2O

4.1.2.12 ketopantoaldolase 2-hydroxy-2-isopropylbutanedioate = 3-methyl-2-oxobutanoate + formaldehyde

4.1.2.13 fructose-bisphosphate aldolase D-fructose 1,6-bisphosphate = glycerone phosphate + D-glyceraldehyde 3-phosphate

4.1.2.14 2-dehydro-3-deoxy-phosphogluconate aldolase 2-dehydro-3-deoxy-D-gluconate 6-phosphate = pyruvate + D-glyceraldehyde 3-phosphate

4.1.2.15 2-dehydro-3-deoxy-phosphoheptonate aldolase 2-dehydro-3-deoxy-D-arabino-heptonate 7-phosphate + phosphate = phosphoenolpyruvate + D-erythrose 4-phosphate + H2O

4.1.3.1 isocitrate lyase isocitrate = succinate + glyoxylate

4.1.3.2 malate synthase L-malate + CoA = acetyl-CoA + H2O + glyoxylate

4.1.3.4 hydroxymethylglutaryl-CoA lyase (S)-3-hydroxy-3-methylglutaryl-CoA = acetyl-CoA + acetoacetate

4.1.3.5 hydroxymethylglutaryl-CoA synthase (S)-3-hydroxy-3-methylglutaryl-CoA + CoA = acetyl-CoA + H2O + acetoacetyl-CoA

4.1.3.7 citrate (si)-synthase citrate + CoA = acetyl-CoA + H2O + oxaloacetate

4.1.3.8 ATP citrate (pro-S)-lyase ATP + citrate + CoA = ADP + phosphate + acetyl-CoA + oxaloacetate

4.1.3.16 4-hydroxy-2-oxoglutarate aldolase 4-hydroxy-2-oxoglutarate = pyruvate + glyoxylate

4.1.3.18 acetolactate synthase 2-acetolactate + CO2 = 2 pyruvate

4.1.3.20 N-acylneuraminate-9-phosphate synthase N-acylneuraminate 9-phosphate + phosphate = N-acyl-D-mannosamine 6-phosphate + phosphoenolpyruvate + H2O

4.1.3.21 homocitrate synthase 2-hydroxybutane-1,2,4-tricarboxylate + CoA = acetyl-CoA + H2O + 2-oxoglutarate

4.1.3.22 citramalate lyase (3S)-citramalate = acetate + pyruvate

4.1.3.27 anthranilate synthase chorismate + L-glutamine = anthranilate + pyruvate + L-glutamate

4.1.99.1 tryptophanase L-tryptophan + H2O = indole + pyruvate + NH3

4.2.1.2 fumarate hydratase (S)-malate = fumarate + H2O

4.2.1.3 aconitate hydratase citrate = cis-aconitate + H2O

4.2.1.4 citrate dehydratase citrate = cis-aconitate + H2O

4.2.1.9 dihydroxy-acid dehydratase 2,3-dihydroxy-3-methylbutanoate = 3-methyl-2-oxobutanoate + H2O

4.2.1.10 3-dehydroquinate dehydratase 3-dehydroquinate = 3-dehydroshikimate + H2O

4.2.1.11 phosphopyruvate hydratase 2-phospho-D-glycerate = phosphoenolpyruvate + H2O

4.2.1.12 phosphogluconate dehydratase 6-phospho-D-gluconate = 2-dehydro-3-deoxy-6-phospho-D-gluconate + H2O

4.2.1.13 L-serine dehydratase L-serine + H2O = pyruvate + NH3 + H2O

4.2.1.16 threonine dehydratase L-threonine + H2O = 2-oxobutanoate + NH3 + H2O

4.2.1.17 enoyl-CoA hydratase (3S)-3-hydroxyacyl-CoA = trans-2(or 3)-enoyl-CoA + H2O

4.2.1.18 methylglutaconyl-CoA hydratase (S)-3-hydroxy-3-methylglutaryl-CoA = trans-3-methylglutaconyl-CoA + H2O

4.2.1.19 imidazoleglycerol-phosphate dehydratase D-erythro-1-(imidazol-4-yl)glycerol 3-phosphate = 3-(imidazol-4-yl)-2-oxopropyl phosphate + H2O

4.2.1.20 tryptophan synthase L-serine + 1-(indol-3-yl)glycerol 3-phosphate = L-tryptophan + glyceraldehyde 3-phosphate

4.2.1.22 cystathionine beta-synthase L-serine + L-homocysteine = cystathionine + H2O

4.2.1.24 porphobilinogen synthase 2 5-aminolevulinate = porphobilinogen + 2 H2O

4.2.1.33 3-isopropylmalate dehydratase 3-isopropylmalate = 2-isopropylmaleate + H2O

4.2.1.34 (S)-2-methylmalate dehydratase (S)-2-methylmalate = 2-methylfumarate + H2O

4.2.1.36 homoaconitate hydratase 2-hydroxybutane-1,2,4-tricarboxylate = but-1-ene-1,2,4-tricarboxylate + H2O

4.2.1.46 dTDPglucose 4,6-dehydratase dTDPglucose = dTDP-4-dehydro-6-deoxy-D-glucose + H2O

4.2.1.47 GDPmannose 4,6-dehydratase GDPmannose = GDP-4-dehydro-6-deoxy-D-mannose + H2O

4.2.1.49 urocanate hydratase 4,5-dihydro-4-oxo-5-imidazolepropanoate = urocanate + H2O

4.2.1.51 prephenate dehydratase prephenate = phenylpyruvate + H2O + CO2

4.2.1.52 dihydrodipicolinate synthase L-aspartate 4-semialdehyde + pyruvate = dihydrodipicolinate + 2 H2O

4.2.1.55 3-hydroxybutyryl-CoA dehydratase (3R)-3-hydroxybutanoyl-CoA = crotonoyl-CoA + H2O

4.2.1.58 crotonoyl-[acyl-carrier-protein] hydratase (3R)-3-hydroxybutanoyl-[acyl-carrier protein] = but-2-enoyl-[acyl-carrier protein] + H2O

4.2.1.59 3-hydroxyoctanoyl-[acyl-carrier-protein] dehydratase (3R)-3-hydroxyoctanoyl-[acyl-carrier protein] = 2-octenoyl-[acyl-carrier protein] + H2O

4.2.1.60 3-hydroxydecanoyl-[acyl-carrier-protein] dehydratase (3R)-3-hydroxydecanoyl-[acyl-carrier protein] = 2-decenoyl-[acyl-carrier protein] (or 3-decenoyl-[acyl-carrier protein]) + H2O

4.2.1.61 3-hydroxypalmitoyl-[acyl-carrier-protein] dehydratase (3R)-3-hydroxypalmitoyl-[acyl-carrier protein] = 2-hexadecenoyl-[acyl-carrier protein] + H2O

4.2.1.75 uroporphyrinogen-III synthase hydroxymethylbilane = uroporphyrinogen III + H2O

4.2.1.80 2-oxopent-4-enoate hydratase 4-hydroxy-2-oxopentanoate = 2-oxopent-4-enoate + H2O

4.2.3.1 threonine synthase O-phospho-L-homoserine + H2O = L-threonine + phosphate

4.2.3.4 3-dehydroquinate synthase 3-deoxy-arabino-heptulosonate 7-phosphate = 3-dehydroquinate + phosphate

4.2.3.5 chorismate synthase 5-O-(1-carboxyvinyl)-3-phosphoshikimate = chorismate + phosphate

4.2.99.8 cysteine synthase O3-acetyl-L-serine + hydrogen sulfide = L-cysteine + acetate

4.2.99.9 O-succinylhomoserine (thiol)-lyase O-succinyl-L-homoserine + L-cysteine = cystathionine + succinate

4.3.1.1 aspartate ammonia-lyase L-aspartate = fumarate + NH3

4.3.1.2 methylaspartate ammonia-lyase L-threo-3-methylaspartate = mesaconate + NH3

4.3.1.3 histidine ammonia-lyase L-histidine = urocanate + NH3

4.3.1.4 formimidoyltetrahydrofolate cyclodeaminase 5-formimidoyltetrahydrofolate = 5,10-methenyltetrahydrofolate + NH3

4.3.2.1 argininosuccinate lyase N-(L-arginino)succinate = fumarate + L-arginine

4.4.1.1 cystathionine gamma-lyase L-cystathionine + H2O = L-cysteine + NH3 + 2-oxobutanoate

4.4.1.8 cystathionine beta-lyase cystathionine + H2O = L-homocysteine + NH3 + pyruvate

4.6.1.1 adenylate cyclase ATP = 3',5'-cyclic AMP + diphosphate

4.99.1.1 ferrochelatase protoporphyrin + Fe2+ = protoheme + 2 H+

5.1.3.1 ribulose-phosphate 3-epimerase D-ribulose 5-phosphate = D-xylulose 5-phosphate

5.1.3.2 UDPglucose 4-epimerase UDPglucose = UDPgalactose

5.1.3.4 L-ribulose-phosphate 4-epimerase L-ribulose 5-phosphate = D-xylulose 5-phosphate

5.1.3.6 UDPglucuronate 4-epimerase UDPglucuronate = UDP-D-galacturonate

5.1.3.7 UDP-N-acetylglucosamine 4-epimerase UDP-N-acetyl-D-glucosamine = UDP-N-acetyl-D-galactosamine

5.1.3.12 UDPglucuronate 5'-epimerase UDPglucuronate = UDP-L-iduronate

5.1.3.13 dTDP-4-dehydrorhamnose 3,5-epimerase dTDP-4-dehydro-6-deoxy-D-glucose = dTDP-4-dehydro-6-deoxy-L-mannose

5.1.3.14 UDP-N-acetylglucosamine 2-epimerase UDP-N-acetyl-D-glucosamine = UDP-N-acetyl-D-mannosamine

5.1.99.1 methylmalonyl-CoA epimerase (R)-2-methyl-3-oxopropanoyl-CoA = (S)-2-methyl-3-oxopropanoyl-CoA

5.2.1.2 maleylacetoacetate isomerase 4-maleylacetoacetate = 4-fumarylacetoacetate

5.2.1.3 retinal isomerase all-trans-retinal = 11-cis-retinal

5.2.1.7 retinol isomerase all-trans-retinol = 11-cis-retinol

5.3.1.1 triose-phosphate isomerase D-glyceraldehyde 3-phosphate = glycerone phosphate

5.3.1.3 arabinose isomerase D-arabinose = D-ribulose

5.3.1.4 L-arabinose isomerase L-arabinose = L-ribulose

5.3.1.5 xylose isomerase D-xylose = D-xylulose

5.3.1.6 ribose-5-phosphate isomerase D-ribose 5-phosphate = D-ribulose 5-phosphate

5.3.1.8 mannose-6-phosphate isomerase D-mannose 6-phosphate = D-fructose 6-phosphate

5.3.1.9 glucose-6-phosphate isomerase D-glucose 6-phosphate = D-fructose 6-phosphate

5.3.1.15 D-lyxose ketol-isomerase D-lyxose = D-xylulose

5.3.1.16 1-(5-phosphoribosyl)-5-[(5-phosphoribosylamino)methylideneamino)imidazole-4-carboxamide isomerase 1-(5-phosphoribosyl)-5-[(5-phosphoribosylamino)methylideneamino)imidazole-4-carboxamide = 5-[(5-phospho-1-deoxyribulos-1-ylamino)methylideneamino]-1-(5-phosphoribosyl)imidazole-4-carboxamide

5.3.99.5 thromboxane-A synthase (5Z,13E)-(15S)-9alpha,11alpha-epidioxy-15-hydroxyprosta-5,13-dienoate = (5Z,13E)-(15S)-9alpha,11alpha-epoxy-15-hydroxythromba-5,13-dienoate

5.4.2.1 phosphoglycerate mutase 2-phospho-D-glycerate = 3-phospho-D-glycerate

5.4.2.2 phosphoglucomutase D-glucose 1-phosphate = D-glucose 6-phosphate

5.4.2.3 phosphoacetylglucosamine mutase N-acetyl-D-glucosamine 1-phosphate = N-acetyl-D-glucosamine 6-phosphate

5.4.2.8 phosphomannomutase D-mannose 1-phosphate = D-mannose 6-phosphate

5.4.3.8 glutamate-1-semialdehyde 2,1-aminomutase (S)-4-amino-5-oxopentanoate = 5-aminolevulinate

5.4.99.1 methylaspartate mutase L-threo-3-methylaspartate = L-glutamate

5.4.99.2 methylmalonyl-CoA mutase (R)-2-methyl-3-oxopropanoyl-CoA = succinyl-CoA

5.4.99.5 chorismate mutase chorismate = prephenate

5.4.99.7 lanosterol synthase (S)-2,3-epoxysqualene = lanosterol

5.5.1.1 muconate cycloisomerase 2,5-dihydro-5-oxofuran-2-acetate = cis,cis-hexadienedioate

5.5.1.4 inositol-3-phosphate synthase D-glucose 6-phosphate = 1D-myo-inositol 3-phosphate

6.2.1.4 succinate-CoA ligase (GDP-forming) GTP + succinate + CoA = GDP + phosphate + succinyl-CoA

6.2.1.25 benzoate-CoA ligase ATP + benzoate + CoA = AMP + diphosphate + benzoyl-CoA

6.3.1.1 aspartate-ammonia ligase ATP + L-aspartate + NH3 = AMP + diphosphate + L-asparagine

6.3.1.2 glutamate-ammonia ligase ATP + L-glutamate + NH3 = ADP + phosphate + L-glutamine

6.3.1.4 aspartate-ammonia ligase (ADP-forming) ATP + L-aspartate + NH3 = ADP + phosphate + L-asparagine

6.3.1.5 NAD synthase ATP + deamido-NAD + NH3 = AMP + diphosphate + NAD

6.3.2.1 pantoate-beta-alanine ligase ATP + (R)-pantoate + beta-alanine = AMP + diphosphate + (R)-pantothenate

6.3.2.2 glutamate-cysteine ligase ATP + L-glutamate + L-cysteine = ADP + phosphate + gamma-L-glutamyl-L-cysteine

6.3.2.3 glutathione synthase ATP + gamma-L-glutamyl-L-cysteine + glycine = ADP + phosphate + glutathione

6.3.2.6 phosphoribosylaminoimidazolesuccinocarboxamide synthase ATP + 5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxylate + L-aspartate = ADP + phosphate + (S)-2-[5-amino-1-(5-phospho-D-ribosyl)imidazole-4-carboxamido]succinate

6.3.2.7 UDP-N-acetylmuramoyl-L-alanyl-D-glutamate-lysine ligase ATP + UDP-N-acetylmuramoyl-L-alanyl-D-glutamate + L-lysine = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysine

6.3.2.8 UDP-N-acetylmuramate-alanine ligase ATP + UDP-N-acetylmuramate + L-alanine = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanine

6.3.2.9 UDP-N-acetylmuramoylalanine-D-glutamate ligase ATP + UDP-N-acetylmuramoyl-L-alanine + glutamate = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-glutamate

6.3.2.10 UDP-N-acetylmuramoylalanyl-D-glutamyl-lysine-D-alanyl-D-alanine ligase ATP + UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysine + D-alanyl-D-alanine = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysyl-D-alanyl-D-alanine

6.3.2.13 UDP-N-acetylmuramoylalanyl-D-glutamate-2,6-diamino-pimelate ligase ATP + UDP-N-acetylmuramoyl-L-alanyl-D-glutamate + meso-2,6-diaminoheptanedioate = ADP + phosphate + UDP-N-acetylmuramoyl-L-alanyl-D-gamma-glutamyl-meso-2,6-diamino-heptanedioate

6.3.3.1 phosphoribosylformylglycinamidine cyclo-ligase ATP + 2-(formamido)-N1-(5-phospho-D-ribosyl)acetamidine = ADP + phosphate + 5-amino-1-(5-phospho-D-ribosyl)imidazole

6.3.4.1 GMP synthase ATP + xanthosine 5'-phosphate + NH3 = AMP + diphosphate + GMP

6.3.4.2 CTP synthase ATP + UTP + NH3 = ADP + phosphate + CTP

6.3.4.3 formate-tetrahydrofolate ligase ATP + formate + tetrahydrofolate = ADP + phosphate + 10-formyltetrahydrofolate

6.3.4.4 adenylosuccinate synthase GTP + IMP + L-aspartate = GDP + phosphate + N6-(1,2-dicarboxyethyl)-AMP

6.3.4.5 argininosuccinate synthase ATP + L-citrulline + L-aspartate = AMP + diphosphate + N-(L-arginino)succinate

6.3.4.7 ribose-5-phosphate-ammonia ligase ATP + ribose 5-phosphate + NH3 = ADP + phosphate + 5-phosphoribosylamine

6.3.4.13 phosphoribosylamine-glycine ligase ATP + 5-phospho-D-ribosylamine + glycine = ADP + phosphate + N1-(5-phospho-D-ribosyl)glycinamide

6.3.4.16 carbamoyl-phosphate synthase (ammonia) 2 ATP + NH3 + CO2 + H2O = 2 ADP + phosphate + carbamoyl phosphate

6.3.4.17 formate-dihydrofolate ligase ATP + formate + dihydrofolate = ADP + phosphate + 10-formyldihydrofolate

6.3.5.1 NAD synthase (glutamine-hydrolysing) ATP + deamido-NAD + L-glutamine + H2O = AMP + diphosphate + NAD + L-glutamate

6.3.5.2 GMP synthase (glutamine-hydrolysing) ATP + xanthosine 5'-phosphate + L-glutamine + H2O = AMP + diphosphate + GMP + L-glutamate

6.3.5.3 phosphoribosylformylglycinamidine synthase ATP + N2-formyl-N1-(5-phospho-D-ribosyl)glycinamide + L-glutamine + H2O = ADP + phosphate + 2-(formamido)-N1-(5-phospho-D-ribosyl)acetamidine + L-glutamate

6.3.5.4 asparagine synthase (glutamine-hydrolysing) ATP + L-aspartate + L-glutamine = AMP + diphosphate + L-asparagine + L-glutamate

6.3.5.5 carbamoyl-phosphate synthase (glutamine-hydrolysing) 2 ATP + L-glutamine + CO2 + H2O = 2 ADP + phosphate + L-glutamate + carbamoyl phosphate

6.4.1.1 pyruvate carboxylase ATP + pyruvate + HCO3- = ADP + phosphate + oxaloacetate

6.4.1.2 acetyl-CoA carboxylase ATP + acetyl-CoA + HCO3- = ADP + phosphate + malonyl-CoA

6.4.1.3 propionyl-CoA carboxylase ATP + propanoyl-CoA + HCO3- = ADP + phosphate + (S)-methylmalonyl-CoA

6.4.1.4 methylcrotonoyl-CoA carboxylase ATP + 3-methylcrotonoyl-CoA + HCO3- = ADP + phosphate + 3-methylglutaconyl-CoA

8.0.0.1 adrenergic receptor adrenaline + adrenergic receptor = adrenaline-(adrenergic receptor)

8.0.0.2 glucagon receptor glucagon + glucagon receptor = glucagon-(glucagon receptor)

Appendix A2

The data in the Pathway and "Effectors.dat" data files are shown below. Each record in a file contains the following information: the operator code, the operator name and the reaction or regulatory action. They were entered manually into the files. These are tab delimited, so they can be directly input to the Java application. Some of the data were taken from the "User.dat" file above. The pathway data files contain the reactions of the relevant pathways in the correct sequence for the convenience of the user. Including membrane operators in the urea cycle represents a compartment (mitochondrion). In this pathway, some compounds use these operators to pass through the mitochondrial membrane (MM). Cellular compartments are indicated by the prefixes m- for mitochondrial and c- for cytosolic. In the "UreaCycle.dat" file, the membrane operators are assigned to the arbitrary class 9. This is not an internationally accepted code but is defined here for the purpose of the project. The "Trehalose.dat" file was automatically created from the data in the "User.dat" file. In the "Effectors.dat" file, the 'equals' sign (=) is replaced with a colon (:). The function of the colon is to separate activators and inhibitors and to distinguish them from metabolites in an equation. The data in the files were organised with reference to standard textbooks such as Lehninger et al. (1993) and Voet and Voet (1995).

Glycolysis.dat

```
2.7.1.1   hexokinase     ATP + D-glucose = ADP + D-glucose
6-phosphate
5.3.1.9   glucose-6-phosphate isomerase D-glucose 6-
phosphate = D-fructose 6-phosphate
2.7.1.11  6-phosphofructokinase    ATP + D-fructose 6-
phosphate = ADP + D-fructose 1,6-bisphosphate
4.1.2.13  fructose-bisphosphate aldolase     D-fructose 1,6-
bisphosphate = glycerone phosphate + D-glyceraldehyde 3-
phosphate
5.3.1.1   triose-phosphate isomerase    glycerone phosphate
= D-glyceraldehyde 3-phosphate
1.2.1.12  glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)   D-glyceraldehyde 3-phosphate + phosphate
+ NAD = 3-phospho-D-glyceroyl phosphate + NADH2
2.7.2.3   phosphoglycerate kinase  ADP + 3-phospho-D-
glyceroyl phosphate = ATP + 3-phospho-D-glycerate
5.4.2.1   phosphoglycerate mutase  3-phospho-D-glycerate =
2-phospho-D-glycerate
4.2.1.11  phosphopyruvate hydratase     2-phospho-D-
glycerate = phosphoenolpyruvate + H2O
2.7.1.40  pyruvate kinase     ADP + phosphoenolpyruvate =
ATP + enolpyruvate
0.0.0.1   tautomerisation     enolpyruvate = ketopyruvate
```

Gluconeogenesis.dat

```
6.4.1.1   pyruvate carboxylase     ATP + pyruvate + HCO3- =
ADP + phosphate + oxaloacetate
4.1.1.32  phosphoenolpyruvate carboxykinase (GTP) GTP +
oxaloacetate = GDP + phosphoenolpyruvate + CO2
4.2.1.11  phosphopyruvate hydratase     phosphoenolpyruvate
+ H2O = 2-phospho-D-glycerate
5.4.2.1   phosphoglycerate mutase  2-phospho-D-glycerate =
3-phospho-D-glycerate
2.7.2.3   phosphoglycerate kinase  ATP + 3-phospho-D-
glycerate = ADP + 3-phospho-D-glyceroyl phosphate
1.2.1.12  glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)   3-phospho-D-glyceroyl phosphate + NADH2
= D-glyceraldehyde 3-phosphate + phosphate + NAD
5.3.1.1   triose-phosphate isomerase    D-glyceraldehyde 3-
phosphate = glycerone phosphate
4.1.2.13  fructose-bisphosphate aldolase     glycerone
phosphate + D-glyceraldehyde 3-phosphate = D-fructose 1,6-
bisphosphate
3.1.3.11  fructose-bisphosphatase  D-fructose 1,6-
bisphosphate + H2O = D-fructose 6-phosphate + orthophosphate
5.3.1.9   glucose-6-phosphate isomerase D-fructose 6-
phosphate = D-glucose 6-phosphate
3.1.3.9   glucose-6-phosphatase    D-glucose 6-phosphate +
H2O = D-glucose + phosphate
```

PPP.dat (PPP = pentose phosphate pathway)

```
1.1.1.49  glucose-6-phosphate 1-dehydrogenase     D-glucose
6-phosphate + NADP = D-glucono-1,5-lactone 6-phosphate +
NADPH2
3.1.1.31  6-phosphogluconolactonase     D-glucono-1,5-
lactone 6-phosphate + H2O = 6-phospho-D-gluconate
1.1.1.44  phosphogluconate dehydrogenase (decarboxylating)
     6-phospho-D-gluconate + NADP = D-ribulose 5-phosphate +
CO2 + NADPH2
5.1.3.1   ribulose-phosphate 3-epimerase     D-ribulose 5-
phosphate = D-xylulose 5-phosphate
5.3.1.6   ribose-5-phosphate isomerase  D-ribulose 5-
phosphate = D-ribose 5-phosphate
2.2.1.1   transketolase1 D-ribose 5-phosphate + D-xylulose
5-phosphate = sedoheptulose 7-phosphate + D-glyceraldehyde
3-phosphate
2.2.1.2   transaldolase  sedoheptulose 7-phosphate + D-
glyceraldehyde 3-phosphate = D-erythrose 4-phosphate + D-
fructose 6-phosphate
2.2.1.1   transketolase2 D-erythrose 4-phosphate + D-
xylulose 5-phosphate = D-fructose 6-phosphate + D-
glyceraldehyde 3-phosphate
```

UreaCycle.dat

```
9.0.0.1   MM1(in)   c-CO2 = m-CO2
9.0.0.2   MM2(in)   c-NH3 = m-NH3
6.3.4.16  carbamoyl-phosphate synthase (ammonia)  2 m-ATP +
m-NH3 + m-CO2 + m-H2O = 2 ADP + phosphate + carbamoyl
phosphate
2.1.3.3   ornithine carbamoyltransferase     carbamoyl
phosphate + m-L-ornithine = phosphate + m-L-citrulline
9.0.0.3   MM(out)   m-L-citrulline = c-L-citrulline
6.3.4.5   argininosuccinate synthase    c-ATP + c-L-
citrulline + L-aspartate = AMP + diphosphate + N-(L-
arginino)succinate
4.3.2.1   argininosuccinate lyase  N-(L-arginino)succinate =
fumarate + L-arginine
3.5.3.1   arginase  L-arginine + c-H2O = c-L-ornithine +
urea
9.0.0.4   MM3(in)   c-L-ornithine = m-L-ornithine
```

Trehalose.dat

3.2.1.28 alpha,alpha-trehalase alpha,alpha-trehalose + H2O = 2 D-glucose

2.7.1.1 hexokinase ATP + D-glucose = ADP + D-glucose 6-phosphate

5.4.2.2 phosphoglucomutase D-glucose 6-phosphate = D-glucose 1-phosphate

2.7.7.9 UTP-glucose-1-phosphate uridylyltransferase UTP + D-glucose 1-phosphate = diphosphate + UDP-D-glucose

2.4.1.15 alpha,alpha-trehalose-phosphate synthase (UDP-forming) UDP-D-glucose + D-glucose 6-phosphate = UDP + alpha,alpha-trehalose 6-phosphate

3.1.3.12 trehalose-phosphatase alpha,alpha-trehalose 6-phosphate + H2O = alpha,alpha-trehalose + phosphate

Effectors.dat

```
1.1.1.41  isocitrate dehydrogenase (NAD)     m {+}ADP : n {-
}ATP + n {-}NADH
2.7.1.1   hexokinase       : n {-}D-glucose 6-phosphate
2.7.1.11  6-phosphofructokinase    m {+}AMP + m {+}fructose
2,6-bisphosphate : n {-}ATP + n {-}citrate
2.7.1.40  pyruvate kinase       : n {-}ATP + n {-}acetylCoA +
n {-}alanine
3.1.3.11  fructose-bisphosphatase  m {+}citrate : n {-}AMP +
n {-}fructose 2,6-bisphosphate
6.3.4.16  carbamoyl-phosphate synthase (ammonia)  m {+}N-
acetylglutamate :
6.4.1.1   pyruvate carboxylase     m {+}acetylCoA : n {-}ADP
```

Appendix A3

The references in the "SpecRef.txt" text file are shown below. This file contains references that are specific to reactions in the data files. These were taken from the handbook on Enzyme Nomenclature (Webb, 1992) and were manually entered into the file. The PMID numbers were not available at the time of compiling the file; otherwise, the process would have been automated using the EC list. Each record contains the following information: EC number, reference number in the handbook and the reference. A few references (highlighted in blue) contain the symbol '#' instead of the reference number because they were not available in the handbook. Instead, they were taken from the ExplorEnz database (http://www.enzyme-database.org). These were all transferred from previously different EC numbers. Some references (highlighted in red) are not available because the operators (e.g., receptors and membrane operators) are assigned to arbitrary classes. These are not internationally accepted but are defined here for the purpose of the project. General literature references in the "GenRef.txt" file are also shown below. These constitute a subset of the "References" section.

SpecRef.txt

0.0.0.1 Not available

0.0.0.2 Not available

1.1.1.1 571. Branden, G.-I., Jornvall, H., Eklund, H., & Furugren, B. (1975) in The Enzymes, 3rd ed (Boyer, P.D., ed) vol. 11, p.103, Academic Press, New York [1.1.1.1]

1.1.1.1 2328. Jornvall, H. (1977) Eur. J. Biochem. 72, 443 [1.1.1.1]

1.1.1.1 3539. Negelein, E. & Wulff, H.-J. (1937) Biochem. Z. 293, 351 [1.1.1.1]

1.1.1.8 287. Baranowski, T. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 7, p.85, Academic Press, New York [1.1.1.8]

1.1.1.9 796. Chiang, C. & Knight, S.G. (1960) Biochem. Biophys. Res. Commun. 3, 554 [1.1.1.9]

1.1.1.10 1133. Doten, R.C. & Mortlock, R.P. (1985) J. Bacteriol. 161, 529 [1.1.1.10]

1.1.1.14 248. Bailey, J.P. Renz, C. & McGuinness, E.T. (1981) Comp. Biochem. Physiol. 69B, 908 [1.1.1.14]

1.1.1.15 4458. Shaw, D.R.D. (1956) Biochem. J. 64, 394 [1.1.1.15]

1.1.1.19 4569. Sivak, A. & Hoffman-Ostenhof, O. (1961) Biochim. Biophys. Acta 53, 426 [1.1.1.19]

1.1.1.21 192. Attwood, M.A. & Doughty, C.C. (1974) Biochim. Biophys. Acta 370, 358 [1.1.1.21]

1.1.1.22 1155. Druzhinina, T.N., Kusov, Y.Y., Shibaev, V.N., Kochetkov, N.K., Biely, P., Kucar, S, & Bauer, S. (1975) Biochim. Biophys. Acta 381, 301 [1.1.1.22]

1.1.1.23 25. Adams, E. (1954) J. Biol. Chem. 209, 829 [1.1.1.23]

1.1.1.25 270. Balinsky, D. & Davies, D.D. (1961) Biochem. J. 80, 292 [1.1.1.25]

1.1.1.27 1050. Dennis, D. & Kaplan, N.O. (1960) J. Biol. Chem. 235, 810 [1.1.1.27]

1.1.1.28 1050. Dennis, D. & Kaplan, N.O. (1960) J. Biol. Chem. 235, 810 [1.1.1.28]

1.1.1.29 2060. Holzer, H. & Holldorf, A. (1967) Biochem. Z. 329, 292 [1.1.1.29]

1.1.1.30 408. Bergmeyer, H.-U., Gawehn, K., Klotzsch, H., Krebs, H.A., & Williamson, D.H. (1967) Biochem. J. 102, 423 [1.1.1.30]

1.1.1.31 4150. Robinson, W.G. & Coon, M.J. (1957) J. Biol. Chem. 225, 511 [1.1.1.31]

1.1.1.32 4348. Schlesinger, M.J. & Coon, M.J. (1961) J. Biol. Chem. 236, 2421 [1.1.1.32]

1.1.1.34 643. Bucher, N.L.R., Overath, P. & Lynen, F. (1960) Biochim. Biophys. Acta 40, 491 [1.1.1.34]

1.1.1.35 2000. Hillmer, P. & Gottschalk, G. (1974) Biochim. Biophys. Acta 334, 12 [1.1.1.35]

1.1.1.37 278. Banaszak, L.J. & Bradshaw, R.A. (1975) in The Enzymes, 3rd ed (Boyer, P.D., ed) vol. 11, p.369, Academic Press, New York [1.1.1.37]

1.1.1.38 2434. Kaufman, S., Korkes, S. & del Campillo, A. (1951) J. Biol. Chem. 192, 301 [1.1.1.38]

1.1.1.39 4309. Saz, H.J. & Hubbard, J.A. (1957) J. Biol. Chem. 225, 921 [1.1.1.39]

1.1.1.41 1874. Hathaway, J.A. & Atkinson, D.E. (1963) J. Biol. Chem. 238, 2875 [1.1.1.41]

1.1.1.43 1407. Frampton, E.W. & Wood, W.A. (1961) J. Biol. Chem. 236, 2571 [1.1.1.43]

1.1.1.44 1080. Dickens, F. & Glock, G.E. (1951) Biochem. J. 50, 81 [1.1.1.44]

1.1.1.45 1102. Divorsky, P. & Hoffman-Ostenhof, O. (1964) Acta. Biochim. Pol. 11, 269 [1.1.1.45]

1.1.1.49 1254. Engel, H.J., Domschke, W., Alberti, M. & Domagk, G.F. (1969) Biochim. Biophys. Acta 191, 509 [1.1.1.49]

1.1.1.56 2049. Hollmann, S. & Touster, O. (1957) J. Biol. Chem. 225, 87 [1.1.1.56]

1.1.1.59 1048. Den, H., Robinson, W.G. & Coon, M.J. (1959) J. Biol. Chem. 234, 1666 [1.1.1.59]

1.1.1.60 1655. Gotto, A.M. & Kornberg, H.L. (1961) Biochem. J. 81, 273 [1.1.1.60]

1.1.1.61 3580. Nirenberg, M.W. & Jacoby, W.B. (1960) J. Biol. Chem. 235, 954 [1.1.1.61]

1.1.1.79 741. Cartwright, L.M. & Hullin, R.P (1966) Biochem. J. 101, 781 [1.1.1.79]

1.1.1.81 2556. Kleczkowski, L.A. & Randall, D.D. (1988) Biochem. J. 250, 145 [1.1.1.81]

1.1.1.82 1866. Hatch, M.D. & Slack, C.R. (1969) Biochem. Biophys. Res. Commun. 34, 589 [1.1.1.81]

1.1.1.85 668. Burns, R.O., Umbarger, H.E. & Gross, S.R. (1963) Biochemistry 2, 1053 [1.1.1.85]

1.1.1.86 158. Arfin, S.M. & Umbarger, H.E. (1969) J. Biol. Chem. 244, 1118 [1.1.1.86]

1.1.1.95 4807. Sugimoto, E. & Pizer, L.I. (1968) J. Biol. Chem. 243, 2081 [1.1.1.95]

1.1.1.100 3934. Prescot, D.J. & Vagelos, P.R. (1972) Adv. Enzymol. Relat. Areas Mol. Biol. 36, 269 [1.1.1.100]

1.1.1.102 4750. Stoffel, W., Le Kim, D. & Sticht, G. (1968) Hoppe-Seyler's Z. Physiol. Chem. 349, 664 [1.1.1.102]

1.1.1.105 2596. Koen, A.L. & Shaw, C.R. (1966) Biochim. Biophys. Acta 128, 48 [1.1.1.105]

1.1.1.155 4204. Rowley, B. & Tucci, A.F. (1970) Arch. Biochem. Biophys. 141, 499 [1.1.1.155]

1.1.1.157 3015. Madan, V.K., Hilmer, P. & Gottschalk, G. (1973) Eur. J. Biochem. 32, 51 [1.1.1.157]

1.1.1.158 4941. Taku, A. & Anwar, R.A. (1973) J. Biol. Chem. 248, 4971 [1.1.1.158]

1.1.1.169 2525. King, H.L.,Jr, & Wilken, D.R. (1972) J. Biol. Chem. 247, 4096 [1.1.1.169]

1.1.1.204 341. Battelli, M.G. & Lorenzoni, E. (1982) Biochem. J. 207, 133 [1.1.1.204]

1.1.1.205 3018. Magasanik, B., Moyed, H.S. & Gehring, L.B. (1957) J. Biol. Chem. 226, 339 [1.1.1.205]

1.1.3.8 2211. Isherwood, F.A., Mapson, L.W. & Chen, Y.T.(1960) Biochem. J. 76, 157 [1.1.3.8]

1.1.3.15 491. Blanchard, M., Green, D.E., Nocito-Carroll, V. & Ratner, S. (1946) J. Biol. Chem. 163, 137 [1.1.3.15]

1.1.3.22 208. Avis, P.G., Bergel, F. & Bray, R.C. (1955) J. Chem. Soc. (Lond.) 1100 [1.1.3.22]

1.1.99.1 108. Ameyama, M., Shinagawa, E., Matsuchita, K., Takimoto, K., Nakashima, K. & Adachi, O. (1985) Agric. Biol. Chem. 49, 3623 [1.1.99.1]

1.1.99.5 4122. Ringler, R.L. (1961) J. Biol. Chem. 236, 1192 [1.1.99.5]

1.2.1.3 2260. Jakoby, W.B. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 7, p.203, Academic Press, New York [1.2.1.3]

1.2.1.7 1743. Gunsalus, C.F., Stanier, R.Y. & Gunsalus, I.C. (1953) J. Bacteriol. 66, 548 [1.2.1.7]

1.2.1.8 4197. Rothschild, H.A. & Barron, E.S.G. (1954) J. Biol. Chem. 209, 511 [1.2.1.8]

1.2.1.11 476. Black, S. & Wright, N.G. (1955) J. Biol. Chem. 213, 39 [1.2.1.11]

1.2.1.12 726. Caputto, R. & Dixon, M. (1945) Nature (Lond.) 156, 630 [1.2.1.12]

1.2.1.13 587. Brenneman, F.N. & Volk, W.A. (1959) J. Biol. Chem. 234, 2443 [1.2.1.13]

1.2.1.16 2260. Jakoby, W.B. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 7, p.203, Academic Press, New York [1.2.1.16]

1.2.1.18 1886. Hayaishi, O., Nishizuka, Y., Tatibana, M., Takeshita, M. & Kuno, S. (1961) J. Biol. Chem. 236, 781 [1.2.1.18]

1.2.1.19 696. Callewaert, D.M., Rosemblatt, M.S. & Tchen, T.T. (1974) J. Biol. Chem. 249, 1737 [1.2.1.19]

1.2.1.21 993. Davies, D.D. (1960) J. Exp. Bot. 11, 289 [1.2.1.21]

1.2.1.23 3347. Monder, C. (1967) J. Biol. Chem. 242, 4603 [1.2.1.23]

1.2.1.24 61. Albers, R.W. & Koval, G.J. (1961) Biochim. Biophys. Acta 52, 29 [1.2.1.24]

1.2.1.25 3516. Namba, Y., Yoshizawa, K., Ejima, A., Hayashi, T. & Kaneda, T. (1969) J. Biol. Chem. 244, 4437 [1.2.1.25]

1.2.1.26 30. Adams, E. & Rosso, G. (1967) J. Biol. Chem. 242, 1803 [1.2.1.26]

1.2.1.27 4639. Sokatch, J.R., Sanders, L.E. & Marshall, V.P. (1968) J. Biol. Chem. 243, 2500 [1.2.1.27]

1.2.1.28 1743. Gunsalus, C.F., Stanier, R.Y. & Gunsalus, I.C. (1953) J. Bacteriol. 66, 548 [1.2.1.28]

1.2.1.31 697. Calvert, A.F. & Rodwell, V.W. (1966) J. Biol. Chem. 241, 409 [1.2.1.31]

1.2.1.32 2161. Ichiyama, A., Nakamura, S., Kawai, H., Honjo, T., Nishizuka, Y., Hayaishi, O. & Senoh, S. (1965) J. Biol. Chem. 240, 740 [1.2.1.32]

1.2.1.36 3338. Moffa, D.J., Lotspeich, F.J. & Krause, R.F. (1970) J. Biol. Chem. 245, 439 [1.2.1.36]

1.2.1.39 1467. Fujioka, M., Morino, Y. & Wada, H. (1970) Methods Enzymol. 17A, 593 [1.2.1.39]

1.2.1.41 242. Baich, A. (1971) Biochim. Biophys. Acta 244, 129 [1.2.1.41]

1.2.1.52 2198. Inui, H., Miyatake, K., Nakano, Y. & Kitaoka, S. (1984) J. Biochem. (Tokyo) 96, 931 [1.2.1.52]

1.2.3.5 2406. Kasai, T., Suzuki, I. & Asai, T. (1962) Koso Kagaku Shimpojiumii 17, 77 [1.2.3.5]

1.2.4.1 3641. Ochoa, S. (1954) Adv. Enzymol. Relat. Subj. Biochem. 15, 183 [1.2.4.1]

1.2.4.2 3124. Massey, V.(1960) Biochim. Biophys. Acta 38, 447 [1.2.4.2]

1.2.7.1 1284. Evans, M.C.W. & Buchanan, B.B. (1965) Proc. Natl Acad. Sci. U.S.A. 53, 1420 [1.2.7.1]

1.2.7.2 640. Buchanan, B.B. (1969) J. Biol. Chem. 244, 4218 [1.2.7.2]

1.3.1.1 706. Campbell, L.L. (1957) J. Biol. Chem. 227, 693 [1.3.1.1]

1.3.1.2 1439. Fritzon, P. (1960) J. Biol. Chem. 235, 719 [1.3.1.2]

1.3.1.6 2072. Hopgood, M.F. & Walker, D.J. (1969) Aust. J. Biol. Sci. 22, 1413 [1.3.1.6]

1.3.1.8 1122. Dommes, V., Luster, W., Cvetanovic, M. & Kunau, W.-H. (1982) Eur. J. Biochem. 125, 335 [1.3.1.8]

1.3.1.9 4488. Shimakata, T. & Stumpf, P.K. (1982) Arch. Biochem. Biophys. 218, 77 [1.3.1.9]

1.3.1.10 4252. Saito, K.Kawaguchi, A., Okuda, S., Seyama, Y. & Yamakawa, T. (1980) Biochim. Biophys. Acta 618, 202 [1.3.1.10]

1.3.1.13 1510. Gamborg, O.L. & Keeley, F.W. (1966) Biochim. Biophys. Acta 115, 65 [1.3.1.13]

1.3.1.14 1434. Friedmann, H.C. & Vennesland, B. (1958) J. Biol. Chem. 233, 1398 [1.3.1.14]

1.3.1.26 1309. Farkas, W. & Gilvarg, C. (1965) J. Biol. Chem. 240, 4717 [1.3.1.26]

1.3.1.35 3941. Pugh, E.L. & Kates. M. (1975) Biochim. Biophys. Acta 380, 442 [1.3.1.35]

1.3.3.3 343. Battle, A.M., Benson, A. & Rimington, C. (1965) Biochem. J. 97, 731 [1.3.3.3]

1.3.3.4 3909. Poulson, R. (1976) J. Biol. Chem. 251, 3730 [1.3.3.4]

1.3.5.1 1870. Hatefi, Y. (1976) in The Enzymes of Biological Membranes (Martonosi, A., ed.) vol.4, p.3, Wiley, New York [1.3.5.1]

1.3.99.1 1870. Hatefi, Y. (1976) in The Enzymes of Biological Membranes (Martonosi, A., ed.) vol.4, p.3, Wiley, New York [1.3.99.1]

1.3.99.2 377. Beinert, H. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.447, Academic Press, New York [1.3.99.2]

1.3.99.3 377. Beinert, H. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.447, Academic Press, New York [1.3.99.3]

1.3.99.7 431. Besrat, A., Polan, C.E. & Henderson, L.M. (1969) J. Biol. Chem. 244, 1461 [1.3.99.7]

1.3.99.10 231. Bachhawat, B.K., Robinson, W.G. & Coon, M.J. (1956) J. Biol. Chem. 219, 539 [1.3.99.10]

1.4.1.1 3646. O'Connor, R.J. & Halvorson, H. (1961) Biochim. Biophys. Acta 48, 47 [1.4.1.1]

1.4.1.2 1425. Frieden, C. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.3. Academic Press, New York [1.4.1.2]

1.4.1.7 2677. Kretovich, V.L. & Stepanovich, K.M. (1966) Izv. Akad. Nauk S.S.S.R. Ser. Biol. No. 2, 295 [1.4.1.7]

1.4.1.8 2343. Kagan, Z.S., Kretovich, V.L. & Polyakov, V.A. (1966) Enzymologia 30, 343 [1.4.1.8]

1.4.1.9 4280. Sanwal, B.D. & Zink, M.W. (1961) Arch. Biochem. Biophys. 94, 430 [1.4.1.9]

1.4.1.10 1628. Goldman, D.S. & Wagner, M.J. (1962) Biochim. Biophys. Acta 65, 297 [1.4.1.10]

1.4.1.13 3284. Miller, R.E. & Stadtman, E.R. (1972) J. 1.4.3.4247, 7407 [1.4.1.13]

1.4.1.14 522. Boland, M.J. & Benny, A.G.(1977) Eur. J. Biochem. 79, 355 [1.4.1.14]

1.4.1.19 5199. Vackova, K., Mehta, A. & Kutacek, M. (1985) Biol. Plant. 27, 154 [1.4.1.19]

1.4.3.1 1104. Dixon, M. & Kenworthy, p. (1967) Biochim. Biophys. Acta 146, 54 [1.4.3.1]

1.4.3.2 3237. Meister, A. & Wellner, D. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.609, Academic Press, New York [1.4.3.2]

1.4.3.4 497. Blaschko, H. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.8, p.337, Academic Press, New York [1.4.3.4]

1.4.3.6 103. Ameyama, M., Hayashi, M., Matsushita, K., Shinagawa, E. & Adachi, O. (1984) Agric. Biol. Chem. 48, 561 [1.4.3.6]

1.4.3.8 3523. Narrod, S.A. & Jakoby, W.B. (1964) J. Biol. Chem. 239, 2189 [1.4.3.8]

1.4.4.2 2011. Hiraga, K. & Kikuchi, G. (1980) J. Biol. Chem. 255, 11671 [1.4.4.2]

1.5.1.1 3236. Meister, A.Radhakrishnan, A.N. & Buckley, S.D. (1957) J. Biol. Chem. 229, 789 [1.5.1.1]

1.5.1.2 27. Adams, E. & Goldstone, A. (1960) J. Biol. Chem. 235, 3499 [1.5.1.2]

1.5.1.3 490. Blakely, R.L. & MacDougall, B.M. (1961) J. Biol. Chem. 236, 1163 [1.5.1.3]

1.5.1.5 1871. Hatefi, Y., Osborn, M.J., Kay, L.D. & Huennekens, F.M. (1957) J. Biol. Chem. 227, 637 [1.5.1.5]

1.5.1.6 2743. Kutzbach, C. & Stokstad, E.L.R. (1968) Biochem. Biophys. Res. Commun. 30, 111 [1.5.1.6]

1.5.1.7 1468. Fujioka, M. & Nakatani, Y. (1974) Eur. J. Biochem. 25, 301 [1.5.1.7]

1.5.1.8 2144. Hutzler, J. & Dancis, J. (1968) Biochim. Biophys. Acta 158, 62 [1.5.1.8]

1.5.1.9 2145. Hutzler, J. & Dancis, J. (1970) Biochim. Biophys. Acta 206, 205 [1.5.1.9]

1.5.1.10 2317. Jones, E.E. & Broquist, H.P. (1966) J. Biol. Chem. 241, 3430 [1.5.1.10]

1.5.1.12 28. Adams, E. & Goldstone, A. (1960) J. Biol. Chem. 235, 3504 [1.5.1.12]

1.5.1.21 3807. Payton, C.W. & Chang, Y.-F. (1982) J. Bacteriol. 149, 864 [1.5.1.21]

1.5.3.1 1900. Hayashi, S., Nakamura, S. & Suzuki, M. (1980) Biochem. Biophys. Res. Commun. 96, 924 [1.5.3.1]

1.5.99.1 1437. Frisell, W.R. & MacKenzie, C.G. (1962) J. Biol. Chem. 237, 94 [1.5.99.1]

1.5.99.2 1437. Frisell, W.R. & MacKenzie, C.G. (1962) J. Biol. Chem. 237, 94 [1.5.99.2]

1.5.99.3 237. Baginsky, B.L. & Rodwell, V.W. (1967) J. Bacteriol. 94, 1034 [1.5.99.3]

1.5.99.8 4316. Scarpulla, R.C. & Sofer, R.L. (1978) J. Biol. Chem. 253, 5997 [1.5.99.8]

1.6.4.1 4168. Romano, A.H. & Nickerson, W.J. (1954) J. Biol. Chem. 208, 409 [1.6.4.1]

1.6.5.3 1870. Hatefi, Y. (1976) in The Enzymes of Biological Membranes (Martonosi, A., ed.) vol.4, p.3, Wiley, New York [1.6.5.3]

1.6.6.1 1332. Fewson, C.A. & Nicholas, D.J.D. (1961) Biochim. Biophys. Acta 49, 335 [1.6.6.1]

1.6.6.2 3526. Nason. A. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.587, Academic Press, New York [1.6.6.2]

1.6.6.3 3526. Nason. A. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.587, Academic Press, New York [1.6.6.3]

1.6.6.4 702. Cammack, R., Jackson, R.H., Cornish-Bowden, A. & Cole, J.A. (1982) Biochem. J. 207, 333 [1.6.6.4]

1.6.6.8 3005. MacKenzie, J.J. & Sorensen, L.B. (1973) Biochim. Biophys. Acta 327, 282 [1.6.6.8]

1.7.3.3 2951. London, M. & Hudson, P.B. (1956) Biochim. Biophys. Acta 21, 290 [1.7.3.3]

1.7.7.1 2334. Joy, K.W. & Hageman, R.H. (1966) Biochem. J. 100, 263 [1.7.7.1]

1.7.99.4 3526. Nason. A. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.587, Academic Press, New York [1.7.99.4]

1.8.1.3 4819. Sumizu, K. (1962) Biochim. Biophys. Acta 63, 210 [1.8.1.3]

1.8.1.4 3125. Massey, V. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.7, p.275, Academic Press, New York [1.8.1.4]

1.8.2.1 766. Charles, A.M. & Suzuki, I. (1966) Biochim. Biophys. Acta 128, 522 [1.8.2.1]

1.8.3.1 2487. Kessel, D.L., Johnston, J.L., Cohen, H.J. & Rajagopalan, K.V. (1974) Biochim. Biophys. Acta 334, 86 [1.8.3.1]

1.8.7.1 4360. Schmidt, A. & Trebst, A. (1969) Biochim. Biophys. Acta 180, 529 [1.8.7.1]

1.8.99.1 170. Asada, K. (1967) J. Biol. Chem. 242, 3646 [1.8.99.1]

1.9.3.1 2457. Keilin, D. & Hartree, E.F. (1938) Proc. R. Soc. Lond. B Biol. Sci. 125, 171 [1.9.3.1]

1.10.2.1 1288. Everling, F.C. Weis, W. & Staudinger, H. (1969) Hoppe-Seyler's Z. Physiol. Chem. 350, 1485 [1.10.2.1]

1.10.2.2 3097. Marres, C.A.M. & Slater, E.C. (1977) Biochim. Biophys. Acta 462, 531 [1.10.2.2]

1.10.3.3 4697. Staek, G.R. & Dawson, C.R. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.8, p.297, Academic Press, New York [1.10.3.3]

1.10.99.1 2133. Hurt, E. & Hauska, G. (1981) Biochem. J. 117, 591 [1.10.99.1]

1.13.11.1 1882. Hayashi, O. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.8, p.353, Academic Press, New York [1.13.11.1]

1.13.11.2 1882. Hayashi, O. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.8, p.353, Academic Press, New York [1.13.11.2]

1.13.11.5 1882. Hayashi, O. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.8, p.353, Academic Press, New York [1.13.11.5]

1.13.11.6 1882. Hayashi, O. (1963) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.8, p.353, Academic Press, New York [1.13.11.6]

1.13.11.11 564. Brady, F.O., Monaco, M.E., Forman, H.J., Schutz, G. & Feigelson, P. (1972) J. Biol. Chem. 247, 7915 [1.13.11.11]

1.13.11.20 2949. Lombardini, J.B., Singer, T.P. & Boyer, P.D. (1969) J. Biol. Chem. 244, 1172 [1.13.11.20]

1.13.11.21 1642. Goodman, D.S., Huang, H.S., Kanai, M. & Shiratori, T. (1967) J. Biol. Chem. 242, 3543 [1.13.11.21]

1.13.11.27 2919. Lindstedt, S. & Rundgren, M. (1982) J. Biol. Chem. 257, 11922 [1.13.11.27]

1.13.99.1 764. Charlalampous, F.C. (1959) J. Biol. Chem. 234, 220 [1.13.99.1]

1.14.11.1 2918. Lindstedt, G. & Lindstedt, S. (1970) J. Biol. Chem. 245, 4178 [1.14.11.1]

1.14.11.2 412. Berk, R.A. & Prockop, D.J. (1973) J. Biol. Chem. 248, 1175 [1.14.11.2]

1.14.11.8 2126. Hulse, J.D., Ellis, S.R. & Henderson, L.M. (1978) J. Biol. Chem. 253, 1654 [1.14.11.8]

1.14.12.1 2589. Kobayashi, S. & Hayashi, O. (1970) Methods Enzymol. 17A, 505 [1.14.12.1]

1.14.12.10 5578. Yamaguchi, M. & Fujisawa, H. (1978) J. Biol. Chem. 253, 8848 [1.14.12.10]

1.14.13.5 3044. Maki, Y., Yamamoto, S., Nozaki, M. & Hayashi, O. (1969) J. Biol. Chem. 244, 2942 [1.14.13.5]

1.14.13.9 1014. de Castro, F.T., Price, J.M. & Brown, R.R. (1956) J. Am. Chem. Soc. 78, 2904 [1.14.13.9]

1.14.13.12 4044. Reddy, C.C. & Vaidyanathan, C.S. (1975) Biochim. Biophys. Acta 384, 46 [1.14.13.12]

1.14.13.39 582, Bredt, D.S. & Snyder, S.H. (1990) Proc. Natl Acad. Sci. U.S.A. 87, 682 [1.14.13.39]

1.14.16.1 1753. Guroff, G. & Rhoads, C.A. (1969) J. Biol. Chem. 244, 142 [1.14.16.1]

1.14.16.2 1238. El Mestikawy, S., Glowinski, J. & Hamon, M. (1983) Nature (Lond.) 302, 830 [1.14.16.2]

1.14.16.4 1429. Friedman, P.A.Kappelman, A.H. & Kaufman, S. (1972) J. Biol. Chem. 247, 4165 [1.14.16.4]

1.14.17.1 1431. Friedman, S. & Kaufman, S. (1965) J. Biol. Chem. 240, 4763 [1.14.17.1]

1.14.18.1 1002. Dawson, C.R. & Tapley, W.B. (1951) in The Enzymes, 1st edn (Sumner, J.B & Myrback, K,. ed.) vol.2, p.454, Academic Press, New York [1.14.18.1]

1.14.19.1 #. Fulco. A.J. & Bloch, K. (1964) J. Biol. Chem. 239, 993 [1.14.19.1]

1.14.19.3 #. Okayasu, T., Nagao, M., Ishibashi, T., & Imai, Y. (1981) Arch. Biochem. Biophys. 206, 21 [1.14.19.3]

1.14.99.1 1076. DeWitt, D.L. & Smith, W.L. (1988) Proc. Natl Acad. Sci. U.S.A. 85, 1412 [1.14.99.1]

1.14.99.7 889. Corey, E.J., Russey, W.E. & Ortiz de Montellano, P.R. (1966) J. Am. Chem. Soc. 88, 4750 [1.14.99.7]

1.17.4.1 2760. Lammers, M. & Follmann, H. (1983) Struct. Bonding 54, 27 [1.17.4.1]

1.18.6.1 5714. ?

2.1.1.1 719. Cantoni, G.L. (1951) J. Biol. Chem. 189, 203 [2.1.1.1]

2.1.1.2 723, Cantoni, G.L. & Scarano, E. (1954) J. Am. Chem. Soc. 76, 4744 [2.1.1.2]

2.1.1.3 2558. Klee, W.A., Richards, H.H. & Cantoni, G.L. (1961) Biochim. Biophys. Acta 54, 157 [2.1.1.3]

2.1.1.4 216. Axelrod, J. & Weissbach, H. (1961) J. Biol. Chem. 236, 211 [2.1.1.4]

2.1.1.5 2558. Klee, W.A., Richards, H.H. & Cantoni, G.L. (1961) Biochim. Biophys. Acta 54, 157 [2.1.1.5]

2.1.1.6 215. Axelrod, J. & Tomchick, R. (1958) J. Biol. Chem. 233, 702 [2.1.1.6]

2.1.1.10 271. Balish, E. & Shapiro, S.K. (1967) Arch. Biochem. Biophys. 119, 62 [2.1.1.10]

2.1.1.13 670. Burton, E.G. & Sakami, W. (1969) Biochem. Biophys. Res. Commun. 36, 229 [2.1.1.13]

2.1.1.14 1735. Guest, J.R., Friedman, S., Foster, M.A., Tejerina, G. & Woods, D.D. (1964) Biochem. J. 92, 497 [2.1.1.14]

2.1.1.17 2016. Hirata, F., Viveros, O.H., Diliberto, E.J.,Jr, & Axelrod, J. (1978) Proc. Natl Acad. Sci. U.S.A. 75, 1718 [2.1.1.17]

2.1.1.20 510. Blumenstein, J. & Williams, G.R. (1963) Can. J. Biochem. Physiol. 41, 201 [2.1.1.20]

2.1.1.28 213. Axelrod, J. (1962) J. Biol. Chem. 237, 1656 [2.1.1.28]

2.1.1.43 3762. Paik, W.K. & Kim, S. (1970) J. Biol. Chem. 245, 6010 [2.1.1.43]

2.1.1.45 488. Blakely, R.L. (1963) J. Biol. Chem. 238, 2113 [2.1.1.45]

2.1.2.1 54. Akhtar, M. & El-Obeid, H.A. (1972) Biochim. Biophys. Acta 258, 791 [2.1.2.1]

2.1.2.2 1831. Hartman, S.C. & Buchanan, J.M. (1959) J. Biol. Chem. 234, 1812 [2.1.2.2]

2.1.2.3 1831. Hartman, S.C. & Buchanan, J.M. (1959) J. Biol. Chem. 234, 1812 [2.1.2.3]

2.1.2.5 3279. Miller, A. & Waelsch, H. (1957) J. Biol. Chem. 228, 397 [2.1.2.5]

2.1.2.10 3687. Okamora-Ikeda, J., Jujiwara, K. & Motokawa, Y. (1982) J. Biol. Chem. 257, 135 [2.1.2.10]

2.1.3.1 2030. Hoffman, A., Hilpert, W. & Dimroth, P. (1989) Eur. J. Biochem. 179, 645 [2.1.3.1]

2.1.3.2 2968. Lowenstein, J.M. & Cohen, P.P. (1956) J. Biol. Chem. 220, 57 [2.1.3.2]

2.1.3.3 464. Bishop, S.H. & Grisolia, S. (1967) Biochim. Biophys. Acta 139, 344 [2.1.3.3]

2.1.4.1 543. Borsook, H. & Dubnoff, J.W. (1941) J. Biol. Chem. 138, 389 [2.1.4.1]

2.2.1.1 1032. de la Haba, G., Leder, I.G. & Racker, E. (1955) J. Biol. Chem. 214, 409 [2.2.1.1]

2.2.1.2 2083. Horecker, B.L. & Smyrniotis, P.Z. (1955) J. Biol. Chem. 212, 811 [2.2.1.2]

2.3.1.1 2995. Maas, W.K., Novelli, G.D. & Lipmann, F. Proc. Natl Acad. Sci. U.S.A. 39, 1004 [2.3.1.1]

2.3.1.4 985. Davidson, E.A. (1966) Methods Enzymol. 9, 704 [2.3.1.4]

2.3.1.5 815. Chou, T.C. & Lipmann, F. (1952) J. Biol. Chem. 196, 89 [2.3.1.5]

2.3.1.6 414. Berman, R., Wilson, I.B. & Nachmansohn, D. (1953) Biochim. Biophys. Acta 12, 315 [2.3.1.6]

2.3.1.7 767. Chase, J.F.A., Pearson, D.J. & Tubbs, P.K. (1965) Biochim. Biophys. Acta 96, 162 [2.3.1.7]

2.3.1.8 410. Bergmeyer, H.-U., Holz, G., Klotzsch, H. & Lang, G. (1963) Biochem. Z. 338, 114 [2.3.1.8]

2.3.1.9 2987. Lynen, F. & Ochoa, S. (1953) Biochim. Biophys. Acta 12, 299 [2.3.1.9]

2.3.1.12 569. Brady, R.O. & Stadtman, E.R. (1954) J. Biol. Chem. 211, 621 [2.3.1.12]

2.3.1.15 429. Bertrams, M. & Heinz, E. (1981) Plant Physiol. 68, 653 [2.3.1.15]

2.3.1.16 378. Beinert, H., Bock, R.M., Goldman, D.S., Green, D.E., Mahler, H.R., Mii, S., Stansly, P.G. & Wakil, S.J. (1953) J. Am. Chem. Soc. 75, 4111 [2.3.1.16]

2.3.1.20 1704. Grigor, M.R. & Bell, R.M. (1982) Biochim. Biophys. Acta 712, 464 [2.3.1.20]

2.3.1.21 1057. Derrick, J.P., Tubbs, P.K. & Ramsay, R.R. (1986) Biochem. Soc. Trans. 14, 698 [2.3.1.21]

2.3.1.23 380. Bell, R.M. & Coleman, R.A. (1980) Annu. Rev. Biochem. 49, 459 [2.3.1.23]

2.3.1.24 4673. Sribney, M. (1966) Biochim. Biophys. Acta 125, 542 [2.3.1.24]

2.3.1.30 2669. Kredich, N.M. & Tomkins, G.M. (1966) J. Biol. Chem. 241, 4955 [2.3.1.30]

2.3.1.32 3760. Paik, W.K. & Kim, S. (1964) Arch. Biochem. Biophys. 108, 221 [2.3.1.32]

2.3.1.35 4702. Staub, M. & Denes, G. (1966) Biochim. Biophys. Acta 128, 82 [2.3.1.35]

2.3.1.37 461. Bishop, D.F. Henderson, A.S. & Astrin, K.H. (1990) Genomics 7, 207 [2.3.1.37]

2.3.1.38 3934. Prescott, D.J. & Vagelos, P.R. (1972) Adv. Enzymol. Areas Mol. Biol. 36, 269 [2.3.1.38]

2.3.1.39 3934. Prescott, D.J. & Vagelos, P.R. (1972) Adv. Enzymol. Areas Mol. Biol. 36, 269 [2.3.1.39]

2.3.1.41 3934. Prescott, D.J. & Vagelos, P.R. (1972) Adv. Enzymol. Areas Mol. Biol. 36, 269 [2.3.1.41]

2.3.1.46 4202. Rowbury, R.J. & Woods, D.D. (1964) J. Gen. Microbiol. 36, 341 [2.3.1.46]

2.3.1.50 565. Brady, R.N., DiMari, S.J. & Snell, E.E. (1969) J. Biol. Chem. 244, 491 [2.3.1.50]

2.3.1.51 1417. Frentzen, M., Heinz, E., McKeon, T.A. & Stumpf, P.K. (1983) Eur. J. Biochem. 129, 629 [2.3.1.51]

2.3.1.76 1938. Helgerud, P., Peterson, L.B. & Norum, K.R. (1983) J. Clin. Invest. 71, 747 [2.3.1.76]

2.4.1.1 352. Baum, H. & Gilbert, G.A. (1953) Nature (Lond.) 171, 983 [2.4.1.1]

2.4.1.4 1323. Feingold, D.S., Avigad, G. & Hestrin, S. (1957) J. Biol. Chem. 224, 295 [2.4.1.4]

2.4.1.9 446. Bhatia, I.S., Satvanaravana, M.N. & Srinivasan, M. (1955) Biochem. J. 61, 171 [2.4.1.9]

2.4.1.11 76. Algranati, I.D. & Cabib, E. (1960) Biochim. Biophys. Acta 43, 141 [2.4.1.11]

2.4.1.13 206. Avigad, G. & Milner, Y. (1966) Methods Enzymol. 8, 341 [2.4.1.13]

2.4.1.17 513. Bock, K.W., Burchell, B., Dutton, D.J., Hanninen, O., Mulder, G.J., Owens, I.S., Siest, G. & Jephly, T.R. (1983) Biochem. Pharmacol. 32, 953 [2.4.1.17]

2.4.1.21 754. Chambers, J.C. & Elbein, A.D. (1970) Arch. Biochem. Biophys. 138, 620 [2.4.1.21]

2.4.1.22 1364. Fitzgerald, D.K., Brodbeck, U., Kiyosawa, I., Mawal, R., Colvin, B. & Ebner, K.E. (1970) J. Biol. Chem. 245, 2103 [2.4.1.22]

2.4.1.23 840. Cleland, W.W. & Kennedy, E.P. (1960) J. Biol. Chem. 235, 45 [2.4.1.23]

2.4.1.29 754. Chambers, J.C. & Elbein, A.D. (1970) Arch. Biochem. Biophys. 138, 620 [2.4.1.29]

2.4.1.32 1225. Elbein, A.D. (1969) J. Biol. Chem. 244, 1608 [2.4.1.32]

2.4.1.41 4809. Sugiura, M., Kawasaki, T. & Yamashina, I. (1982) J. Biol. Chem. 257, 9501 [2.4.1.41]

2.4.1.62 337. Basu, S., Kaufman, B. & Roseman, S. (1965) J. Biol. Chem. 240, 4115 [2.4.1.62]

2.4.1.68 2953. Longmore, G.D. & Schachter, H. (1982) Carbohydr. Res. 100, 365 [2.4.1.68]

2.4.1.69 336. Basu, S., Basu, M. & Chien, J.L. (1975) J. Biol. Chem. 250, 2956 [2.4.1.69]

2.4.2.1 43. Agarwal, R.P. & Parks, R.E. (1969) J. Biol. Chem. 244, 644 [2.4.2.1]

2.4.2.2 1427. Friedkin, M. & Kalckar, H. (1961) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol.5, p.237, Academic Press, New York [2.4.2.2]

2.4.2.4 1428. Friedkin, M. & Roberts, D. (1954) J. Biol. Chem. 207, 245 [2.4.2.4]

2.4.2.8 1366. Flaks, J.G. (1963) Methods Enzymol. 6, 136 [2.4.2.8]

2.4.2.9 915. Crawford, I., Kornberg, A. & Simms, E.S. (1967) J. Biol. Chem. 226, 1093 [2.4.2.9]

2.4.2.10 2319. Jones, M. E., Kavipurapu, P.R. & Traut, T.W. (1978) Methods Enzymol. 51, 155 [2.4.2.10]

2.4.2.11 2187. Imsande, J. (1961) J. Biol. Chem. 236, 1494 [2.4.2.11]

2.4.2.14 742. Caskey, C.T., Ashton, D.M. & Wyngaarden, J.B. (1964) J. Biol. Chem. 239, 2570 [2.4.2.14]

2.4.2.15 5556. Yamada, E.W. (1961) J. Biol. Chem. 236, 3043 [2.4.2.15]

2.4.2.17 98. Ames, B.N., Martin, R.G. & Garry, B.J. (1961) J. Biol. Chem. 236, 2019 [2.4.2.17]

2.4.2.18 919. Creighton, T.E. & Yanofsky, C. (1970) Methods Enzymol. 17A, 365 [2.4.2.18]

2.4.2.19 1550. Gholson, R.K., Ueda, I., Ogasawara, N. & Henderson, L.M. (1964) J. Biol. Chem. 239, 1208 [2.4.2.19]

2.4.99.1 323. Bartholomew, B.A., Jourdian, G.W. & Roseman, S. (1973) J. Biol. Chem. 248, 5751 [2.4.99.1]

2.4.99.2 4037. Rearick, J.I., Sadler, J.E., Paulson, J.C. & Hill, R.L. (1979) J. Biol. Chem. 254, 4444 [2.4.99.2]

2.4.99.3 4243. Sadler, J.E., Rearick, J.I., & Hill, R.L. (1979) J. Biol. Chem. 254, 5934 [2.4.99.3]

2.4.99.4 4037. Rearick, J.I., Sadler, J.E., Paulson, J.C. & Hill, R.L. (1979) J. Biol. Chem. 254, 4444 [2.4.99.4]

2.4.99.5 3851. Pierenger, J., Keech, S. & Pierenger, R.A. (1981) J. Biol. Chem. 256, 12306 [2.4.99.5]

2.4.99.6 5219. Van den Eijnden, D.H. & Schiphorst, W.E.C.M. (1981) J. Biol. Chem. 256, 3159 [2.4.99.6]

2.4.99.7 403. Bergh, M.L.E., Hooghwinkel, G.J.M. & Van den Eijnden, D.H. (1983) J. Biol. Chem. 258, 7430 [2.4.99.7]

2.4.99.8 1265. Eppler, M.C., Morre, J.D. & Keenan, T.W. (1980) Biochim. Biophys. Acta 619, 332 [2.4.99.8]

2.4.99.9 338. Basu, S., Kaufman, B. & Roseman, S. (1973) J. Biol. Chem. 248, 1388 [2.4.99.9]

2.4.99.10 335. Basu, M., Basu, S., Stoffyn, A. & Stoffyn, P. (1982) J. Biol. Chem. 257, 12765 [2.4.99.10]

2.4.99.11 60. Albarracin, I., Lassaga, F.E. & Caputto, R. (1988) Biochem. J. 254, 559 [2.4.99.11]

2.5.1.1 285. Banthorpe, D.V., Bucknall, G.A., Doonan, H.J., Doonan, S. & Rowan, M.G. (1976) Phytochemistry 15, 91 [2.5.1.1]

2.5.1.6 720. Cantoni, G.L. (1953) J. Biol. Chem. 204, 403 [2.5.1.6]

2.5.1.10 2984. Lynen, F., Agranoff, B.W., Eggerer, H., Henning, V. & Moslein, E.M. (1959) Angew. Chem. 71, 657 [2.5.1.10]

2.5.1.16 1798. Hannonen, P. Janne, J. & Raina, A. (1972) Biochim. Biophys. Acta 289, 225 [2.5.1.16]

2.5.1.19 3365. Morell, H., Clark, M.J., Knowles, P.F. & Sprinson, D.B. (1967) J. Biol. Chem. 242, 82 [2.5.1.19]

2.5.1.21 2742. Kuswick-Rabiega, G. & Rilling, H.C. (1987) J. Biol. Chem. 262, 1505 [2.5.1.21]

2.5.1.22 1983. Hibasami, H., Borchardt, R.T., Chen, S.-Y., Coward, J.K. & Pegg, A.E. (1980) Biochem. J. 187, 419 [2.5.1.22]

2.5.1.29 4246. Sagami, H., Ishii, K. & Ogura, K. (1981) Biochem. Int. 3, 669 [2.5.1.29]

2.5.1.32 1694. Gregonis, D.E. & Rilling, H.C. (1974) Biochemistry 13, 1538 [2.5.1.32]

2.6.1.1 284. Banks, B.E.C. & Vernon, C.A. (1961) J. Chem. Soc. (Lond) 1698, [2.6.1.1]

2.6.1.2 1172. Dumitru, I.F., Iordachescu, D. & Niculescu, S. (1970) Experientia 26, 837 [2.6.1.2]

2.6.1.4 3482. Nakada, H.I. (1964) J. Biol. Chem. 239, 468 [2.6.1.4]

2.6.1.5 714. Canellakis, Z.N. & Cohen, P.P. (1956) J. Biol. Chem. 222, 53 and 63 [2.6.1.5]

2.6.1.6 55. Aki, K., Ogawa, K. & Ichihara, A. (1968) Biochim. Biophys. Acta 159, 276 [2.6.1.6]

2.6.1.9 97. Ames, B.N. & Horecker, B.L. (1956) J. Biol. Chem. 220, 113 [2.6.1.9]

2.6.1.11 66. Albrecht, A. & Vogel, H.J. (1964) J. Biol. Chem. 239, 1872 [2.6.1.11]

2.6.1.13 1345. Fincham, J.R.S. (1953) Biochem. J. 53, 313 [2.6.1.13]

2.6.1.16 1552. Ghosh, S., Blumenthal, H.J., Davidson, E. & Roseman, S. (1960) J. Biol. Chem. 235, 1265 [2.6.1.16]

2.6.1.17 3825. Peterofsky, B. & Gilvarg, C. (1961) J. Biol. Chem. 236, 1432 [2.6.1.17]

2.6.1.18 1886. Hayaishi, O., Nishizuka, Y., Tatibana, M., Takeshita, M. & Kuno, S. (1961) J. Biol. Chem. 236, 781 [2.6.1.18]

2.6.1.19 200. Aurich, H. (1961) Hoppe-Seyler's Z. Physiol. Chem. 326, 25 [2.6.1.19]

2.6.1.22 2354. Kakimoto, Y., Kanazawa, A., Taniguchi, K. & Sano, I. (1968) Biochim. Biophys. Acta 156, 374 [2.6.1.22]

2.6.1.23 1636. Goldstone, A. & Adams, E. (1962) J. Biol. Chem. 237, 3476 [2.6.1.23]

2.6.1.27 1540. George, H. & Gabay, S. (1968) Biochim. Biophys. Acta 167, 555 [2.6.1.27]

2.6.1.32 2342. Kagan, Z.S., Dronov, A.S. & Kretovich, V.L. (1968) Dokl. Akad. Nauk. S.S.S.R. 179, 1236 [2.6.1.32]

2.6.1.36 4626. Soda, K. & Misono, H. (1968) Biochemistry 7, 4110 [2.6.1.36]

2.6.1.39 3141. Matsuda, M. & Ogur, M. (1969) J. Biol. Chem. 244, 3352 [2.6.1.39]

2.6.1.42 55. Aki, K., Ogawa, K. & Ichihara, A. (1968) Biochim. Biophys. Acta 159, 276 [2.6.1.42]

2.6.1.51 795. Cheung, G.P.Rosenblum, I. & Sallach, H.J. (1968) Plant Physiol. 43, 1813 [2.6.1.51]

2.6.1.52 2019. Hirsch, H. & Greenberg, D.M. (1967) J. Biol. Chem. 242, 2283 [2.6.1.52]

2.6.1.65 560. Bozler, G.Robertson, J.M., Ohsugi, M., Hensley, C. & Barker, H.A. (1979) Arch. Biochem. Biophys. 197, 226 [2.6.1.65]

2.6.1.66 1301. Falkinham, J.O.,III, (1979) Mol. Gen. Genet. 176, 147 [2.6.1.66]

2.7.1.1 250. Bailey, K. & Webb, E.C. (1948) Biochem. J. 42, 60 [2.7.1.1]

2.7.1.2 353. Baumann, P. (1969) Biochemistry 8, 5011 [2.7.1.2]

2.7.1.3 890. Cori, G.T., Ochoa, S., Slein, M.W. & Cori, C.F. (1951) Biochim. Biophys. Acta 7, 304 [2.7.1.3]

2.7.1.4 654. Bueding, E. & MacKinnon, J.A. (1955) J. Biol. Chem. 215, 495 [2.7.1.4]

2.7.1.6 728. Cardini, C.E. & Leloir, L.F. (1953) Arch. Biochem. Biophys. 45, 55 [2.7.1.6]

2.7.1.7 654. Bueding, E. & MacKinnon, J.A. (1955) J. Biol. Chem. 215, 495 [2.7.1.7]

2.7.1.11 212. Axelrod, B., Saltman, P., Bandurski, R.S. & Baker, R.S. (1952) J. Biol. Chem. 197, 89 [2.7.1.11]

2.7.1.15 45. Agranoff, B.W. & Brady, R.O. (1956) J. Biol. Chem. 219, 221 [2.7.1.15]

2.7.1.16 664. Burma, D.P. & Horecker, B.L. (1958) J. Biol. Chem. 231, 1039 [2.7.1.16]

2.7.1.17 1985. Hickman, J. & Ashwell, G. (1958) J. Biol. Chem. 232, 737 [2.7.1.17]

2.7.1.19 2138. Hurwitz, J., Weissbach, A., Horecker, B.L. & Smyrniotis, P.Z. (1956) J. Biol. Chem. 218, 769 [2.7.1.19]

2.7.1.23 676. Butler, J.R. & McGuinness, E.T. (1982) Int. J. Biochem. 14, 839 [2.7.1.23]

2.7.1.24 8. Abiko, Y. (1970) Methods Enzymol. 18A, 358 [2.7.1.24]

2.7.1.25 280. Bandurski, R.S., Wilson, L.G. & Squires, C.L. (1956) J. Am. Chem. Soc. 78, 6408 [2.7.1.25]

2.7.1.28 1970. Hers, H.G. & Kusaka, T. (1953) Biochim. Biophys. Acta 11, 427 [2.7.1.28]

2.7.1.30 409. Bergmeyer, H.-U., Holz, G., Kauder, E.M., Mollering, H. & Wieland, O. (1961) Biochem. Z. 333, 471 [2.7.1.30]

2.7.1.31 1135. Doughty, C.C., Hayashi, J.A. & Guenther, H.L. (1966) J. Biol. Chem. 241, 568 [2.7.1.31]

2.7.1.32 1899. Hayashi, S. & Lin, E.C.C. (1967) J. Biol. Chem. 242, 1030 [2.7.1.32]

2.7.1.33 9. Abiko, Y., Ashida, S.-I. & Shimizu, M. (1972) Biochim. Biophys. Acta 268, 364 [2.7.1.33]

2.7.1.34 3623. Novelli, G.D.(1953) Fed. Proc. 12, 675 [2.7.1.34]

2.7.1.36 1942. Hellig, H. & Popjak, G. (1961) J. Lipid Res. 2, 235 [2.7.1.36]

2.7.1.39 1371. Flavin, M. & Slaughter, C. (1960) J. Biol. Chem. 235, 1103 [2.7.1.39]

2.7.1.40 559. Boyer, P.D. (1962) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 6, p.95, Academic Press, New York [2.7.1.40]

2.7.1.47 1441. Fromm, H.J. (1959) J. Biol. Chem. 234, 3097 [2.7.1.47]

2.7.1.53 121. Anderson, R.L. & Wood, W.A. (1962) J. Biol. Chem. 237, 1029 [2.7.1.53]

2.7.1.60 281. Banerjee, S. & Ghosh, S. (1969) Eur. J. Biochem. 8, 200 [2.7.1.60]

2.7.1.64 1256. English, P.D., Dietz, M. & Albersheim, P. (1966) Science (Wash. D.C.) 151, 198 [2.7.1.64]

2.7.1.71 3366. Morell, H. & Sprinson, D.B. (1968) J. Biol. Chem. 243, 676 [2.7.1.71]

2.7.1.80 695. Cagen, L.M. & Friedmann, H.C. (1972) J. Biol. Chem. 247, 3382 [2.7.1.80]

2.7.1.107 961. Daleo, G.R., Piras, M.M. & Piras, R. (1976) Eur. J. Biochem. 68, 339 [2.7.1.107]

2.7.2.1 4724. Stern, J.R. & Ochoa, S. (1951) J. Biol. Chem. 191, 161 [2.7.2.1]

2.7.2.3 211. Axelrod, B. & Bandurski, R.S. (1953) J. Biol. Chem. 204, 939 [2.7.2.3]

2.7.2.4 474. Black, S. (1962) Methods Enzymol. 5, 820 [2.7.2.4]

2.7.2.6 4591. Sly, W.S. & Stadtman, E.R. (1963) J. Biol. Chem. 238, 2639 [2.7.2.6]

2.7.2.11 241. Baich, A. (1969) Biochim. Biophys. Acta 192, 462 [2.7.2.11]

2.7.3.2 1258. Ennor, A.H., Rosenberg, H. & Armstrong, M.D. (1955) Nature (Lond.) 175, 120 [2.7.3.2]

2.7.4.2 506. Bloch, K., Chaykin, S., Phillips, A.H. & de Waard, A. (1959) J. Biol. Chem. 234, 2595 [2.7.4.2]

2.7.4.3 801. Chiga, M. & Plaut, G.W.E. (1960) J. Biol. Chem. 235, 3260 [2.7.4.3]

2.7.4.4 1567. Gibson, D.M., Ayengar, P. & Sanadi, D.R. (1956) Biochim. Biophys. Acta 21, 86 [2.7.4.4]

2.7.4.6 1567. Gibson, D.M., Ayengar, P. & Sanadi, D.R. (1956) Biochim. Biophys. Acta 21, 86 [2.7.4.6]

2.7.4.9 2134. Hurwitz, J. (1959) J. Biol. Chem. 234, 2351 [2.7.4.9]

2.7.4.14 2134. Hurwitz, J. (1959) J. Biol. Chem. 234, 2351 [2.7.4.14]

2.7.6.1 2123. Hughes, D.E. & Williamson, D.H. (1952) Biochem. J. 51, 45 [2.7.6.1]

2.7.7.3 3623. Novelli, G.D.(1953) Fed. Proc. 12, 675 [2.7.7.3]

2.7.7.4 280. Bandurski, R.S., Wilson, L.G. & Squires, C.L. (1956) J. Am. Chem. Soc. 78, 6408 [2.7.7.4]

2.7.7.6 2658. Krakow, J.S. & Ochoa, S. (1963) Methods Enzymol. 6, 11 [2.7.7.6]

2.7.7.7 526. Bollum, F.J. (1960) J. Biol. Chem. 235, 2399 [2.7.7.7]

2.7.7.9 2359. Kalckar, H.M. (1953) Biochim. Biophys. Acta 12, 250 [2.7.7.9]

2.7.7.10 2359. Kalckar, H.M. (1953) Biochim. Biophys. Acta 12, 250 [2.7.7.10]

2.7.7.12 2360. Kalckar, H.M., Braganca, B. & Munch-Peterson, A. (1953) Nature (Lond.) 172, 1038 [2.7.7.12]

2.7.7.13 3426. Munch-Peterson, A. (1955) Arch. Biochem. Biophys. 55, 592 [2.7.7.13]

2.7.7.14 2476. Kennedy, E.P. & Weiss, S.B. (1956) J. Biol. Chem. 222, 193 [2.7.7.14]

2.7.7.15 2476. Kennedy, E.P. & Weiss, S.B. (1956) J. Biol. Chem. 222, 193 [2.7.7.15]

2.7.7.18 2187. Imsande, J. (1961) J. Biol. Chem. 236, 1494 [2.7.7.18]

2.7.7.22 733. Carminatti, H. & Cabib, E. (1961) Biochim. Biophys. Acta 53, 417 [2.7.7.22]

2.7.7.23 3793. Pattabiramin, T.N. & Bachhawat, B.K. (1961) Biochim. Biophys. Acta 50, 129 [2.7.7.23]

2.7.7.24 2644. Kornfeld, S. & Glaser, L. (1961) J. Biol. Chem. 236, 1791 [2.7.7.24]

2.7.7.27 1551. Ghosh, H.P. & Preiss, J. (1966) J. Biol. Chem. 241, 4491 [2.7.7.27]

2.7.7.34 968. Danishefsky, I. & Heritier-Watkins, O. (1967) Biochim. Biophys. Acta 139, 349 [2.7.7.34]

2.7.7.41 740. Carter, J.R. & Kennedy, E.P. (1966) J. Lipid Res. 7, 678 [2.7.7.41]

2.7.7.43 2450. Kean, E.L. & Roseman, S. (1966) J. Biol. Chem. 241, 5643 [2.7.7.43]

2.7.8.1 2476. Kennedy, E.P. & Weiss, S.B. (1956) J. Biol. Chem. 222, 193 [2.7.8.1]

2.7.8.2 852. Coleman, R. & Bell, R.M. (1977) J. Biol. Chem. 252, 3050 [2.7.8.2]

2.7.8.3 2475. Kennedy, E.P. (1962) Methods Enzymol. 5, 484 [2.7.8.3]

2.7.8.5 501. Bleasdale, J.E. & Johnston, J.M.(1982) Biochim. Biophys. Acta 710, 377 [2.7.8.5]

2.7.8.8 2780. Larson, T.J. & Dowhan, W. (1976) Biochemistry 15, 5212 [2.7.8.8]

2.7.8.11 502. Bleasdale, J.E., Wallis, P., MacDonald, P.C. & Johnston, J.M. (1979) Biochim. Biophys. Acta 575, 135 [2.7.8.11]

2.7.8.13 1981. Heydanek, M.G., Jr, & Neuhaus, F.C.(1969) Biochemistry 8, 1474 [2.7.8.13]

2.8.3.1 4682. Stadtman, E.R. (1952) Fed. Proc. 11, 291 [2.8.3.1]

2.8.3.5 1973. Hersh, L.B. & Jencks, W.P. (1967) J. Biol. Chem. 242, 3468 [2.8.3.5]

2.8.3.6 2410. Katagiri, M. & Hayaishi, O. (1957) J. Biol. Chem. 226, 439 [2.8.3.6]

3.1.1.3 2632. Korn, E.D. & Quigley, T.W. (1957) J. Biol. Chem. 226, 833 [3.1.1.3]

3.1.1.4 1114. Doery, H.M. & Pearson, J.E. (1961) Biochem. J. 78, 820 [3.1.1.4]

3.1.1.5 3. Abe, M., Ohno, K. & Sato, R. (1974) Biochim. Biophys. Acta 369, 361 [3.1.1.5]

3.1.1.7 196. Augustinsson, K.-B. (1948) Acta Physiol. Scand. 15, Suppl. 52 [3.1.1.7]

3.1.1.17 602. Brodie, A.F. & Lipmann, F. (1955) J. Biol. Chem. 212, 677 [3.1.1.17]

3.1.1.21 3023. Mahadevan, S., Ayyoub, N.I. & Roels, O.A. (1966) J. Biol. Chem. 241, 57 [3.1.1.21]

3.1.1.24 3720. Ornston, L.N. (1966) J. Biol. Chem. 241, 3787 [3.1.1.24]

3.1.1.28 3024. Mahadevan, S. & Sauer, F. (1969) J. Biol. Chem. 244, 4448 [3.1.1.28]

3.1.1.31 2439. Kawada, M., Kagawa, Y., Takiguchi, H. & Shimazono, N. (1962) Biochim. Biophys. Acta 57, 404 [3.1.1.31]

3.1.1.32 1524. Gatt, S. (1968) Biochim. Biophys. Acta 159, 304 [3.1.1.32]

3.1.2.1 1542. Gergely, J., Hele, P. & Ramakrishnan, C.V. (1952) J. Biol. Chem. 198, 323 [3.1.2.1]

3.1.2.3 1542. Gergely, J., Hele, P. & Ramakrishnan, C.V. (1952) J. Biol. Chem. 198, 323 [3.1.2.3]

3.1.2.4 4091. Rendina, G. & Coon, M.J. (1957) J. Biol. Chem. 225, 523 [3.1.2.4]

3.1.2.11 148. Aragon, J.J. & Lowenstein, J.M. (1983) J. Biol. Chem. 258, 4725 [3.1.2.11]

3.1.3.2 2336. Joyce,, B.K. & Grisolia, S. (1960) J. Biol. Chem. 235, 2278 [3.1.3.2]

3.1.3.3 542. Borkenhagen, L.F. & Kennedy, E.P. (1959) J. Biol. Chem. 234, 849 [3.1.3.3]

3.1.3.4 4614. Smith, S.W., Weiss, S.B. & Kennedy, E.P. (1957) J. Biol. Chem. 228, 915 [3.1.3.4]

3.1.3.5 1738. Gulland, J.M. & Jackson, E.M. (1938) Biochem. J. 32, 597 [3.1.3.5]

3.1.3.9 113. Anchors, J.M. & Karnovsky, N.L. (1975) J. Biol. Chem. 250, 6048 [3.1.3.9]

3.1.3.10 1318. Faulkner, P. (1955) Biochem. J. 60, 590 [3.1.3.10]

3.1.3.11 1223. El-Badry, A.M. (1974) Biochim. Biophys. Acta 333, 366 [3.1.3.11]

3.1.3.15 95. Ames, B.N.(1957) J. Biol. Chem. 226, 583 [3.1.3.15]

3.1.3.25 1217. Eisenberg, P.,Jr, (1967) J. Biol. Chem. 242, 1375 [3.1.3.25]

3.1.3.27 761. Chang, Y.Y. & Kennedy, E.P. (1967) J. Lipid Res. 8, 456 [3.1.3.27]

3.1.3.29 2333. Jourdian, G.W., Swanson, A., Watson, D. & Roseman, S. (1966) Methods Enzymol. 8, 205 [3.1.3.29]

3.1.3.31 165. Arsenis, C. & Touster, O. (1968) J. Biol. Chem. 243, 5702 [3.1.3.31]

3.1.4.2 1003. Dawson, R.M. C. (1956) Biochem. J. 62, 689 [3.1.4.2]

3.1.4.3 1154. Druzhinina, K.V. & Kritzman, M.G. (1952) Biokhimiya 17, 77 [3.1.4.3]

3.1.4.4 186. Astrachan, L. (1973) Biochim. Biophys. Acta 296, 79 [3.1.4.4]

3.1.4.10 79. Allan, D. & Mitchell, R.H. Biochem. J. 142, 591 [3.1.4.10]

3.1.4.12 309. Barnholz, Y., Roitman, A. & Gatt, S. (1966) J. Biol. Chem. 241, 3731 [3.1.4.12]

3.2.1.3 1415. French, D. & Knapp, D.W. (1950) J. Biol. Chem. 187, 463 [3.2.1.3]

3.2.1.21 807. Chinchetru, M.A., Cabezas, J.A. & Calvo, P. (1989) Int. J. Biochem. 21, 469 [3.2.1.21]

3.2.1.23 485. Blakely, J.A. & MacKenzie, S.L. (1969) Can. J. Biochem. 47, 1021 [3.2.1.23]

3.2.1.26 3454. Myrback, K. (1960) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 4, p.379, Academic Press, New York [3.2.1.26]

3.2.1.45 568. Brady, R.O., Kanfer, J.N. & Shapiro, D. (1966) J. Biol. Chem. 240, 39 [3.2.1.45]

3.2.1.46 567. Brady, R.O., Gal, A.E., Kanfer, J.N. & Bradley, R.M. (1965) J. Biol. Chem. 240, 3766 [3.2.1.46]

3.2.1.48 864. Conklin, K.A., Yamashiro, K.M. & Gray, G.M. (1975) J. Biol. Chem. 250, 5735 [3.2.1.48]

3.3.1.1 1031. de la Haba, G. & Cantoni, G.L. (1959) J. Biol. Chem. 234, 603 [3.3.1.1]

3.5.1.1 1783. Halpern, Y.S. & Grossowicz, N. (1957) Biochem. J. 65, 716 [3.5.1.1]

3.5.1.2 2711. Kung, H.-F. & Wagner, C. (1969) J. Biol. Chem. 244, 4136 [3.5.1.2]

3.5.1.6 707. Campbell, L.L.(1960) J. Biol. Chem. 235, 2375 [3.5.1.6]

3.5.1.7 2898. Lieberman, I. & Kornberg, A. (1955) J. Biol. Chem. 212, 909 [3.5.1.7]

3.5.1.8 3672. Ohmura, E. & Hayaishi, O. (1957) J. Biol. Chem. 227, 181 [3.5.1.8]

3.5.1.9 1889. Hayaishi, O. & Stanier, R.Y. (1951) J. Bacteriol. 62, 691 [3.5.1.9]

3.5.1.16 5280. Vogel, H.J. (1953) Proc. Natl Acad. Sci. U.S.A. 39, 578 [3.5.1.16]

3.5.1.18 2524. Kindler, S.H. & Gilvarg, C. (1960) J. Biol. Chem. 235, 3532 [3.5.1.18]

3.5.1.19 3833.Petrack, B., Greengard, P., Craston, A. & Sheppy, F. (1965) J. Biol. Chem. 240, 1725 [3.5.1.19]

3.5.1.20 1996. Hill, D.L. & Chambers, P. (1967) Biochim. Biophys. Acta 148, 435 [3.5.1.20]

3.5.1.22 3631. Nurmikko, V., Salo, E., Hakola, H., Makinen, K. & Snell, E.E. (1966) Biochemistry 5, 399 [3.5.1.22]

3.5.1.23 3576. Nilsson, A. (1969) Biochim. Biophys. Acta 176, 339 [3.5.1.23]

3.5.1.32 4165. Rohr, M. (1968) Mh Chem. 99, 2255 and 2278 [3.5.1.32]

3.5.2.2 606. Brooks, K.P., Jones, E.A., Kim, B.-D. & Sander, E.G. (1983) Arch. Biochem. Biophys. 226, 469 [3.5.2.2]

3.5.2.3 879. Cooper, C. & Wilson, D.W. (1954) Fed. Proc. 13, 194 [3.5.2.3]

3.5.2.5 1379. Florkin, M. & Duchateau-Bosson, G. (1940) Enzymologia 9, 5 [3.5.2.5]

3.5.2.7 4007. Rao, D.R. & Greenberg, D.M. (1961) J. Biol. Chem. 236, 1758 [3.5.2.7]

3.5.2.10 5125. Tsuru, D., Oka, I. & Yoshimoto, T.(1976) Agric. Biol. Chem. 40, 1011 [3.5.2.10]

3.5.3.1 229. Bach, S.J. & Killip, J.D. (1961) Biochim. Biophys. Acta 47, 336 [3.5.3.1]

3.5.3.4 1379. Florkin, M. & Duchateau-Bosson, G. (1940) Enzymologia 9, 5 [3.5.3.4]

3.5.3.5 1227. Elodi, P. & Szorenyi, E,T. (1956) Acta. Physiol. Acad. Sci. (Hung.) 9, 367 [3.5.3.5]

3.5.3.6 3664. Oginsky, E.L. & Gehrig, R.F. (1952) J. Biol. Chem. 198, 799 [3.5.3.6]

3.5.4.1 846. Cohen, S.S. & Barner, H.D. (1957) J. Biol. Chem. 226, 631 [3.5.4.1]

3.5.4.3 2021. Hitchings, G.H. & Falco, E.A. (1944) Proc. Natl Acad. Sci. U.S.A. 30, 294 [3.5.4.3]

3.5.4.6 2356. Kalckar, H.M. (1947) J. Biol. Chem. 167, 461 [3.5.4.6]

3.5.4.9 3956. Rabinowitz, J.C. & Pricer, W.E. (1956) J. Am. Chem. Soc. 78, 4176 [3.5.4.9]

3.5.4.10 1368. Flaks, J.G., Erwin, M.J. & Buchanan, J.M. (1957) J. Biol. Chem. 229, 603 [3.5.4.10]

3.5.4.12 4313. Scarano, E. (1960) J. Biol. Chem. 235, 706 [3.5.4.12]

3.5.4.19 3294. Minson, A.C. & Creaser, E.H. (1969) Biochem. J. 114, 49 [3.5.4.19]

3.5.4.22 4563. Singh, R.M.M. & Adams, E. (1965) J. Biol. Chem. 240, 4344 and 4352 [3.5.4.22]

3.6.1.3 1535. Geider, K. & Hofmann-Berling, H. (1981) Annu. Rev. Biochem. 50, 233 [3.6.1.3]

3.6.1.15 593. Brightwell, R. & Tappel, A.L. (1968) Arch. Biochem. Biophys. 124, 333 [3.6.1.15]

3.6.1.31 4598. Smith, D.W.E & Ames, B.N. (1965) J. Biol. Chem. 240, 3056 [3.6.1.31]

3.7.1.2 870. Connors, W.M. & Stotz, E. (1949) J. Biol. Chem. 178, 881 [3.7.1.2]

3.7.1.3 2263. Jakoby, W.B. & Bonner, D.M. (1953) J. Biol. Chem. 205, 699 [3.7.1.3]

3.7.1.9 1815. Harayama, S., Rekik, M., Wasserfallen, A. & Bairoch, A. (1987) MGG Mol. Gen. Genet. 210, 241 [3.7.1.9]

3.9.1.1 3784. Parvin, R. & Smith, R.A. (1969) Biochemistry 8, 1748 [3.9.1.1]

4.1.1.1 4557. Singer, T.P. & Pensky, J. (1952) J. Biol. Chem. 196, 375 [4.1.1.1]

4.1.1.3 1092. Dimroth, P. (1981) Eur. J. Biochem. 115, 353 [4.1.1.3]

4.1.1.4 996. Davies, R.(1943) Biochem. J. 37, 230 [4.1.1.4]

4.1.1.9 651. Buckner, J.S., Kolattudy, P.E. & Poulose, A.J. (1976) Arch. Biochem. Biophys. 177, 539 [4.1.1.9]

4.1.1.11 5477. Williamson, J.M. & Brown, G.M. (1979) J. Biol. Chem. 254, 8074 [4.1.1.11]

4.1.1.12 2353. Kakimoto, T., Kato, J., Shibitani, T., Nishimura, N. & Chibata, I. (1969) J. Biol. Chem. 244, 353 [4.1.1.12]

4.1.1.15 94. Ambe, L. & Sohonie, K. (1963) Enzymologia 26, 98 [4.1.1.15]

4.1.1.17 3711. Ono, M., Inoue, H., Suzuki, F. & Takeda, Y. (1972) Biochim. Biophys. Acta 284, 285 [4.1.1.17]

4.1.1.20 1049. Denman, R.F., Hoare, D.S. & Work, E. (1955) Biochim. Biophys. Acta 16, 442 [4.1.1.20]

4.1.1.21 2978. Lukens, L.n. & Buchanan, J.M. (1959) J. Biol. Chem. 234, 1799 [4.1.1.21]

4.1.1.22 1266. Epps, H.M.R. (1945) Biochem. J. 39, 42 [4.1.1.22]

4.1.1.23 2319. Jones, M. E., Kavipurapu, P.R. & Traut, T.W. (1978) Methods Enzymol. 51, 155 [4.1.1.23]

4.1.1.25 3201. McGilvery, R.W. & Cohen, P.P (1948) J. Biol. Chem. 174, 813 [4.1.1.25]

4.1.1.28 3201. McGilvery, R.W. & Cohen, P.P (1948) J. Biol. Chem. 174, 813 [4.1.1.28]

4.1.1.29 1737. Guion-Rain, M.C., Portemer, C. & Chatagner, F. (1975) Biochim. Biophys. Acta 384, 265 [4.1.1.29]

4.1.1.31 784. Chen, J.H. & Jones, R.F. (1970) Biochim. Biophys. Acta 214, 318 [4.1.1.31]

4.1.1.32 762. Change, H.-C. & Lane, M.D. (1966) J. Biol. Chem. 241, 2413 [4.1.1.32]

4.1.1.33 506. Bloch, K., Chaykin, S., Phillips, A.H. & de Waard, A. (1959) J. Biol. Chem. 234, 2595 [4.1.1.33]

4.1.1.34 4596. Smiley, J.D. & Ashwell, G. (1961) J. Biol. Chem. 236, 357 [4.1.1.34]

4.1.1.36 617. Brown, G.M. (1958) J. Am. Chem. Soc. 80, 3161 [4.1.1.36]

4.1.1.37 3163. Mauzerall, D. & Granick, S. (1958) J. Biol. Chem. 232, 1141 [4.1.1.37]

4.1.1.38 2944. Lochmuller, H., Wood, H.G & Davis, J.J. (1966) J. Biol. Chem. 241, 5678 [4.1.1.38]

4.1.1.39 555. Bowless, G., Ogren, W.L. & Hageman, R.H. (1971) Biochem. Biophys. Res. Commun. 45, 716 [4.1.1.39]

4.1.1.41 1503. Galivan, J.H. & Allen, S.H.G. (1968) J. Biol. Chem. 243, 1253 [4.1.1.41]

4.1.1.43 172. Asakawa, T., Wada, H. & Yamano, T. (1968) Biochim. Biophys. Acta 170, 375 [4.1.1.43]

4.1.1.45 2161. Ichiyama, A., Nakamura, S., Kawai, H., Honjo, T., Nishizuka, Y., Hayaishi, O. & Senoh, S. (1965) J. Biol. Chem. 240, 740 [4.1.1.45]

4.1.1.47 1747. Gupta, N.K. & Vennesland, B. (1964) J. Biol. Chem. 239, 3787 [4.1.1.47]

4.1.1.48 918. Creighton, T.E. & Yanofsky, C. (1966) J. Biol. Chem. 241, 4616 [4.1.1.48]

4.1.1.49 715. Cannata, J.J.B. (1970) J. Biol. Chem. 245, 792 [4.1.1.49]

4.1.1.50 139. Anton, D.A. & Kutny, R. (1987) J. Biol. Chem. 262, 2817 [4.1.1.50]

4.1.1.53 2963. Lovenberg, W., Weissbach, H. & Udenfriend, S. (1962) J. Biol. Chem. 237, 89 [4.1.1.53]

4.1.1.65 2390. Kanfer, J. & Kennedy, E.P. (1964) J. Biol. Chem. 239, 1720 [4.1.1.65]

4.1.1.71 4483. Shigeoka, S., Onishi, T., Maeda, K., Nakano, Y. & Kitaoka, S. (1986) FEBS Lett. 195, 43 [4.1.1.71]

4.1.2.5 381. Bell, S.C. & Turner, J.M. (1973) Biochem. Soc. Trans. 1, 678 [4.1.2.5]

4.1.2.9 1911. Heath, E.C., Hurwitz, J., Horecker, B.L. & Ginsburg, A. (1958) J. Biol. Chem. 231, 1009 [4.1.2.9]

4.1.2.12 3209. McIntosh, E.N., Purko, M. & Wood, W.A. (1957) J. Biol. Chem. 228, 499 [4.1.2.12]

4.1.2.13 2086. Horecker, B.L., Tsolas, O. & Lai, C.Y. (1972) in The Enzymes, 3rd edn (Boyer, P.D., ed.) vol. 7, p.213, Academic Press, New York [4.1.2.13]

4.1.2.14 3242. Meloche, H.P. & Wood, W.A. (1964) J. Biol. Chem. 239, 3515 [4.1.2.14]

4.1.2.15 4677. Srinivasin, P.R. & Sprinson, D.B. (1959) J. Biol. Chem. 234, 716 [4.1.2.15]

4.1.3.1 3200. McFadden, B.A. & Howes, W.V. (1963) J. Biol. Chem. 238, 1737 [4.1.3.1]

4.1.3.2 1103. Dixon, G.H., Kornberg, H.L. & Lund, P. (1960) Biochim. Biophys. Acta 41, 217 [4.1.3.2]

4.1.3.4 230. Bachhawat, B.K., Robinson, W.G. & Coon, M.J. (1955) J. Biol. Chem. 216, 727 [4.1.3.4]

4.1.3.5 4216. Rudney, H.(1957) J. Biol. Chem. 227, 363 [4.1.3.5]

4.1.3.7 1657. Gottschalk, G. (1969) Eur. J. Biochem. 7, 301 [4.1.3.7]

4.1.3.8 2905. Lill, U., Schreil, A. & Eggerer, H. (1982) Eur. J. Biochem. 125, 645 [4.1.3.8]

4.1.3.16 2728. Kuratomi, K. & Fukunaga, K. (1963) Biochim. Biophys. Acta 78, 617 [4.1.3.16]

4.1.3.18 349. Bauerle, R.H., Freundlich, M., Stormer, F.C. & Umbarger, H.E. (1964) Biochim. Biophys. Acta 92, 142 [4.1.3.18]

4.1.3.20 4184. Roseman, S., Jourdian, G.W., Watson, D. & Rood, R. (1961) Proc. Natl Acad. Sci. U.S.A. 47, 958 [4.1.3.20]

4.1.3.21 4768. Strassman, M. & Ceci, L.N. (1964) Biochem. Biophys. Res. Commun. 14, 262 [4.1.3.21]

4.1.3.22 295. Barker, H.A. (1967) Arch. Microbiol. 59, 4 [4.1.3.22]

4.1.3.27 265. Baker, T. & Crawford, I.P. (1966) J. Biol. Chem. 241, 5577 [4.1.3.27]

4.1.99.1 667. Burns, R.O. & DeMoss, R.D. (1962) Biochim. Biophys. Acta 65, 233 [4.1.99.1]

4.2.1.2 65. Alberty, R.A. (1961) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 5, p.531, Academic Press, New York [4.2.1.2]

4.2.1.3 1083. Dickman, S.R. (1961) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 5, p.495, Academic Press, New York [4.2.1.3]

4.2.1.4 3546. Neilson, N.E. (1955) Biochim. Biophys. Acta 17, 139 [4.2.1.4]

4.2.1.9 2384. Kanamori, M. & Wixom, R.L.(1963) J. Biol. Chem. 238, 998 [4.2.1.9]

4.2.1.10 3305. Mitsuhashi, S. & Davis, B.D. (1954) Biochim. Biophys. Acta 15, 54 [4.2.1.10]

4.2.1.11 2057. Holt, A. & Wold, F. (1961) J. Biol. Chem. 236, 3227 [4.2.1.11]

4.2.1.12 3241. Meloche, H.P. & Wood, W.A. (1964) J. Biol. Chem. 239, 3505 [4.2.1.12]

4.2.1.13 3487. Nakagawa, H., Kimura, H. & Suda, M. (1968) in Symposium on Pyridoxal Enzymes, 3rd edn, p.101 Wiley, New York [4.2.1.13]

4.2.1.16 848. Cohn, M.S. & Phillips, A.T. (1974) Biochemistry 13, 1208 [4.2.1.16]

4.2.1.17 3396. Moskowitz, G.J. & Merrick, J.M. (1969) Biochemistry 8, 2748 [4.2.1.17]

4.2.1.18 2003. Hilz, H., Knappe, J., Ringelmann, E. & Lynen, F. (1958) Biochem. Z. 329, 476 [4.2.1.18]

4.2.1.19 96. Ames, B.N. (1957) J. Biol. Chem. 228, 131 [4.2.1.19]

4.2.1.20 916. Crawford, I.P. & Yanofsky, C. (1958) Proc. Natl Acad. Sci. U.S.A. 44, 1161 [4.2.1.20]

4.2.1.22 575. Braunstein, A.E., Goryachinkova, E.V., Tolosa, E.A., Willhardt, I.H. & Yefremova, L.L. (1971) Biochim. Biophys. Acta 242, 247 [4.2.1.22]

4.2.1.24 1572. Gibson, K.D., Neuberger, A. & Scott, J.J. (1955) Biochem. J. 61, 618 [4.2.1.24]

4.2.1.33 1727. Gross, S.R., Burns, R.O. & Umbarger, H.E. (1963) Biochemistry 2, 1046 [4.2.1.33]

4.2.1.34 482. Blair, A.H. & Barker, H.A. (1966) J. Biol. Chem. 241, 400 [4.2.1.34]

4.2.1.36 4770. Strassman, M. & Ceci, L.N. (1966) J. Biol. Chem. 241, 5401 [4.2.1.36]

4.2.1.46 1578. Gilbert, J.M., Matsuhashi, M. & Strominger, J.L. (1965) J. Biol. Chem. 240, 1305 [4.2.1.46]

4.2.1.47 1226. Elbein, A.D. & Heath, E.C. (1965) J. Biol. Chem. 240, 1926 [4.2.1.47]

4.2.1.49 1539. George, D.J. & Phillips, A.T. (1970) J. Biol. Chem. 245, 528 [4.2.1.49]

4.2.1.51 752. Cerutti, P. & Guroff, G. (1965) J. Biol. Chem. 240, 3034 [4.2.1.51]

4.2.1.52 4467. Shedlarski, J.G. & Gilvarg, C. (1970) J. Biol. Chem. 245, 1362 [4.2.1.52]

4.2.1.55 3396. Moskowitz, G.J. & Merrick, J.M. (1969) Biochemistry 8, 2748 [4.2.1.55]

4.2.1.58 3042. Majerus, P.W., Alberts, A.W. & Vagelos, P.R. (1965) J. Biol. Chem. 240, 618 [4.2.1.58]

4.2.1.59 3324. Mizugaki, M., Swindell, A.C. & Wkil, S.J. (1968) Biochem. Biophys. Res. Commun. 33, 520 [4.2.1.59]

4.2.1.60 596. Brock, D.J.H., Kass, L.R. & Bloch, K. (1967) J. Biol. Chem. 242, 4432 [4.2.1.60]

4.2.1.61 3324. Mizugaki, M., Swindell, A.C. & Wkil, S.J. (1968) Biochem. Biophys. Res. Commun. 33, 520 [4.2.1.61]

4.2.1.75 342. Battersby, A.R., Fookes, C.J.R., Matcham, G.W.J. & McDonald, E. (1980) Nature (Lond.) 285, 17 [4.2.1.75]

4.2.1.80 2717. Kunz, D.A., Ribbons, D.W. & Chapman, P.J. (1981) J. Bacteriol. 148, 72 [4.2.1.80]

4.2.3.1 #. Flavin, M. & Slaughter, C. (1960) J. Biol. Chem. 235, 1103 [4.2.3.1]

4.2.3.4 #. Rotenberg, S.L. & Sprinson, D.B. (1970) Proc. Natl Acad. Sci. U.S.A. 67, 1669 [4.2.3.4]

4.2.3.5 #. Gaertner, F.H. & Cole. K.W. (1973) 248, 4602 [4.2.3.5]

4.2.99.8 366. Becker, M.A., Kredich, N.M. & Tomkins, G.M. (1969) J. Biol. Chem. 244, 2418 [4.2.99.8]

4.2.99.9 1373. Flavin, M. & Slaughter, C. (1967) Biochim. Biophys. Acta 132, 400 [4.2.99.9]

4.3.1.1 1230. Ellfolk, N. (1953) Acta Chem. Scand. 7, 824 [4.3.1.1]

4.3.1.2 299. Barker, H.A., Smyth, R.O., Wawszkiewicz, E.J., Lee, M.N. & Wilson, R.M. (1958) Arch. Biochem. Biophys. 78, 468 [4.3.1.2]

4.3.1.3 3226. Mehler, A.H. & Tabor, H. (1953) J. Biol. Chem. 201, 775 [4.3.1.3]

4.3.1.4 3957. Rabinowitz, J.C. & Pricer, W.E. (1956) J. Am. Chem. Soc. 78, 5702 [4.3.1.4]

4.3.1.8 342. Battersby, A.R., Fookes, C.J.R., Matcham, G.W.J. & McDonald, E. (1980) Nature (Lond.) 285, 17 [4.3.1.8]

4.3.2.1 1001. Davison, D.C. & Elliott, W.H. (1952) Nature (Lond.) 169, 313 [4.3.2.1]

4.3.2.2 739. Carter, C.E. & Cohen, L.H. (1956) J. Biol. Chem. 222, 17 [4.3.2.2]

4.4.1.1 573. Braunstein, A.E. & Azarkh, R.M. (1950) Dokl. Akad. Nauk. S.S.S.R. 71, 93 [4.4.1.1]

4.4.1.8 116. Anderson, N.W. & Thompson, J.F. (1979) Phytochemistry 18, 1953 [4.4.1.8]

4.6.1.1 2017. Hirata, M. & Hayaishi, O. (1967) Biochim. Biophys. Acta 149, 1 [4.6.1.1]

4.99.1.1 507. Bloomer, J.R., Hill, H.D., Morton, K.O., Anderson-Burnham, L.A. & Straka, J.G. (1987) J. Biol. Chem. 262, 667 [4.99.1.1]

5.1.3.1 179. Ashwell, G. & Hickman, J. (1957) J. Biol. Chem. 226, 65 [5.1.3.1]

5.1.3.2 2834. Leloir, L.F. (1953) Adv. Enzymol. Relat. Subj. Biochem. 14, 193 [5.1.3.2]

5.1.3.4 665. Burma, D.P. & Horecker, B.L. (1958) J. Biol. Chem. 231, 1053 [5.1.3.4]

5.1.3.6 1324. Feingold, D.S., Neufeld, E.F. & Hassid, W.Z. (1960) J. Biol. Chem. 235, 910 [5.1.3.6]

5.1.3.7 1595. Glaser, L. (1959) J. Biol. Chem. 234, 2801 [5.1.3.7]

5.1.3.12 2245. Jacobsen, B. & Davidson, E.A. (1962) J. Biol. Chem. 237, 638 [5.1.3.12]

5.1.3.13 1526. Gaugler, R.W. & Gabriel, O. (1973) J. Biol. Chem. 248, 6041 [5.1.3.13]

5.1.3.14 2510. Kikuchi, K. & Tsuiki, S. (1973) Biochim. Biophys. Acta 327, 193 [5.1.3.14]

5.1.99.1 3178. Mazumder, R., Sasakawa, T., Kaziro, Y. & Ochoa, S. (1962) J. Biol. Chem. 237, 3065 [5.1.99.1]

5.2.1.2 1206. Edwards, S.W. & Knox, W.E. (1956) J. Biol. Chem. 220, 79 [5.2.1.2]

5.2.1.3 2116. Hubbard, R. (1956) J. Gen. Physiol. 39, 935 [5.2.1.3]

5.2.1.7 422. Bernstein, P.S., Law, W.C. & Rando, R.R. (1987) Proc. Natl Acad. Sci. U.S.A. 84, 1849 [5.2.1.7]

5.3.1.1 3267. Meyerhof, O. & Beck, L.V. (1944) J. Biol. Chem. 156, 109 [5.3.1.1]

5.3.1.3 845. Cohen, S.S. (1953) J. Biol. Chem. 201, 71 [5.3.1.3]

5.3.1.4 1910. Heath, E.C., Horecker, B.L., Smyrniotis, P.Z. & Takagi, Y. (1958) J. Biol. Chem. 231, 1031 [5.3.1.4]

5.3.1.5 2027. Hochster, R.M. & Watson, R.W. (1954) Arch. Biochem. Biophys. 48, 120 [5.3.1.5]

5.3.1.6 2138. Hurwitz, J., Weissbach, A., Horecker, B.L. & Smyrniotis, P.Z. (1956) J. Biol. Chem. 218, 769 [5.3.1.6]

5.3.1.8 635. Bruns, F.H., Noltmann, E. & Willemsen, A. (1958) Biochem. Z. 330, 411 [5.3.1.8]

5.3.1.9 244. Baich, A., Wolfe, R.G. & Reithel, F.J. (1960) J. Biol. Chem. 235, 3130 [5.3.1.9]

5.3.1.15 119. Anderson, R.L. & Allison, D.P. (1965) J. Biol. Chem. 240, 2367 [5.3.1.15]

5.3.1.16 3083. Margolies, M.N. & Goldberger, R.F. (1966) J. Biol. Chem. 241, 3262 [5.3.1.16]

5.3.99.5 4472. Shen, R.-F. & Tai, H.-H. (1986) J. Biol. Chem. 261, 11592 [5.3.99.5]

5.4.2.1 1711. Grisolia, S. (1962) Methods Enzymol. 5, 236 [5.4.2.1]

5.4.2.2 2331. Joshi, J.G. & Handler, P. (1964) J. Biol. Chem. 239, 2741 [5.4.2.2]

5.4.2.3 731. Carlson, D.M. (1966) Methods Enzymol. 8, 183 [5.4.2.3]

5.4.2.8 4592. Small, D.M. & Matheson, N.K. (1979) Phytochemistry 18, 1147 [5.4.2.8]

5.4.3.8 1660. Gough, S.P. & Kannangara, C.G. Carlsberg Res. Commun. 43, 185 [5.4.3.8]

5.4.99.1 298. Barker, H.A., Rooze, V., Suzuki, F. & Iodice, A.A. (1964) J. Biol. Chem. 239, 3260 [5.4.99.1]

5.4.99.2 296. Barker, H.A., (1972) in The Enzymes, 3rd ed (Boyer, P.D., ed) vol. 6, p.509, Academic Press, New York [5.4.99.2]

5.4.99.5 901. Cotton, R.G.H. & Gibson, F. (1965) Biochim. Biophys. Acta 100, 76 [5.4.99.5]

5.4.99.7 1010. Dean, P.D.G., Oritz de Montellano, P.R., Bloch, K. & Corey, E.J. (1967) J. Biol. Chem. 242, 3014 [5.4.99.7]

5.5.1.1 3721. Ornston, L.N. (1966) J. Biol. Chem. 241, 3795 [5.5.1.1]

5.5.1.4 1217. Eisenberg, P.,Jr, (1967) J. Biol. Chem. 242, 1375 [5.5.1.4]

6.2.1.4 1764. Hager, L.P.(1962) in The Enzymes, 2nd edn (Boyer, P.D., Lardy, H. & Myrback, K,. ed.) vol. 6, p.387, Academic Press, New York [6.2.1.4]

6.2.1.25 2140. Hutber, G.N. & Ribbons, D.W. (1983) J. Gen. Microbiol. 129, 2413 [6.2.1.25]

6.3.1.1 4020. Ravel, J.M., Norton, S.J., Humphreys, J.S. & Shive, W. (1962) J. Biol. Chem. 237, 2845 [6.3.1.1]

6.3.1.2 1234. Elliott, W.H. (1953) J. Biol. Chem. 201, 661 [6.3.1.2]

6.3.1.4 3474. Nair, P.M. (1969) Arch. Biochem. Biophys. 133, 208 [6.3.1.4]

6.3.1.5 4662. Spencer, R.L. & Preiss, J. (1967) 3066.385 [6.3.1.5]

6.3.2.1 1584. Ginoza, H.S. & Altenbern, R.A. (1955) Arch. Biochem. Biophys. 56, 537 [6.3.2.1]

6.3.2.2 3066. Mandeles, S. & Bloch, K. (1955) J. Biol. Chem. 214, 639 [6.3.2.2]

6.3.2.3 2790. Law, M.Y. & Halliwell, B. (1986) Plant Sci. (Limerick, Ireland) 43, 185 [6.3.2.3]

6.3.2.5 618. Brown, G.M. (1959) J. Biol. Chem. 234, 370 [6.3.2.5]

6.3.2.6 2978. Lukens, L.n. & Buchanan, J.M. (1959) J. Biol. Chem. 234, 1799 [6.3.2.6]

6.3.2.7 2223. Ito, E. & Strominger, J.L. (1962) J. Biol. Chem. 237, 2689 [6.3.2.7]

6.3.2.8 2223. Ito, E. & Strominger, J.L. (1962) J. Biol. Chem. 237, 2689 [6.3.2.8]

6.3.2.9 223. Ito, E. & Strominger, J.L. (1962) J. Biol. Chem. 237, 2689 [6.3.2.9]

6.3.2.10 2224. Ito, E. & Strominger, J.L. (1962) J. Biol. Chem. 237, 2696 [6.3.2.10]

6.3.2.13 3329. Mizuno, Y. & Ito, E. (1968) J. Biol. Chem. 243, 2665 [6.3.2.13]

6.3.3.1 2862. Levenberg, B. & Buchanan, J.M. (1957) J. Biol. Chem. 224, 1005 and 1018 [6.3.3.1]

6.3.4.1 3404. Moyed, H.S. & Magasanik, B. (1957) J. Biol. Chem. 226, 351 [6.3.4.1]

6.3.4.2 2894. Lieberman, I. (1956) J. Biol. Chem. 222, 765 [6.3.4.2]

6.3.4.3 2253. Jaenicke, L. & Brode, E. (1961) Biochem. Z. 334, 108 [6.3.4.3]

6.3.4.4 982. Davey, C.L. (1959) Nature (Lond.) 183, 995 [6.3.4.4]

6.3.4.5 4016. Ratner, S.(1954) Adv. Enzymol. Relat. Subj. Biochem. 15, 319 [6.3.4.5]

6.3.4.7 4049. Reem, G.H. (1968) J. Biol. Chem. 243, 5695 [6.3.4.7]

6.3.4.13 1637. Goldthwait, D.A., Peabody, R.A. & Greenberg, G.R. (1956) J. Biol. Chem. 221, 569 [6.3.4.13]

6.3.4.16 1292. Fahien, L.A. & Cohen, P.P. (1964) J. Biol. Chem. 239, 1925 [6.3.4.16]

6.3.4.17 1144. Drake, J.C., Baram, J. & Allegra, C.J. (1990) Biochem. Pharmacol. 39, 615 [6.3.4.17]

6.3.5.1 2187. 2187. Imsande, J. (1961) J. Biol. Chem. 236, 1494 [6.3.5.1]

6.3.5.2 12. Abrams, R. & Bentley, M. (1959) Arch. Biochem. Biophys. 79, 91 [6.3.5.2]

6.3.5.3 3238. Melnick, I. & Buchanan, J.M. (1957) J. Biol. Chem. 225, 157 [6.3.5.3]

6.3.5.4 3798. Patterson, M.K.,Jr, & Orr, G.R. (1968) J. Biol. Chem. 243, 376 [6.3.5.4]

6.3.5.5 118. Anderson, P.M. & Meister, A. (1965) Biochemistry 4, 2803 [6.3.5.5]

6.4.1.1 3181. McClure, W.R., Lardy, H.A. & Kneifel, H.P. (1971) J. Biol. Chem. 246, 3569 [6.4.1.1]

6.4.1.2 1867. Hatch, M.D. & Stumpf, P.K. (1961) J. Biol. Chem. 236, 2879 [6.4.1.2]

6.4.1.3 2448. Kaziro, Y., Ochoa, S., Warner, R.C. & Chen, J. (1961) J. Biol. Chem. 236, 1917 [6.4.1.3]

6.4.1.4 2572. Knappe, J., Schlegel, H.-G. & Lynen, F. (1961) Biochem. Z. 335, 101 [6.4.1.4]

8.0.0.1 Not available

8.0.0.2 Not available

9.0.0.1 Not available

9.0.0.2 Not available

9.0.0.3 Not available

9.0.0.4 Not available

GenRef.txt

Campbell, M.K. (1995). Biochemistry. Saunders College Publishing, Orlando.

Campbell, N.A. (1996). Biology. Benjamin/Cummings, Menlo Park, California.

Coffee, C.J. (1998). Metabolism. Fence Creek Publishing, Madison, Connecticut.

Givan, C.V. (1999). Evolving concepts in plant glycolysis: two centuries of progress. Biol. Rev. 74, 277 – 309.

Heinrich, R., Schuster, S. (1996). The Regulation of Cellular Systems. Chapman & Hall, New York.

Kahn, D., Westerhoff, H.V. (1991). Control Theory of Regulatory Cascades. J. Theor. Biol. 153, 255 – 285.

Kanehisa, M., Goto, S., Kawashima, S., Nakaya, A. (2002). The KEGG databases at GenomeNet.Nucleic Acids Res., 30, 42 – 46.

Karp, P.D., Riley, M., Paley, S.M., Pellegrini-Toole, A., Krummenacker, M. (1998). EcoCyc: Encyclopaedia of Escherichia coli genes and metabolism. Nucleic Acids Res., 26, 50 – 53.

Karp, P.D., Riley, M., Paley, S.M., Pellegrini-Toole, A. (2002). The MetaCyc Database. Nucleic Acids Res., 30, 49 – 61.

Klipp, E., Herwig, R., Kowald, A., Wierling, C., Lehrach, H. (2005). Systems Biology in Practice. WILEY-VCH,

Weinheim.

Krishnamurthy, L., Nadeau, J., Ozsoyoglu, G., Ozsoyoglu, M., Schaeffer, M., Tasan, M., Xu, W. (2003). Pathways database system: an integrated system for biological pathways. Bioinformatics 19, 930–937.

Lehninger, A.L., Nelson, D.L., Cox, M.M. (1993). Principles of Biochemistry, Worth, New York.

Michal, G. (Ed.), (1999). Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology. Wiley, New York.

Murray, R.K., Granner, D.K., Mayes, P.A., Rodwell, V.W. (2003). Harper's Illustrated Biochemistry, twenty-sixth edition. McGraw-Hill, New York.

Nicholson, D.E. (2000). Metabolic pathways chart (21st Edition). International Union of Biochemistry and Molecular Biology, www.iubmb.unibe.ch/.

Opperdoes, F.R., Borst, P. (1977). Localization of nine glycolytic enzymes in a microbody-like organelle in Trypanosoma Brucei: the glycosome. FEBS Lett. 80, 360 – 364.

Palmer, T. (1995). Understanding Enzymes. Prentice Hall/Ellis Horwood, London.

Papin J.A., Price N.D., Wilback S.J., Fell D.A., Palsson B.O. (2003). Metabolic pathways in the post-genome era. Trends Biochem. Sci. 28, 250–258.

Pfeiffer, T., Sanchez-Valdenebro, I., Nuño, J.C., Montero, F., Schuster, S. (1999). METATOOL: for studying metabolic networks. Bioinformatics 15, 251 – 257.

Rashidi, H.H., Buehler, L.K. (2000). Bioinformatics Basics: Applications in Biological Science and Medicine. CRC Press LLC, Boca Raton, Florida.

Salamonsen, W.B., Mok, K.Y.C., Kolatkar, P., Subbiah, S. (1999). BioJAKE: A tool for the creation, visualisation and manipulation of metabolic pathways. Pacific Symposium on Biocomputing. Hawaii, pp. 392 – 400.

Schomberg, I., Chang, A., Schomberg, D. (2002). BRENDA, enzyme data and metabolic information. Nucleic Acids Res., 30, 47 – 49.

Schuster, S., Hilgetag, C. (1994). On elementary flux modes in biochemical reaction systems at steady state. J. Biol. Syst. 2, 165 – 182.

Selkov, E., Maltsev, N., Olsen, G.J., Overbeek, R., Whitman, W.B. (1997). A reconstruction of the metabolism of Methanococcus jannaschii from sequence data. Gene 197, GC11 – 26.

Sirava, M., Schafer, T. Eiglsperger, M., Kaufmann, M., Kohlbacher, O., Bornberg-Bauer, E., Lenhof, H. (2002). BioMiner – modelling, analyzing, and visualizing biochemical pathways and networks. Bioinformatics 18, S219 – S230.

Sprott, G.D., Ekiel, I., Patel, G.B. (1993). Metabolic pathways in Methanococcus jannaschii and other

methanogenic bacteria. Appl. Environ. Microbiol. 59, 1092 – 1098.

Stephanopulous, G.N., Aristidou, A.A., Nielsen, J. (1998). Metabolic Engineering. Academic Press, San Diego.

Tipton, K.F., O'Sullivan, M.I., Davey, G.P., O'Sullivan, J. (2003). It can be a complicated life being an enzyme. Biochem. Soc. Trans. 31, 711 – 715.

Voet, D., Voet, J.G. (1995). Biochemistry. Wiley, New York.

Voit, E.O. (2000). Computational Analysis of Biochemical Systems. Cambridge University Press, New York.

Webb, E.C. (1992). Enzyme Nomenclature: Recommendations of the nomenclature committee of the International Union of Biochemistry and Molecular Biology on the nomenclature and classificationof enzymes. Academic Press Limited, London.

Appendix A4

The pathway information text files shown below contain brief descriptions of four classical pathways: glycolysis, gluconeogenesis, the pentose phosphate pathway, and the urea cycle. There is also a text file that contains pathway EC numbers ("EC Pathways.txt") and one that contains a list of the data files ("DataFileNames.txt").

GlycolInfo.txt

In GLYCOLYSIS, glucose is degraded in ten sequential enzyme-catalysed reactions to two molecules of pyruvate, supplying energy in the form of ATP, and reducing equivalents in the form of NADH (see Lehninger et al, 1993). Pyruvate is at a crossroads in metabolism and may be converted to lactate, alanine, acetyl CoA or oxaloacetate, while many reactions require second substrates that are converted from a high- to a low-energy form, as in the case of ATP and ADP, or from a reduced to an oxidised form, as in the case of NADH and NAD^+. In the last step of glycolysis the product pyruvate is believed to first appear in its enol form. However, the enol form tautomerises rapidly and nonenzymatically to yield the keto form of pyruvate. The glucose transporter GLUT1 is responsible for the low level of basal glucose uptake required to sustain respiration in all cells.

PPPInfo.txt (PPP = pentose phosphate pathway)

The PENTOSE PHOSPHATE PATHWAY provides an alternative pathway for the oxidation of glucose. Its major function is the production of reducing equivalents in the form of NADPH and of pentoses and tetroses for nucleoside and amino acid syntheses. It is also important in preventing oxygen insult to cells, such as the red blood cell (RBC), because NADPH reduces GSSG (oxidised form of glutathione) back to GSH (reduced glutathione), which protects against oxidative damage by reducing hydrogen peroxide (derived from molecular oxygen) to water. This happens in all cells but it is particularly noticeable in the RBC because otherwise haemoglobin would go to the Fe(III) form. The enzymes are present in the cytosol (Michal, 1999; Coffee, 1998).

GluconeoInfo.txt

GLUCONEOGENESIS is defined as the synthesis of glucose from small non-carbohydrate sources. Both glycolysis and gluconeogenesis, which are opposing pathways, share many of the same enzymes, except for four enzymes that are unique to the latter. These are required to by pass three essentially irreversible steps inglycolysis. Its major function is to provide a source of blood glucose that can augment the role of glycogenin preventing hypoglycaemia (Coffee, 1998). However, not all tissues can contribute to the blood glucose – only those with glucose-6-phosphatase. Skeletal muscle,for example, can convert small gluconeogenicprecursors to glucose 6-phosphate but, as mentioned above, cannot release glucose into the blood due to theabsence of glucose-6-phosphatase. In animals, gluconeogenesis takes place in the liver and to a minor extent in the kidney cortex. The first step occurs in the mitochondrion where pyruvate carboxylase convertspyruvate to oxalacetate. The final step occurs in the endoplasmic reticulum where glucose 6-phosphate ishydrolysed to glucose by glucose-6-phosphatase; this bypasses the essentially irreversible reaction inglycolysis catalysed by glucokinase in liver and hexokinase in kidney. All of the intervening steps occur in the cytosol (Coffee, 1998).

UreaCycleInfo.txt

In the UREA CYCLE ammonia is first combined with CO2 to form carbamoyl phosphate. This then combines with ornithine to form citrulline. Citrulline then condenses with aspartate, the source of the second nitrogen atom in urea, to form argininosuccinate. This compound is in turn split to arginine and fumarate, and the arginine then splits to form urea and regenerate ornithine. The first two reactions take place in the mitochondria of liver cells, the remaining three in the cytosol. Ornithine carbamoyltransferase catalyses transfer of the carbamoyl group of carbamoyl phosphate to ornithine, forming citrulline and phosphate. While the reaction occurs in the mitochondrial matrix, both the formation of ornithine and the subsequent metabolism of citrulline take place in the cytosol. Entry of ornithine into mitochondria and exodus of citrulline from mitochondria therefore involve mitochondrial inner membrane transport systems. The activity of carbamoyl-phosphate synthase (ammonia) is determined (activated) by N-acetylglutamate, whose steady-state level is dictated by its rate of synthesis from acetyl-CoA and glutamate and its rate of hydrolysis to acetate and glutamate. These reactions are catalysed by N-acetylglutamate synthase and N-acetylglutamate hydrolase, respectively. N-acetylglutamate synthase is in turn activated by arginine, which accumulates when urea production is too slow to accommodate the ammonia produced by amino acid catabolism.

Major changes in diet can increase the concentrations of individual urea cycle enzymes 10-fold to 20-fold. Starvation, for example, elevates enzyme levels, presumably to cope with the increased production of ammonia that accompanies enhanced protein degradation. Metabolic disorders are associated with each reaction of the urea cycle. All defects in urea synthesis result in ammonia intoxication. Significant improvement and minimisation of brain damage accompany a low-protein diet ingested as frequent small meals to avoid sudden increases in blood ammonia levels (see Murray et al., 2003).

EC Pathways.txt

```
Glycolysis Gluconeogenesis Pentose phosphate Urea cycle
```

2.7.1.2	3.1.3.9	1.1.1.49	6.3.4.16
5.3.1.9	5.3.1.9	3.1.1.31	2.1.3.3
2.7.1.11	3.1.3.11	1.1.1.44	6.3.4.5
4.1.2.13	4.1.2.13	5.1.3.1	4.3.2.1
5.3.1.1	5.3.1.1	5.3.1.6	3.5.3.1
1.2.1.12	1.2.1.12	2.2.1.1	
2.7.2.3	2.7.2.3	2.2.1.2	
5.4.2.1	5.4.2.1		
4.2.1.11	4.2.1.11		
2.7.1.40	4.1.1.32		
	6.4.1.1		

DataFileNames.txt

```
Effectors.dat
General.dat
Glycolysis.dat
Gluconeogenesis.dat
PPP.dat
UreaCycle.dat
User.dat
```

Appendix A5

Shown below is the source code for a Java application to demonstrate computational approaches to the display and manipulation of metabolic pathways. It consists of a number of inter-dependent classes, which contain methods (procedures). The computer starts the execution of the program at the first statement in the method main() in the class TrivialApplication, which is the name assigned to it by the compiler. Single line comments are delimited with // and multiple line comments are delimited with /* and */. Both are printed in dark red font. The dot operator (.) calls a method for a particular object, which is a combination of data (variables) and actions (methods). A semi-colon terminates an instruction and two curly brackets enclose a *block* of code. Reaction information is automatically input to the program as tab-delimited reaction strings with the tabs separating operator code (e.g., EC number), operator name (e.g., enzyme name) and reaction or regulatory action. A powerful *ReactionParser* class breaks down the reaction information into basic components and concatenates them into n-ary relations. It then sends the relations to the *Relation* class, which decodes them and sends the resulting data to the drawing classes for display in a window. The program generates graphical user interfaces on demand, which permit the user to select reactions in different ways.

Source code for the RSD Java application

```
/* The following is a Java application to demonstrate
computational approaches to the display and manipulation of
metabolic pathways.

Author: James Joseph Connolly
Submitted for Ph.D. Thesis 31/10/08
*/

//These packages are used
import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
```

```
/* Class to set up the main GUI (System Gateway) with menus */
public class TrivialApplication extends Frame implements
ActionListener, WindowListener {

    //Declaration of objects
    DataFile data;
     MessageDialog md;
     ClassicalPathway classic;
     StandardPathway standard;
     StandardSubnet subnet;
     UserSearch search;
     NewPathway np;
     PathwayInfo pinfo;
     Reference ref;
     ReactionString view;
     SubStringOne window1;
     SubStringTwo window2;
     SubStringThree window3;
     SubStringFour window4;
     String text;
     String st;
     String st1;
     String st2;

     //Declaration and creation of objects
     Reaction reaction = new Reaction();//reaction references a
new Reaction object
     ReactionParser parser = new ReactionParser();
     FileView screen = new FileView();
     GraphPanel panel = new GraphPanel("Drawing Area");
     GuiSelection g = new GuiSelection("");
     String str = "Reactions.txt";
     String eStr = "Effectors.dat";
     String newString;

     //Declaration of variables
     int counter;
     int nTabs;

//Contents of drop-down menus stored in arrays
     String menuFile[] = {"View data file names","Select any
data file","Select general data file","Select user data file"};

     String menuSelectedFile[] = {"On the Java Console","In
another scrolling window"};

     String menuOperator[] = {"By code","By name"};

     String menuEvent[] = {"Reactions or regulatory actions",
"All regulatory actions"};
```

```
     String menuReactSystem[] = {"Search for pathways","Search
for subnets","User-defined search","Make pathway"};

    String menuSubstring[] = {"Find","Find and display","Find
by inclusion/exclusion","Find by inclusion/exclusion &
display","All of above"};

    String menuClasPathway[] = {"Classical pathways","Pathway
information"};

    String menuInfo[] = {"About using the system","About the
application"};

    String menuRef[] = {"General","Reaction-specific","Search-
specific"};

    String menuTool [] = {"Clear info about using the system"};

    String menuShow [] = {"HTML code"};

//Information about using the system
     String userInfo =
     " ReactionString Database allows the automated drawing of
reactions, pathways and subnets. The\n"+
     " database may be searched using the menus above, the
results of which are given as text and/or \n"+
     " displayed in the form of maps.\n\n"+
     " The FILES menu allows access to all the data files.
These consist of general, user and pathway files. \n"+
     " A file is selected via a GUI that is generated by
clicking the 'Any data file' menu item, but the   \n"+
     " general and user data files may be selected directly by
clicking the corresponding menu item. \n"+
     " Selecting an option imports the corresponding file and
writes it to the Reactions.txt file, which is \n"+
     " used by the program. There is also a file containing a
list of the data files. \n\n"+
     " The VIEW FILE menu allows one to view the selected file
on the Java console or in another scrolling \n"+
     " window.\n\n"+
     " OPERATORS allow the options of searching the selected
file by operator code or operator name. The \n"+
     " four-element EC numbers, classes 1 to 6, constitute a
subset of the operator codes.  Class 0 \n"+
     " represents uncatalysed reactions while classes 8 and 9
deal with receptors and membranes, \n"+
     " respectively.  In the case of an uncatalysed reaction,
an operator may be represented by its code, \n"+
     " the reactive process or the activation energy.  A five-
element TC code represents a transporter. \n"+
```

```
     " The results are given as text and also displayed in the
form of maps.\n\n"+
     " EVENTS allow one to search the selected file by reaction
or regulatory action. The results are given \n"+
     " as text and also displayed in the form of maps. This
menu also allows one to access all the regulatory \n"+
     " actions (effectors) together and display them on screen.
\n\n"+
     " The SYSTEMS SEARCH menu allows one to search the
selected file for defined standard metabolic \n"+
     " pathways and subnets. Also, the selected file may be
searched using input from a user-defined file \n"+
     " and a pathway-creating file. The results are given as
text and also displayed in the form of maps. \n\n"+
     " SUBSTRING options allow one to search the selected file
for a substring and give the result as text \n"+
     " and/or display it in the form of a map. Another option
allows the simultaneous inclusion and \n"+
     " exclusion of substrings, the results of which may also
be given as text and/or displayed as maps on \n"+
     " the screen. \n\n"+
     " The CLASSICAL PATHWAYS menu allows one to access and
display defined classical pathways (with or \n"+
     " without effectors) from specific pathway files (and
effectors file).  A class frequency histogram is \n"+
     " generated for each pathway.  Also, descriptions of these
pathways may be obtained via this menu. \n\n"+
     " INFORMATION on using the system and about the
application may be found via the menu of the same \n"+
     " name. \n\n"+
     " There is a separate menu for REFERENCES, with general
literature and reaction-specific references, \n"+
     " in addition to search-specific references that are
generated when selections are made via the \n"+
     " Operators menu or the 'Reactions or regulatory actions'
item in the Events menu . \n\n"+
     " The SHOW menu allows access to the HTML code that is
generated by the application. This consists of \n"+
     " applet tags and reaction code. The user may copy this
and paste it into the applet (WebPathway), \n"+
     " which is part of the overall system. Then Internet
Explorer may be opened and TrivialApplet.html \n"+
     " dragged  into the browser. \n\n"+
     " To erase this text area, choose 'Clear info about using
the system' from the TOOLS menu\n.";

//Component for displaying information about using
ReactionString Database System
     TextArea systemInfo = new
TextArea(userInfo,0,0,TextArea.SCROLLBARS_VERTICAL_ONLY);
```

```
//Component for displaying information about the application
     Frame about = new Frame("About Application");

    //Method for entry point to application
    public static void main(String args[]) {
     //Create a graphics frame
     Frame f = new TrivialApplication();
     int w = 900; //900 pixels wide
     int h = 500; //500 pixels high
     f.setSize(w,h); //set size of frame
     f.setVisible(true); //display frame on screen
     }
     //Constructor method
     public TrivialApplication() {
          setTitle("System Gateway");
          addWindowListener(this);
          setLayout(null); //set layout manager (none)
          setBackground(Color.blue); //colour the frame bright
blue

//Set up text area for information about using ReactionString
Database System
     systemInfo.setLocation(200,270);
     systemInfo.setSize(520,160);
     systemInfo.setBackground(Color.white);
     systemInfo.setEditable(false);

    //Create a new menu bar
     MenuBar bar = new MenuBar();

     //Create a new menu with the title "Files"
     Menu filesMenu = new Menu("Files");
     //Add the menu items to the menu
     for(int index = 0;index != menuFile.length;index++) {
          filesMenu.add(menuFile[index]);
     }
     //Add the menu to the menu bar
     bar.add(filesMenu);
     //Register this object as a listener
     filesMenu.addActionListener(this);

     //Add 'view file' menu
     Menu selectedFileMenu = new Menu("View File");
     for(int index = 0;index !=
menuSelectedFile.length;index++) {
          selectedFileMenu.add(menuSelectedFile[index]);
     }
     bar.add(selectedFileMenu);
     selectedFileMenu.addActionListener(this);

     //Add operators menu
```

```
        Menu operatorsMenu = new Menu("Operators");
        for(int index = 0;index != menuOperator.length;index++) {
             operatorsMenu.add(menuOperator[index]);
        }
        bar.add(operatorsMenu);
        operatorsMenu.addActionListener(this);

        //Add events menu
        Menu eventsMenu = new Menu("Events");
        for(int index = 0;index != menuEvent.length;index++) {
             eventsMenu.add(menuEvent[index]);
        }
        bar.add(eventsMenu);
        eventsMenu.addActionListener(this);

        //Add 'systems search' menu
        Menu reactSystemsMenu = new Menu("Systems Search");
        for(int index = 0;index != menuReactSystem.length;index++)
{
             reactSystemsMenu.add(menuReactSystem[index]);
        }
        bar.add(reactSystemsMenu);
        reactSystemsMenu.addActionListener(this);

        //Add substrings menu
        Menu substringMenu = new Menu("Substrings");
        for(int index = 0;index != menuSubstring.length;index++) {
             substringMenu.add(menuSubstring[index]);
        }
        bar.add(substringMenu);
        substringMenu.addActionListener(this);

        //Add 'classical pathways' menu
        Menu clasPathwaysMenu = new Menu("Classical Pathways");
        for(int index = 0;index != menuClasPathway.length;index++)
{
             clasPathwaysMenu.add(menuClasPathway[index]);
        }
        bar.add(clasPathwaysMenu);
        clasPathwaysMenu.addActionListener(this);

        //Add information menu
        Menu infoMenu = new Menu("Information");
        for(int index = 0;index != menuInfo.length;index++) {
             infoMenu.add(menuInfo[index]);
        }
        bar.add(infoMenu);
        infoMenu.addActionListener(this);

        //Add references menu
        Menu refMenu = new Menu("References");
```

```
for(int index = 0;index != menuRef.length;index++) {
     refMenu.add(menuRef[index]);
}
bar.add(refMenu);
refMenu.addActionListener(this);

//Add tools menu
Menu toolsMenu = new Menu("Tools");
for(int index = 0;index != menuTool.length;index++) {
     toolsMenu.add(menuTool[index]);
}
bar.add(toolsMenu);
toolsMenu.addActionListener(this);

//Add 'show' menu
Menu showMenu = new Menu("Show");
for(int index = 0;index != menuShow.length;index++) {
     showMenu.add(menuShow[index]);
}
bar.add(showMenu);
showMenu.addActionListener(this);

setMenuBar(bar);
}

//Method to display a string
public void paint( Graphics g ) {
     String title = "ReactionString Database";
     //Create and set font
     Font font = new Font("Serif", Font.BOLD, 72);
     g.setFont(font);
     //Get font metrics
     FontMetrics fm = g.getFontMetrics();
     //Calculate starting position of string
     Dimension d = getSize();
     int x = d.width/2 - fm.stringWidth(title)/2;
     int y = d.height/2 - fm.getDescent();
     //Display string on screen
     g.setColor(Color.orange);
     g.drawString(title, x, y);
}

//Method to detect which item is chosen from the menu
public void actionPerformed(ActionEvent ae) {

Graphics g = getGraphics();
Object source = ae.getActionCommand();

//Check for items chosen from the files menu
for(int index = 0;index != menuFile.length;index++)
     if(source.equals(menuFile[index])) {
```

```
                String files = menuFile[index];

                //Select data file names
                if(files.equals("View data file names")) {
                view = new ReactionString("Data file names");
//Wrap this in a try...catch block to deal with exceptions
                     try{
                     //Read file
                     File f1 = new File("DataFileNames.txt");
                     FileReader fr = new FileReader(f1);//create
a file reader
                     BufferedReader in = new BufferedReader(fr);
//create a buffered reader

                st = in.readLine();
                     while(st!= null) {
                          view.setVisible(true);
                          view.showReactions(st);
                          st = in.readLine();
                     }
                in.close();
                }
                catch(Exception e) {
                     System.out.println(e.getMessage());
                     }
                }
                //Select any data file
                else if(files.equals("Select any data file")) {
                     //Create a window
                     data = new DataFile("Select a data file");
                     data.setSize(440, 80);
                     data.setVisible(true);
                     data.setLocation(800,550);
                     System.out.println();
                     System.out.println(" Select a file with
'.dat' appended to it.");
                     System.out.println();
                }
                else {
                //Select the general data file
                if(files.equals("Select general data file")) {
                     newString = "General.dat";
                     System.out.println();
                     System.out.println(" "+newString+" file
selected and loaded. ");
                     System.out.println(" To view this file,
click on the 'View File' menu and select an option.");
                }
                //Select the user data file
                else if(files.equals("Select user data file")) {
                     newString = "User.dat";
```

```
                    System.out.println();
                    System.out.println(" "+newString+" file
selected and loaded.");
                    System.out.println(" To view this file,
click on the 'View File' menu and select an option.");
               }
//Wrap this in a try...catch block to deal with exceptions
               try{
                    //Read selected file
                    File f = new File(newString);
                    FileReader fr = new FileReader(f);//create
a file reader
                    BufferedReader in = new BufferedReader(fr);
//create a buffered reader

//Write selected file to Reactions.txt file
                    FileOutputStream ostream = new
FileOutputStream("Reactions.txt");
                    PrintWriter pw = new PrintWriter(ostream);

               st = in.readLine();
                    while(st!= null) {
                         pw.println(st);//write data
                         st = in.readLine();
                    }
               //Close file
               in.close();
                    pw.flush();
                    ostream.close();
               }
               catch(Exception e) {
               System.out.println(e.getMessage());
               }
          }
     }

     //Check for items chosen from the 'view file' menu
     for(int index = 0;index !=
menuSelectedFile.length;index++)
          if(source.equals(menuSelectedFile[index])) {
               String file = menuSelectedFile[index];

               if(file.equals("On the Java Console")) {
                    screen.consoleView(str);
               }
               else if(file.equals("In another scrolling
window")) {
                    screen.windowView(str);
               }
          }
```

```
//Check for items chosen from the operators menu
     for(int index = 0;index != menuOperator.length;index++)
          if(source.equals(menuOperator[index])) {
               String operators = menuOperator[index];
               boolean tabs = true;
               counter = 0;
//Wrap this in a try...catch block to deal with exceptions
               try{
                    File f = new File(str);
                    FileReader fr = new FileReader(f);//create
a file reader
                    BufferedReader in = new BufferedReader(fr);
//create a buffered reader
                    //Read the data
               st = in.readLine();
                    while(st!= null) {
                         counter++;//increment counter
                         for(int i=0;i<st.length();i++) {
                              if(st.charAt(i) == '     ')
nTabs++;
                         }
                         if (nTabs != 2) {
                              System.out.println(" Input format
error in tab(s) in line "+counter+" of "+str+" file!");
                              System.out.println();
                              tabs = false;
                         }
                              nTabs = 0;
                         st = in.readLine();
                    }
               in.close();
               }
               catch(Exception e) {
               System.out.println(e.getMessage());
          }

          if(tabs) {
               if(operators.equals("By code")) {
                    String message = "Please wait for complete
list of codes.";
                    md = new MessageDialog(this, "Message",
true, message);
                    md.show();//make dialogue box visible
                    reaction.opCode(str,counter);
               }
               if(operators.equals("By name")) {
                    String message = "Please wait for complete
list of names.";
                    md = new MessageDialog(this, "Message",
true, message);
                    md.show();
```

```
                    reaction.opName(str,counter);
               }
          }
     }

     //Check for items chosen from the events menu
     for(int index = 0;index != menuEvent.length;index++)
          if(source.equals(menuEvent[index])) {
               String events = menuEvent[index];
               boolean tabs = true;
               counter = 0;
               if(events.equals("Reactions or regulatory
actions")) {
//Wrap this in a try...catch block to deal with exceptions
               try{
                    File f = new File(str);
                    FileReader fr = new FileReader(f);//create
a file reader
                    BufferedReader in = new BufferedReader(fr);
//create a buffered reader
               st = in.readLine();
                    while(st!= null) {
                         counter++;
                         for(int i=0;i<st.length();i++) {
                              if(st.charAt(i) == '     ')
nTabs++;
                         }
                         if (nTabs != 2) {
                              System.out.println(" Input format
error in tab(s) in line "+counter+" of "+str+" file!");
                              System.out.println();
                              tabs = false;
                         }
                              nTabs = 0;
                         st = in.readLine();
                    }
               in.close();
               }
               catch(Exception e) {
               System.out.println(e.getMessage());
          }
          System.out.println(" Number of records in selected
file: "+counter);
          System.out.println();
          if(tabs) {
               String message = "Please wait for complete list
of Reactions or Regulatory Actions.";
               md = new MessageDialog(this, "Message", true,
message);
               md.show();
               reaction.event(str,counter);
```

```
          }
        }
        if(events.equals("All regulatory actions")) {
//Wrap this in a try...catch block to deal with exceptions
            try{
                File f = new File(eStr);
                FileReader fr = new FileReader(f);//create
a file reader
                BufferedReader in = new BufferedReader(fr);
//create a buffered reader
            st = in.readLine();
                while(st!= null) {
                    counter++;
                    for(int i=0;i<st.length();i++) {
                        if(st.charAt(i) == '     ')
nTabs++;
                    }
                    if (nTabs != 2) {
                        System.out.println(" Input format
error in tab(s) in line "+counter+" of "+str+" file!");
                        System.out.println();
                        tabs = false;
                    }
                        nTabs = 0;
                    st = in.readLine();
                }
            in.close();
            }
            catch(Exception e) {
            System.out.println(e.getMessage());
        }
        System.out.println(" Number of records in
Effectors.dat file: "+counter);
        System.out.println();
        if(tabs) {
                view = new ReactionString("Regulatory
actions");
//Wrap this in a try...catch block to deal with exceptions
                try{
                //Input effectors file
                File f1 = new File("Effectors.dat");
                FileReader fr = new FileReader(f1);
                BufferedReader in = new BufferedReader(fr);
            st = in.readLine();
                while(st!= null) {
                    parser.ReactionParser(st);
                    view.setVisible(true);
                    view.showReactions(st);
                    st = in.readLine();
                }
            in.close();
```

```
                }
                catch(Exception e) {
                     System.out.println(e.getMessage());
                     }
                }
           }
     }

//Check for items chosen from the 'systems search' menu
     for(int index = 0;index != menuReactSystem.length;index++)
          if(source.equals(menuReactSystem[index])) {
               String systems = menuReactSystem[index];

               if(systems.equals("Search for pathways")) {
                    //Create a window
                    standard = new StandardPathway("Standard
Pathway Search");
                    standard.setSize(360,150);
                    standard.setVisible(true);
                    standard.setLocation(910,0);
               }
               if(systems.equals("Search for subnets")) {
                    //Create a window
                    subnet = new StandardSubnet("Standard
Subnet Search");
                    subnet.setSize(360,150);
                    subnet.setVisible(true);
                    subnet.setLocation(910,193);
               }
               if(systems.equals("User-defined search")) {
                    //Create a window
                    search = new UserSearch("User-defined
search");
                    search.setSize(360,150);
                    search.setVisible(true);
                    search.setLocation(910,386);
               }
               if(systems.equals("Make pathway")) {
                    //Create a window
                    np = new NewPathway("New pathway search");
                    np.setSize(360,150);
                    np.setVisible(true);
                    np.setLocation(910,579);
               }
          }

//Check for items chosen from the substring menu
     for(int index = 0;index != menuSubstring.length;index++)
          if(source.equals(menuSubstring[index])) {
               String sub = menuSubstring[index];
```

```
                if(sub.equals("Find")) {
                     //Create a window
                     window1 = new SubStringOne("Find");
                     window1.setSize(360,60);
                     window1.setVisible(true);
                     window1.setLocation(910,0);
                }
                if(sub.equals("Find and display")) {
                     //Create a window
                     window2 = new SubStringTwo("Find and
display");
                     window2.setSize(360,60);
                     window2.setVisible(true);
                     window2.setLocation(910,103);
                }
                if(sub.equals("Find by inclusion/exclusion")) {
                     //Create a window
                     window3 = new SubStringThree("Find by
inclusion/exclusion");
                     window3.setSize(360,80);
                     window3.setVisible(true);
                     window3.setLocation(910,206);
                }
                if(sub.equals("Find by inclusion/exclusion &
display")) {
                     //Create a window
                     window4 = new SubStringFour("Find by
inclusion/exclusion and display");
                     window4.setSize(360,80);
                     window4.setVisible(true);
                     window4.setLocation(910,328);
                }
                if(sub.equals("All of above")) {
                     //Create windows
                     window1 = new SubStringOne("Find");
                     window1.setSize(360,60);
                     window1.setVisible(true);
                     window1.setLocation(910,0);
                     window2 = new SubStringTwo("Find and
display");
                     window2.setSize(360,60);
                     window2.setVisible(true);
                     window2.setLocation(910,103);
                     window3 = new SubStringThree("Find by
inclusion/exclusion");
                     window3.setSize(360,80);
                     window3.setVisible(true);
                     window3.setLocation(910,206);
                     window4 = new SubStringFour("Find by
inclusion/exclusion and display");
                     window4.setSize(360,80);
```

```
                    window4.setVisible(true);
                    window4.setLocation(910,328);
               }
          }
//Check for items chosen from the 'classical pathways' menu
     for(int index = 0;index != menuClasPathway.length;index++)
          if(source.equals(menuClasPathway[index])) {
               String clasPathway = menuClasPathway[index];

           if(clasPathway.equals("Classical pathways")) {
               //Create a window
               classic = new ClassicalPathway("Classical
Pathways");
               classic.setSize(360,170);
               classic.setLocation(910,0);
               classic.setVisible(true);
           }
           else if(clasPathway.equals("Pathway information")) {
               //Create a window
               pinfo = new PathwayInfo ("Pathway Information");
               pinfo.setSize(360,60);
               pinfo.setLocation(910,212);
               pinfo.setVisible(true);
          }
     }

     //Check for items chosen from the information menu
     for(int index = 0;index != menuInfo.length;index++)
          if(source.equals(menuInfo[index])) {
               String info = menuInfo[index];

               if(info.equals("About using the system")) {
                    add(systemInfo);
                    return;
               }
               else if(info.equals("About the application")) {
                    aboutApplication(about);
               }
          }

     //Check for items chosen from the references menu
     for(int index = 0;index != menuRef.length;index++)
          if(source.equals(menuRef[index])) {
               String info = menuRef[index];

               if(info.equals("General")) {
                    ref = new Reference();
                    ref.genRef();
               }
               if(info.equals("Reaction-specific")) {
                    ref = new Reference();
```

```
                    ref.specifRef();
               }
               else if(info.equals("Search-specific")) {
                    view = new ReactionString("Selected
Reference(s)");
                    try{
                    //Input search-specific references file
                    File f1 = new File("selectedRef.txt");
                    FileReader fr = new FileReader(f1);
                    BufferedReader in = new BufferedReader(fr);
               st = in.readLine();
                    while(st!= null) {
                         view.setVisible(true);
                         view.showReactions(st);
                         st = in.readLine();
                    }
               in.close();
               }
               catch(Exception e) {
                    System.out.println(e.getMessage());
                    }
               }
          }

     //Check for items chosen from the tools menu
     for(int index = 0;index != menuTool.length;index++)
          if(source.equals(menuTool[index])) {
               String tool = menuTool[index];

               if(tool.equals("Clear info about using the
system")) {
                    remove (systemInfo);
                    return;
               }
          }

     //Check for items chosen from the show menu
     for(int index = 0;index != menuShow.length;index++)
          if(source.equals(menuShow[index])) {
               String show = menuShow[index];

               if(show.equals("HTML code")) {
               view = new ReactionString("HTML");
                    try{
                    //Input html file
                    File f = new File("html.txt");
                    FileReader fr = new FileReader(f);
                    BufferedReader in = new BufferedReader(fr);
               st = in.readLine();
                    while(st!= null) {
                         view.setVisible(true);
```

```
                    view.showReactions(st);
                    st = in.readLine();
                }
            in.close();
            }
            catch(Exception e) {
                System.out.println(e.getMessage());
                }
            }
        }
    }

//Method to display information about application in a new
window
    protected void aboutApplication(Frame about) {
        about.setLocation(665,20);
        about.setSize(230,110);
        about.setBackground(Color.cyan);
        about.setFont(new Font("Serif",Font.ITALIC,15));
        about.setLayout(new FlowLayout(FlowLayout.LEFT));

        about.add(new Label("Title: ReactionString
Database"));
        about.add(new Label("Submitted: 31st October 2008"));
        about.add(new Label("Use: Automated pathway
drawing"));
        about.add(new Label("Author: James Joseph
Connolly"));
        about.setVisible(true);
        about.addWindowListener(this);
    }

    //Method to check which window is closing
    public void windowClosing(WindowEvent we) {
        if (we.getWindow() == about) {
            about.dispose();
            return;
        }
        else {
        System.exit(0);
        }
    }
    //Implement blank (empty) WindowListener methods
    public void windowClosed(WindowEvent we){}
    public void windowDeiconified(WindowEvent we){}
    public void windowIconified(WindowEvent we){}
    public void windowActivated(WindowEvent we){}
    public void windowDeactivated(WindowEvent we){}
    public void windowOpened(WindowEvent we){}
}
```

```
//Class to select a data file
class DataFile extends Frame implements ActionListener,
WindowListener {

     private Button button; //declares button
     private TextField fileNameField;
     private TextField messageField;
     private String fileName;
     String st;
     int len;
     String endSeg;

     //Constructor
     public DataFile(String title1) {
          super(title1);
          setBackground(Color.orange);
          setLayout(new FlowLayout());
          messageField = new TextField("Enter the file name ---
-->");
          messageField.setEditable(false);
          add(messageField);
          fileNameField = new TextField(25);
          fileNameField.setText("");
          add(fileNameField);
          button = new Button("LOAD FILE");//creates button
          add(button);//makes button a component
          button.addActionListener(this);
          this.addWindowListener(this);
     }
     //Respond to action events
     public void actionPerformed(ActionEvent ae) {
          //Select file
          if(ae.getSource() == button) {
               fileName = fileNameField.getText();
               if(fileName.equals("")) {
                    String message = "Please enter a file
name!";
                    MessageDialog md = new MessageDialog(this,
"Message", true, message);
                    md.show();
               }
               else {
                    len = fileName.length();
                    endSeg =  fileName.substring(len-3);
               }
               if(endSeg.equals("txt")) {
                    System.out.println(" Invalid choice of
file!");
                    System.out.println(" Select a file with
'.dat' appended to it!");
                    System.out.println();
```

```
                }
                else {
                try{
                     BufferedReader in = new BufferedReader(new
FileReader(fileName));
                }
                catch(IOException e) {
                     messageField.setText("Try again!");
                     System.out.println(" Can't find file.");
                     System.out.println(" File names may be
accessed via the 'View data file names' menu item in the
'Files' menu.");
                     return;
                }
                messageField.setText("Enter the file name -----
>");
                System.out.println();
                System.out.println(" "+fileName+" file selected
and loaded.");
                System.out.println(" To view this file, click on
the 'View File' menu and select an option.");
                System.out.println();
                }
          }
          try{
             //Read selected file
             File f = new File(fileName);
             FileReader fr = new FileReader(f);//create a file
reader
                BufferedReader in = new
BufferedReader(fr);//create a buffered reader

//Write selected file to Reactions.txt file
                FileOutputStream ostream = new
FileOutputStream("Reactions.txt");
                PrintWriter pw = new PrintWriter(ostream);

               st = in.readLine();
                while(st!= null) {
                     pw.println(st);
                     st = in.readLine();
                }
               in.close();
                pw.flush();
                ostream.close();
          }
          catch(Exception e) {
                System.out.println(e.getMessage());
          }
     }
     //Method to check which window is closing
```

```
        public void windowClosing(WindowEvent we) {
             dispose();
             return;
        }
        //Implement blank WindowListener methods
        public void windowClosed(WindowEvent we){}
        public void windowDeiconified(WindowEvent we){}
        public void windowIconified(WindowEvent we){}
        public void windowActivated(WindowEvent we){}
        public void windowDeactivated(WindowEvent we){}
        public void windowOpened(WindowEvent we){}
}
//Class to view a selected data file
class FileView {

        ReactionString view = new ReactionString("Selected File");

        String st;
        int counter;

        //Method to display a data file on the Java console
        public void consoleView(String s) {
             System.out.println();
             System.out.println(" Selected File:");

             try{
                  File f = new File(s);
                  FileReader fr = new FileReader(f);
                  BufferedReader in = new BufferedReader(fr);

                  st = in.readLine();
                       while(st!= null) {
                       System.out.println();
                       System.out.println(" "+st);
                            st = in.readLine();
                            counter++;
                  }
                  in.close();
                  System.out.println();
                  }
                  catch(Exception e) {
                  System.out.println(e.getMessage());
             }
        }
//Method to display a data file in another scrolling window
        public void windowView(String s) {

             try{
                  File f = new File(s);
                  FileReader fr = new FileReader(f);
                  BufferedReader in = new BufferedReader(fr);
```

```
                st = in.readLine();
                while(st!= null) {
                view.setVisible(true);
                     view.showReactions(st);
                     st = in.readLine();
                     counter++;
                }
                in.close();
                }
                catch(Exception e) {
                System.out.println(e.getMessage());
           }
     }
}

/* Class to carry out primary processing for selection of
operator codes and names and reactions/regulatory actions via
scrolling lists*/
class Reaction {

     String str;
     String st;
     int len;
     int pos1;
     String eReaction = "";
     int nrs;

//Method to isolate operator codes and send them to
GuiSelection class
     public void opCode(String str, int counter) {
          nrs = counter; //number of records in Reactions.txt
file

          try{
                File f = new File(str);
                FileReader fr = new FileReader(f);
                BufferedReader in = new BufferedReader(fr);

                GuiSelection window5 = new
GuiSelection("Operator codes");
                window5.setSize(170,232);//set size of window
                window5.setVisible(true);

               st = in.readLine();

                while(st!= null) {
                     len = st.length();
                     pos1 = st.indexOf("\t");
                     String oc =
st.substring(0,pos1);//operator code (oc)
```

```
                    window5.reactions1(oc,nrs);
                    st = in.readLine();
               }
                in.close();
          }
          catch(Exception e) {
               System.out.println(e.getMessage());
          }
     }
//Method to isolate & sort operator names and send them to
GuiSelection class
     public void opName(String str,int counter) {
          nrs = counter;
          String nameArray[] = new String[nrs];//array for
sorting operator names
          int i=0;

          try{
               File f1 = new File(str);
               FileReader fr1 = new FileReader(f1);
               BufferedReader in1 = new BufferedReader(fr1);

//Write the sorted operator names to a file and access them
again
               FileOutputStream ostream = new
FileOutputStream("OperatorNames.txt");
               PrintWriter pw = new PrintWriter(ostream);

              st = in1.readLine();

               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String OC =  st.substring(0,pos1);
                    eReaction = st.substring(pos1+1,len);
                    len = eReaction.length();
                    pos1 = eReaction.indexOf("\t");
                    String on =
eReaction.substring(0,pos1);//operator name (on)
                    nameArray[i] = on;//add operator name to
array
                    i++;
                    st = in1.readLine();
               }

//Send array of operator names to sorter method
               namesSort(nameArray);

               for(int index = 0;index !=
nameArray.length;index++) {
                    pw.println(nameArray[index]);
```

```
                }

                in1.close();
                pw.flush();
                ostream.close();
            }
            catch(Exception e) {
                System.out.println(e.getMessage());
                }

            try{
                File f2 = new File("OperatorNames.txt");
                FileReader fr2 = new FileReader(f2);
                BufferedReader in2 = new BufferedReader(fr2);

                GuiSelection window6 = new
GuiSelection("Operator names");
                window6.setSize(620,232);//set size of window
                window6.setVisible(true);

               st = in2.readLine();

                while(st!= null) {
                    window6.reactions2(st, nrs);
                    st = in2.readLine();
                }
                in2.close();
            }
            catch(Exception e) {
                System.out.println(e.getMessage());
            }
        }

        //Method to sort operator names stored in array
        static void namesSort(String[] nameArray) {
            String temp;
            int chosen;

            for(int leftmost = 0;leftmost <
nameArray.length;leftmost++) {
                chosen = leftmost;
                for(int j = leftmost+1;j < nameArray.length;j++)

        if(nameArray[j].compareTo(nameArray[chosen]) < 0)
                            chosen = j;
                            temp = nameArray[chosen];
                            nameArray[chosen] =
nameArray[leftmost];
                            nameArray[leftmost] = temp;
            }
        }
```

```
//Method to isolate events and send them to GuiSelection class

     public void event(String str,int counter) {
          nrs = counter;
          try{
               File f = new File(str);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

               GuiSelection window7 = new
GuiSelection("Reactions/Regulatory Actions");
               window7.setSize(1250,232);//set size of window
               window7.setVisible(true);

               st = in.readLine();

               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String OC =  st.substring(0,pos1);
                    eReaction = st.substring(pos1+1,len);
                    len = eReaction.length();
                    pos1 = eReaction.indexOf("\t");
                    String enzyme =
eReaction.substring(0,pos1);
                    String re =
eReaction.substring(pos1+1,len);//reaction/regulatory action
(re)
                    window7.reactions3(re, nrs);
                    st = in.readLine();
               }
               in.close();
          }
          catch(Exception e) {
               System.out.println(e.getMessage());
          }
     }
}

//Class to select reaction strings via scrolling selection
lists
class GuiSelection extends Frame implements ActionListener,
WindowListener {

     ReactionParser parser = new ReactionParser();
     ReactionString view = new ReactionString("Reaction(s) or
Regulatory Action(s) Selected");
     Reference ref = new Reference();
     Button button = new Button("SUBMIT SELECTION");
     List list = new List(10,true); //create a list
```

```
        String s = "Reactions.txt";
        String st;
        String code;
        int index = 0;
        int pos1;
        int pos2;
        String pathway;
        String enzymeReaction;
        String eReact;
        int len;
        int counter1;
        Label label;
        String message;
        int nreactions;

//Arrays to hold incoming items from Reaction class
        String reaction1[] = new String[10000];//operator codes
        String reaction2[] = new String[10000];//operator names
        String reaction3[] = new
String[10000];//reactions/regulatory actions

        //Constructor
        public GuiSelection(String title2) {

             super(title2);
             setBackground(Color.pink);
             setLayout(null); //set layout manager (none)
             addWindowListener(this);

             //Display label
             Label selectLabel = new Label("SELECT ITEM(S):");
             selectLabel.setLocation(15, 3);
            selectLabel.setSize(100,20);
            add(selectLabel);

             //Display button
            button.setLocation(10,190);
            button.setSize(120,20);
            add(button);
            button.addActionListener(this);
        }
        //Method to check which window is closing
        public void windowClosing(WindowEvent event) {
             dispose();
             return;
        }

        //Implement blank WindowListener methods
        public void windowClosed(WindowEvent event) {}
        public void windowDeiconified(WindowEvent event) {}
        public void windowIconified(WindowEvent event) {}
```

```
    public void windowActivated(WindowEvent event) {}
    public void windowDeactivated(WindowEvent event) {}
    public void windowOpened(WindowEvent event) {}

    //Method to display list of operator codes in window
    public void reactions1(String enzCode, int nrs) {
        reaction1[index] = enzCode;
        nreactions = nrs;
        list.add(reaction1[index]);
        list.setLocation(5,20);
        list.setSize(130,160); //internal window
        add(list);
        list.addActionListener(this);
        list.setBackground(Color.pink);
        int nitems = list.getItemCount();
        Graphics g = getGraphics();
        Color backgroundColour = getBackground();
        g.setColor(backgroundColour);
        g.fillRect(90,220,35,10);
        g.setColor(Color.black);
        g.drawString("No. of items: "+nitems, 30, 230);

        index++;
//Create dialogue boxes and send messages to be displayed to
MessageDialog class
        if(index >= nreactions) {
            message = "The list is complete.";
            MessageDialog md1 = new MessageDialog(this,
"Message", true, message);
            md1.show();
            message = "To exit this drawing session, close
Operator codes window.";
            MessageDialog md2 = new MessageDialog(this,
"Message", true, message);
            md2.show();

//Print a mesage on the Java console
            System.out.println(" SUCCESSIVE SELECTIONS WILL
BE ADDED TO THE SAME DISPLAY WINDOW.");
            System.out.print(" TO START A NEW SELECTION,
FIRST CLOSE Operator codes WINDOW");
            System.out.println(" AND THEN RESELECT IT FROM
THE Operators MENU.");
            System.out.println();
        }
    }

    //Method to display list of operator names in window
    public void reactions2(String enzName, int nrs) {
        reaction2[index] = enzName;
        nreactions = nrs;
```

```
            list.add(reaction2[index]);
            list.setLocation(5,20);
            list.setSize(600,160); //internal window
            add(list);
            list.addActionListener(this);
            list.setBackground(Color.pink);
            int nitems = list.getItemCount();
            Graphics g = getGraphics();
            Color backgroundColour = getBackground();
            g.setColor(backgroundColour);
            g.fillRect(90,220,35,10);
            g.setColor(Color.black);
            g.drawString("No. of items: "+nitems, 30, 230);

            index++;
//Create dialogue boxes and send messages to be displayed to
MessageDialog class
            if(index >= nreactions) {
                  message = "The list is complete.";
                  MessageDialog md1 = new MessageDialog(this,
"Message", true, message);
                  md1.show();
                  message = "To exit this drawing session, close
Operator names window.";
                  MessageDialog md2 = new MessageDialog(this,
"Message", true, message);
                  md2.show();
//Print a mesage on the Java console
                  System.out.println(" SUCCESSIVE SELECTIONS WILL
BE ADDED TO THE SAME DISPLAY WINDOW.");
                  System.out.print(" TO START A NEW SELECTION,
FIRST CLOSE Operator names WINDOW");
                  System.out.println(" AND THEN RESELECT IT FROM
THE Operators MENU.");
                  System.out.println();
            }
      }

//Method to display lists of reactions/regulatory actions in
window
      public void reactions3(String ev, int nrs) {
            reaction3[index] = ev;
            nreactions = nrs;
            list.add(reaction3[index]);
            list.setLocation(5,20);
            list.setSize(1245,160); //internal window
            add(list);
            list.addActionListener(this);
            list.setBackground(Color.pink);
            int nitems = list.getItemCount();
            Graphics g = getGraphics();
```

```
            Color backgroundColour = getBackground();
            g.setColor(backgroundColour);
            g.fillRect(90,220,35,10);
            g.setColor(Color.black);
            g.drawString("No. of items: "+nitems, 30, 230);

            index++;
//Create dialogue boxes and send messages to be displayed to
MessageDialog class
            if(index >= nreactions) {
                  message = "The list is complete.";
                  MessageDialog md1 = new MessageDialog(this,
"Message", true, message);
                  md1.show();
                  message = "To exit this drawing session, close
Reactions/Regulatory Actions window.";
                  MessageDialog md2 = new MessageDialog(this,
"Message", true, message);
                  md2.show();
//Print a mesage on the Java console
                  System.out.println(" SUCCESSIVE SELECTIONS WILL
BE ADDED TO THE SAME DISPLAY WINDOW.");
                  System.out.print(" TO START A NEW SELECTION,
FIRST CLOSE Reactions/Regulatory Actions WINDOW");
                  System.out.println(" AND THEN RESELECT IT FROM
THE Events MENU.");
                  System.out.println();
            }
      }
      //Method to submit selected items
      public void actionPerformed(ActionEvent ae) {
            Object source = ae.getActionCommand();

            if(source.equals("SUBMIT SELECTION")) {
                  getSelection();
            }
      }
      //Method to get selected items
      private void getSelection() {
            int listArray[] = list.getSelectedIndexes();
            for(index=0;index!=listArray.length;index++) {
                  String component =
list.getItem(listArray[index]);
                  code=component;
                  getReactions(code, s);
            }
      }
//Method to get corresponding reaction strings and send them
for display
      public void getReactions(String code, String s) {
```

```
        try{
             File f = new File(s);
             FileReader fr = new FileReader(f);
             BufferedReader in = new BufferedReader(fr);

            st = in.readLine();
             while(st!= null) {
                  len = st.length();
                  pos1 = st.indexOf("\t");
                  //Operator code
                  String OC =  st.substring(0,pos1);
                  eReact = st.substring(pos1+1,len);
                  len = eReact.length();
                  //Operator name
                  pos2 = eReact.indexOf("\t");
                  String enzyme =  eReact.substring(0,pos2);
//Reaction/regulatory action
                  String evt = eReact.substring(pos2+1,len);

                  if(OC.equals(code)) {
                  parser.ReactionParser(st);
                  view.setVisible(true);
                       view.showReactions(st);
                       ref.getOC(OC);
             }
             else if(enzyme.equals(code)) {
                  parser.ReactionParser(st);
                  view.setVisible(true);
                       view.showReactions(st);
                       ref.getOC(OC);
             }
             else if(evt.equals(code)) {
                  parser.ReactionParser(st);
                  view.setVisible(true);
                       view.showReactions(st);
                       ref.getOC(OC);
             }
                  st = in.readLine();
                  counter1++;
            }
            System.out.println();

            in.close();
            System.out.println(" Number of reactions or
regulatory actions filtered: "+counter1);
            System.out.println();
            counter1 = 0;
        }
        catch(Exception e) {
             System.out.println(e.getMessage());
        }
```

```
    }
}

//Class to search for defined standard pathways
class StandardPathway extends Frame implements ActionListener,
WindowListener {

          GraphPanel panel = new GraphPanel("Drawing Area");
          ReactionParser parser = new ReactionParser();
          ReactionString view;

          String str = "Reactions.txt";
          String st;
          String EC;
          String eReaction = "";
          int pos1;
          int counter1;
          int counter2;
          int counter3;
          int counter4;
          int len;

//Declaration & creation of buttons with pathway names
          Button pushButton1 = new Button("Glycolysis");
          Button pushButton2 = new Button("Gluconeogenesis");
          Button pushButton3 = new Button("Pentose phosphate
pathway");
          Button pushButton4 = new Button("Urea cycle");

     //Constructor
     public StandardPathway(String title4) {

          super(title4);
          setBackground(Color.red);
          setLayout(new FlowLayout());

//Add the buttons to the 'Standard Pathway Search' window

          add(pushButton1);
          add(pushButton2);
          add(pushButton3);
          add(pushButton4);

          pushButton1.addActionListener(this);
          pushButton2.addActionListener(this);
          pushButton3.addActionListener(this);
          pushButton4.addActionListener(this);
          addWindowListener(this);
     }
//Method to display message in 'Standard Pathway Search' window
     public void paint( Graphics g ) {
```

```
          g.drawRect(65,70,230,70);
          g.drawString( "SUCCESSIVE SELECTIONS WILL BE
ADDED",75,84);
          g.drawString( "TO THE SAME DISPLAY WINDOW. ",75,96);
          g.drawString( "TO START A NEW SELECTION, FIRST
CLOSE",75,120);
          g.drawString( "THIS WINDOW, AND THEN RESELECT IT.
",75,132);
     }
     //Method to check which window is closing
     public void windowClosing(WindowEvent we) {
          dispose();
          return;
     }

     //Implement blank WindowListener methods
     public void windowClosed(WindowEvent we) {}
     public void windowDeiconified(WindowEvent we) {}
     public void windowIconified(WindowEvent we) {}
     public void windowActivated(WindowEvent we) {}
     public void windowDeactivated(WindowEvent we) {}
     public void windowOpened(WindowEvent we) {}

     //Respond to action events
     public void actionPerformed(ActionEvent ae) {
               if(ae.getActionCommand().equals("Glycolysis")) {
                    glycolysis(str);
               }
               else
if(ae.getActionCommand().equals("Gluconeogenesis")) {
                    gluconeogenesis(str);
               }
               else if(ae.getActionCommand().equals("Pentose
phosphate pathway")) {
                    pentosePP(str);
               }
               else if(ae.getActionCommand().equals("Urea
cycle")) {
                    ureaCycle(str);
               }
          }
     //Method to find standard glycolysis pathway
     public void glycolysis(String s) {
          view = new ReactionString("Standard GLYCOLYSIS
reactions found");
          String enzGlycol[] = new String[10];
          String glycol[] = new String[10];
          String message;
          int eGlyNumber = 10;
```

```
//Glycolysis enzymes assigned to companion array in correct
sequence
          enzGlycol[0] = "glucokinase";
          enzGlycol[1] = "glucose-6-phosphate isomerase";
          enzGlycol[2] = "6-phosphofructokinase";
          enzGlycol[3] = "fructose-bisphosphate aldolase";
          enzGlycol[4] = "triose-phosphate isomerase";
          enzGlycol[5] = "glyceraldehyde-3-phosphate
dehydrogenase (phosphorylating)";
          enzGlycol[6] = "phosphoglycerate kinase";
          enzGlycol[7] = "phosphoglycerate mutase";
          enzGlycol[8] = "phosphopyruvate hydratase";
          enzGlycol[9] = "pyruvate kinase";

          try{
               File f = new File(s);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

              st = in.readLine();

               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String EC =  st.substring(0,pos1);//EC
number

//Glycolysis reaction strings assigned to array by EC numbers
in correct sequence
                    if(EC.equals("2.7.1.2")) glycol[0] = st;
                    if(EC.equals("5.3.1.9")) glycol[1] = st;
                    if(EC.equals("2.7.1.11")) glycol[2] = st;
                    if(EC.equals("4.1.2.13")) glycol[3] = st;
                    if(EC.equals("5.3.1.1")) glycol[4] = st;
                    if(EC.equals("1.2.1.12")) glycol[5] = st;
                    if(EC.equals("2.7.2.3")) glycol[6] = st;
                    if(EC.equals("5.4.2.1")) glycol[7] = st;
                    if(EC.equals("4.2.1.11")) glycol[8] = st;
                    if(EC.equals("2.7.1.40")) glycol[9] = st;

                    st = in.readLine();
                    counter1++;
               }
              in.close();
              for(int i=0;i< glycol.length;i++) {
               st = glycol[i];
               if(st!= null) {
                    parser.ReactionParser(st);
                    view.setVisible(true);
                    view.showReactions(st);
                    counter2++;
```

```
                    }
                    if(st == null) {
                    System.out.println(" ** N.B:
"+enzGlycol[i]+" NOT FOUND for standard GLYCOLYSIS Pathway.
**");
                    message = "The operator "+enzGlycol[i]+"
could not be found. See Java console";
                    MessageDialog md = new MessageDialog(this,
"Message", true, message);
                    md.show();
                    }
                    System.out.println();
                }

                System.out.println(" Number of reactions
filtered: "+counter1);
                System.out.println(" Number of reactions
selected: "+counter2);
                System.out.println(" Number of reactions
expected: "+eGlyNumber);
                System.out.println(" Number of reactions missing:
"+(eGlyNumber-counter2));
                System.out.println();
                if(counter2 > 0) {
                 message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
                 MessageDialog md = new MessageDialog(this,
"Message", true, message);
                 md.show();
                 }
                counter1 = 0;
                counter2 = 0;
             }
           catch(Exception e) {
                System.out.println(e.getMessage());
           }
     }
     //Method to find standard gluconeogenesis pathway
     public void gluconeogenesis(String s) {
          view = new ReactionString("Standard GLUCONEOGENESIS
reactions found");
          String enzGluconeo[] = new String[11];
          String gluconeo[] = new String[11];
          String message;
          int eGluconeoNumber = 11;

//Gluconeogenesis enzymes assigned to companion array in
correct sequence
          enzGluconeo[0] = "glucose-6-phosphatase";
          enzGluconeo[1] = "glucose-6-phosphate isomerase";
          enzGluconeo[2] = "fructose-bisphosphatase";
```

```
          enzGluconeo[3] = "fructose-bisphosphate aldolase";
          enzGluconeo[4] = "triose-phosphate isomerase";
          enzGluconeo[5] = "glyceraldehyde-3-phosphate
dehydrogenase (phosphorylating)";
          enzGluconeo[6] = "phosphoglycerate kinase";
          enzGluconeo[7] = "phosphoglycerate mutase";
          enzGluconeo[8] = "phosphopyruvate hydratase";
          enzGluconeo[9] = "phosphoenolpyruvate carboxykinase
(GTP)";
          enzGluconeo[10] = "pyruvate carboxylase";

          try{
               File f = new File(s);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

              st = in.readLine();

               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String EC =  st.substring(0,pos1);//EC
number

//Gluconeogenesis reaction strings assigned to array by EC
number in correct sequence
                    if(EC.equals("3.1.3.9")) gluconeo[0] = st;
                    if(EC.equals("5.3.1.9")) gluconeo[1] = st;
                    if(EC.equals("3.1.3.11")) gluconeo[2] = st;
                    if(EC.equals("4.1.2.13")) gluconeo[3] = st;
                    if(EC.equals("5.3.1.1")) gluconeo[4] = st;
                    if(EC.equals("1.2.1.12")) gluconeo[5] = st;
                    if(EC.equals("2.7.2.3")) gluconeo[6] = st;
                    if(EC.equals("5.4.2.1")) gluconeo[7] = st;
                    if(EC.equals("4.2.1.11")) gluconeo[8] = st;
                    if(EC.equals("4.1.1.32")) gluconeo[9] = st;
                    if(EC.equals("6.4.1.1")) gluconeo[10] = st;

                    st = in.readLine();
                    counter1++;
               }
              in.close();

              for(int i=0;i< gluconeo.length;i++) {
               st = gluconeo[i];
               if(st!= null) {
                    parser.ReactionParser(st);
                    view.setVisible(true);
                    view.showReactions(st);
                    counter2++;
                    }
```

```
                    if(st == null) {
                    System.out.println(" ** N.B:
"+enzGluconeo[i]+" NOT FOUND for standard GLUCONEOGENESIS
Pathway. **");
                    message = "The operator "+enzGluconeo[i]+"
could not be found. See Java console";
                    MessageDialog md = new MessageDialog(this,
"Message", true, message);
                    md.show();
                    }
                    System.out.println();
                }

                System.out.println(" Number of reactions
filtered: "+counter1);
                System.out.println(" Number of reactions
selected: "+counter2);
                System.out.println(" Number of reactions
expected: "+eGluconeoNumber);
                System.out.println(" Number of reactions missing:
"+(eGluconeoNumber-counter2));
                System.out.println();
                if(counter2 > 0) {
                 message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
                 MessageDialog md = new MessageDialog(this,
"Message", true, message);
                 md.show();
                 }
                counter1 = 0;
                counter2 = 0;

          }
          catch(Exception e) {
               System.out.println(e.getMessage());
          }
     }
//Method to find standard pentose phosphate pathway
     public void pentosePP(String s) {
          view = new ReactionString("Standard PENTOSE PHOSPHATE
PATHWAY reactions found");
          String enzPentose[] = new String[8];
          String pentose[] = new String[8];
          String message;
          int ePentoseNumber = 8;

//Pentose phosphate pathway enzymes assigned to associated
array in correct sequence
          enzPentose[0] = "glucose-6-phosphate 1-
dehydrogenase";
          enzPentose[1] = "6-phosphogluconolactonase";
```

```
          enzPentose[2] = "phosphogluconate dehydrogenase
(decarboxylating)";
          enzPentose[3] = "ribose-5-phosphate isomerase";
          enzPentose[4] = "ribulose-phosphate 3-epimerase";
          enzPentose[5] = "transketolase1";
          enzPentose[6] = "transaldolase";
          enzPentose[7] = "transketolase2";

          try{
               File f = new File(s);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

              st = in.readLine();

               while(st!= null) {
                 len = st.length();
                 pos1 = st.indexOf("\t");
                 String EC =  st.substring(0,pos1);//EC
number
                     eReaction = st.substring(pos1+1,len);
                     len = eReaction.length();
                     pos1 = eReaction.indexOf("\t");
                     String enzyme =
eReaction.substring(0,pos1);//enzyme

//Pentose phosphate pathway reaction strings assigned to array
by EC number in correct sequence
                     if(EC.equals("1.1.1.49")) pentose[0] = st;
                     if(EC.equals("3.1.1.31")) pentose[1] = st;
                     if(EC.equals("1.1.1.44")) pentose[2] = st;
                     if(EC.equals("5.3.1.6")) pentose[3] = st;
                     if(EC.equals("5.1.3.1")) pentose[4] = st;
                     if(enzyme.equals("transketolase1"))
pentose[5] = st;
                     if(EC.equals("2.2.1.2")) pentose[6] = st;
                     if(enzyme.equals("transketolase2"))
pentose[7] = st;

                     st = in.readLine();
                     counter1++;
               }
              in.close();
              for(int i=0;i< pentose.length;i++) {
               st = pentose[i];
               if(st!= null) {
                     parser.ReactionParser(st);
                     view.setVisible(true);
                     view.showReactions(st);
                     counter2++;
                     }
```

```
                    if(st == null) {
                    System.out.println(" ** N.B:
"+enzPentose[i]+" NOT FOUND for standard PENTOSE PHOSPHATE
PATHWAY. **");
                    message = "The operator "+enzPentose[i]+"
could not be found. See Java console";
                    MessageDialog md = new MessageDialog(this,
"Message", true, message);
                    md.show();
                    }
                    System.out.println();
                }

               System.out.println(" Number of reactions
filtered: "+counter1);
               System.out.println(" Number of reactions
selected: "+counter2);
               System.out.println(" Number of reactions
expected: "+ePentoseNumber);
               System.out.println(" Number of reactions missing:
"+(ePentoseNumber-counter2));
               System.out.println();
               if(counter2 > 0) {
                message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
                MessageDialog md = new MessageDialog(this,
"Message", true, message);
                md.show();
                }
               counter1 = 0;
               counter2 = 0;
          }
          catch(Exception e) {
               System.out.println(e.getMessage());
          }
     }
//Method to find standard urea cycle pathway
     public void ureaCycle(String s) {
          view = new ReactionString("Standard UREA-CYCLE
reactions found");
          String enzUCycle[] = new String[5];
          String uCycle[] = new String[5];
          String message;
          int eUCycleNumber = 5;

//Urea cycle enzymes assigned to companion array in correct
sequence
          enzUCycle[0] = "carbamoyl-phosphate synthase
(ammonia)";
          enzUCycle[1] = "ornithine carbamoyltransferase";
          enzUCycle[2] = "argininosuccinate synthase";
```

```
            enzUCycle[3] = "argininosuccinate lyase";
            enzUCycle[4] = "arginase";

            try{
                 File f = new File(s);
                 FileReader fr = new FileReader(f);
                 BufferedReader in = new BufferedReader(fr);

                st = in.readLine();

                 while(st!= null) {
                      len = st.length();
                      pos1 = st.indexOf("\t");
                      String EC =  st.substring(0,pos1);//EC
number

//Urea cycle reaction strings assigned to array by EC number in
correct sequence
                      if(EC.equals("6.3.4.16")) uCycle[0] = st;
                      if(EC.equals("2.1.3.3")) uCycle[1] = st;
                      if(EC.equals("6.3.4.5")) uCycle[2] = st;
                      if(EC.equals("4.3.2.1")) uCycle[3] = st;
                      if(EC.equals("3.5.3.1")) uCycle[4] = st;

                      st = in.readLine();
                      counter1++;
                 }
                in.close();
                for(int i=0;i< uCycle.length;i++) {
                 st = uCycle[i];
                 if(st!= null) {
                      parser.ReactionParser(st);
                      view.setVisible(true);
                      view.showReactions(st);
                      counter2++;
                      }
                      if(st == null) {
                      System.out.println(" ** N.B:
"+enzUCycle[i]+" NOT FOUND for the standard UREA CYCLE. **");
                      message = "The operator "+enzUCycle[i]+"
could not be found. See Java console";
                      MessageDialog md = new MessageDialog(this,
"Message", true, message);
                      md.show();
                      }
                      System.out.println();
                 }

               System.out.println(" Number of reactions
filtered: "+counter1);
```

```
                System.out.println(" Number of reactions
selected: "+counter2);
                System.out.println(" Number of reactions
expected: "+eUCycleNumber);
                System.out.println(" Number of reactions missing:
"+(eUCycleNumber-counter2));
                System.out.println();
                if(counter2 > 0) {
                 message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
                 MessageDialog md = new MessageDialog(this,
"Message", true, message);
                 md.show();
                 }
                counter1 = 0;
                counter2 = 0;
           }
           catch(Exception e) {
                System.out.println(e.getMessage());
           }
     }
}

//Class to search for subnets of defined standard pathways
class StandardSubnet extends Frame implements ActionListener,
WindowListener {

     GraphPanel panel = new GraphPanel("Drawing Area");
     ReactionParser parser = new ReactionParser();
     ReactionString view;

     String str = "Reactions.txt";
     String st;
     String EC;
     int pos1;
     String eReaction = "";
     int counter1;
    int counter2;
     int len;

     //Declaration & creation of buttons with subnet names
     Button pushButton11 = new
Button("Glycolysis/Gluconeogenesis");
     Button pushButton12 = new Button("Glycolysis/Pentose
phosphate");

     //Constructor
     public StandardSubnet(String title5) {

          super(title5);
          setBackground(Color.red);
```

```
            setLayout(new FlowLayout());

            add(pushButton11);
            add(pushButton12);
            pushButton11.addActionListener(this);
            pushButton12.addActionListener(this);
            addWindowListener(this);
      }
//Method to display message in 'Standard Subnet Search' window
      public void paint( Graphics g ) {
            g.drawRect(65,70,230,70);
            g.drawString( "SUCCESSIVE SELECTIONS WILL BE
ADDED",75,84);
            g.drawString( "TO THE SAME DISPLAY WINDOW. ",75,96);
            g.drawString( "TO START A NEW SELECTION, FIRST
CLOSE",75,120);
            g.drawString( "THIS WINDOW, AND THEN RESELECT IT.
",75,132);
      }
      //Method to check which window is closing
      public void windowClosing(WindowEvent we) {
            dispose();
            return;
      }

      //Implement blank WindowListener methods
      public void windowClosed(WindowEvent we) {}
      public void windowDeiconified(WindowEvent we) {}
      public void windowIconified(WindowEvent we) {}
      public void windowActivated(WindowEvent we) {}
      public void windowDeactivated(WindowEvent we) {}
      public void windowOpened(WindowEvent we) {}

      //Respond to action events
      public void actionPerformed(ActionEvent ae) {

      if(ae.getActionCommand().equals("Glycolysis/Gluconeogenesi
s")) {
                        glycolGluconeo(str);
                  }
                  else
if(ae.getActionCommand().equals("Glycolysis/Pentose
phosphate")) {
                        GlycolPP(str);
                  }
            }
      //Method to find glycolysis/gluconeogenesis subnet
      public void glycolGluconeo(String s) {

            view = new ReactionString("GLYCOLYSIS/GLUCONEOGENESIS
subnet reactions found");
```

```
          String enzGlylGlu[] = new String[14];
          String glyGlu[] = new String[14];
          String message;
          int eGlyGluNumber = 14;

//Glycolysis/gluconeogenesis enzymes assigned to companion
array in correct sequence
          enzGlylGlu[0] = "glucokinase";
          enzGlylGlu[1] = "glucose-6-phosphatase";
          enzGlylGlu[2] = "glucose-6-phosphate isomerase";
          enzGlylGlu[3] = "6-phosphofructokinase";
          enzGlylGlu[4] = "fructose-bisphosphatase";
          enzGlylGlu[5] = "fructose-bisphosphate aldolase";
          enzGlylGlu[6] = "triose-phosphate isomerase";
          enzGlylGlu[7] = "glyceraldehyde-3-phosphate
dehydrogenase (phosphorylating)";
          enzGlylGlu[8] = "phosphoglycerate kinase";
          enzGlylGlu[9] = "phosphoglycerate mutase";
          enzGlylGlu[10] = "phosphopyruvate hydratase";
          enzGlylGlu[11] = "pyruvate kinase";
          enzGlylGlu[12] = "phosphoenolpyruvate carboxykinase
(GTP)";
          enzGlylGlu[13] = "pyruvate carboxylase";

          try{
               File f = new File(s);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

              st = in.readLine();

               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String EC =  st.substring(0,pos1);//EC
number

//Glycolysis/gluconeogenesis reaction strings assigned to array
by EC number in correct sequence
                    if(EC.equals("2.7.1.2")) glyGlu[0] = st;
                    if(EC.equals("3.1.3.9")) glyGlu[1] = st;
                    if(EC.equals("5.3.1.9")) glyGlu[2] = st;
                    if(EC.equals("2.7.1.11")) glyGlu[3] = st;
                    if(EC.equals("3.1.3.11")) glyGlu[4] = st;
                    if(EC.equals("4.1.2.13")) glyGlu[5] = st;
                    if(EC.equals("5.3.1.1")) glyGlu[6] = st;
                    if(EC.equals("1.2.1.12")) glyGlu[7] = st;
                    if(EC.equals("2.7.2.3")) glyGlu[8] = st;
                    if(EC.equals("5.4.2.1")) glyGlu[9] = st;
                    if(EC.equals("4.2.1.11")) glyGlu[10] = st;
                    if(EC.equals("2.7.1.40")) glyGlu[11] = st;
```

```
                    if(EC.equals("4.1.1.32")) glyGlu[12] = st;
                    if(EC.equals("6.4.1.1")) glyGlu[13] = st;

                    st = in.readLine();
                    counter1++;
                }
               in.close();
               for(int i=0;i< glyGlu.length;i++) {
                st = glyGlu[i];
                if(st!= null) {
                    parser.ReactionParser(st);
                    view.setVisible(true);
                    view.showReactions(st);
                    counter2++;
                    }
                    if(st == null) {
                    System.out.println(" ** N.B:
"+enzGlylGlu[i]+" NOT FOUND. **");
                    message = "The operator "+enzGlylGlu[i]+"
could not be found. See Java console";
                    MessageDialog md = new MessageDialog(this,
"Message", true, message);
                    md.show();
                    }
                    System.out.println();
                }

               System.out.println(" Number of reactions
filtered: "+counter1);
               System.out.println(" Number of reactions
selected: "+counter2);
               System.out.println(" Number of reactions
expected: "+eGlyGluNumber);
               System.out.println(" Number of reactions missing:
"+(eGlyGluNumber-counter2));
               System.out.println();
               if(counter2 > 0) {
                message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
                MessageDialog md = new MessageDialog(this,
"Message", true, message);
                md.show();
                }
               counter1 = 0;
               counter2 = 0;
            }
          catch(Exception e) {
               System.out.println(e.getMessage());
          }
     }
//Method to find glycolysis/pentose phosphate subnet
```

```
     public void GlycolPP(String s) {

          view = new ReactionString("GLYCOLYSIS/PENTOSE
PHOSPHATE PATHWAY subnet reactions found");
          String enzGlyPP[] = new String[18];
          String glyPP[] = new String[18];
          String message;
          int eGlyPPNumber = 18;

//Glycolysis/pentose phosphate enzymes assigned to companion
array in correct sequence
          enzGlyPP[0] = "glucokinase";
          enzGlyPP[1] = "glucose-6-phosphate isomerase";
          enzGlyPP[2] = "6-phosphofructokinase";
          enzGlyPP[3] = "fructose-bisphosphate aldolase";
          enzGlyPP[4] = "triose-phosphate isomerase";
          enzGlyPP[5] = "glyceraldehyde-3-phosphate
dehydrogenase (phosphorylating)";
          enzGlyPP[6] = "phosphoglycerate kinase";
          enzGlyPP[7] = "phosphoglycerate mutase";
          enzGlyPP[8] = "phosphopyruvate hydratase";
          enzGlyPP[9] = "pyruvate kinase";
          enzGlyPP[10] = "glucose-6-phosphate 1-dehydrogenase";
          enzGlyPP[11] = "6-phosphogluconolactonase";
          enzGlyPP[12] = "phosphogluconate dehydrogenase
(decarboxylating)";
          enzGlyPP[13] = "ribose-5-phosphate isomerase";
          enzGlyPP[14] = "ribulose-phosphate 3-epimerase";
          enzGlyPP[15] = "transketolase1";
          enzGlyPP[16] = "transaldolase";
          enzGlyPP[17] = "transketolase2";

          try{
               File f = new File(s);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

              st = in.readLine();

               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String EC =  st.substring(0,pos1);//EC
number
                    eReaction = st.substring(pos1+1,len);
                    len = eReaction.length();
                    pos1 = eReaction.indexOf("\t");
                    String enzyme =
eReaction.substring(0,pos1);//enzyme
```

```
//Glycolysis/pentose phosphate reaction strings assigned to
array by EC number in correct sequence
                    if(EC.equals("2.7.1.2")) glyPP[0] = st;
                    if(EC.equals("5.3.1.9")) glyPP[1] = st;
                    if(EC.equals("2.7.1.11")) glyPP[2] = st;
                    if(EC.equals("4.1.2.13")) glyPP[3] = st;
                    if(EC.equals("5.3.1.1")) glyPP[4] = st;
                    if(EC.equals("1.2.1.12")) glyPP[5] = st;
                    if(EC.equals("2.7.2.3")) glyPP[6] = st;
                    if(EC.equals("5.4.2.1")) glyPP[7] = st;
                    if(EC.equals("4.2.1.11")) glyPP[8] = st;
                    if(EC.equals("2.7.1.40")) glyPP[9] = st;
                    if(EC.equals("1.1.1.49")) glyPP[10] = st;
                    if(EC.equals("3.1.1.31")) glyPP[11] = st;
                    if(EC.equals("1.1.1.44")) glyPP[12] = st;
                    if(EC.equals("5.3.1.6")) glyPP[13] = st;
                    if(EC.equals("5.1.3.1")) glyPP[14] = st;
                    if(enzyme.equals("transketolase1"))
glyPP[15] = st;
                    if(EC.equals("2.2.1.2")) glyPP[16] = st;
                    if(enzyme.equals("transketolase2"))
glyPP[17] = st;

                    st = in.readLine();
                    counter1++;
                }
               in.close();
               for(int i=0;i< glyPP.length;i++) {
                st = glyPP[i];
                if(st!= null) {
                    parser.ReactionParser(st);
                    view.setVisible(true);
                    view.showReactions(st);
                    counter2++;
                    }
                    if(st == null) {
                    System.out.println(" ** N.B:
"+enzGlyPP[i]+" NOT FOUND. **");
                    message = "The operator "+enzGlyPP[i]+"
could not be found. See Java console";
                    MessageDialog md = new MessageDialog(this,
"Message", true, message);
                    md.show();
                    }
                    System.out.println();
                }
               System.out.println(" Number of reactions
filtered: "+counter1);
               System.out.println(" Number of reactions
selected: "+counter2);
```

```
                System.out.println(" Number of reactions
expected: "+eGlyPPNumber);
                System.out.println(" Number of reactions missing:
"+(eGlyPPNumber-counter2));
                System.out.println();
                if(counter2 > 0) {
                 message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
                 MessageDialog md = new MessageDialog(this,
"Message", true, message);
                 md.show();
                 }
                counter1 = 0;
                counter2 = 0;
             }
           catch(Exception e) {
                System.out.println(e.getMessage());
           }
     }
}

//Class to search for reaction strings as defined in a query
file
class UserSearch extends Frame implements ActionListener,
ItemListener, WindowListener {

     ReactionParser parser = new ReactionParser();
     ReactionString view = new ReactionString("Reactions
found");;

     String s = "Reactions.txt";
     String st;
     String str;
     String uds;
     String EC;
     String eReaction = "";
     int difference;
     int pos1;
     int len;
     int nTabs;
     int counter;
     int counter1;
     int j = 0;
     int found = 0;
     Label label1;
     Label label2;
     boolean tabs = true;
     boolean opcode;
     boolean opname;
     CheckboxGroup field;
     Checkbox codeField, nameField, reactionField;
```

```
//Arrays to hold operator codes & names and events from search
file
     String codeArray[] = new String[100];
     String opArray[]  = new String[100];
     String reArray[]  = new String[100];

//Arrays to check for operator codes or names or events NOT
found
     String cArray[] = new String[100];
     String nArray[] = new String[100];
     String rArray[] = new String[100];

     //Declaration & creation of 'Submit ...' button
     Button pushButton = new Button("Submit user-defined
search");

     //Constructor
     public UserSearch(String title14) {

          super(title14);
          setBackground(Color.red);
          setLayout(new FlowLayout());

          //Display labels
          label1 = new Label("First enter your reaction
string(s) into the UserSearch.txt file.");
          add(label1);
          label2 = new Label("Search by: ");
          add(label2);

          //Create radio buttons
          field = new CheckboxGroup();
          codeField = new Checkbox("Operator code",field,true);
          add(codeField);
          codeField.addItemListener(this);
          nameField = new Checkbox("Operator
name",field,false);
          add(nameField);
          nameField.addItemListener(this);
          reactionField = new Checkbox("Reaction",field,false);
          add(reactionField);
          reactionField.addItemListener(this);

//Add the 'Submit ...' button to the user search window
          add(pushButton);

          pushButton.addActionListener(this);
          addWindowListener(this);
     }
     //Method to display message in user search window
```

```
     public void paint( Graphics g ) {
          g.drawRect(65,70,230,76);
          g.drawString( "SUCCESSIVE SELECTIONS WILL BE
ADDED",75,90);
          g.drawString( "TO THE SAME DISPLAY WINDOW. ",75,102);
          g.drawString( "TO START A NEW SELECTION, FIRST
CLOSE",75,126);
          g.drawString( "THIS WINDOW, AND THEN RESELECT IT.
",75,138);
     }
     //Method to check which window is closing
     public void windowClosing(WindowEvent we) {
          dispose();
          return;
     }
     //Implement blank WindowListener methods
     public void windowClosed(WindowEvent we) {}
     public void windowDeiconified(WindowEvent we) {}
     public void windowIconified(WindowEvent we) {}
     public void windowActivated(WindowEvent we) {}
     public void windowDeactivated(WindowEvent we) {}
     public void windowOpened(WindowEvent we) {}

     //Respond to item events
     public void itemStateChanged(ItemEvent ie) {
          if(ie.getSource() == codeField) opcode = true;
          else opcode = false;
          if(ie.getSource() == nameField) opname = true;
          else opname = false;
          System.out.println(" CheckboxGroup state =
"+codeField.getState()+" "+nameField.getState()+"
"+reactionField.getState());
     }
     //Respond to action event
     public void actionPerformed(ActionEvent ae) {
          if(ae.getActionCommand().equals("Submit user-defined
search")) {
               uds = "UserSearch.txt";
          }
          //Read user search file
          try{
               File f = new File(uds);
               FileReader fr = new FileReader(f);//create a
file reader
               BufferedReader in = new
BufferedReader(fr);//create a buffered reader

              str = in.readLine();
               while(str!= null) {
                    counter++;
                    for(int i=0;i<str.length();i++) {
```

```
                    if(str.charAt(i) == '    ') nTabs++;
                }
                if(nTabs != 2) {
                    System.out.println(" Input format
error in tab(s) in line "+counter+" of "+uds+" file!");
                    System.out.println();
                    tabs = false;
                }
                nTabs = 0;

                len = str.length();
                pos1 = str.indexOf("\t");
                String oc =
str.substring(0,pos1);//operator code (oc)
                codeArray[j] = oc;//add operator codes to
codeArray
                eReaction = str.substring(pos1+1,len);
                len = eReaction.length();
                pos1 = eReaction.indexOf("\t");
                String on =
eReaction.substring(0,pos1);//operator name (on)
                opArray[j] = on;//add operator names to
opArray
                String re =
eReaction.substring(pos1+1,len);//reaction/regulatory action
(re)
                reArray[j] = re;//add reactions/regulatory
actions to reArray

                counter1++;
                j++;
                str = in.readLine();
            }
            //Close file
           in.close();
            j = 0;
        }
        catch(Exception e) {
        System.out.println(e.getMessage());
        }
        if(tabs) {
            System.out.println();
            System.out.println(" Number of records in
"+uds+" file: "+counter1);
            System.out.println();
            System.out.println(" Parsing results for
corresponding reactions found in "+s+" file:");
        }
        //Read 'Reactions.txt' file
        try{
            File f = new File(s);
```

```
                FileReader fr = new FileReader(f);
                BufferedReader in = new BufferedReader(fr);

               st = in.readLine();

                while(st!= null) {
                     len = st.length();
                     pos1 = st.indexOf("\t");
                String ec =  st.substring(0,pos1);//code (ec)
                     eReaction = st.substring(pos1+1,len);
                     len = eReaction.length();
                     pos1 = eReaction.indexOf("\t");
                     String op =
eReaction.substring(0,pos1);//operator (op)
                     String re =
eReaction.substring(pos1+1,len);//reaction/regulatory action
(re)

                     opcode = field.getSelectedCheckbox() ==
codeField;
                     opname = field.getSelectedCheckbox() ==
nameField;

                //Compare codes
                if(opcode) {
                     for(int i = 0;i!= counter1;i++) {
                          if(ec.compareTo(codeArray[i]) == 0) {
                               parser.ReactionParser(st);
                               view.setVisible(true);
                               view.showReactions(st);
                               cArray[i] = codeArray[i];
                               found++;
                               }
                          }
                     //Compare operators
                     else if(opname) {
                          for(int i = 0;i!= counter1;i++) {
                               if(op.compareTo(opArray[i]) == 0)
{
                                    parser.ReactionParser(st);
                                    view.setVisible(true);
                                    view.showReactions(st);
                                    nArray[i] = opArray[i];
                                    found++;
                               }
                          }
                     }
                     //Compare reactions
                     else {
                          for(int i = 0;i!= counter1;i++) {
```

```
                                        if(re.compareTo(reArray[i]) == 0)
{
                                             parser.ReactionParser(st);
                                             view.setVisible(true);
                                             view.showReactions(st);
                                             rArray[i] = reArray[i];
                                             found++;
                                        }
                                   }
                              }
                              st = in.readLine();
                         }
                         System.out.println(" Number of reactions found:
"+found);
                         //Close file
                         in.close();
                         if(found > 0) {
                         String message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
                         MessageDialog md = new MessageDialog(this,
"Message", true, message);
                         md.show();
                         }

                    //Check for reactions not found using operator code
                         if(opcode) {
                         System.out.println();
                         for(int index = 0;index != counter1;index++) {
                                   if(cArray[index] == null) {
                                        System.out.println(" No reaction
found for operator code "+codeArray[index]);
                                   }
                              }
                          }
               //Check for reactions not found using operator name
                         else if(opname) {
                              System.out.println();
                              for(int index = 0;index !=
counter1;index++) {
                                   if(nArray[index] == null) {
                                        System.out.println(" No reaction
found for operator name "+opArray[index]);
                                        System.out.println();
                                        System.out.println(" The
operator(s) may have synonyms");
                                        System.out.println(" Use search
option 'Operator code'");
                                   }
                              }
                         }
                         //Check for reactions not found
```

```
                else {
                     System.out.println();
                     for(int index = 0;index !=
counter1;index++) {
                          if(rArray[index] == null) {
                               System.out.println(" The reaction
"+reArray[index]+"  was NOT found");
                               System.out.println(" The reaction
may be in the database with L and R side(s) exchanged");
                               System.out.println(" Use search
options 'Operator code' or 'Operator name'");
                               System.out.println();
                          }
                     }

                }
                 System.out.println();
                counter1 = 0;
                found = 0;
                for(int index = 0;index != counter1;index++) {
                     codeArray[index] = "";
                     opArray[index] = "";
                     cArray[index] = "";
                     nArray[index] = "";
                     reArray[index] = "";
                     rArray[index] = "";
                }
           }
           catch(Exception e) {
                System.out.println(e.getMessage());
           }
     }
}

//Class to make a 'new' pathway from a data file
class NewPathway extends Frame implements ActionListener,
ItemListener, WindowListener {

     ReactionParser parser = new ReactionParser();
     ReactionString view = new ReactionString("Pathway
reactions found");;

     String s = "Reactions.txt";
     String st;
     String str;
     String uds;
     String EC;
     String mp;
     String eReaction = "";
     int difference;
     int pos1;
```

```
      int len;
      int nTabs;
      int counter;
      int counter1;
      int j = 0;
      int found = 0;
      Label label1;
      Label label2;
      boolean tabs = true;
      boolean opcode;
      boolean opname;
      CheckboxGroup field;
      Checkbox codeField, nameField, reactionField;

//Arrays to hold operator codes & names from MakePathway file
      String codeArray[] = new String[100];
      String opArray[]  = new String[100];

      //Arrays to check for operator codes or names NOT found
      String cArray[] = new String[100];
      String nArray[] = new String[100];

      //Declaration & creation of 'Submit ...' button
      Button pushButton = new Button("Submit pathway search");

      //Constructor
      public NewPathway(String title14) {

            super(title14);
            setBackground(Color.red);
            setLayout(new FlowLayout());

            //Display labels
            label1 = new Label("First enter operator code(s) &
name(s) into the Pathway.txt file.");
            add(label1);
            label2 = new Label("Search by: ");
            add(label2);

            //Create radio buttons
            field = new CheckboxGroup();
            codeField = new Checkbox("Operator code",field,true);
            add(codeField);
            codeField.addItemListener(this);
            nameField = new Checkbox("Operator
name",field,false);
            add(nameField);
            nameField.addItemListener(this);

//Add the 'Submit ...' button to the MakePathway window
            add(pushButton);
```

```
            pushButton.addActionListener(this);
            addWindowListener(this);
        }
        //Method to check which window is closing
        public void windowClosing(WindowEvent we) {
            dispose();
            return;
        }
        //Implement blank WindowListener methods
        public void windowClosed(WindowEvent we) {}
        public void windowDeiconified(WindowEvent we) {}
        public void windowIconified(WindowEvent we) {}
        public void windowActivated(WindowEvent we) {}
        public void windowDeactivated(WindowEvent we) {}
        public void windowOpened(WindowEvent we) {}

        //Respond to item events
        public void itemStateChanged(ItemEvent ie) {
            if(ie.getSource() == codeField)
                opcode = true;
            else opcode = false;
            System.out.println(" CheckboxGroup state =
"+codeField.getState()+" "+nameField.getState());
        }
        //Respond to action event
        public void actionPerformed(ActionEvent ae) {
            if(ae.getActionCommand().equals("Submit pathway
search")) {
                mp = "Pathway.txt";
            }
            //Read Pathway.txt file
            try{
                File f = new File(mp);
                FileReader fr = new FileReader(f);//create a
file reader
                BufferedReader in = new
BufferedReader(fr);//create a buffered reader

               str = in.readLine();
                while(str!= null) {
                    counter++;
                    for(int i=0;i<str.length();i++) {
                        if(str.charAt(i) == '    ') nTabs++;
                    }
                    if(nTabs != 1) {
                        System.out.println(" Input format
error in tab(s) in line "+counter+" of "+uds+" file!");
                        System.out.println();
                        tabs = false;
                    }
```

```
                    nTabs = 0;

                    len = str.length();
                    pos1 = str.indexOf("\t");
                    String oc =
str.substring(0,pos1);//operator code (oc)
                    codeArray[j] = oc;//add operator codes to
codeArray
                    String on =
str.substring(pos1+1,len);//operator name (on)
                    opArray[j] = on;//add operator names to
opArray
                    counter1++;
                    j++;
                    str = in.readLine();
               }
               //Close file
              in.close();

               j = 0;
          }
          catch(Exception e) {
          System.out.println(e.getMessage());
          }
          if(tabs) {
               System.out.println();
               System.out.println(" Number of records in "+mp+"
file: "+counter1);
               System.out.println();
          }
          //Read 'Reactions.txt' file
          try{
               File f = new File(s);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

              st = in.readLine();

               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String ec =  st.substring(0,pos1);//code
(ec)
                    eReaction = st.substring(pos1+1,len);
                    len = eReaction.length();
                    pos1 = eReaction.indexOf("\t");
                    String op =
eReaction.substring(0,pos1);//operator (op)

                    opcode = field.getSelectedCheckbox() ==
codeField;
```

```
                    //Compare codes
                    if(opcode) {
                         for(int i = 0;i!= counter1;i++) {
                              if(ec.compareTo(codeArray[i]) ==
0) {
                                   parser.ReactionParser(st);
                                   view.setVisible(true);
                                   view.showReactions(st);
                                   cArray[i] = codeArray[i];
                                   found++;
                              }
                         }
                    }
                    //Compare operators
                    else {
                         for(int i = 0;i!= counter1;i++) {
                              if(op.compareTo(opArray[i]) == 0)
{
                                   parser.ReactionParser(st);
                                   view.setVisible(true);
                                   view.showReactions(st);
                                   nArray[i] = opArray[i];
                                   found++;
                              }
                         }
                    }
                    st = in.readLine();
               }
               System.out.println(" Number of reactions found
in target file: "+found);
               //Close file
               in.close();
               if(found > 0) {
               String message = "CLICK IN DRAWING AREA TO VIEW
DIAGRAM";
               MessageDialog md = new MessageDialog(this,
"Message", true, message);
               md.show();
               }
//Check for reactions not found using operator code
               if(opcode) {
               System.out.println();
               for(int index = 0;index != counter1;index++) {
                    if(codeArray[index].equals("****"))
                              System.out.println(" Operator
code for "+opArray[index]+" not in MakePathway.txt file");
                         else if(cArray[index] == null) {
                              System.out.println(" No reaction
found for operator code "+codeArray[index]);
                         }
```

```
                    }
                }
//Check for reactions not found using operator name
                else if(opname) {
                    System.out.println();
                    for(int index = 0;index !=
counter1;index++) {
                        if(opArray[index].equals("****"))
                            System.out.println(" Operator
name for "+codeArray[index]+" not in MakePathway.txt file");
                        else if(nArray[index] == null) {
                            System.out.println(" No reaction
found for operator name "+opArray[index]);
                            System.out.println(" The
operator(s) may have synonyms");
                            System.out.println(" Use search
option 'Operator code'");
                            System.out.println();
                        }
                    }
                }
                 System.out.println();
                //Reinitialise variables
                counter1 = 0;
                found = 0;
                //Reinitialise arrays
                for(int index = 0;index != counter1;index++) {
                    codeArray[index] = "";
                    opArray[index] = "";
                    cArray[index] = "";
                    nArray[index] = "";
                }
            }
            catch(Exception e) {
                System.out.println(e.getMessage());
            }
        }
}

//Class to find reaction string(s) containing a given substring
class SubStringOne extends Frame implements ActionListener,
WindowListener {

    ReactionString view = new ReactionString("Reaction(s)
Selected");
    int counter;
    TextField sub;
    String subStr;
    String line;

    //Constructor
```

```
    public SubStringOne(String title6) {

        super(title6);
        setBackground(Color.cyan);
        setLayout(new FlowLayout());

        //Create the text field and the labels
        sub = new TextField(20);
        add(new Label("Enter substring: "));//label to prompt
the user
        add(sub);
        sub.addActionListener(this);

        add(new Label("Press enter to view results.
"));//label to prompt the user
        addWindowListener(this);
    }
    //Method to check which window is closing
    public void windowClosing(WindowEvent we) {
        dispose();
        return;
    }

    //Implement blank WindowListener methods
    public void windowClosed(WindowEvent we) {}
    public void windowDeiconified(WindowEvent we) {}
    public void windowIconified(WindowEvent we) {}
    public void windowActivated(WindowEvent we) {}
    public void windowDeactivated(WindowEvent we) {}
    public void windowOpened(WindowEvent we) {}

    public void actionPerformed(ActionEvent ae) {
        subStr = sub.getText();
        sub.setText("");//clear input
        System.out.println();

        if(subStr.equals("")) {
            String message = "Please enter a substring!";
            MessageDialog md = new MessageDialog(this,
"Message", true, message);
            md.show();
        }
        else try {
            File f = new File("Reactions.txt");
            ReactionFinder in = new ReactionFinder(new
FileReader(f),subStr);
            System.out.println(" Reaction string(s)
containing the substring "+subStr+":");
            System.out.println();
            while((line = in.readLine()) != null) {
                System.out.println(" "+line);
```

```
                    view.setVisible(true);
                    view.showReactions(line);
               }
               in.close();
          }
          catch (Exception e) {
               System.out.println(e);
          }
     }
}

//Show reaction string(s) containing a given substring and map
the reaction(s)
class SubStringTwo extends Frame implements ActionListener,
WindowListener {

     ReactionString view = new ReactionString("Reaction(s)
Selected");
     ReactionParser parser = new ReactionParser();
     int counter;
     TextField sub;
     String subStr;

     //Constructor
     public SubStringTwo(String title7) {

          super(title7);
          setBackground(Color.cyan);
          setLayout(new FlowLayout());

          //Create the text field and the labels
          sub = new TextField(20);
          add(new Label("Enter substring: "));
          add(sub);
          sub.addActionListener(this);
          add(new Label("Press enter to view results. "));
          addWindowListener(this);
     }
     //Method to check which window is closing
     public void windowClosing(WindowEvent we) {
          dispose();
          return;
     }

     //Implement blank WindowListener methods
     public void windowClosed(WindowEvent we) {}
     public void windowDeiconified(WindowEvent we) {}
     public void windowIconified(WindowEvent we) {}
     public void windowActivated(WindowEvent we) {}
     public void windowDeactivated(WindowEvent we) {}
     public void windowOpened(WindowEvent we) {}
```

```
     public void actionPerformed(ActionEvent ae) {
          subStr = sub.getText();
          sub.setText("");//clear input
          System.out.println();
          System.out.println();

          if(subStr.equals("")) {
               String message = "Please enter a substring!";
               MessageDialog md = new MessageDialog(this,
"Message", true, message);
               md.show();
          }
          else try {
               System.out.println(" Parsing of reaction
string(s) containing the substring "+subStr+":");
               System.out.println();
               File f = new File("Reactions.txt");
               ReactionFinder in = new ReactionFinder(new
FileReader(f),subStr);
               String line;
               while((line = in.readLine()) != null) {
               System.out.println(" "+line);
               view.setVisible(true);
               view.showReactions(line);
               parser.ReactionParser(line);
               }
               in.close();
          }
          catch (Exception e) {
               System.out.println(e);
          }
     }
}

//Show reaction string(s) containing/excluding a given
substring
class SubStringThree extends Frame implements ActionListener,
WindowListener {

     ReactionString view = new ReactionString("Reaction(s)
Selected");
     int counter;
     TextField sub1,sub2;
     String subStr1,subStr2;

     //Constructor
     public SubStringThree(String title8) {

          super(title8);
          setBackground(Color.cyan);
```

```
        setLayout(new FlowLayout());

        //Create the text fields and the labels
        sub1 = new TextField(20);
        add(new Label("INCLUDE substring: "));
        add(sub1);
        sub1.addActionListener(this);

        sub2 = new TextField(20);
        add(new Label("EXCLUDE substring: "));
        add(sub2);
        sub2.addActionListener(this);
        add(new Label("Press enter to view results. "));
        addWindowListener(this);
    }
    //Method to check which window is closing
    public void windowClosing(WindowEvent we) {
        dispose();
        return;
    }

    //Implemented blank WindowListener methods
    public void windowClosed(WindowEvent we) {}
    public void windowDeiconified(WindowEvent we) {}
    public void windowIconified(WindowEvent we) {}
    public void windowActivated(WindowEvent we) {}
    public void windowDeactivated(WindowEvent we) {}
    public void windowOpened(WindowEvent we) {}

    public void actionPerformed(ActionEvent ae) {
        subStr1 = sub1.getText();
        subStr2 = sub2.getText();

        if(subStr1.equals("")) {
            String message = "Please enter a substring for
INCLUSION!";
            MessageDialog md = new MessageDialog(this,
"Message", true, message);
            md.show();
        }
        else if(subStr2.equals("")) {
            String message = "Please enter a substring for
EXCLUSION!";
            MessageDialog md = new MessageDialog(this,
"Message", true, message);
            md.show();
        }

        else try {
            System.out.println();
```

```
                System.out.println(" Reaction string(s)
containing the substring "+subStr1+":");
                System.out.println();
                File f1 = new File("Reactions.txt");
                ReactionFinder in1 = new ReactionFinder(new
FileReader(f1),subStr1);

//Write the first result to the 'Inclusion.txt' file and filter
this again
                FileOutputStream ostream = new
FileOutputStream("Inclusion.txt");
                PrintWriter pw = new PrintWriter(ostream);

                String line1;
                while((line1 = in1.readLine()) != null) {
                System.out.println(" "+line1);
                pw.println(line1);
                }
                in1.close();
                pw.flush();
                ostream.close();

                System.out.println();
                System.out.println();
                System.out.println(" Reaction string(s)
containing the substring "+subStr1+" but excluding the
substring "+subStr2+":");
                System.out.println();
                File f2 = new File("Inclusion.txt");
                ReactionExcluder in2 = new ReactionExcluder(new
FileReader(f2),subStr2);
                String line2;
                while((line2 = in2.readLine()) != null) {
                System.out.println(" "+line2);
                view.setVisible(true);
                view.showReactions(line2);
                }
                in2.close();
                sub1.setText("");//clear input
                sub2.setText("");//clear input
          }
          catch (Exception e) {
                System.out.println(e);
          }
     }
}

//Show reaction string(s) containing/excluding a given
substring and map the reaction(s)
class SubStringFour extends Frame implements ActionListener,
WindowListener {
```

```
     ReactionString view = new ReactionString("Reaction(s)
Selected");
     ReactionParser parser = new ReactionParser();
     int counter;
     TextField sub1,sub2;
     String subStr1,subStr2;

     //Constructor
     public SubStringFour(String title9) {

          super(title9);
          setBackground(Color.cyan);
          setLayout(new FlowLayout());

          //Create the text fields and the labels
          sub1 = new TextField(20);
          add(new Label("INCLUDE substring: "));
          add(sub1);
          sub1.addActionListener(this);

          sub2 = new TextField(20);
          add(new Label("EXCLUDE substring: "));
          add(sub2);
          sub2.addActionListener(this);
          add(new Label("Press enter to view results. "));
          addWindowListener(this);
     }
     //Method to check which window is closing
     public void windowClosing(WindowEvent we) {
          dispose();
          return;
     }

     //Implement blank WindowListener methods
     public void windowClosed(WindowEvent we) {}
     public void windowDeiconified(WindowEvent we) {}
     public void windowIconified(WindowEvent we) {}
     public void windowActivated(WindowEvent we) {}
     public void windowDeactivated(WindowEvent we) {}
     public void windowOpened(WindowEvent we) {}

     public void actionPerformed(ActionEvent ae) {
          subStr1 = sub1.getText();
          subStr2 = sub2.getText();

          if(subStr1.equals("")) {
               String message = "Please enter a substring for
INCLUSION!";
               MessageDialog md = new MessageDialog(this,
"Message", true, message);
```

```
                md.show();
            }
            else if(subStr2.equals("")) {
                String message = "Please enter a substring for
EXCLUSION!";
                MessageDialog md = new MessageDialog(this,
"Message", true, message);
                md.show();
            }

            else try {
                System.out.println();
                System.out.println(" Reaction string(s)
containing the substring "+subStr1+":");
                System.out.println();
                File f1 = new File("Reactions.txt");
                ReactionFinder in1 = new ReactionFinder(new
FileReader(f1),subStr1);

//Write the first result to the 'Inclusion.txt' file and filter
this again
                FileOutputStream ostream = new
FileOutputStream("Inclusion.txt");
                PrintWriter pw = new PrintWriter(ostream);

                String line1;
                while((line1 = in1.readLine()) != null) {
                    System.out.println(" "+line1);
                    pw.println(line1);
                }
                in1.close();
                pw.flush();
                ostream.close();

                System.out.println();
                System.out.println();
                System.out.println(" Parsing of reaction
string(s) containing the substring "+subStr1+" but excluding
the substring "+subStr2+":");
                System.out.println();
                File f2 = new File("Inclusion.txt");
                ReactionExcluder in2 = new ReactionExcluder(new
FileReader(f2),subStr2);
                String line2;
                while((line2 = in2.readLine()) != null) {
                    System.out.println(" "+line2);
                    System.out.println();
                    view.setVisible(true);
                    view.showReactions(line2);
                    parser.ReactionParser(line2);
                }
```

```
                in2.close();
                sub1.setText("");//clear input
                sub2.setText("");//clear input
            }
            catch (Exception e) {
                System.out.println(e);
            }
        }
}

//Class for the selection of classical or user-defined pathways
class ClassicalPathway extends Frame implements ActionListener
{

    GraphPanel panel = new GraphPanel("Drawing Area");
    ReactionParser parser = new ReactionParser();
    Analysis an = new Analysis();
    ReactionString view;

    Label label;
    private List list;
    String st;
    String pathway;
    String pathName;
    int len;
    int pos1;
    String message;
    int number;

    int nTabs;
    boolean modulator =false;
    String rCode[] = new String[100];

    //Constructor
    public ClassicalPathway(String title10) {

        super(title10);
        setBackground(Color.green);
        setLayout(new FlowLayout());

        add(new Checkbox("Check if you wish to to include
effectors."));

        label = new Label("Double-click pathway name: ");
        add(label);

        list = new List(4,false);
        list.add("Glycolysis");
        list.add("Gluconeogenesis");
        list.add("Pentose phosphate pathway");
        list.add("Urea cycle");
```

```
          add(list);
          list.addActionListener(this);

//Anonymous (unnamed) inner class handles window events
          addWindowListener(new WindowAdapter() {
               public void windowClosing(WindowEvent we) {
                    dispose();
          return;
               }
          });
     }
//Method to display message in 'Classical Pathways' window
     public void paint( Graphics g ) {
          g.drawRect(65,104,230,64);
          g.drawString( "SUCCESSIVE SELECTIONS WILL BE
ADDED",75,118);
          g.drawString( "TO THE SAME DISPLAY WINDOW. ",75,130);
          g.drawString( "TO START A NEW SELECTION, FIRST
CLOSE",75,154);
          g.drawString( "THIS WINDOW, AND THEN RESELECT IT.
",75,166);
     }

     public boolean action(Event evt, Object arg) {
                    if(arg instanceof Boolean) {

     if(((Checkbox)evt.target).getLabel().equals("Check if you
wish to to include effectors.")) {
                              modulator =
((Boolean)arg).booleanValue();
                         }
                         return true;
                    }
                    return false;
               }

     public void actionPerformed(ActionEvent ae) {

          if(ae.getSource() == list) {
               String listSelect = list.getSelectedItem();
               boolean tabs = true;
               int counter1 = 0;
               int counter2 = 0;
               int counter3 = 0;
               int j = 0;

          //Select Glycolysis file
          if(listSelect.equals("Glycolysis")) {
               pathway = "Glycolysis.dat";
```

```
                view = new ReactionString("Classical Glycolysis
Pathway Reactions");
                pathName = "Classical Glycolysis Pathway";
                System.out.println(" Glycolysis pathway
selected.");
                System.out.println();
           }
           //Select Gluconeogenesis file
           if(listSelect.equals("Gluconeogenesis")) {
                pathway = "Gluconeogenesis.dat";
                view = new ReactionString("Classical
Gluconeogenesis Pathway Reactions");
                pathName = "Classical Gluconeogenesis Pathway";
                System.out.println(" Gluconeogenesis pathway
selected.");
                System.out.println();
           }
           //Select Pentose phosphate pathway file
           else if(listSelect.equals("Pentose phosphate
pathway")) {
                pathway = "PPP.dat";
                view = new ReactionString("Classical Pentose
Phosphate Pathway Reactions");
                pathName = "Classical Pentose Phosphate
Pathway";
                System.out.println(" Pentose phosphate pathway
selected.");
                System.out.println();
           }
           //Select Urea cycle file
           if(listSelect.equals("Urea cycle")) {
                pathway = "UreaCycle.dat";
                view = new ReactionString("Classical Urea Cycle
Pathway Reactions");
                pathName = "Classical Urea Cycle Pathway";
                System.out.println(" Urea cycle pathway
selected.");
                System.out.println();
           }

           //Read pathway and effectors files
           if(modulator) {
                try{
                     File f = new File(pathway);//pathway file
                     FileReader fr = new FileReader(f);//create
a file reader
                     BufferedReader in = new
BufferedReader(fr);//create a buffered reader

                st = in.readLine();
                     while(st!= null) {
```

```
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String EC1 =  st.substring(0,pos1);
                    rCode[j] = EC1;
                    counter1++;
                    for(int i=0;i<st.length();i++) {
                         if(st.charAt(i) == '     ')
nTabs++;
                    }
                    if(nTabs != 2) {
                         System.out.println(" Input format
error in tab(s) in line "+counter1+" of "+pathway+" file!");
                         System.out.println();
                         tabs = false;
                    }

                    nTabs = 0;
                    parser.ReactionParser(st);
                    view.setVisible(true);
                    view.showReactions(st);
                    j++;
                    st = in.readLine();
               }
               System.out.println("Value of J: "+j);
          in.close();

          }
          catch(Exception e) {
               System.out.println(e.getMessage());
          }
          if(tabs) {
               System.out.println(" Number of records in
"+pathway+" file: "+counter1);
               System.out.println();
          }
          try{
               File f = new
File("Effectors.dat");//effectors file
               FileReader fr = new FileReader(f);//create
a file reader
               BufferedReader in = new
BufferedReader(fr);//create a buffered reader

          st = in.readLine();
               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String EC2 =  st.substring(0,pos1);
                    counter2++;
                    for(int i=0;i<st.length();i++) {
```

```
                    if(st.charAt(i) == '	')
nTabs++;
               }
               if(nTabs != 2) {
                    System.out.println(" Input format
error in tab(s) in line "+counter2+" of Effectors.dat file!");
                    System.out.println();
                    tabs = false;
               }

               nTabs = 0;
               for(int k=0;k<counter1;k++) {
                    if(EC2.equals(rCode[k])) {
                         parser.ReactionParser(st);
                         view.setVisible(true);
                         view.showReactions(st);
                    }
               }
               st = in.readLine();
          }
     in.close();
     message = "Substrate(s) & activator(s) on left
and product(s) & inhibitor(s) on right of operator.";
          MessageDialog md = new MessageDialog(this,
"Message", true, message);
          md.show();
          an.histogram(pathway,pathName);
     }
     catch(Exception e) {
          System.out.println(e.getMessage());
     }
     if(tabs) {
          System.out.println(" Number of records in
Effectors.dat file: "+counter2);
          System.out.println();
     }
   }//Read pathway file only
     else {
          try{
               File f = new File(pathway);
               FileReader fr = new
FileReader(f);//create a file reader
               BufferedReader in = new
BufferedReader(fr);//create a buffered reader

          st = in.readLine();
               while(st!= null) {
                    len = st.length();
                    pos1 = st.indexOf("\t");
                    String OC =
st.substring(0,pos1);//Operator code
```

```
                                        counter3++;
                                        for(int i=0;i<st.length();i++) {
                                             if(st.charAt(i) == '     ')
nTabs++;
                                        }
                                        if (nTabs != 2) {
                                             System.out.println(" Input
format error in tab(s) in line "+counter3+" of "+pathway+"
file!");
                                             System.out.println();
                                             tabs = false;
                                        }

                                        nTabs = 0;
                                        parser.ReactionParser(st);
                                        view.setVisible(true);
                                        view.showReactions(st);

                                        st = in.readLine();
                                   }
                              in.close();
                              message = "Substrate(s) on left and
product(s) on right of operator.";
                                   MessageDialog md = new
MessageDialog(this, "Message", true, message);
                                   md.show();
                                   an.histogram(pathway,pathName);
                              }
                              catch(Exception e) {
                              System.out.println(e.getMessage());
                              }
                              if(tabs) {
                                   System.out.println(" Number of records
in "+pathway+" file: "+counter3);
                                   System.out.println();
                              }
                         }
                    }
               }
          }
     }
}

//Class to filter out all reaction strings that do NOT contain
a given substring
public class ReactionFinder extends BufferedReader {

     String substring;//substring to be matched

//Pass the stream to the superclass and remember the substring
     public ReactionFinder(Reader in, String substring) {
          super(in);
          this.substring = substring;
```

```
    }

/*This filter calls the superclass's readline() to get the
actual reaction strings, but only returns reactionStrings that
contain the substring.*/
    public final String readLine() throws IOException {
        String reactionString;
        do {
            reactionString = super.readLine();
        }
        while((reactionString != null) &&
reactionString.indexOf(substring) == -1);
            return reactionString;
    }
}
//Class to filter out all reaction strings that contain a given
substring
public class ReactionExcluder extends BufferedReader {

    String substring;//substring to be matched

//Pass the stream to the superclass and remember the substring
    public ReactionExcluder(Reader in, String substring) {
        super(in);
        this.substring = substring;
    }

/*This filter calls the superclass's readline() to get the
actual reaction strings, but only returns reactionStrings that
do NOT contain the substring.*/
    public final String readLine() throws IOException {
        String reactionString;
        do {
            reactionString = super.readLine();
        }
        while((reactionString != null) &&
reactionString.indexOf(substring) != -1);
            return reactionString;
    }
}

//Subclass of Dialog to display messages to the user
class MessageDialog extends Dialog implements ActionListener {

    Button ok;

    //Constructor
    MessageDialog(Frame parent, String title, boolean mode,
String message) {

        super(parent, title, mode);
```

```
        setLocation(350,300);
        setBackground(Color.pink);

        //Create and add centre panel
        Panel pc = new Panel();
        Label label = new Label(message);
        pc.add(label);
        add(pc, "Center");

        //Create and add south panel
        Panel ps = new Panel();
        ok = new Button("OK");
        ok.addActionListener(this);
        ps.add(ok);
        add(ps, "South");

//Lay out components and set initial size of this dialogue box
        pack();

        //Anonymous inner class handles window events
        addWindowListener(new WindowAdapter() {
             public void windowClosing(WindowEvent we) {
                  System.exit(0);
             }
        });
    }

    public Insets getInsets() {
        return new Insets(10, 20, 10, 20);
    }

    public void actionPerformed(ActionEvent ae) {
        dispose();//remove the dialogue
    }
}
//Class to display simple histogram of operator code classes
class Analysis {

    ReactionString view;
    String pathway;
    String pathwayName;
    String st;
    int pos1;
    String ocNumber;
    int c1;
    int classNumber;
    int counter = 0;
    String output;
    //Method to receive pathway name and file name
    public void histogram(String path, String name) {
        pathway = path;
```

```
          pathwayName = name;
          int classNumber, frequency[] = new int[10];
          //Read pathway file
          try{
               File f = new File(pathway);
               FileReader fr = new FileReader(f);//create a
file reader
               BufferedReader in = new
BufferedReader(fr);//create a buffered reader

              st = in.readLine();
               while(st!= null) {
                    pos1 = st.indexOf("\t");
                    String ocNumber =
st.substring(0,pos1);//Operator code
                    c1 = ocNumber.indexOf(".");
                    String ocClass =
ocNumber.substring(0,c1);//Operator code class
                    classNumber = Integer.parseInt(ocClass);

     //Use class number as subscript for frequency array
                    ++frequency[classNumber];

                    counter++;
                    st = in.readLine();
               }
              in.close();

          }
          catch(Exception e) {
               System.out.println(e.getMessage());
          }
          view = new ReactionString("Class frequency histogram
for "+pathwayName);
          String output = "Class\t\tFrequency\tHistogram";

//Append frequencies to String output (outer 'for' loop)
          for(classNumber = 0;classNumber <
frequency.length;classNumber++) {
               output += "\n" + classNumber + "\t\t" +
frequency[classNumber]+"\t\t";

          //Print bar of asterisks (inner 'for' structure)
               for(int star = 0;star <
frequency[classNumber];star++)
               output += "*";
          }
          view.setVisible(true);
          view.showReactions(output);
     }
}
```

```
//Class to access information on named textbook pathways
class PathwayInfo extends Frame implements ActionListener,
WindowListener {

     ReactionString pathInfo = new ReactionString("Pathway
Information");

     String s;
     String st;
     int counter;

     //Create buttons with the names of pathways
     Button pushButton1 = new Button("Glycolysis");
     Button pushButton2 = new Button("Gluconeogenesis");
     Button pushButton3 = new Button("Pentose phosphate
pathway");
     Button pushButton4 = new Button("Urea cycle");
     Button pushButton5 = new Button("Pathway EC numbers");

     //Constructor
     public PathwayInfo(String title11) {
          super(title11);
          setBackground(Color.green);
          setLayout(new FlowLayout());

          add(pushButton1);
          add(pushButton2);
          add(pushButton3);
          add(pushButton4);
          add(pushButton5);
          pushButton1.addActionListener(this);
          pushButton2.addActionListener(this);
          pushButton3.addActionListener(this);
          pushButton4.addActionListener(this);
          pushButton5.addActionListener(this);
          addWindowListener(this);
     }
     //Method to check which window is closing
     public void windowClosing(WindowEvent we) {
          dispose();
          return;
     }

     //Implement blank WindowListener methods
     public void windowClosed(WindowEvent we) {}
     public void windowDeiconified(WindowEvent we) {}
     public void windowIconified(WindowEvent we) {}
     public void windowActivated(WindowEvent we) {}
     public void windowDeactivated(WindowEvent we) {}
     public void windowOpened(WindowEvent we) {}
```

```
     //Select pathway information file
     public void actionPerformed(ActionEvent ae) {
          if(ae.getActionCommand().equals("Glycolysis")) {
               s = "GlycolInfo.txt";
          }
          if(ae.getActionCommand().equals("Gluconeogenesis")) {
               s ="GluconeoInfo.txt";
          }
          if(ae.getActionCommand().equals("Pentose phosphate
pathway")) {
               s ="PPPInfo.txt";
          }
          if(ae.getActionCommand().equals("Urea cycle")) {
               s ="UreaCycleInfo.txt";
          }
          if(ae.getActionCommand().equals("Pathway EC
numbers")) {
               s ="EC Pathways.txt";
          }
          //Read selected file
          try{
               File f = new File(s);
               FileReader fr = new FileReader(f);
               BufferedReader in = new BufferedReader(fr);

              st = in.readLine();

               while(st!= null) {
                    pathInfo.setVisible(true);
                    pathInfo.showReactions(" "+st);
                    st = in.readLine();
              }
              in.close();
          }

          catch(Exception e) {
               System.out.println(e.getMessage());
          }
     }
}
//Class to access references
class Reference {

     ReactionString view;

     String s = "GenRef.txt";
     String ss = "SpecRef.txt";
     String st;
     String sst;
     int pos1;
```

```
String selRef[] = new String[100];
int i;
int counter;
String ocNumber;

//Method to access general literature references
public void genRef() {

     try{
          File f = new File(s);
          FileReader fr = new FileReader(f);
          BufferedReader in = new BufferedReader(fr);
          view = new ReactionString("General References");
         st = in.readLine();

          while(st!= null) {
               view.setVisible(true);
               view.showReactions(" "+st);
               st = in.readLine();
         }
          in.close();
               }

     catch(Exception e) {
          System.out.println(e.getMessage());
     }
}
//Method to access reaction-specific references
public void specifRef() {

     try{
          File f = new File(ss);
          FileReader fr = new FileReader(f);
          BufferedReader in = new BufferedReader(fr);
          view = new ReactionString("Reaction-specific
References");
         st = in.readLine();

          while(st!= null) {
               view.setVisible(true);
               view.showReactions(" "+st);
               st = in.readLine();
         }
          in.close();
               }

     catch(Exception e) {
          System.out.println(e.getMessage());
     }
}
//Method to access search-specific references
```

```
    public void getOC(String oc) {
        ocNumber = oc;

        try{
            File f = new File(ss);
            FileReader fr = new FileReader(f);
            BufferedReader in = new BufferedReader(fr);
            view = new ReactionString("Selected
Reference(s)");

//Write the first result to the 'selectedRef.txt' file and
access this again
            FileOutputStream ostream = new
FileOutputStream("selectedRef.txt");
            PrintWriter pw = new PrintWriter(ostream);

           sst = in.readLine();

            while(sst!= null) {
                pos1 = sst.indexOf("\t");
                String OC =  sst.substring(0,pos1);
                if(OC.equals(ocNumber)) {
                selRef[i] = sst;
                i++;
                counter++;
            }
                sst = in.readLine();
           }
           for(int j=0;j<counter;j++) {
                pw.println(selRef[j]);
                }
            in.close();
            pw.flush();
                ostream.close();
            }

        catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
//Class to display reaction-string information in a scrolling
window
class ReactionString extends Frame implements WindowListener {

    private static final int H = 200;//height of window
    private static final int W = 600;//width of window

    private TextArea outputArea;//visual text area (output)

    //Constructor
```

```
	public ReactionString(String title11) {
		super(title11);
		setLayout(null);//set layout manager (none)
		setSize(W,H);//set size of window
		setLocation(650,550);
		outputArea = new TextArea(10,40);//output area
		outputArea.setBounds(10,5,W-15,H-10);//size
		add(outputArea);
		outputArea.setEditable(false);//set the text area to
non-edit mode
		addWindowListener(this);
	}
	//Method to check which window is closing
	public void windowClosing(WindowEvent we) {
		dispose();
		return;
	}
	//Implement blank WindowListener methods
	public void windowClosed(WindowEvent we) {}
	public void windowDeiconified(WindowEvent we) {}
	public void windowIconified(WindowEvent we) {}
	public void windowActivated(WindowEvent we) {}
	public void windowDeactivated(WindowEvent we) {}
	public void windowOpened(WindowEvent we) {}

	public void showReactions(String output) {
		String display = output;
		outputArea.append(display+"\n");
	}
}

//Class to break down reaction strings and assemble n-ary
relations
public class ReactionParser extends Frame {

	HTML code = new HTML();
	Relation rel = new Relation();

	//Declare variables and initialise some of them
	String s;
	String origS;
	String newString;
	String opReaction = "";
	int position1;
	int position2 = 0;
	int position3 = 0;
	int position4;
	int len;
	int p2 = 1;
	int p3 = 1;
	int separateLeft;
```

```
        int separateRight;
        int h = 1;
        int i = 1;
        int j = 1;
        int k = 1;
        String leftBinary = "";
        String rightBinary = "";
        String leftRightBinary = "";
        String nAry = "";
        String newBinLeft = "";
        String newBinRight = "";
        String binary = "";

//Arrays to hold parts, compounds and stoichiometric
coefficients
        String partLeftSide[] = new String[100];
        String partRightSide[] = new String[100];
        String cmpdLeftSide[] = new String[100];
        String cmpdLeftSideL[] = new String[100];
        String cmpdRightSide[] = new String[100];
        String cmpdRightSideR[] = new String[100];
        String stoichLeftSide[] = new String[100];
        String stoichRightSide[] = new String[100];

//Method to parse reaction strings and form n-ary relations
        public void ReactionParser(String str) {
                origS = str;
                s = str;
                len = s.length();//length of original string
                System.out.println();
                System.out.println(" Original reaction string: "+s);
                System.out.println();
                System.out.println();

                //Operator code (OC)
                position1 = s.indexOf("\t");
                String numberOC =  s.substring(0,position1);
                System.out.println(" Operator code: "+numberOC+"
(stored in variable numberOC)");
                System.out.println();

                //Operator reaction/regulatory action
                opReaction = s.substring(position1+1,len);
                System.out.println(" Operator reaction/regulatory
action: "+opReaction+" (stored in variable opReaction)");
                len = opReaction.length();
                System.out.println();
                System.out.println();

                //Operator
                position1 = opReaction.indexOf("\t");
```

```
          String Operator =  opReaction.substring(0,position1);
          System.out.println(" Operator: "+Operator+" (stored
in variable Operator)");
          System.out.println();

          //Reaction/regulatory action
          String reaction =
opReaction.substring(position1+1,len);
          System.out.println(" Reaction/regulatory action:
"+reaction+" (stored in variable reaction)");
          len = reaction.length();
          System.out.println();
          System.out.println();

          //Replace ':' with '='
          reaction = reaction.replace(':','=');

          int p1 = reaction.indexOf(" = ");
          int leftEqual = reaction.indexOf("= ");
          int rightEqual = reaction.indexOf(" =");
          int nld = reaction.indexOf("=");

          //Case where there is no equals sign
          if(nld == -1) {
               System.out.println(" This reaction is only
available as a natural language description! ");
               System.out.println(" It cannot be displayed.");
               System.out.println();
               String message = "Reaction "+numberOC+" is only
available as a natural language description! See Java
Console.";
               MessageDialog md = new MessageDialog(this,
"Message", true, message);
               md.show();
               }

//Case where there is no space on either side of 'equals' sign
if((leftEqual == -1 || rightEqual == -1) && nld != -1) {
               System.out.println(" Input format error in
substring containing equality sign in reaction "+numberOC);
               String message = "Input format error in reaction
"+numberOC+". See Java Console.";
               MessageDialog md = new MessageDialog(this,
"Message", true, message);
               md.show();
          }
          //LHS of reaction
          String left = reaction.substring(0,p1);
          System.out.println(" Left side of reaction/regulatory
action: "+left+" (stored in variable left)");
          System.out.println();
```

```
            int lenLeft = left.length();

            int indexLeft = left.indexOf(" + ");

            /*Case where there is no plus sign*/
            if(indexLeft == -1) {
                 System.out.println();
                 String newStringLeft = left.substring(0,p1);
                 partLeftSide[i] = newStringLeft;
                 System.out.println(" Part "+i+" of left side:
"+partLeftSide[i]+" (stored at index "+i+" of array
partLeftSide)");
                 System.out.println();

                 int lenPartLeft = partLeftSide[i].length();
                 separateLeft = partLeftSide[i].indexOf(" ");

//No space in compound name and stoichiometry is implicit
                 if(separateLeft == -1) {
                      cmpdLeftSide[i] = partLeftSide[i];
                      cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
                      stoichLeftSide[i] = "1";//implicit
stoichiometry (as string)
                      System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                      System.out.println();
                      System.out.println(" Coefficient "+i+" of
left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                      System.out.println();
                      System.out.println();
                 }
            //There may be a space in the compound name
                 else {
            //Explicit stoichiometry between 1 and 9

      if((partLeftSide[i].substring(0,separateLeft).length() ==
1)) {
                      cmpdLeftSide[i] =
partLeftSide[i].substring(2,lenPartLeft);
                      cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
                      System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                      System.out.println();
                      stoichLeftSide[i] =
partLeftSide[i].substring(0,separateLeft);//explicit
stoichiometry (as string)
```

```
                    if(stoichLeftSide[i].equals("A") ||
stoichLeftSide[i].equals("a")) stoichLeftSide[i] = "1";
                    System.out.println(" Coefficient "+i+" of
left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                    System.out.println();
                    System.out.println();
               }
//Explicit stoichiometry between 10 and 99
               else
if((partLeftSide[i].substring(0,separateLeft).length() == 2)) {
                    cmpdLeftSide[i] =
partLeftSide[i].substring(3,lenPartLeft);
                    cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
                    System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                    System.out.println();
                    stoichLeftSide[i] =
partLeftSide[i].substring(0,separateLeft);//explicit
stoichiometry (as string)
                    if(stoichLeftSide[i].equals("An")
||stoichLeftSide[i].equals("an")) stoichLeftSide[i] = "1";
                    System.out.println(" Coefficient "+i+" of
left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                    System.out.println();
                    System.out.println();
               }
               //Implicit stoichiometry
               else {
                    cmpdLeftSide[i] =
partLeftSide[i].substring(0,lenPartLeft);
                    cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
                    System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                    System.out.println();
                    stoichLeftSide[i] = "1";//implicit
stoichiometry (as string)
                    System.out.println(" Coefficient "+i+" of
left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                    System.out.println();
                    System.out.println();
               }
          }

//Case for no activator - only applies before plus sign(s)
          if(cmpdLeftSideL[i].equals("|L")) leftBinary = "";
```

```
          else leftBinary =
Operator+"$"+cmpdLeftSideL[i]+"/"+stoichLeftSide[i]+";";
     }
     /*Case where there is at least one plus sign*/
     else {
     while(position2 < p2) {

          position2 = left.indexOf(" + ");
          p2 = left.lastIndexOf(" + ");
          System.out.println();

         String newStringLeft =
left.substring(0,position2);//substring from 0 to " + "
          partLeftSide[i] = newStringLeft;
          int lenPartLeft = partLeftSide[i].length();
          left =
left.substring(position2+3,lenLeft);//remainder of string from
" + "

          System.out.println(" Part "+i+" of left side:
"+partLeftSide[i]+" (stored at index "+i+" of array
partLeftSide)");
          System.out.println();

          separateLeft = partLeftSide[i].indexOf(" ");
//There is no space in the compound name and the stoichiometry
is implicit
          if(separateLeft == -1) {
               cmpdLeftSide[i] = partLeftSide[i];
               cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
               stoichLeftSide[i] = "1";//implicit stoichiometry
(as string) on LHS
               System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
               System.out.println();
               System.out.println(" Coefficient "+i+" of left
side: "+stoichLeftSide[i]+" (stored at index "+i+" of array
stoichLeftSide)");
               System.out.println();
          }
          //There may be a space in the compound name
          else {
          //Explicit stoichiometry between 1 and 9

     if((partLeftSide[i].substring(0,separateLeft).length() ==
1)) {
               cmpdLeftSide[i] =
partLeftSide[i].substring(2,lenPartLeft);
               cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
```

```
                System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                System.out.println();
                stoichLeftSide[i] =
partLeftSide[i].substring(0,separateLeft);//as string
                if(stoichLeftSide[i].equals("A") ||
stoichLeftSide[i].equals("a")) stoichLeftSide[i] = "1";
                System.out.println(" Coefficient "+i+" of left
side: "+stoichLeftSide[i]+" (stored at index "+i+" of array
stoichLeftSide)");
                System.out.println();
                System.out.println();
            }
            //Explicit stoichiometry between 10 and 99
            else
if((partLeftSide[i].substring(0,separateLeft).length() == 2)) {
                cmpdLeftSide[i] =
partLeftSide[i].substring(3,lenPartLeft);
                cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
                System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                System.out.println();
                stoichLeftSide[i] =
partLeftSide[i].substring(0,separateLeft);//as string
                if(stoichLeftSide[i].equals("an"))
stoichLeftSide[i] = "1";
                System.out.println(" Coefficient "+i+" of left
side: "+stoichLeftSide[i]+" (stored at index "+i+" of array
stoichLeftSide)");
                System.out.println();
                System.out.println();
            }
            //Implicit stoichiometry
            else {
                cmpdLeftSide[i] =
partLeftSide[i].substring(0,lenPartLeft);
                cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
                System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                System.out.println();
                stoichLeftSide[i] = "1";//implicit stoichiometry
(as string) on LHS
                System.out.println(" Coefficient "+i+" of left
side: "+stoichLeftSide[i]+" (stored at index "+i+" of array
stoichLeftSide)");
                System.out.println();
                }
            }
```

```
                lenLeft = left.length();
                System.out.println(" Remainder: "+left);

     //Case where an operator equals a non-operator
     if(Operator.equals(cmpdLeftSide[i])) leftBinary += "";
                else leftBinary +=
Operator+"$"+cmpdLeftSideL[i]+"/"+stoichLeftSide[i]+";";

                i++;
                h++;

                }
/*Case where there are no more plus signs left*/
                System.out.println();
                partLeftSide[i] = left;
                System.out.println(" Part "+i+" of left side:
"+partLeftSide[i]+" (stored at index "+i+" of array
partLeftSide)");
                System.out.println();

                int lenPartLeft = partLeftSide[i].length();
                separateLeft = partLeftSide[i].indexOf(" ");
//There is no space in the compound name and the stoichiometry
is implicit
                if(separateLeft == -1) {
                     cmpdLeftSide[i] = partLeftSide[i];
                     cmpdLeftSideL[i] = cmpdLeftSide[i]+"|L";
                     stoichLeftSide[i] = "1";//implicit
stoichiometry (as string) on LHS
                     System.out.println(" Compound "+i+" of left
side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                     System.out.println();
                     System.out.println(" Coefficient "+i+" of
left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                     System.out.println();
                     System.out.println();
                }
                //There may be a space in the compound name
                else {
                //Explicit stoichiometry between 1 and 9

     if((partLeftSide[i].substring(0,separateLeft).length() ==
1)) {
                          cmpdLeftSide[i] =
partLeftSide[i].substring(2,lenPartLeft);
                          cmpdLeftSideL[i] =
cmpdLeftSide[i]+"|L";
```

```
                        System.out.println(" Compound "+i+" of
left side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                        System.out.println();
                        stoichLeftSide[i] =
partLeftSide[i].substring(0,separateLeft);//as string
                        if(stoichLeftSide[i].equals("A") ||
stoichLeftSide[i].equals("a")) stoichLeftSide[i] = "1";
                        System.out.println(" Coefficient "+i+"
of left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                        System.out.println();
                        System.out.println();
                    }
//Explicit stoichiometry between 10 and 99
                    else
if((partLeftSide[i].substring(0,separateLeft).length() == 2)) {
                        cmpdLeftSide[i] =
partLeftSide[i].substring(3,lenPartLeft);
                        cmpdLeftSideL[i] =
cmpdLeftSide[i]+"|L";
                        System.out.println(" Compound "+i+" of
left side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                        System.out.println();
                        stoichLeftSide[i] =
partLeftSide[i].substring(0,separateLeft);//as string
                        if(stoichLeftSide[i].equals("an"))
stoichLeftSide[i] = "1";
                        System.out.println(" Coefficient "+i+"
of left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                        System.out.println();
                        System.out.println();
                    }
                    //Implicit stoichiometry
                    else {
                        cmpdLeftSide[i] =
partLeftSide[i].substring(0,lenPartLeft);
                        cmpdLeftSideL[i] =
cmpdLeftSide[i]+"|L";
                        System.out.println(" Compound "+i+" of
left side: "+cmpdLeftSide[i]+" (stored at index "+i+" of array
cmpdLeftSide)");
                        System.out.println();
                        stoichLeftSide[i] = "1";//implicit
stoichiometry (as string) on LHS
                        System.out.println(" Coefficient "+i+"
of left side: "+stoichLeftSide[i]+" (stored at index "+i+" of
array stoichLeftSide)");
                        System.out.println();
```

```
                    System.out.println();
                    }
                }

     //Case where an operator equals a non-operator
     if(Operator.equals(cmpdLeftSide[i])) leftBinary += "";
                    else leftBinary +=
Operator+"$"+cmpdLeftSideL[i]+"/"+stoichLeftSide[i]+";";

                }
                System.out.println(" Number of compounds on LHS
= "+i);
                System.out.println();
                System.out.println();

          //RHS of reaction
          String right = reaction.substring(p1+3,len);
          System.out.println(" Right side of
reaction/regulatory action: "+right+" (stored in variable
right)");
          System.out.println();

          int lenRight = right.length();

          int indexRight = right.indexOf(" + ");

          /*Case where there is no plus sign*/
          if(indexRight == -1) {
               System.out.println();
               String newStringRight =
right.substring(0,lenRight);
               partRightSide[j] = newStringRight;
               System.out.println(" Part "+j+" of right side:
"+partRightSide[j]+" (stored at index "+j+" of array
partRightSide)");
               System.out.println();

               int lenPartRight = partRightSide[j].length();
               separateRight = partRightSide[j].indexOf(" ");

//There is no space in the compound name and the stoichiometry
is implicit
               if(separateRight == -1) {
                    cmpdRightSide[j] = partRightSide[j];
                    cmpdRightSideR[j] = cmpdRightSide[j]+"|R";
                    stoichRightSide[j] = "1";//implicit
stoichiometry (as string) on RHS
                    System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                    System.out.println();
```

```
                    System.out.println(" Coefficient "+j+" of
right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                    System.out.println();
                    System.out.println();
                }
                //There may be a space in the compound name
                else {
                //explicit stoichiometry between 1 and 9

     if((partRightSide[j].substring(0,separateRight).length()
== 1)) {
                        cmpdRightSide[j] =
partRightSide[j].substring(2,lenPartRight);
                        cmpdRightSideR[j] =
cmpdRightSide[j]+"|R";
                        System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                        System.out.println();
                        stoichRightSide[j] =
partRightSide[j].substring(0,separateRight);//as string
                        if(stoichRightSide[j].equals("A") ||
stoichRightSide[j].equals("a")) stoichRightSide[j] = "1";
                        System.out.println(" Coefficient "+j+"
of right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                        System.out.println();
                        System.out.println();
                    }
//Explicit stoichiometry between 10 and 99
                    else
if((partRightSide[j].substring(0,separateRight).length() == 2))
{
                        cmpdRightSide[j] =
partRightSide[j].substring(3,lenPartRight);
                        cmpdRightSideR[j] =
cmpdRightSide[j]+"|R";
                        System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                        System.out.println();
                        stoichRightSide[j] =
partRightSide[j].substring(0,separateRight);//as string
                        if(stoichRightSide[j].equals("An") ||
stoichRightSide[j].equals("an")) stoichRightSide[j] = "1";
                        System.out.println(" Coefficient "+j+"
of right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                        System.out.println();
                        System.out.println();
```

```
                    }
                    //Implicit stoichiometry
                    else {
                         cmpdRightSide[j] =
partRightSide[j].substring(0,lenPartRight);
                         cmpdRightSideR[j] =
cmpdRightSide[j]+"|R";
                         System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                         System.out.println();
                         stoichRightSide[j] = "1";//implicit
stoichiometry (as string) on RHS
                         System.out.println(" Coefficient "+j+"
of right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                         System.out.println();
                         System.out.println();
                    }
               }
//Case for no inhibitor - only applies before plus sign(s)
if(cmpdRightSideR[j].equals("|R")) rightBinary = "";

               else rightBinary =
Operator+"$"+cmpdRightSideR[j]+"/"+stoichRightSide[j]+";";
          }
          /*Case where there is at least one plus sign*/
          else {
          while(position3 < p3) {

               position3 = right.indexOf(" + ");
               p3 = right.lastIndexOf(" + ");
               System.out.println();

          String newStringRight =
right.substring(0,position3);
               partRightSide[j] = newStringRight;
               right = right.substring(position3+3,lenRight);
               int lenPartRight = partRightSide[j].length();

               System.out.println(" Part "+j+" of right side:
"+partRightSide[j]+" (stored at index "+j+" of array
partRightSide)");
               System.out.println();
               separateRight = partRightSide[j].indexOf(" ");
//There is no space in the compound name and the stoichiometry
is implicit
               if(separateRight == -1) {
                    cmpdRightSide[j] = partRightSide[j];
                    cmpdRightSideR[j] = cmpdRightSide[j]+"|R";
```

```
                    stoichRightSide[j] = "1";//implicit
stoichiometry (as string) on RHS
                    System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                    System.out.println();
                    System.out.println(" Coefficient "+j+" of
right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                    System.out.println();
                }
                //There may be a space in the compound name
                else {
                //Explicit stoichiometry between 1 and 9

     if((partRightSide[j].substring(0,separateRight).length()
== 1)) {
                        cmpdRightSide[j] =
partRightSide[j].substring(2,lenPartRight);
                        cmpdRightSideR[j] =
cmpdRightSide[j]+"|R";
                        System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                        System.out.println();
                        stoichRightSide[j] =
partRightSide[j].substring(0,separateRight);//as string
                        if(stoichRightSide[j].equals("A") ||
stoichRightSide[j].equals("a")) stoichRightSide[j] = "1";
                        System.out.println(" Coefficient "+j+"
of right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                        System.out.println();
                        System.out.println();
                    }
//Explicit stoichiometry between 10 and 99
                    else
if((partRightSide[j].substring(0,separateRight).length() == 2))
{
                        cmpdRightSide[j] =
partRightSide[j].substring(3,lenPartRight);
                        cmpdRightSideR[j] =
cmpdRightSide[j]+"|R";
                        System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                        System.out.println();
                        stoichRightSide[j] =
partRightSide[j].substring(0,separateRight);//as string
                        if(stoichRightSide[j].equals("An") ||
stoichRightSide[j].equals("an")) stoichRightSide[j] = "1";
```

```
                    System.out.println(" Coefficient "+j+"
of right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                    System.out.println();
                    System.out.println();
               }
               //Implicit stoichiometry
               else {
                    cmpdRightSide[j] =
partRightSide[j].substring(0,lenPartRight);
                    cmpdRightSideR[j] =
cmpdRightSide[j]+"|R";
                    System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                    System.out.println();
                    stoichRightSide[j] = "1";//implicit
stoichiometry (as string) on RHS
                    System.out.println(" Coefficient "+j+"
of right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                    System.out.println();
               }
          }
          lenRight = right.length();
          System.out.println(" Remainder: "+right);

     //Case where an operator equals a non-operator
     if(Operator.equals(cmpdRightSide[j])) rightBinary += "";
          else rightBinary +=
Operator+"$"+cmpdRightSideR[j]+"/"+stoichRightSide[j]+";";

          j++;
          k++;
     }
/*Case where there there are no more plus signs*/
          System.out.println();
          partRightSide[j] = right;
          System.out.println(" Part "+j+" of right side:
"+partRightSide[j]+" (stored at index "+j+" of array
partRightSide)");
          System.out.println();

          int lenPartRight = partRightSide[j].length();
          separateRight = partRightSide[j].indexOf(" ");
//Case where there is no space in the compound name and the
stoichiometry is implicit
          if(separateRight == -1) {
               cmpdRightSide[j] = partRightSide[j];
               cmpdRightSideR[j] = cmpdRightSide[j]+"|R";
```

```
                    stoichRightSide[j] = "1";//implicit
stoichiometry (as string) on RHS
                    System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                    System.out.println();
                    System.out.println(" Coefficient "+j+" of
right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                    System.out.println();
                }
                //There may be a space in the compound name
                else {
                //Explicit stoichiometry between 1 and 9

     if((partRightSide[j].substring(0,separateRight).length()
== 1)) {
                    cmpdRightSide[j] =
partRightSide[j].substring(2,lenPartRight);
                    cmpdRightSideR[j] = cmpdRightSide[j]+"|R";
                    System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                    System.out.println();
                    stoichRightSide[j] =
partRightSide[j].substring(0,separateRight);//as string
                    if(stoichRightSide[j].equals("A") ||
stoichRightSide[j].equals("a")) stoichRightSide[j] = "1";
                    System.out.println(" Coefficient "+j+" of
right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                    System.out.println();
                    System.out.println();
                }
                //Explicit stoichiometry between 10 and 99
                    else
if((partRightSide[j].substring(0,separateRight).length() == 2))
{
                        cmpdRightSide[j] =
partRightSide[j].substring(3,lenPartRight);
                        cmpdRightSideR[j] =
cmpdRightSide[j]+"|R";
                        System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                        System.out.println();
                        stoichRightSide[j] =
partRightSide[j].substring(0,separateRight);//as string
                        if(stoichRightSide[j].equals("An") ||
stoichRightSide[j].equals("an")) stoichRightSide[j] = "1";
```

```
                        System.out.println(" Coefficient "+j+"
of right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                        System.out.println();
                        System.out.println();
                    }
                //Implicit stoichiometry
                else {
                    cmpdRightSide[j] =
partRightSide[j].substring(0,lenPartRight);
                    cmpdRightSideR[j] = cmpdRightSide[j]+"|R";
                    stoichRightSide[j] = "1";//implicit
stoichiometry (as string) on RHS
                    System.out.println(" Compound "+j+" of
right side: "+cmpdRightSide[j]+" (stored at index "+j+" of
array cmpdRightSide)");
                    System.out.println();
                    System.out.println(" Coefficient "+j+" of
right side: "+stoichRightSide[j]+" (stored at index "+j+" of
array stoichRightSide)");
                    System.out.println();
                    }
                }
     //Case where an operator equals a non-operator
     if(Operator.equals(cmpdRightSide[j])) rightBinary += "";
                else rightBinary +=
Operator+"$"+cmpdRightSideR[j]+"/"+stoichRightSide[j]+";";
          }
System.out.println(" Number of compounds on RHS = "+j);
                System.out.println();
                System.out.println();

                leftRightBinary = leftBinary + rightBinary;
                nAry += leftRightBinary;

                //Reinitialise variables
                leftBinary = "";
                rightBinary = "";
                leftRightBinary = "";

                //Reinitialise parameters
                position2 = 0;
                position3 = 0;
                p2 = 1;
                p3 = 1;

                //Reinitialise arrays
                for(i=1;i<partLeftSide.length;i++) {
                partLeftSide[i] = "";
                cmpdLeftSide[i] = "";
                cmpdLeftSideL[i] = "";
```

```
                stoichLeftSide[i] = "";
                }
                for(j=1;i<partRightSide.length;j++) {
                partRightSide[j] = "";
                cmpdRightSide[j] = "";
                cmpdRightSideR[j] = "";
                stoichRightSide[j] = "";
                }

                //Reinitialise indices
                h = 1;
                i = 1;
                j = 1;
                k = 1;

               System.out.println(" N-ary relation(s):");
               System.out.println();
               System.out.println(" "+nAry);
               System.out.println();
//The n-ary relation(s) are passed to getCode() method of class
HTML
          code.getCode(nAry);
//The n-ary relation(s) are passed to getRelations() method of
class Relation
          rel.getRelations(nAry);
     }
}
//Class to generate an HTML document
class HTML {

     public HTML() {}

     public void getCode(String nAry) {

          try {
               //Write html code to html.txt file
               FileOutputStream ostream = new
               FileOutputStream("html.txt");
               PrintWriter pw = new PrintWriter(ostream);

               //Html code
              pw.println("  <title>TrivialApplet</title>");
              pw.println("  <hr>");
              pw.println("  <applet codebase=\"Java Classes\"
code=\"TrivialApplet.class\" width=1150 height=700>");
              pw.print("  <param name=GET_EVENT(S) value=\"");
              pw.print(nAry);//n-ary relations
              pw.print("\">");
              pw.println();
              pw.println("  </applet>");
              pw.println("  <hr>");
```

```
                pw.flush();
                 ostream.close();
                }
            catch (Exception e) {
                 System.out.println(e);
            }
        }
}
//Class to process n-ary relations
class Relation extends Frame {

     GraphPanel panel = new GraphPanel("Drawing Area");

     int width = 1100;
     int height = 700;

     public Relation() {}

     public void getRelations(String nAry) {
          panel.setSize(width,height);
          panel.setVisible(true);
          panel.message1();//call the 'CLICK HERE!' message

//Create a string tokenizer and process tokens
for(StringTokenizer t = new StringTokenizer(nAry, ";");
t.hasMoreTokens();) {
                String str = t.nextToken();
                //Separate components of each n-ary relation
                int i = str.indexOf('$');
                if(i > 0) {
                     int len = 50;
                     int j = str.indexOf('|');
                     if(j > 0) {
                          String side = str.substring(j+1,j+2);
                          int k = str.indexOf('/');
                         if(k > 0) {
                          String stoich = str.substring(k+1);
                               str = str.substring(0,j);

     panel.addEdge(str.substring(0,i),str.substring(i+1),len,si
de,stoich);

     panel.fixedNode(str.substring(0,i));
                          }
                     }
                }
          }
     }
}
```

```
/* Drawing classes: Node, Edge and GraphPanel */
class Node {

     double x;
     double y;
     double dx;
     double dy;
     boolean fixed;
     String lbl;
}

class Edge {

     int from;
     int to;
     String stoich;
     double len;
     String side;
}

class GraphPanel extends Frame implements Runnable,
ActionListener {

     TrivialApplication graph;
     Thread relaxer;
     String stoich;
     boolean relaxed;
     boolean reactionSide;
     boolean reactionColour;
     boolean stoichiometry;
     boolean thick;
     boolean showNode;
     int nnodes;
     int nedges;

     //Constructor
     public GraphPanel(String title13) {
          super(title13);
          setLayout(new FlowLayout());

          //Check boxes
          add(new Checkbox("Side indicator (L or R)"));
          add(new Checkbox("Side indicator (Colour)"));
          add(new Checkbox("Reaction stoichiometry"));
          add(new Checkbox("Thick edge"));
          add(new Checkbox("Show node borders"));
          add(new Checkbox("Allow NOPs to move"));

          //Buttons
          add(new Button("Shake NOPs"));
          Button pushButton = new Button("Close window");
```

```
            pushButton.setSize(100,40);
            add(pushButton);
            pushButton.addActionListener(this);

            //Start thread
            relaxer = new Thread(this);
            relaxer.start();
      }
      //Method to close window
      public void actionPerformed(ActionEvent ae) {
            if(ae.getActionCommand().equals("Close window")) {
                  dispose();
                  return;
            }
      }
//Method to change boolean value when check box or button is
clicked
      public boolean action(Event evt, Object arg) {
            if(arg instanceof Boolean) {

      if(((Checkbox)evt.target).getLabel().equals("Side
indicator (L or R)")) {
                  reactionSide = ((Boolean)arg).booleanValue();
                  }
                  else
if(((Checkbox)evt.target).getLabel().equals("Side indicator
(Colour)")) {
                  reactionColour = ((Boolean)arg).booleanValue();
                  }
                  else
if(((Checkbox)evt.target).getLabel().equals("Reaction
stoichiometry")) {
                  stoichiometry = ((Boolean)arg).booleanValue();

                  }
                  else
if(((Checkbox)evt.target).getLabel().equals("Thick edge")) {
                        thick = ((Boolean)arg).booleanValue();
                  }
                  else
if(((Checkbox)evt.target).getLabel().equals("Show node
borders")) {
                        showNode = ((Boolean)arg).booleanValue();

                  }
                  else
if(((Checkbox)evt.target).getLabel().equals("Allow NOPs to
move")) {
                        relaxed = ((Boolean)arg).booleanValue();

                  }
```

```
                return true;
            }
            if("Shake NOPs".equals(arg)) {
                Dimension d = size();
                for(int i = 0;i < nnodes;i++) {
                    Node n = nodes[i];
                    if(!n.fixed) {
                        n.x += 80*Math.random() - 40;
                        n.y += 80*Math.random() - 40;
                    }
                }
                return true;
            }
            return false;
        }

        int numberNodes;
        Node nodes[] = new Node[100000];


        Edge edges[] = new Edge[200000];

        int j;

        GraphPanel(TrivialApplication graph) {
            this.graph = graph;
        }

        //Method to fix position of an operator node
        public void fixedNode(String op) {
            if(op != null) {
                Node n = nodes[findNode(op)];
                n.fixed = true;
            }
        }
    //Method to search for a node or call addNode() to create it
        int findNode(String lbl) {
            for(int i = 0;i < nnodes;i++) {
                if(nodes[i].lbl.equals(lbl)) {
                    return i;
                }
            }
        return addNode(lbl);
        }

        //Method to create a new node
        int addNode(String lbl) {
            Node n = new Node();

            int k=1;
           n.x = 158;
```

```
        //Node layout algorithm
        for(int i=-1;i<j;i++) {
              n.y = 100 + 16*k;
              k++;
              if (n.y >= 650) {
                   n.x = n.x + 200;
                   k=1;
             }
          }

          n.lbl = lbl;

          j++;

          nodes[nnodes] = n;
          return nnodes++;
     }
//Method to create a new edge with labels and send node names
to findNode()
     void addEdge(String from, String to, int len, String
side,String stoich) {
          Edge e = new Edge();
          e.from = findNode(from);
          e.to = findNode(to);
          e.len = len;
          e.side = side;
          e.stoich = stoich;
          edges[nedges++] = e;
     }
     //Method that contains the logic for the thread
     public void run() {
          while(true) {
               if(relaxed) {
               relax();
               }
                    try {
                         Thread.sleep(100);
                    }
                    catch(InterruptedException e) {
                         break;
                    }
               }
          }
//Algorithm for the motion of nodes
     synchronized void relax() {
          for(int i = 0;i < nedges;i++) {
               Edge e = edges[i];
               double vx = nodes[e.to].x - nodes[e.from].x;
               double vy = nodes[e.to].y - nodes[e.from].y;
               double len = Math.sqrt(vx * vx + vy * vy);
               double f = (edges[i].len - len)/(len*3);
```

```
        double dx = f*vx;
        double dy = f*vy;

        nodes[e.to].dx += dx;
        nodes[e.to].dy += dy;
        nodes[e.from].dx += -dx;
        nodes[e.from].dy += -dy;
    }

    for(int i = 0;i < nnodes;i++) {
        Node n1 = nodes[i];
        double dx = 0;
        double dy = 0;

        for(int j = 0;j < nnodes;j++) {
            if(i == j) {
                continue;
            }

            Node n2 = nodes[j];
            double vx = n1.x - n2.x;
            double vy = n1.y - n2.y;
            double len = vx*vx + vy*vy;
            if(len == 0) {
                dx += Math.random();
                dy += Math.random();
                }
                else if(len < 100*100) {
                    dx += vx/len;
                    dy += vy/len;
                }
            }

            double dlen = dx*dx + dy*dy;
            if(dlen > 0) {
                dlen = Math.sqrt(dlen)/2;
                n1.dx += dx/dlen;
                n1.dy += dy/dlen;
            }
        }

        Dimension d = size();
        int width = 1000;
        int height = 500;
        for(int i = 0;i < nnodes;i++) {
            Node n = nodes[i];
            if(!n.fixed) {
                n.x += Math.max(-5,Math.min(5,n.dx));
                n.y += Math.max(-5,Math.min(5,n.dy));
                if(n.x < 0) {
                    n.x = 0;
```

```
                    }else if(n.x > d.width) {
                        n.x = d.width;
                    }
                    if(n.y < 0) {
                        n.y = 0;
                    }else if(n.y > d.height) {
                        n.y = d.height;
                    }
                }
                n.dx /= 2;
                n.dy /= 2;
            }
            repaint();
        }

        Node pick;
        Node n;
        boolean pickfixed;
        Image offscreen;
        Dimension offscreensize;
        Graphics offgraphics;
        int pos;

        //Node colours
        final Color fixedColor = Color.yellow;
        final Color nodeColor1 = Color.white;
        final Color nodeColor2 = Color.magenta;
        final Color selectColor = Color.pink;
        //Edge colours
        final Color edgeColor = Color.black;
        final Color arcColor1 = Color.black;
        final Color arcColor2 = Color.blue;
        final Color arcColor3 = Color.green;

        //Method to display message
        public void message1() {
        Graphics g = getGraphics();
        g.setFont(new Font("Serif", Font.BOLD, 40));//set
font
        g.setColor(Color.black);
        g.drawString("CLICK HERE!",400, 300);

    }
    //Method to draw nodes
    public void paintNode(Graphics g, Node n, FontMetrics fm)
{
            int x = (int)n.x;
            int y = (int)n.y;

            int membrane = 0;
            String prefix = n.lbl.substring(0, 2);
```

```
                if(prefix.equals("MM")) {
                     membrane = 1;
                }

/* ?: is a ternary (3-way) if-then-else operator; if expression
1 is true, then expression 2 is evaluated, otherwise,expression
3 is evaluated*/
g.setColor((n == pick) ? selectColor : (membrane == 1 ?
nodeColor2: (n.fixed ? fixedColor : nodeColor1)));
                int w = fm.stringWidth(n.lbl) + 10;
                int h = fm.getHeight() + 4;

                g.fillRect(x - w/2, y - h/2, w, h);

                if(n.lbl.indexOf("{-}") != -1) {
                     g.setColor(Color.red);
                }
                else if(n.lbl.indexOf("{+}") != -1) {
                     g.setColor(Color.blue);
                }
                else g.setColor(Color.black);
                if(showNode) {
                g.drawRect(x - w/2, y - h/2, w - 1, h - 1);
                }
                g.drawString(n.lbl, x - (w-10)/2, (y - (h-4)/2)
+ fm.getAscent());
           }
//Method to draw edges & edge labels and call paintNode()
     public synchronized void update(Graphics g) {
           Dimension d = size();
           if((offscreen == null) || (d.width !=
offscreensize.width) || (d.height != offscreensize.height)) {
                offscreen = createImage(d.width,d.height);
                offscreensize = d;
                offgraphics = offscreen.getGraphics();
                offgraphics.setFont(getFont());
           }

           offgraphics.setColor(getBackground());
           offgraphics.fillRect(0,0,d.width,d.height);
           for(int i = 0;i < nedges;i++) {
                Edge e = edges[i];
                int x1 = (int)nodes[e.from].x;
                int y1 = (int)nodes[e.from].y;
                int x2 = (int)nodes[e.to].x;
                int y2 = (int)nodes[e.to].y;
                String reactSide = e.side;
                String stoichValue = e.stoich;

                offgraphics.setColor(arcColor1);
```

```
                if(reactionSide) {

     offgraphics.setColor((reactSide.equals("L")) ? arcColor2 :
(reactSide.equals("R") ? arcColor3 : arcColor1));
                }
                if(reactionColour) {

     offgraphics.setColor((reactSide.equals("L")) ? arcColor2 :
(reactSide.equals("R") ? arcColor3 : arcColor1));
                }

//Need to keep this line here to use the above colour changes
          offgraphics.drawLine(x1,y1,x2,y2);//draws the edges

                if(thick) {

     offgraphics.drawLine(x1+1,y1+1,x2+1,y2+1);//draws a second
line to thicken edge
                }
                if(reactionSide) {
                     String lbl = reactSide;
                     offgraphics.drawString(lbl,x1 + (x2-x1)/2,
y1 + (y2-y1)/2);
                }
                else if(stoichiometry) {
                     String lbl = stoichValue;
                     offgraphics.drawString(lbl,x1 + (x2-x1)/2,
y1 + (y2-y1)/2);
                }
          }
          FontMetrics fm = offgraphics.getFontMetrics();
          for(int i = 0;i < nnodes;i++) {
               paintNode(offgraphics,nodes[i],fm);
          }

          g.drawImage(offscreen,0,0,null);
          }
//x and y are the coordinates at which the mouse is pressed
public synchronized boolean mouseDown(Event evt, int x, int y)
{
double bestdist = Double.MAX_VALUE;
for(int i = 0;i <nnodes;i++) {
     Node n = nodes[i];
     double dist = (n.x - x)*(n.x - x) + (n.y - y)*(n.y - y);
                     if(dist < bestdist) {
                          pick = n;
                          bestdist = dist;
                     }
                }
                pickfixed = pick.fixed;
                pick.fixed = true;
```

```
                pick.x = x;
                pick.y = y;
                repaint();
                return true;
        }
//Method to allow continuous smooth drag of nodes
public synchronized boolean mouseDrag(Event evt, int x, int y)
{
                pick.x = x;//drag in x direction
                pick.y = y;//drag in y direction
                repaint();
                return true;
        }
//x and y are the coordinates at which the mouse is released
public synchronized boolean mouseUp(Event evt, int x, int y) {
                pick.x = x;
                pick.y = y;
                pick.fixed = pickfixed;
                pick = null;
                repaint();
                return true;
        }
    }
```

Appendix A6

Shown below is the source code for a Java applet to display and interact with metabolic pathways on-line. This applet uses the HTML format generated by the 'ReactionString Database' application. It consists of a number of inter-dependent classes, which contain methods (procedures). Single line comments are delimited with // and multiple line comments are delimited with /* and */. Both are printed in dark red font. The dot operator (.) calls a method for a particular object, which is a combination of data (variables) and actions (methods). A semi-colon terminates an instruction and two curly brackets enclose a *block* of code.

Source code for the WP applet

```
/* The following is a Java applet to display and interact with
metabolic pathways on-line. This applet uses the HTML format
generated by the 'ReactionString Database' application.

Author: James Joseph Connolly
Submitted for Ph.D. Thesis 31/10/08*/

//Import statements
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.net.*;

/* Class to set up a GUI for the applet */
public class TrivialApplet extends Applet implements
ActionListener {

     //Initialise an array with the name(s) of the pathway(s)
     String[] pathwayName = {"GET_EVENT(S)"};

     //Initialise an array with the names of the Web sites

     String[] webSiteName =
{"KEGG","LIGAND","ENZYME","ExplorEnz","Enz Nomen","UM-
BBD","WIT","PathDB","BioCyc","BRENDA","EMP","Agora"};
```

```
     //Initialise an array with the URLS of the Web sites
     String[] webURL =
{"http://www.genome.ad.jp/kegg","http://www.genome.ad.jp/ligand
",

"http://www.expasy.ch/enzyme","http://www.enzyme-database.org",

"http://www.chem.qmul.ac.uk/iubmb/enzyme/","http://umbbd.ahc.um
n.edu",

"http://wit.mcs.anl.gov/WIT2","http://www.ncgr.org/pathdb",

"http://BioCyc.org/","http://www.brenda.uni-koeln.de",

"http://www.empproject.com","http://www.the-agora.org/"};

//Array of buttons that will denote the different reactions
     Button[] button = new Button[pathwayName.length];

//Array of buttons that will denote the different Web sites
     Button[] button2 = new Button[webSiteName.length];

     GraphPanel panel;

     //Override method init() to display the group of buttons
     public void init() {

          for(int index=0;index != pathwayName.length;index++)
{
               button[index] = new Button(pathwayName[index]);
               button[index].setLocation(74,25*index);
               button[index].setSize(85,20);
               button[index].addActionListener(this);
               add(button[index]);
          }
          for(int index=0;index != webSiteName.length;index++)
{
               button2[index] = new Button(webSiteName[index]);
               button2[index].setLocation(1,25*index);
               button2[index].setSize(70,20);
               button2[index].addActionListener(this);
               add(button2[index]);
          }
          setLayout(new BorderLayout());

          panel = new GraphPanel(this);
          add("Center",panel);

          Panel p = new Panel();
          add("South",p);
          p.add(new Button("Shake NOPs"));//button
```

```
          //Check boxes
          p.add(new Checkbox("Side indicator (L or R)"));
          p.add(new Checkbox("Side indicator (Colour)"));
          p.add(new Checkbox("Reaction stoichiometry"));
          p.add(new Checkbox("Thick edge"));
          p.add(new Checkbox("Maths"));
          p.add(new Checkbox("Show node borders"));
          p.add(new Checkbox("Allow NOPs to move"));

          Color background = new Color(180,220,220);
          setBackground(background);//colour appears before
clicking display area - see "paintNode" below

          Panel l = new Panel();
          add("North",l);
          Graphics g = getGraphics();
          l.add(new Label("Select a LAYOUT and/or click
GET_EVENT(S) button."));
     }
//Use an implementation of the actionPerformed method taken
from the ActionListener abstract class
     public void actionPerformed(ActionEvent event) {

          //Find the name of the button that was pressed
          Object source = event.getActionCommand();

          for(int index=0;index != webSiteName.length;index++)
{

//Inspect each web site name in the array of web site names
          if(source.equals(webSiteName[index])) {
               //Display the relevant page from the Web site
                    try {
                         URL site = new URL(webURL[index]);
                         getAppletContext().showDocument(site);
                         }
                         catch (MalformedURLException m) {
                         System.exit(1);
                         }
                         return;
                    }
               }

          for(int index=0;index != pathwayName.length;index++)
{

//Inspect each pathway name in the array of pathway names
          if(source.equals(pathwayName[index])) {
               panel.j=0;
               panel.message2();
               String edges = getParameter(pathwayName[index]);
```

```
//Prevents extra unwanted nodes and edges if button reclicked
          button[index].removeActionListener(this);

//Create a string tokenizer and process tokens
for(StringTokenizer t = new StringTokenizer(edges, ";");
t.hasMoreTokens();) {
          String str = t.nextToken();
          int i = str.indexOf('$');
          if(i > 0) {
               int len = 50;
               int j = str.indexOf('|');
               if(j > 0) {
                    String side = str.substring(j+1,j+2);
                    int k = str.indexOf('/');
                    if(k > 0) {
                         String stoich = str.substring(k+1);
                              str = str.substring(0,j);
                              panel.addEdge(str.substring(0,i),
str.substring(i+1),len,side,stoich);

     panel.fixedNode(str.substring(0,i));
                              }
                    }
               }
          }
     }
}
          Dimension d = size();
          String center = getParameter("center");
          if(center != null) {
               Node n = panel.nodes[panel.findNode(center)];
               n.x = d.width/2;
               n.y = d.height/2;
               n.fixed = true;
          }
     }
     public void start() {
          panel.start();
     }
     public void stop() {
          panel.stop();
     }
//Method to change boolean value when check box or button is
clicked
     public boolean action(Event evt, Object arg) {
          if(arg instanceof Boolean) {

     if(((Checkbox)evt.target).getLabel().equals("Side
indicator (L or R)")) {
```

```
                    panel.reactionSide =
((Boolean)arg).booleanValue();
               }
               else
if(((Checkbox)evt.target).getLabel().equals("Side indicator
(Colour)")) {
                    panel.reactionColour =
((Boolean)arg).booleanValue();
               }
               else
if(((Checkbox)evt.target).getLabel().equals("Reaction
stoichiometry")) {
                    panel.stoichiometry =
((Boolean)arg).booleanValue();
               }
               else
if(((Checkbox)evt.target).getLabel().equals("Thick edge")) {
                    panel.thick =
((Boolean)arg).booleanValue();
               }
               else
if(((Checkbox)evt.target).getLabel().equals("Maths")) {
                    panel.math = ((Boolean)arg).booleanValue();
               }
               else
if(((Checkbox)evt.target).getLabel().equals("Show node
borders")) {
                    panel.showNode =
((Boolean)arg).booleanValue();
               }
               else {
                    panel.relaxed =
((Boolean)arg).booleanValue();
               }
               return true;
          }

          if("Shake NOPs".equals(arg)) {
               Dimension d = size();
               for(int i = 0;i < panel.nnodes;i++) {
                    Node n = panel.nodes[i];
                    if(!n.fixed) {
                         n.x += 80*Math.random() - 40;
                         n.y += 80*Math.random() - 40;
                    }
               }
               return true;
          }
          return false;
     }
}
```

```
/* Drawing classes: Node, Edge and GraphPanel */
class Node {

     double x;
     double y;

     double dx;
     double dy;

     boolean fixed;

     String lbl;
}

class Edge {

     int from;
     int to;
     String side;
     String stoich;
     double len;
}

class GraphPanel extends Panel implements Runnable {

     TrivialApplet graph;

     int nnodes;
     Node nodes[] = new Node[100000];

     int nedges;
     Edge edges[] = new Edge[200000];

     Thread relaxer;
     boolean stoichiometry;
     boolean thick;
     boolean direction;
     boolean relaxed;
     boolean reactionSide;
     boolean reactionColour;
     boolean linDiag;
     boolean linVert;
     boolean sCircular;
     boolean mCircular;
     boolean lCircular;
     boolean math;
     boolean showNode;

     private Choice choice;
     String operation;
```

```
int xRStart = 300;
int yRStart = 50;
int xlevel = 0;
int yLevel = 0;

int j;

GraphPanel(TrivialApplet graph) {
     this.graph = graph;

     add(new Label("LAYOUTS"));
     //Create a dropdown list of choices
     choice = new Choice();
     choice.addItem("Random");
     choice.addItem("Linear Diagonal");
     choice.addItem("Linear Vertical");
     choice.addItem("Circular (small)");
     choice.addItem("Circular (medium)");
     choice.addItem("Circular (large)");
     add(choice);

     //Handle events on this choice
     choice.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {

               Graphics g = getGraphics();

               Choice c = (Choice)e.getItemSelectable();
          operation = c.getSelectedItem();

               if(operation.equals("Linear Diagonal")) {
                    linDiag = true;
               }
               if(operation.equals("Linear Vertical")) {
                    linVert = true;
               }
               if(operation.equals("Circular (small)")) {
                    sCircular = true;
               }
               if(operation.equals("Circular (medium)")) {
                    mCircular = true;
               }
               if(operation.equals("Circular (large)")) {
                    lCircular = true;
               }
          }
     });
}
//Method to fix position of an operator node
public void fixedNode(String enzyme) {
     if(enzyme != null) {
```

```
                Node n = nodes[findNode(enzyme)];
                n.fixed = true;
            }
        }
//Method to search for a node or call addNode() to create it
        int findNode(String lbl) {
            for(int i = 0;i < nnodes;i++) {
                if(nodes[i].lbl.equals(lbl)) {
                    return i;
                }
            }
        return addNode(lbl);
        }
        //Method to create a new node
        int addNode(String lbl) {
            Node n = new Node();

            /* node layout algorithms */

            int i;
            int d;

            //separation factors
            int s1 = 2;
            int s2 = 3;

            //linear diagonal layout
            if(linDiag) {
                for(i=-1;i<j;i++) {
                    n.x = 300 + 8*s1*i;
                    n.y = 50 + 8*s1*i;
                    }
                }

            //linear vertical layout
            else if(linVert) {
            int k=1;
          n.x = 158;
          for(i=-1;i<j;i++) {
                n.y = 40 + 16*k;
                k++;
                if (n.y >= 610) {
                    n.x = n.x + 200;
                    k=1;
              }
            }
            }

            //circular layout (small)
            else if(sCircular) {
                d =120;
```

```
                for(i=-1;i<j;i++) {
                     n.x = 450 + (int)(d*Math.cos(i));
                     n.y = 350 + (int)(d*Math.sin(i));
                     }
                }
           //circular layout (medium)
           else if(mCircular) {
                d =200;;
                for(i=-1;i<j;i++) {
                     n.x = 450 + (int)(d*Math.cos(i));
                     n.y = 350 + (int)(d*Math.sin(i));
                     }
                }
           //circular layout (large)
           else if(lCircular) {
                d =260;;
                for(i=-1;i<j;i++) {
                     n.x = 450 + (int)(d*Math.cos(i));
                     n.y = 350 + (int)(d*Math.sin(i));
                     }
                }

           //random layout
           else {
           n.x = xRStart + 350*Math.random();
           n.y = yRStart + 350*Math.random();
           }

           n.lbl = lbl;

                j++;
           nodes[nnodes] = n;
           return nnodes++;
     }
//Method to create a new edge with labels and send node names
to findNode()
     void addEdge(String from, String to, int len,String
side,String stoich) {
           Edge e = new Edge();
           e.from = findNode(from);
           e.to = findNode(to);
           e.len = len;
           e.side = side;
           e.stoich = stoich;
           edges[nedges++] = e;
     }
     //Method that contains the logic for the thread
     public void run() {
           while(true) {
                if(relaxed) {
                relax();
```

```
            }
                try {
                    Thread.sleep(100);
                }
                catch(InterruptedException e) {
                    break;
                }
            }
        }
//Algorithm for the motion of (effectively non-operator) nodes

    synchronized void relax() {
        for(int i = 0;i < nedges;i++) {
            Edge e = edges[i];
            double vx = nodes[e.to].x - nodes[e.from].x;
            double vy = nodes[e.to].y - nodes[e.from].y;
            double len = Math.sqrt(vx * vx + vy * vy);
            double f = (edges[i].len - len)/(len*3);
            double dx = f*vx;
            double dy = f*vy;

            nodes[e.to].dx += dx;
            nodes[e.to].dy += dy;
            nodes[e.from].dx += -dx;
            nodes[e.from].dy += -dy;
        }

        for(int i = 0;i < nnodes;i++) {
            Node n1 = nodes[i];
            double dx = 0;
            double dy = 0;

            for(int j = 0;j < nnodes;j++) {
                if(i == j) {
                    continue;
                }

                Node n2 = nodes[j];
                double vx = n1.x - n2.x;
                double vy = n1.y - n2.y;
                double len = vx*vx + vy*vy;
                if(len == 0) {
                        dx += Math.random();
                        dy += Math.random();
                        }
                        else if(len < 100*100) {
                            dx += vx/len;
                            dy += vy/len;
                        }
                }
```

```
                double dlen = dx*dx + dy*dy;
                if(dlen > 0) {
                        dlen = Math.sqrt(dlen)/2;
                        n1.dx += dx/dlen;
                        n1.dy += dy/dlen;
                }
        }

        Dimension d = size();
        for(int i = 0;i < nnodes;i++) {
                Node n = nodes[i];
                if(!n.fixed) {
                        n.x += Math.max(-5,Math.min(5,n.dx));
                        n.y += Math.max(-5,Math.min(5,n.dy));
                        if(n.x < 0) {
                                n.x = 0;
                        }else if(n.x > d.width) {
                                n.x = d.width;
                        }
                        if(n.y < 0) {
                                n.y = 0;
                        }else if(n.y > d.height) {
                                n.y = d.height;
                        }
                }
                n.dx /= 2;
                n.dy /= 2;
        }
        repaint();
}

Node pick;
boolean pickfixed;
Image offscreen;
Dimension offscreensize;
Graphics offgraphics;

//Node colours
final Color fixedColor = Color.yellow;
final Color nodeColor1 = Color.magenta;
final Color nodeColor2 = Color.white;
final Color selectColor = Color.pink;
//Edge colours
final Color arcColor1 = Color.black;
final Color arcColor2 = Color.magenta;
final Color arcColor3 = Color.blue;
final Color arcColor4 = Color.green;
final Color edgeColor = Color.black;
final Color reactionSideColorL = Color.blue;
final Color reactionSideColorR = Color.green;
```

```
     //Method to display message
     public void message1() {
          Graphics g = getGraphics();
          g.setFont(new Font("Serif", Font.BOLD, 30));
          g.setColor(Color.red);
          g.drawString("Select a LAYOUT and/or click
GET_EVENT(S) button!",200, 300);
     }
     //Method to display message
     public void message2() {
          Graphics g = getGraphics();
          g.setFont(new Font("Serif", Font.BOLD, 30));
          g.setColor(Color.black);
          g.drawString("NOW CLICK WINDOW OR CLICK 'Allow NOPs
to move' CHECK BOX!",110, 350);
     }
     //Method to draw nodes
     public void paintNode(Graphics g, Node n, FontMetrics fm)
{
               int x = (int)n.x;
               int y = (int)n.y;
               int membrane = 0;
               String mito = n.lbl.substring(0, 2);
               if(mito.equals("MM")) {
                    membrane = 1;
               }
/* ?: is a ternary (3-way) if-then-else operator; if expression
1 is true, then expression 2 is evaluated, otherwise,
expression 3 is evaluated*/
g.setColor((n == pick) ? selectColor : (membrane == 1 ?
nodeColor1: (n.fixed ? fixedColor : nodeColor2)));
               int w = fm.stringWidth(n.lbl) + 10;
               int h = fm.getHeight() + 4;
               Color background = new Color(199,200,150);
               setBackground(background);//colour only appears
after clicking display area
               g.fillRect(x - w/2, y - h/2, w, h);

               g.setColor(Color.black);

               double ratio = (double)nedges/nnodes;

//Function to adjust a double value to two decimal places with
rounding
               if(math) {
               final double roundingFraction = 0.005;
               double newRatio;
               ratio = ratio + roundingFraction;
               newRatio = (int)(100.0*ratio);
               newRatio = (double)newRatio/100;
```

```
            g.drawString("Number of edges: "+nedges, 5, 600 );
            g.drawString("Number of nodes: "+nnodes, 5, 612 );
            g.drawString("Edge/node ratio : "+newRatio, 5, 624 );
                }

                if(n.lbl.indexOf("{-}") != -1) {
                    g.setColor(Color.red);
                }
                else if(n.lbl.indexOf("{+}") != -1) {
                    g.setColor(Color.blue);
                }
                else g.setColor(Color.black);
                if(showNode) {
                g.drawRect(x - w/2, y - h/2, w - 1, h - 1);
                }
                g.drawString(n.lbl, x - (w-10)/2, (y - (h-4)/2)
+ fm.getAscent());
                if(linDiag) {
                g.setColor(Color.red);
                g.drawString("TO CHANGE LAYOUT, SELECT Reload
Applet FROM Applet MENU OR Refresh FROM BROWSER.", 670, 25 );
                }
                else if(linVert) {
                g.setColor(Color.red);
                g.drawString("TO CHANGE LAYOUT, SELECT Reload
Applet FROM Applet MENU OR Refresh FROM BROWSER.", 670, 25 );
                }
                else if(sCircular) {
                g.setColor(Color.red);
                g.drawString("TO CHANGE LAYOUT, SELECT Reload
Applet FROM Applet MENU OR Refresh FROM BROWSER.", 670, 25 );
                }
                else if(mCircular) {
                g.setColor(Color.red);
                g.drawString("TO CHANGE LAYOUT, SELECT Reload
Applet FROM Applet MENU OR Refresh FROM BROWSER.", 670, 25 );
                }
                else if(lCircular) {
                g.setColor(Color.red);
                g.drawString("TO CHANGE LAYOUT, SELECT Reload
Applet FROM Applet MENU OR Refresh FROM BROWSER.", 670, 25 );
                }
                else {
                g.setColor(Color.red);
                g.drawString("TO CHANGE LAYOUT, SELECT Reload
Applet FROM Applet MENU OR Refresh FROM BROWSER.", 670, 25 );
                }
            }

//Method to draw edges & edge labels and call paintNode()
    public synchronized void update(Graphics g) {
```

```
          Dimension d = size();
          if((offscreen == null) || (d.width !=
offscreensize.width) || (d.height != offscreensize.height)) {
               offscreen = createImage(d.width,d.height);
               offscreensize = d;
               offgraphics = offscreen.getGraphics();
               offgraphics.setFont(getFont());
          }
          offgraphics.setColor(getBackground());
          offgraphics.fillRect(0,0,d.width,d.height);
          for(int i = 0;i < nedges;i++) {
               Edge e = edges[i];
               int x1 = (int)nodes[e.from].x;
               int y1 = (int)nodes[e.from].y;
               int x2 = (int)nodes[e.to].x;
               int y2 = (int)nodes[e.to].y;
               String reactSide = e.side;
               String stoichValue = e.stoich;
               offgraphics.setColor(arcColor1);
               if(reactionSide) {
               offgraphics.setColor((reactSide.equals("L")) ?
arcColor3 : (reactSide.equals("R") ? arcColor4 : arcColor1));
               }
               if(reactionColour) {
               offgraphics.setColor((reactSide.equals("L")) ?
arcColor3 : (reactSide.equals("R") ? arcColor4 : arcColor1));
               }
//Need to keep this line here to use the above colour changes
          offgraphics.drawLine(x1,y1,x2,y2);//draws the edges
     if(thick) {
          offgraphics.drawLine(x1+1,y1+1,x2+1,y2+1);//draws a
second line to thicken edge
               }
               if(reactionSide) {
                    String lbl = reactSide;
                    if(reactSide.equals("L")) {
                    offgraphics.setColor(reactionSideColorL);
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
                         }
                    else
offgraphics.setColor(reactionSideColorR);
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
               }

               else if(stoichiometry) {
                    String lbl = stoichValue;
                    offgraphics.setColor(edgeColor);
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
               }
          }
          FontMetrics fm = offgraphics.getFontMetrics();
```

```
            for(int i = 0;i < nnodes;i++) {
                paintNode(offgraphics,nodes[i],fm);
            }
            g.drawImage(offscreen,0,0,null);
            }
//x and y are the coordinates at which the mouse is pressed
public synchronized boolean mouseDown(Event evt, int x, int y)
{
                if(nnodes == 0) message1();
                double bestdist = Double.MAX_VALUE;
                for(int i = 0;i <nnodes;i++) {
                    Node n = nodes[i];
      double dist = (n.x - x)*(n.x - x) + (n.y - y)*(n.y - y);
                    if(dist < bestdist) {
                        pick = n;
                        bestdist = dist;
                    }
                }
                pickfixed = pick.fixed;
                pick.fixed = true;
                pick.x = x;
                pick.y = y;
                repaint();
                return true;
            }
//Method to allow continuous smooth drag of nodes
public synchronized boolean mouseDrag(Event evt, int x, int y)
{
                pick.x = x;//drag in x direction
                pick.y = y;//drag in y direction
                repaint();
                return true;
            }
//x and y are the coordinates at which the mouse is released
public synchronized boolean mouseUp(Event evt, int x, int y) {
                pick.x = x;
                pick.y = y;
                pick.fixed = pickfixed;
                pick = null;
                repaint();
                return true;
            }
            //Start thread
            public void start() {
                relaxer = new Thread(this);
                relaxer.start();
            }
            public void stop() {
                relaxer.stop();
            }
    }
```

Appendix A7

Shown below is the source code for an interactive Java applet to visualise and explore the glycolytic pathway over the Internet. It is independent of the application. Some data are input to the applet via the applet argument in the html source, while other data are contained within the code. The applet consists of a number of inter-dependent classes, which contain methods (procedures). Single line comments are delimited with // and multiple line comments are delimited with /* and */. Both are printed in dark red. The dot operator (.) calls a method for a particular object, which is a combination of data (variables) and actions (methods). A semi-colon terminates an instruction and two curly brackets enclose a *block* of code.

Source code for the Glycolysis pathway applet

```
/* This program displays the glycolysis pathway. It uses
predefined coordinates for the placement of the enzyme nodes in
a snake layout.

Author: James Joseph Connolly
Submitted for Ph.D. Thesis 31/10/08
*/

//Import statements
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.net.*;

/* Class to set up a GUI for the applet */
public class TrivialApplet extends Applet {

     GraphPanel panel;
     Information documentation;

//Override method init() to display group of check boxes
     public void init() {

//Border layout divides  window into different areas
          setLayout(new BorderLayout());
```

```
        panel = new GraphPanel(this);
        add("Center",panel);

        documentation = new Information();
        add("North",documentation);

        //Create a new panel p
        Panel p = new Panel();

        //Add components to panel p
        p.add(new Label("GRAPH PROPERTIES: "));
        p.add(new Checkbox("Thick edge"));
        p.add(new Checkbox("Node border"));
        p.add(new Checkbox("NOP fill"));
        p.add(new Checkbox("Red font"));
        p.add(new Checkbox("Blue font"));
        p.add(new Checkbox("Maths"));
        p.add(new Checkbox("NOP movement"));
        p.add(new Checkbox("Pick"));

        //Add panel p to applet window
        add("South",p);

        String edges = getParameter("glycolysis");

     //Create a string tokenizer and process tokens
    for(StringTokenizer t = new StringTokenizer(edges, ";");
t.hasMoreTokens();) {
              String str = t.nextToken();
              int i = str.indexOf('$');
              if(i > 0) {
                   int len = 50;
                   int j = str.indexOf('/');
                   if(j > 0) {
                        int carbon =
Integer.valueOf(str.substring(j+1,j+2)).intValue();
                       int k = str.indexOf('|');
                       if(k > 0) {
                        int phosphorous =
Integer.valueOf(str.substring(k+1,k+2)).intValue();
                        int r = str.indexOf('&');
                        if(r > 0) {
                             String role =
str.substring(r+1,r+2);
                             int z = str.indexOf('#');
                             if(z > 0) {
                             String rev =
str.substring(z+1,z+2);
                             int q = str.indexOf('!');
                             if(q > 0) {
```

```
                                        int stoich =
Integer.valueOf(str.substring(q+1)).intValue();
                                            str =
str.substring(0,j);

     panel.addEdge(str.substring(0,i),
str.substring(i+1),len,carbon,phosphorous,role,rev,stoich);
                                            }
                                        }
                                    }
                                }
                            }
                        }

        Dimension d = size();

        //Assign names to fixed (operator) nodes
        String anchor1 = "HK";
        String anchor2 = "PGI";
        String anchor3 = "PFK";
        String anchor4 = "aldolase";
        String anchor5 = "TPI";
        String anchor6 = "G3PDH";
        String anchor7 = "PGK";
        String anchor8 = "PGM";
        String anchor9 = "enolase";
        String anchor10 = "PK";

        //Assign coordinates to fixed (operator) nodes
        if(anchor1 != null) {
            Node n = panel.nodes[panel.findNode(anchor1)];
            n.x = d.width/6;//position of x coordinate of
fixed node 1
            n.y = d.height/5.7;//position of y coordinate of
fixed node 1
            n.fixed1 = true;
        }
        if(anchor2 != null) {
            Node n = panel.nodes[panel.findNode(anchor2)];
            n.x = d.width/3;
            n.y = d.height/5.7;
            n.fixed2 = true;
        }
        if(anchor3 != null) {
            Node n = panel.nodes[panel.findNode(anchor3)];
            n.x = d.width/2;
            n.y = d.height/5.7;
            n.fixed3= true;
        }
        if(anchor4 != null) {
```

```
            Node n = panel.nodes[panel.findNode(anchor4)];
            n.x = d.width/1.5;
            n.y = d.height/5.7;
            n.fixed4 = true;
        }
        if(anchor5 != null) {
            Node n = panel.nodes[panel.findNode(anchor5)];
            n.x = d.width/1.3;
            n.y = d.height/5.7;
            n.fixed5 = true;
        }
        if(anchor6 != null) {
            Node n = panel.nodes[panel.findNode(anchor6)];
            n.x = d.width/1.4;
            n.y = d.height/2.8;
            n.fixed6 = true;
        }
        if(anchor7 != null) {
            Node n = panel.nodes[panel.findNode(anchor7)];
            n.x = d.width/1.4;
            n.y = d.height/1.7;
            n.fixed7 = true;
        }
        if(anchor8 != null) {
            Node n = panel.nodes[panel.findNode(anchor8)];
            n.x = d.width/1.9;
            n.y = d.height/1.7;
            n.fixed8 = true;
        }
        if(anchor9 != null) {
            Node n = panel.nodes[panel.findNode(anchor9)];
            n.x = d.width/3;
            n.y = d.height/1.7;
            n.fixed9 = true;
        }
        if(anchor10 != null) {
            Node n = panel.nodes[panel.findNode(anchor10)];
            n.x = d.width/6;
            n.y = d.height/1.7;
            n.fixed10 = true;
        }
    }

    public void start() {
        panel.start();
        panel.message();//call the 'CLICK HERE!' message
    }
    public void stop() {
        panel.stop();
    }
    //Change boolean value when check box is clicked
```

```
    public boolean action(Event evt, Object arg) {
        if(arg instanceof Boolean) {

    if(((Checkbox)evt.target).getLabel().equals("Thick edge"))
{
                    panel.thick =
((Boolean)arg).booleanValue();
                }
                else
if(((Checkbox)evt.target).getLabel().equals("Node border")) {
                    panel.showNode =
((Boolean)arg).booleanValue();
                }
                else
if(((Checkbox)evt.target).getLabel().equals("NOP fill")) {
                    panel.metFill =
((Boolean)arg).booleanValue();
                }
                else
if(((Checkbox)evt.target).getLabel().equals("Red font")) {
                    panel.redFont =
((Boolean)arg).booleanValue();
                }
                else
if(((Checkbox)evt.target).getLabel().equals("Blue font")) {
                    panel.blueFont =
((Boolean)arg).booleanValue();
                }
                else
if(((Checkbox)evt.target).getLabel().equals("Maths")) {
                    panel.math = ((Boolean)arg).booleanValue();
                }
                else
if(((Checkbox)evt.target).getLabel().equals("NOP movement")) {
                    panel.relaxed =
((Boolean)arg).booleanValue();
                }
                else
if(((Checkbox)evt.target).getLabel().equals("Pick")) {
                    panel.select =
((Boolean)arg).booleanValue();
                }
                return true;
            }
            return false;
    }
}
//Class to add information components
class Information extends Panel {
    //Information at top of applet window
    InfoOption infopt = new InfoOption(this);
```

```
    InfoText infotext = new InfoText();

    Information() {
        setLayout(new BorderLayout());
        add("West", infopt);
        add("East", infotext);
    }
}
//Class to create choices for information
class InfoOption extends Panel {

     Choice inf = new Choice();

    Information option;

    InfoOption(Information selection) {
        option = selection;

        //Create dropdown list of choices for information
        inf.addItem("Begin");
        inf.addItem("Model");
        inf.addItem("Pathway");
        inf.addItem("Regulation");
        inf.addItem("Lab");
        inf.addItem("Input");
        inf.addItem("Stoich");
        inf.addItem("Metab flow");
        inf.addItem("Metab role");
        inf.addItem("Vector");
        inf.addItem("Weight");
        inf.addItem("TD");
        inf.addItem("EC Class");
        inf.addItem("HK");
        inf.addItem("PGI");
        inf.addItem("PFK");
        inf.addItem("aldolase");
        inf.addItem("TPI");
        inf.addItem("G3PDH");
        inf.addItem("PGK");
        inf.addItem("PGM");
        inf.addItem("enolase");
        inf.addItem("PK");
        inf.addItem("glucose");
        inf.addItem("G6P");
        inf.addItem("ATP");
        inf.addItem("ADP");
        inf.addItem("Mg++");
        inf.addItem("F6P");
        inf.addItem("FBP");
        inf.addItem("DHAP");
          inf.addItem("G3P");
```

```
            inf.addItem("Pi");
            inf.addItem("NAD+");
            inf.addItem("NADH");
            inf.addItem("H+");
            inf.addItem("BPG");
            inf.addItem("3PG");
            inf.addItem("2PG");
            inf.addItem("PEP");
            inf.addItem("K+");
            inf.addItem("pyruvate");

        add(inf);
        add(new Label("<-Select [INFORMATION] View->"));
    }
    //Method to get selected item
    public boolean action(Event evt, Object arg) {
        if (evt.target instanceof Choice) {
            String str = new String(inf.getSelectedItem());
            option.infotext.showline(str);
        }
        return true;
    }
}
//Class to display information
class InfoText extends TextArea {
     //Information to be displayed
     final String Begin = new String(" TO BEGIN:\n"+
          " Click in window and/or click 'NOP movement'
checkbox at bottom of window. \n"+
          " To display information as edge or node attributes,
use drop down lists below this text area. \n"+
          " To activate a graph property, click a checkbox at
bottom of window. \n"+
          " To view result(s), click in window or activate 'NOP
movement' checkbox.\n"+
          " To stop motion of nodes, deactivate  'NOP movement'
checkbox.\n"+
          " All nodes may be dragged by the mouse, but only the
yellow ones will retain their new positions in the relaxed
state.\n"+
          " To restart, press the command and R keys; or go to
the menubar, click Applet, and then click Reload Applet.\n"+
          " If using a browser, click the 'Refresh'
icon.\n\n");
     final String Model = new String(" MODEL:\n"+
           " A metabolic pathway is modelled as an object
bipartite graph.  The pathway consists of a sequence of
connected star-graphs.\n"+
          " Each star-graph represents a biochemical
reaction.\n\n");
    final String Pathway = new String(" GLYCOLYSIS PATHWAY\n"+
```

```
          " Glycolysis occurs in the cytosol of all cells.  It
is a 10-step pathway that converts one molecule of glucose to
two molecules \n"+
          " of pyruvate, with the concomitant generation of 2
molecules of ATP from ADP and Pi, and 2 molecules of NADH from
NAD+.\n"+
          " All intermediates are phosphorylated, so they stay
in the cell.\n"+
          " The first 5 reactions constitute an energy
investment phase, in which sugar phosphates are synthesised at
the expense of 2 moles of ATP\n"+
          " (converted to ADP), and  the 6-carbon substrate is
split into two 3-carbon sugar phosphates.\n"+
          " The last 5 reactions represent an energy generation
phase, in which the triose phosphates are converted to energy-
rich compounds.\n"+
          " These transfer 4 moles of phosphate to ADP, leading
to 4 moles of ATP.\n"+
          " Glycolysis has two major functions: \n"+
          " (1) to provide energy for the cell in the form of
ATP.\n"+
          " (2) to generate intermediates as starting points
for other pathways.\n"+
          " The primary role of glycolysis may vary in
different cells and tissues:\n"+
          " RBCs depend entirely on glycolysis for ATP
synthesis. They have no mitochondria and cannot synthesise
ATP\n"+
          " by the oxidative pathways that are available to
most cells.\n"+
          " SKELETAL MUSCLE uses glycolysis to generate ATP for
muscle contraction. Although other oxidative pathways for\n"+
          " generating ATP are present in muscle, glycolysis is
the major pathway under conditions of oxygen deprivation.\n"+
          " ADIPOSE TISSUE specialises in the storage and
mobilisation of triglyceride. One of the major functions of
glycolysis.\n"+
          " in adipocytes is to provide DHAP, the precursor of
the glycerol backbone of triglycerides.\n"+
          " LIVER derives most of its energy from
noncarbohydrate sources. The role of glycolysis in liver
depends on the \n"+
          " nutritional state of the organism. In the well-fed
state , dietary carbohydrate is stored either as glycogen or as
fat.\n"+
          " The pyruvate resulting from glycolysis may be used
to generate precursors for fatty acid synthesis. These energy-
storage\n"+
          " pathways are stimulated by insulin. In the fasting
state, pyruvate and other low moledcular weight precursors are
used \n"+
```

```
          " for the de novo synthesis of glucose. This process
is stimulated by glucagon, and it requires all of the enzymes
in the \n"+
          " glycolytic pathway that catalyse reversible
reactions.\n\n");
    final String Regulation = new String(" REGULATION\n"+
          " The entry of free glucose into the glycolytic
pathway is regulated by the allosteric enzyme HEXOKINASE.  The
enzyme is inhibited\n"+
          " by its product, glucose 6-phosphate.  This mode of
metabolic control (feedback inhibition) regulates the
intracellular concentration\n"+
          " of glucose 6-phosphate. Once glucose is
phosphorylated by hexokinase, it continues through glycolysis
to produce ATP.  If high levels\n"+
          " of ATP are already present, the glycolytic pathway
is closed by the key regulatory enzyme, PHOSPHOFRUCTOKINASE
(PFK), which catalyses\n"+
          " the phosphorylation of fructose 6-phosphate.  PFK
is allosterically inhibited by high levels of ATP.  The
effector ATP binds to the\n"+
          " enzyme at two sites, the active site (as a
substrate) and a regulatory site (as a modulator).  When bound
at the regulatory site,\n"+
          " it lowers the affinity of the enzyme for the
substrate fructose 6-phosphate.  A high level of ATP in the
cell makes glycolysis unnecessary,\n"+
          " so the pathway is closed at PFK. PFK is also
inhibited by citrate, the physiological form of citric acid.
This compound is synthesised by\n"+
          " the citric acid cycle in later aerobic metabolism
of pyruvate.  If cellular levels of citrate are sufficient,
then glycolysis is slowed\n"+
          " to reduce the rate  of production of pyruvate and
citrate.\n"+
          " The compound fructose 2,6-bisphosphate (F2,6BP)
acts to stimulate PFK in liver cells.\n"+
          " Pyruvate kinase (PK) is a tetrameric protein that
requires a monovalent ion, Na+ or K+, for activity.\n"+
          " High levels of ATP inhibit pyruvate kinase, and
hence, slow pyruvate formation.\n\n");
    final String Lab = new String(" Laboratory:\n"+
          " Measurement of blood glucose concentration is used
in the diagnosis of diabetes mellitus, \n"+
          " which is caused by insulin deficiency. Because the
red blood cell (RBC) can metabolise glucose to lactate, \n"+
          " even in a sealed, anaerobic vial, the glucose in
blood will be consumed, with concomitant production of
lactate,\n"+
          " which will lead to acidification of of the blood
sample.\n"+
```

```
        " This may be prevented by adding a small amount of
sodium fluoride, an inhibitor of glycolysis, . \n"+
        " to the sample-collection vial. Fluoride is a strong
competitive inhibitor of enolase, \n"+
        " blocking glycolysis and lactate production in the
RBC. Fluoride forms a complex with phosphate \n"+
        " and Mg++ in the active site of the enzyme, blocking
access of substrate.\n\n");
    final String Input = new String(" INPUT FORMAT from html
file (e.g., hexokinase reaction):\n"+
        " HK$glucose/6|0&s#n!-1\n"+
        " HK$ATP1/0|3&c#n!-1 \n"+
        " HK$ADP1/0|2&c#n!1 \n"+
        " HK$G6P/6|1&p#n!1 \n"+
        " HK$H+1/0|0&p#n!1 \n"+
        " Enzyme$metabolite/C atoms|P atoms&metabolite
role#reversibility!signed stoichiometry.\n"+
        " To see complete list, press the command and T keys
at the same time; or go to the menubar, click Applet, and then
click Show Applet Tag.\n\n");
    final String Stoich = new String(" STOICHIOMETRY:\n"+
        " Stoichiometry indicates the proportions of
molecularities with which the reactants and products enter the
reactions.\n"+
        " For example, -n indicates the consumption of n
molecules, and n the production of n molecules.\n\n");
    final String Dir = new String(" METABOLITE FLOW:\n"+
        " A blue edge represents the direction of a molecule
in towards an enzyme (substrate).\n"+
        " A green edge represents the direction of a molecule
out from an enzyme (product).\n"+
        " These colours are computed from the sign of the
stoichiometric coefficient.\n\n");
    final String Role = new String(" METABOLITE ROLE:\n"+
        " The role of a metabolite as substrate (s), product
(p), currency metabolite (c) or cofactor (f)\n"+
        " in a biochemical reaction represents a relationship
with its enzyme and is, therefore, a property of the
corresponding edge.\n\n");
    final String Vector = new String(" COMPOSITION VECTORS\n"+
      " The molecular formula of a compound indicates the
proportions of atoms combined to make up the molecule.\n"+
      " The molecular formula of a compound on a metabolic
network is represented by a composition vector\n"+
      " whose components give the number of each atom type on
the network which is present in the link molecule:\n"+
      " X = (x1, x2,...,xn). In glycolysis, X = (#C,#H,#O,#P),
where C = carbon, H = hydrogen, O = oxygen and\n"+
      " P = phosphorous. For example, the molecular formula
for glucose is (6,12,6) while the composition vector is\n"+
```

```
       " (6,12,6,0). The number of components of a composition
vector is equal to the number of atom types on the network.\n"+
          " As a molecule moves through glycolysis, the values
of the components of the composition vector change,\n"+
          " but the number of components remains the same.\n"+
          " The vector representation is more convenient when
working with metabolic networks. \n\n");
    final String Weight = new String(" FORMULA WEIGHTS:\n"+
          " These are computed in the program by adding the
atomic weights in the composition vectors. For example,\n"+
          " glucose (6,12,6,0) has a formula weight of (6 x
12.01) + (12 x 1.01) + (6 x 16.00) + (0 x 30.97) = 180.18\n"+
          " The resulting values are corrected to two decimal
places.\n\n");
    final String Tdynam = new String(" THERMODYNAMICS\n"+
          " DG0'= 0: The reactants and products are at the same
energy level.  The reaction under standard conditions is at
equilibrium.\n"+
          " There is no release of or requirement for
energy.\n"+
          " DG0'< 0: The reaction releases energy as it
approaches equilibrium.  The reactants are at a higher energy
level than the products.\n"+
          " Energy is released, which may be sufficient for
useful work.\n"+
          " DG0'> 0: The reactants are at a lower energy level
than products.  The reaction requires an input of energy to
proceed as written.\n"+
          " However, reactions within a pathway cannot be
considered in isolation.\n\n");
    final String Code = new String(" EC CLASSES\n"+
          " 1 Oxidoreductases:  Transfer of electrons usually
in the form of hydride ions or hydrogen atoms.\n"+
          " 2 Transferases:      Transfer of functional groups
from one molecule to another.\n"+
          " 3 Hydrolases:          Cleavage of bonds by
hydrolysis.\n"+
          " 4 Lyases:                 Formation of double bonds
by removal of groups or addition of groups to double bonds.\n"+
          " 5 Isomerases:          Transfer of groups within a
molecule to yield isomeric forms.\n"+
          " 6 Ligases:                Formation of C-C, C-S, C-
O, and C-N bonds by condensation coupled to ATP
cleavage.\n\n");
     final String hexokinase = new String(" HEXOKINASE\n"+
            " An enzyme\n"+
          " Reaction  : glucose + ATP -> ADP + glucose 6-
phosphate + H+\n"+
          " ATP donates PO3(2-) to glucose to give glucose 6-
phosphate, ADP and H+.\n"+
          " Cell space: cytosol\n\n");
```

```
    final String PGI = new String(" PHOSPHOGLUCOSE
ISOMERASE\n"+
          " An enzyme\n"+
          " Reaction  : glucose 6-phosphate -> fructose 6-
phosphate\n"+
          " Cell space: cytosol\n\n");
    final String PFK = new String(" PHOSPHOFRUCTOKINASE\n"+
          " An enzyme\n"+
          " Reaction  : fructose 6-phosphate + ATP -> ADP +
fructose 1,6-bisphosphate + H+\n"+
          " Cell space: cytosol\n\n");
    final String aldolase = new String(" ALDOLASE\n"+
          " An enzyme\n"+
          " Reaction  : fructose 1,6-bisphosphate ->
glyceraldehyde 3-phosphate + dihydroxyacetone phosphate\n"+
          " Cell space: cytosol\n\n");
    final String TPI = new String(" TRIOSE-PHOSPHATE
ISOMERASE\n"+
          " An enzyme\n"+
          " Reaction  : dihydroxyacetone phosphate ->
glyceraldehyde 3-phosphate\n"+
          " Cell space: cytosol\n\n");
    final String G3PDH = new String(" GLYCERALDEHYDE-3-
PHOSPHATE DEHYDROGENASE\n"+
          " An enzyme\n"+
          " Reaction  : glyceraldehyde 3-phosphate + Pi + NAD+
-> 1,3-bisphoglycerate + NADH + H+\n"+
          " Cell space: cytosol\n\n");
    final String PGK = new String(" PHOSPHOGLYCERATE KINASE\n"+
          " An enzyme\n"+
          " Reaction  : 1,3-bisphoglycerate + ADP -> 3-
phosphoglycerate + ATP\n"+
          " ADP accepts PO3(2-) to give 3-phosphoglycerate and
ATP\n"+
          " Cell space: cytosol\n\n");
    final String PGM = new String(" PHOSPHOGLYCERATE MUTASE\n"+
          " An enzyme\n"+
          " Reaction  : 3-phosphoglycerate -> 2-
phosphoglycerate\n"+
          " Cell space: cytosol\n\n");
    final String enolase = new String(" ENOLASE\n"+
          " An enzyme\n"+
          " Reaction  : 2-phosphoglycerate -> H2O +
phosphoenolpyruvate\n"+
          " Cell space: cytosol\n");
    final String PK = new String(" PYRUVATE KINASE\n" +
          " An enzyme\n"+
          " Reaction  : phosphoenolpyruvate + ADP + H+ -> ATP +
enolpyruvate\n"+
          " Cell space: cytosol\n\n");
    final String glucose = new String(" GLUCOSE\n" +
```

```
            " A Metabolite.\n"+
            " Cas Number: 50-99-7.\n\n");
    final String G6P = new String(" GLUCOSE 6-PHOSPHATE\n" +
            " A Metabolite.\n"+
            " Cas Number: 56-73-5.\n\n");
    final String F6P = new String(" FRUCTOSE 6-PHOSPHATE\n" +
            " A Metabolite.\n"+
            " Cas Number: 643-13-0.\n\n");
    final String ATP = new String(" ADENOSINE TRIPHOSPHATE\n" +
            " A ribonucleoside 5'-triphosphate functioning as a
phosphate group donor in the cell energy cycle;\n"+
            " carries chemical energy between metabolic pathways
by serving as a shared intermediate coupling \n"+
            " endergonic and exergonic reactions.\n"+
            " Cas Number: 56-65-5.\n\n");
    final String ADP = new String(" ADENOSINE DIPHOSPHATE\n" +
            " A ribonucleoside 5'-diphosphate serving as
phosphate group acceptor in the cell energy cycle.\n"+
            " Cas Number: 58-64-0.\n\n");
    final String Mg = new String(" MAGNESIUM CATION\n" +
            " Required by hexokinase, phosphoglucose isomerase,
phosphofructokinase, phosphoglycerate kinase,\n"+
            " phosphoglycerate mutase, and pyruvate kinase, for
their activity.\n"+
            " Cas Number: 7439-95-4.\n\n");
    final String ThreePG = new String(" 3-PHOSPHOGLYCERATE\n" +
            " A Metabolite.\n"+
            " Cas Number: not available.\n\n");
    final String TwoPG = new String(" 2-PHOSPHOGLYCERATE\n" +
            " A Metabolite.\n"+
            " Cas Number: not available.\n\n");
    final String PEP = new String(" PHOSPHOENOLPYRUVATE\n" +
            " A Metabolite.\n"+
            " Cas Number: not available.\n\n");
    final String K = new String(" POTASSIUM CATION\n" +
            " The pyruvate kinase reaction requires the
participation of the monovalent K+ cation.\n"+
            " Cas Number: 7440-09-7.\n\n");
    final String pyruvate = new String(" PYRUVATE\n" +
            " A Metabolite.\n"+
            " Cas Number: 127-17-3.\n\n");
    final String G3P = new String(" GLYCERALDEHYDE 3-
PHOSPHATE\n" +
            " A Metabolite.\n"+
            " Cas Number: 142-10-9.\n\n");
    final String Pi = new String(" INORGANIC PHOSPHATE\n" +
            " A Metabolite.\n"+
            " Cas Number: not available.\n\n");
    final String NAD = new String(" NICOTINAMIDE ADENINE
DINUCLEOTIDE\n" +
            " A coenzyme.\n"+
```

```
            " Cas Number: 53-84-9.\n\n");
    final String NADH = new String(" NICOTINAMIDE ADENINE
DINUCLEOTIDE (reduced form)\n" +
            " A reduced coenzyme.\n"+
            " Cas Number: 58-68-4.\n\n");
    final String H = new String(" HYDROGEN ION\n" +
            " A Metabolite.\n"+
            " Cas Number: 1333-74-0.\n\n");
    final String BPG = new String(" 1,3-BISPHOSPHOGLYCERATE\n"
+
            " A Metabolite.\n"+
            " Cas Number: not available.\n\n");
    final String FBP = new String(" FRUCTOSE 1,6-
BISPHOSPHATE\n" +
            " A Metabolite.\n"+
            " Cas Number: 488-69-7.\n\n");
    final String DHAP = new String(" DIHYDROXYACETONE
PHOSPHATE\n" +
            " A Metabolite.\n"+
            " Cas Number: 102783-56-2.\n\n");

final String blah = Begin + Model + Pathway + Regulation +
                    Lab + Input + Stoich + Dir + Role + Vector +
                    Weight + Tdynam + Code + hexokinase + PGI +
PFK + aldolase + TPI + G3PDH + PGK +
                      PGM + enolase + PK + K + glucose + G6P +
                      ATP + ADP + Mg + ThreePG + TwoPG + PEP +
                      pyruvate + G3P + Pi + NAD +
                      NADH + H + BPG + FBP + DHAP + F6P;

     //Method to place information in text area
     InfoText() {
        super(5, 90);//set size of text area
        setText(blah);//place text in text area
    }

    //Method to select information
    public void showline(String str) {
          if (str.equals("Begin"))           setText(Begin);
     else if (str.equals("Model"))           setText(Model);
     else if (str.equals("Pathway"))       setText(Pathway);
     else if (str.equals("Regulation")) setText(Regulation);
     else if (str.equals("Lab"))              setText(Lab);
     else if (str.equals("Input"))          setText(Input);
     else if (str.equals("Stoich"))        setText(Stoich);
     else if (str.equals("Metab flow"))       setText(Dir);
     else if (str.equals("Metab role"))      setText(Role);
     else if (str.equals("Vector"))        setText(Vector);
     else if (str.equals("Weight"))        setText(Weight);
     else if (str.equals("TD"))          setText(Tdynam);
     else if (str.equals("EC Class"))        setText(Code);
```

```
      else if (str.equals("HK"))          setText(hexokinase);
      else if (str.equals("PGI"))               setText(PGI);
      else if (str.equals("PFK"))               setText(PFK);
      else if (str.equals("aldolase"))      setText(aldolase);
      else if (str.equals("TPI"))               setText(TPI);
      else if (str.equals("G3PDH"))           setText(G3PDH);
      else if (str.equals("PGK"))               setText(PGK);
      else if (str.equals("PGM"))               setText(PGM);
      else if (str.equals("enolase"))        setText(enolase);
      else if (str.equals("PK"))                 setText(PK);
      else if (str.equals("glucose"))        setText(glucose);
      else if (str.equals("G6P"))               setText(G6P);
      else if (str.equals("F6P"))               setText(F6P);
      else if (str.equals("ATP"))               setText(ATP);
      else if (str.equals("ADP"))               setText(ADP);
      else if (str.equals("Mg++"))               setText(Mg);
      else if (str.equals("3PG"))            setText(ThreePG);
      else if (str.equals("2PG"))              setText(TwoPG);
      else if (str.equals("PEP"))               setText(PEP);
      else if (str.equals("K+"))                 setText(K);
      else if (str.equals("pyruvate"))      setText(pyruvate);
      else if (str.equals("G3P"))               setText(G3P);
      else if (str.equals("Pi"))                 setText(Pi);
      else if (str.equals("NAD+"))              setText(NAD);
      else if (str.equals("NADH"))            setText(NADH);
      else if (str.equals("H+"))                   setText(H);
      else if (str.equals("BPG"))               setText(BPG);
      else if (str.equals("FBP"))               setText(FBP);
      else if (str.equals("DHAP"))            setText(DHAP);
      else setText(str);
    }
}
/* Drawing classes: Node, Edge and GraphPanel */
class Node {

      double x;
      double y;

      double dx;
      double dy;

      boolean fixed1;
      boolean fixed2;
      boolean fixed3;
      boolean fixed4;
      boolean fixed5;
      boolean fixed6;
      boolean fixed7;
      boolean fixed8;
      boolean fixed9;
      boolean fixed10;
```

```
        boolean fixed11;
        boolean fixed12;
        boolean fixed13;
        boolean fixed14;
        boolean fixed15;

        String lbl;
}

class Edge {

        int from;
        int to;
        int values;
        int stoich;
        int carbon;
        int phosphorous;
        String role;
        String rev;
        double len;
}

class GraphPanel extends Panel implements Runnable {

        TrivialApplet graph;

        int nnodes;
        int i;

        Node nodes[] = new Node[100];

        int nedges;

        Edge edges[] = new Edge[200];

        boolean stoichiometry;
        boolean direction;
        boolean metabRole;
        boolean vectors;
        boolean thermo;
        boolean ion;
        boolean molWeight;
        boolean EC;
        boolean gene;
        boolean inhibByG6P;
        boolean inhibByATP;
        boolean inhibByCitrate;
        boolean inhibByAcetylCoA;
        boolean inhibByAla;
        boolean inhibByFluoride;
        boolean activByAMP;
```

```
    boolean activByF2BP;
    boolean isomerase;
    boolean transferase;
    boolean lyase;
    boolean metFill;
    boolean irreverseFill;
    boolean r;
    boolean redFont;
    boolean blueFont;
    boolean random;
    boolean relaxed;
    boolean thick;
    boolean showNode;
    boolean math;
    boolean select;
    boolean carbonColour;
    boolean phosphorousColour;
    boolean carbonFlow;
    boolean phosphorousFlow;

    Thread relaxer;
    int stoich;
    int carbon;
    int phosphorous;
    private Choice choice1, choice2, choice3, choice4,
choice5;
    String operation;
    String fullName;

    GraphPanel(TrivialApplet graph) {
        this.graph = graph;

        add(new Label("Edge label"));
        //Create dropdown list of choices for edge label
        choice1 = new Choice();
        choice1.addItem("None");
        choice1.addItem("Stoichiometry");
        choice1.addItem("Metabolite role");
        choice1.addItem("C value");
        choice1.addItem("P value");
        add(choice1);

        add(new Label("NOP label"));
//Create dropdown list of choices for non-operator label
        choice2 = new Choice();
        choice2.addItem("None");
        choice2.addItem("Vector");
        choice2.addItem("Formula Wt");
        add(choice2);

        add(new Label("OP attribute"));
```

```
//Create dropdown list of choices for operator attribute
          choice3 = new Choice();
          choice3.addItem("None");
          choice3.addItem("Ion");
          choice3.addItem("TD");
          choice3.addItem("EC number");
          choice3.addItem("Gene name");
          choice3.addItem("Isomerases");
          choice3.addItem("Transferases");
          choice3.addItem("Lyases");
          choice3.addItem("Irreversible");

          add(choice3);

          add(new Label("Edge function"));
         //Create dropdown list of choices for edge function
          choice4 = new Choice();
          choice4.addItem("None");
          choice4.addItem("Metabolite flow");
          choice4.addItem("C flow");
          choice4.addItem("P flow");
          choice4.addItem("Reversibility");
          add(choice4);

          add(new Label("Effector"));
          //Create dropdown list of choices for effector
          choice5 = new Choice();
          choice5.addItem("None");
          choice5.addItem("G6P");
          choice5.addItem("ATP");
          choice5.addItem("Citrate");
          choice5.addItem("AcetylCoA");
          choice5.addItem("Alanine");
          choice5.addItem("Fluoride");
          choice5.addItem("AMP");
          choice5.addItem("F2,6BP");
          add(choice5);

          //Handle events on edge label choices
          choice1.addItemListener(new ItemListener() {
               public void itemStateChanged(ItemEvent e) {

                    Graphics g = getGraphics();

                    Choice c1 = (Choice)e.getItemSelectable();
               operation = c1.getSelectedItem();

               if(operation.equals("None")) {
                         stoichiometry = false;
                         metabRole = false;
                         carbonColour = false;
```

```
                phosphorousColour = false;
            }
        else if(operation.equals("Stoichiometry")) {
                stoichiometry = true;
                metabRole = false;
                carbonColour = false;
                phosphorousColour = false;
            }
        else if(operation.equals("Metabolite role")) {
                stoichiometry = false;
                metabRole = true;
                carbonColour = false;
                phosphorousColour = false;
            }
            else if(operation.equals("C value")) {
                stoichiometry = false;
                metabRole = false;
                carbonColour = true;
                phosphorousColour = false;
            }
            else if(operation.equals("P value")) {
                stoichiometry = false;
                metabRole = false;
                carbonColour = false;
                phosphorousColour = true;
            }
        }
});
//Handle events on NOP label choices
choice2.addItemListener(new ItemListener() {
        public void itemStateChanged(ItemEvent e) {

            Graphics g = getGraphics();

            Choice c2 = (Choice)e.getItemSelectable();
        operation = c2.getSelectedItem();

        if(operation.equals("None")) {
                vectors = false;
                molWeight = false;
            }
            else if(operation.equals("Vector")) {
                vectors = true;
                molWeight = false;
            }
            else if(operation.equals("Formula Wt")) {
                vectors = false;
                molWeight = true;
            }
        }
});
```

```
//Handle events on OP attribute choices
choice3.addItemListener(new ItemListener() {
     public void itemStateChanged(ItemEvent e) {

          Graphics g = getGraphics();

          Choice c3 = (Choice)e.getItemSelectable();
     operation = c3.getSelectedItem();

     if(operation.equals("None")) {
          ion = false;
               thermo = false;
               EC = false;
               gene = false;
               isomerase = false;
               transferase = false;
               lyase = false;
               irreverseFill = false;
          }
          else if(operation.equals("Ion")) {
               ion = true;
               thermo = false;
               EC = false;
               gene = false;
               isomerase = false;
               transferase = false;
               lyase = false;
               irreverseFill = false;
          }
          else if(operation.equals("TD")) {
               ion = false;
               thermo = true;
               EC = false;
               gene = false;
               isomerase = false;
               transferase = false;
               lyase = false;
               irreverseFill = false;
          }
          else if(operation.equals("EC number")) {
               ion = false;
               thermo = false;
               EC = true;
               gene = false;
               isomerase = false;
               transferase = false;
               lyase = false;
               irreverseFill = false;
          }
          else if(operation.equals("Gene name")) {
               ion = false;
```

```
                thermo = false;
                EC = false;
                gene = true;
                isomerase = false;
                transferase = false;
                lyase = false;
                irreverseFill = false;
            }
            else if(operation.equals("Isomerases")) {
                ion = false;
                thermo = false;
                EC = false;
                gene = false;
                isomerase = true;
                transferase = false;
                lyase = false;
                irreverseFill = false;
            }
            else if(operation.equals("Transferases")) {
                ion = false;
                thermo = false;
                EC = false;
                gene = false;
                isomerase = false;
                transferase = true;
                lyase = false;
                irreverseFill = false;
            }
            else if(operation.equals("Lyases")) {
                ion = false;
                thermo = false;
                EC = false;
                gene = false;
                isomerase = false;
                transferase = false;
                lyase = true;
                irreverseFill = false;
            }
            else if(operation.equals("Irreversible")) {
                ion = false;
                thermo = false;
                EC = false;
                gene = false;
                isomerase = false;
                transferase = false;
                lyase = false;
                irreverseFill = true;
            }
        }
    });
```

```
        //Handle events on edge function choices
        choice4.addItemListener(new ItemListener() {
             public void itemStateChanged(ItemEvent e) {

                  Graphics g = getGraphics();

                  Choice c4 = (Choice)e.getItemSelectable();
             operation = c4.getSelectedItem();

             if(operation.equals("None")) {
                       direction = false;
                       carbonFlow = false;
                       phosphorousFlow = false;
                       r = false;
                  }
             else if(operation.equals("Metabolite flow")) {
                       direction = true;
                       carbonFlow = false;
                       phosphorousFlow = false;
                       r = false;
                  }
                  else if(operation.equals("C flow")) {
                       direction = false;
                       carbonFlow = true;
                       phosphorousFlow = false;
                       r = false;
                  }
                  else if(operation.equals("P flow")) {
                       direction = false;
                       carbonFlow = false;
                       phosphorousFlow = true;
                       r = false;
                  }
                  else if(operation.equals("Reversibility"))
{
                       direction = false;
                       carbonFlow = false;
                       phosphorousFlow = false;
                       r = true;
                  }
             }
        });
        //Handle events on effector choices
        choice5.addItemListener(new ItemListener() {
             public void itemStateChanged(ItemEvent e) {

                  Graphics g = getGraphics();

                  Choice c5 = (Choice)e.getItemSelectable();
             operation = c5.getSelectedItem();
```

```
if(operation.equals("None")) {
          inhibByG6P = false;
          inhibByATP = false;
          inhibByCitrate = false;
          inhibByAcetylCoA = false;
          inhibByAla = false;
          inhibByFluoride = false;
          activByAMP = false;
          activByF2BP = false;
     }
     else if(operation.equals("G6P")) {
          inhibByG6P = true;
          inhibByATP = false;
          inhibByCitrate = false;
          inhibByAcetylCoA = false;
          inhibByAla = false;
          inhibByFluoride = false;
          activByAMP = false;
          activByF2BP = false;
     }
     else if(operation.equals("ATP")) {
          inhibByG6P = false;
          inhibByATP = true;
          inhibByCitrate = false;
          inhibByAcetylCoA = false;
          inhibByAla = false;
          inhibByFluoride = false;
          activByAMP = false;
          activByF2BP = false;
     }
     else if(operation.equals("Citrate")) {
          inhibByG6P = false;
          inhibByATP = false;
          inhibByCitrate = true;
          inhibByAcetylCoA = false;
          inhibByAla = false;
          inhibByFluoride = false;
          activByAMP = false;
          activByF2BP = false;
     }
     else if(operation.equals("AcetylCoA")) {
          inhibByG6P = false;
          inhibByATP = false;
          inhibByCitrate = false;
          inhibByAcetylCoA = true;
          inhibByAla = false;
          inhibByFluoride = false;
          activByAMP = false;
          activByF2BP = false;
     }
     else if(operation.equals("Alanine")) {
```

```
                    inhibByG6P = false;
                    inhibByATP = false;
                    inhibByCitrate = false;
                    inhibByAcetylCoA = false;
                    inhibByAla = true;
                    inhibByFluoride = false;
                    activByAMP = false;
                    activByF2BP = false;
               }
               else if(operation.equals("Fluoride")) {
                    inhibByG6P = false;
                    inhibByATP = false;
                    inhibByCitrate = false;
                    inhibByAcetylCoA = false;
                    inhibByAla = false;
                    inhibByFluoride = true;
                    activByAMP = false;
                    activByF2BP = false;
               }
               else if(operation.equals("AMP")) {
                    inhibByG6P = false;
                    inhibByATP = false;
                    inhibByCitrate = false;
                    inhibByAcetylCoA = false;
                    inhibByAla = false;
                    inhibByFluoride = false;
                    activByAMP = true;
                    activByF2BP = false;
               }
               else if(operation.equals("F2,6BP")) {
                    inhibByG6P = false;
                    inhibByATP = false;
                    inhibByCitrate = false;
                    inhibByAcetylCoA = false;
                    inhibByAla = false;
                    inhibByFluoride = false;
                    activByAMP = false;
                    activByF2BP = true;
               }
          }
     });
}

//Search for a node or call addNode() to create it
int findNode(String lbl) {
     for(int i = 0;i < nnodes;i++) {
          if(nodes[i].lbl.equals(lbl)) {
               return i;
          }
     }
```

```
        return addNode(lbl);
        }

        //Create a new node
        int addNode(String lbl) {
             Node n = new Node();
             n.x = 300 + 380*Math.random();
             n.y = 100 + 380*Math.random();
             n.lbl = lbl;

             //Code for repetition of side compounds
             if((n.lbl).length()>3) {
                  if((n.lbl).substring(0,3).equals("ATP")) {
                       n.lbl = "ATP";
                       }
                  }
             if((n.lbl).length()>3) {
                  if((n.lbl).substring(0,3).equals("ADP")) {
                       n.lbl = "ADP";
                       }
                  }
             if((n.lbl).length()>2) {
                  if((n.lbl).substring(0,2).equals("H+")) {
                       n.lbl = "H+";
                       }
                  }

             nodes[nnodes] = n;
             return nnodes++;

        }
//Create a new edge with labels and send node names to
findNode()
        void addEdge(String from,String to,int len,int carbon,int
phosphorous,String role,String rev,int stoich) {
             Edge e = new Edge();
             e.from = findNode(from);
             e.to = findNode(to);
             e.len = len;
             e.stoich = stoich;
             e.rev = rev;
             e.role = role;
             e.carbon = carbon;
             e.phosphorous = phosphorous;
             edges[nedges++] = e;
        }
        //Logic for the thread
        public void run() {
             while(true) {
                  if(relaxed) {
                  relax();
```

```
            }
                try {
                    Thread.sleep(100);
                }
                catch(InterruptedException e) {
                    break;
                }
            }
        }
    //Algorithm for motion of nodes
    synchronized void relax() {
        for(int i = 0;i < nedges;i++) {
            Edge e = edges[i];
            double vx = nodes[e.to].x - nodes[e.from].x;
            double vy = nodes[e.to].y - nodes[e.from].y;
            double len = Math.sqrt(vx * vx + vy * vy);
            double f = (edges[i].len - len)/(len*3);
            double dx = f*vx;
            double dy = f*vy;

            nodes[e.to].dx += dx;
            nodes[e.to].dy += dy;
            nodes[e.from].dx += -dx;
            nodes[e.from].dy += -dy;
        }

        for(int i = 0;i < nnodes;i++) {
            Node n1 = nodes[i];
            double dx = 0;
            double dy = 0;

            for(int j = 0;j < nnodes;j++) {
                if(i == j) {
                    continue;
                }

                Node n2 = nodes[j];
                double vx = n1.x - n2.x;
                double vy = n1.y - n2.y;
                double len = vx*vx + vy*vy;
                if(len == 0) {
                        dx += Math.random();
                        dy += Math.random();
                        }
                        else if(len < 100*100) {
                            dx += vx/len;
                            dy += vy/len;
                        }
                }

                double dlen = dx*dx + dy*dy;
```

```
                    if(dlen > 0) {
                        dlen = Math.sqrt(dlen)/2;
                        n1.dx += dx/dlen;
                        n1.dy += dy/dlen;
                    }
                }

                Dimension d = size();
                for(int i = 0;i < nnodes;i++) {
                    Node n = nodes[i];

    if(!n.fixed1&&!n.fixed2&&!n.fixed3&&!n.fixed4&&!n.fixed5&&
!n.fixed6&&!n.fixed7&&!n.fixed8&&!n.fixed9&&!n.fixed10&&!n.fixe
d11) {
                        n.x += Math.max(-5,Math.min(5,n.dx));
                        n.y += Math.max(-5,Math.min(5,n.dy));
                        if(n.x < 0) {
                            n.x = 0;
                        }else if(n.x > d.width) {
                            n.x = d.width;
                        }
                        if(n.y < 0) {
                            n.y = 0;
                        }else if(n.y > d.height) {
                            n.y = d.height;
                        }
                    }
                    n.dx /= 2;
                    n.dy /= 2;
                }
                repaint();
            }

            Node pick;
            boolean pickfixed;
            Image offscreen;
            Dimension offscreensize;
            Graphics offgraphics;

            //Node colours
            final Color fixedColor1 = Color.yellow;
            final Color fixedColor2 = new Color(255,90,150);
            final Color fixedColor3 = Color.cyan;
            final Color fixedColor4 = Color.magenta;
            final Color inhibColor1 = Color.red;
            final Color activColor1 = Color.green;
            final Color selectColor = Color.pink;
            final Color nodeColor1 = Color.lightGray;
            final Color nodeColor2 = new Color(250,220,100);
            final Color nodeColor3 = Color.cyan;
            final Color nodeColor4 = Color.white;
```

```
            final Color nodeColor5 = Color.yellow;
            final Color nodeColor6 = Color.lightGray;
            final Color nodeColor7 = new Color(255,150,50);

            //Edge colours
            final Color edgeColor = Color.black;
            final Color stoichiometryColor1 = Color.blue;
            final Color stoichiometryColor2 = Color.green;
            final Color carbonValueColor = Color.magenta;
            final Color phosphorousValueColor = Color.red;
            final Color arcColor1 = Color.black;
            final Color arcColor2 = Color.blue;
            final Color arcColor3 = Color.green;
            final Color arcColor4 = Color.red;
            final Color arcColor5 = Color.magenta;

            //Display message
            public void message() {
            Graphics g = getGraphics();
      g.setFont(new Font("Serif", Font.BOLD, 40));//set font
            g.setColor(Color.black);
            g.drawString("Click window and/or 'NOP movement'
check box!",130, 300);

      }
      //Method to draw nodes
      public void paintNode(Graphics g, Node n, FontMetrics fm)
{
            int x = (int)n.x;
            int y = (int)n.y;

/* ?: is a ternary (3-way) if-then-else operator; if expression
1 is true, then expression 2 is evaluated, otherwise,
expression 3 is evaluated */
            g.setColor((n == pick) ? selectColor:(n.fixed1 ?
fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:
```

```
(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));

          if(metFill) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor6))))))))))));
          }
          if(irreverseFill) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? nodeColor7:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? nodeColor7:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:
```

```
(n.fixed10 ? nodeColor7:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }



          if(isomerase) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor3:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor3:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor3:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }

          else if(transferase) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor2:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor2:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor2:

(n.fixed8 ? fixedColor1:
```

```
(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor2:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }

          else if(lyase) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor4:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor4:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }
          if(inhibByATP) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? inhibColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:
```

```
(n.fixed10 ? inhibColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }

          else if(inhibByG6P) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? inhibColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }
          else if(inhibByCitrate) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? inhibColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor1:
```

```
(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }
          else if(inhibByAcetylCoA) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? inhibColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }
          else if(inhibByAla) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? inhibColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }
```

```
          else if(inhibByFluoride) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? fixedColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? inhibColor1:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }
          else if(activByAMP) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:

(n.fixed2 ? fixedColor1:

(n.fixed3 ? activColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }
          else if(activByF2BP) {
               g.setColor((n == pick) ? selectColor :(n.fixed1
? fixedColor1:
```

```
(n.fixed2 ? fixedColor1:

(n.fixed3 ? activColor1:

(n.fixed4 ? fixedColor1:

(n.fixed5 ? fixedColor1:

(n.fixed6 ? fixedColor1:

(n.fixed7 ? fixedColor1:

(n.fixed8 ? fixedColor1:

(n.fixed9 ? fixedColor1:

(n.fixed10 ? fixedColor1:

(n.fixed11 ? fixedColor1:nodeColor4))))))))))));
          }


          int w = fm.stringWidth(n.lbl) + 10;
          int h = fm.getHeight() + 4;
          int xCoord = x - w/2;
          int yCoord = y - h/2;

          g.fillRect(x - w/2, y - h/2, w, h);

          g.setColor(Color.black);

          if(redFont) {
               g.setColor(Color.red);
          }

          else if(blueFont) {
               g.setColor(Color.blue);
          }
          if(showNode) {
            g.drawRect(x - w/2, y - h/2, w - 1, h - 1);
          }
          g.drawString(n.lbl, x - (w-10)/2, (y - (h-4)/2) +
fm.getAscent());

        int vector[] = new int[4];
          double formulaWeight = 0.0;
          String ionName = "";
          String ECNumber = "";
          String geneName = "";
          double tDynamics = 0.0;
```

```
          String effector1 = "";
          String effector2 = "";
          String effector3 = "";
          String effector4 = "";
          String effector5 = "";
          String effector6 = "";
          String comment = "";

          if(n.lbl.equals("ATP")) {
               comment = "ATP inhibits PFK and PK";
          }
          else if(n.lbl.equals("G6P")) {
               comment = "G6P inhibits HK";
          }

          //Assign NOP data & evaluate formula weights
          if(n.lbl.equals("glucose")) {
             vector[0] = 6;
             vector[1] = 12;
             vector[2] = 6;
             vector[3] = 0;
            fullName = "glucose";
             formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("G6P")) {
              vector[0] = 6;
               vector[1] = 11;
               vector[2] = 9;
               vector[3] = 1;
               fullName = "glucose 6-phosphate";
               formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
          }
          else if(n.lbl.equals("F6P")) {
               vector[0] = 6;
               vector[1] = 11;
               vector[2] = 9;
               vector[3] = 1;
               fullName = "fructose 6-phosphate";
               formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
          }
          else if(n.lbl.equals("FBP")) {
               vector[0] = 6;
               vector[1] = 10;
               vector[2] = 12;
               vector[3] = 2;
```

```
                fullName = "fructose 1,6-bisphosphate";
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("DHAP")) {
                vector[0] = 3;
                vector[1] = 5;
                vector[2] = 6;
                vector[3] = 1;
                fullName = "dihydroxyacetone phosphate";
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("G3P")) {
                vector[0] = 3;
                vector[1] = 5;
                vector[2] = 6;
                vector[3] = 1;
                fullName = "glyceraldehyde 3-phosphate";
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("BPG")) {
                vector[0] = 3;
                vector[1] = 4;
                vector[2] = 10;
                vector[3] = 2;
                fullName = "1,3-bisphosphoglycerate";
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("3PG")) {
                vector[0] = 3;
                vector[1] = 4;
                vector[2] = 7;
                vector[3] = 1;
                fullName = "3-phosphoglycerate";
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("2PG")) {
                vector[0] = 3;
                vector[1] = 4;
                vector[2] = 7;
                vector[3] = 1;
                fullName = "2-phosphoglycerate";
```

```
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("PEP")) {
                vector[0] = 3;
                vector[1] = 2;
                vector[2] = 6;
                vector[3] = 1;
                fullName = "phosphoenolpyruvate";
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }
           else if(n.lbl.equals("pyruvate")) {
                vector[0] = 3;
                vector[1] = 3;
                vector[2] = 3;
                vector[3] = 0;
                fullName = "pyruvate";
                formulaWeight =
twoDecimalPlaces(vector[0]*12.01+vector[1]*1.01+vector[2]*16.00
+vector[3]*30.97);
           }

           else if(n.lbl.equals("ATP")) {
                fullName = "adenosine triphosphate";
           }
           else if(n.lbl.equals("ADP")) {
                fullName = "adenosine diphosphate";
           }
           else if(n.lbl.equals("H+")) {
                fullName = "hydrogen ion";
           }
           else if(n.lbl.equals("Pi")) {
                fullName = "inorganic phosphate";
           }
           else if(n.lbl.equals("H2O")) {
                fullName = "water";
           }
           else if(n.lbl.equals("NAD+")) {
                fullName = "nicotinamide adenine dinucleotide
(oxidised)";
           }
           else if(n.lbl.equals("NADH")) {
                fullName = "nicotinamide adenine dinucleotide
(reduced)";
           }

           //Assignment of operator data
           if(n.lbl.equals("HK")) {
```

```
        ECNumber = "2.7.1.1";
        geneName = "HK1";
        tDynamics = -16.7;
        ionName = "Mg++";
        fullName = "hexokinase";
        effector1 = "{-} G6P";
        effector2 = "";
        effector3 = "";
        effector4 = "";
        effector5 = "";
    }
    else if(n.lbl.equals("PGI")) {
        ECNumber = "5.3.1.9";
        geneName = "GPI";
        tDynamics = 1.7;
        ionName = "Mg++";
        fullName = "phosphoglucose isomerase";
        effector1 = "";
        effector2 = "";
        effector3 = "";
        effector4 = "";
        effector5 = "";
    }
    else if(n.lbl.equals("PFK")) {
        ECNumber = "2.7.1.11";
        geneName = "PFKL";
        tDynamics = -14.2;
        ionName = "Mg++";
        fullName = "phosphofructokinase";
        effector1 = "{-} ATP";
        effector2 = "{-} Citrate";
        effector3 = "";
        effector4 = "{+} AMP";
        effector5 = "{+} Fructose 2,6-bisphosphate";
    }
    else if(n.lbl.equals("aldolase")) {
        ECNumber = "4.1.2.13";
        geneName = "ALDOA";
        tDynamics = 23.8;
        ionName = "";
        fullName = "aldolase";
        effector1 = "";
        effector2 = "";
        effector3 = "";
        effector4 = "";
        effector5 = "";
    }
    else if(n.lbl.equals("TPI")) {
        ECNumber = "5.3.1.1";
        geneName = "TPI";
        tDynamics = 7.5;
```

```
            ionName = "";
            fullName = "triose-phosphate isomerase";
            effector1 = "";
            effector2 = "";
            effector3 = "";
            effector4 = "";
            effector5 = "";
      }
      else if(n.lbl.equals("G3PDH")) {
            ECNumber = "1.2.1.12";
            geneName = "GAPD";
            tDynamics = 6.3;
            ionName = "";
fullName = "glyceraldehyde-3-phosphate dehydrogenase";
            effector1 = "";
            effector2 = "";
            effector3 = "";
            effector4 = "";
            effector5 = "";
      }
      else if(n.lbl.equals("PGK")) {
            ECNumber = "2.7.2.3";
            geneName = "PGK1";
            tDynamics = -18.5;
            ionName = "Mg++";
            fullName = "phosphoglycerate kinase";
            effector1 = "";
            effector2 = "";
            effector3 = "";
            effector4 = "";
            effector5 = "";
      }
      else if(n.lbl.equals("PGM")) {
            ECNumber = "5.4.2.1";
            geneName = "PGAM1";
            tDynamics = 4.4;
            ionName = "Mg++";
            fullName = "phosphoglycerate mutase";
            effector1 = "";
            effector2 = "";
            effector3 = "";
            effector4 = "";
            effector5 = "";
      }
      else if(n.lbl.equals("enolase")) {
            ECNumber = "4.2.1.11";
            geneName = "ENO";
            tDynamics = 7.5;
            ionName = "";
            fullName = "enolase";
            effector1 = "{-} fluoride (See Lab)";
```

```
                effector2 = "";
                effector3 = "";
                effector4 = "";
                effector5 = "";
            }
            else if(n.lbl.equals("PK")) {
                ECNumber = "2.7.1.40";
                geneName = "PKL,PKM";
                tDynamics = -31.4;
                ionName = "Mg++,K+";
                fullName = "pyruvate kinase";
                effector1 = "{-} ATP";
                effector2 = "{-} acetylCoA";
                effector3 = "{-} alanine";
                effector4 = "";
                effector5 = "";
            }

            if(n == pick) {
                g.setColor(Color.black);
                g.drawString("Node label   :  "+n.lbl, 260, 226);
            g.drawString("Compound   :  "+fullName, 260, 238);
                for(i=0;i<nnodes;i++) {
                    g.setColor(Color.black);
                if(n.equals(nodes[i])) g.drawString("Node name :
"+i, 260, 250);
                     if(n.equals(nodes[1])) {
                          g.setColor(Color.blue);
                     g.drawString("Pathway starts here.", 260, 262);
                     }
                     if(n.equals(nodes[nnodes-1])) {
                          g.setColor(Color.green);
                g.drawString("Pathway ends here.", 260, 262);
                     }
                }
            g.setColor(Color.red);
            g.drawString(comment, 260, 262);
            g.drawString(effector1, 260, 274 );
            g.drawString(effector2, 260, 286 );
            g.drawString(effector3, 260, 298 );
            g.setColor(Color.green);
            g.drawString(effector4, 260, 310 );
            g.drawString(effector5, 260, 322 );
            g.drawString(effector6, 260, 334 );
            }

            double ratio = (double)nedges/nnodes;
            double newRatio = twoDecimalPlaces(ratio);
            double ratio2 = (double)nnodes/nedges;
            double newRatio2 = twoDecimalPlaces(ratio2);
```

```
          if(math) {
               g.setColor(Color.black);
          g.drawString("Number of nodes: "+nnodes, 550, 250);
          g.drawString("Number of edges: "+nedges, 550, 262);
     g.drawString("Edge/node ratio : "+newRatio, 550, 274);
          }
          if(stoichiometry) {
               g.setColor(Color.blue);
               g.drawString("Pathway stoichiometry", 885, 173);
          }
          else if(carbonColour) {
               g.setColor(Color.magenta);
     g.drawString("Carbon value:  number of atoms", 490, 502);
          }
          else if(phosphorousColour) {
               g.setColor(Color.red);
g.drawString("Phosphorous value:  number of atoms", 490, 526);
          }

          if(vectors) {
               g.setColor(Color.blue);
               if(vector[0] != 0 || vector[1] != 0 || vector[2]
!= 0 || vector[3] != 0) {

     g.drawString("("+vector[0]+","+vector[1]+","+vector[2]+","
+vector[3]+")",x - (w-20), (y - (h+7)) + fm.getAscent());
                    g.drawString("Composition vectors
(C,H,O,P)", 885, 111);
                    g.drawString("ATP donates PO3(2-)", 300,
170);
                    g.drawString("ADP accepts PO3(2-)", 490,
454);
                    g.drawString("Pi = HOPO3(2-)", 885, 214);
                    }
                    else if(vector[0] == 0 && vector[1] == 0 &&
vector[2] == 0 && vector[3] == 0) {
     g.drawString("",x - (w-20), (y - (h+7)) + fm.getAscent());
                    }
          }

          else if(molWeight) {
                g.setColor(Color.red);
                if(formulaWeight != 0) {
g.drawString(Double.toString(formulaWeight),x - (w-20), (y -
(h+7)) + fm.getAscent());
                }
else g.drawString("",x - (w-20), (y - (h+7)) + fm.getAscent());
          }
          if(ion) {
               g.setColor(Color.blue);
               if(!ionName.equals("")) {
```

```
g.drawString(ionName,x - (w-20), (y - (h+7)) + fm.getAscent());
                }
          else g.drawString("",x - (w-20), (y - (h+7)) +
fm.getAscent());
          }
          else if(EC) {
                g.setColor(Color.magenta);
                if(!ECNumber.equals("")) {
                  g.drawString(ECNumber,x - (w-20), (y - (h+7))
+ fm.getAscent());
                }
          else g.drawString("",x - (w-20), (y - (h+7)) +
fm.getAscent());
          }

          else if(thermo) {
                g.setColor(Color.magenta);
                if(tDynamics != 0) {
          g.drawString(Double.toString(tDynamics),x - (w-20),
(y - (h+7)) + fm.getAscent());
     g.drawString("Thermodynamics:  DG0' (kJ/mol)", 885, 312);
                }
else g.drawString("",x - (w-20), (y - (h+7)) + fm.getAscent());
          }
          else if(gene) {
               g.setColor(Color.magenta);
               if(!geneName.equals("")) {
                  g.drawString(geneName,x - (w-20), (y - (h+7))
+ fm.getAscent());
               }
else g.drawString("",x - (w-20), (y - (h+7)) + fm.getAscent());
          }
          else if(isomerase) {
               g.setColor(Color.blue);
               g.drawString("Isomerases", 885, 123);
          }

          else if(transferase) {
               g.setColor(new Color(255,90,150));
               g.drawString("Transferases", 885, 410);
          }

          else if(lyase) {
               g.setColor(Color.magenta);
               g.drawString("Lyases", 885, 135);
          }
          else if(irreverseFill) {
               g.setColor(Color.black);
               g.drawString("Essentially irreversible enzymes:
dark orange nodes", 200, 514);
          }
```

```
          if(direction) {
               g.setColor(Color.blue);
g.drawString("Substrate consumption:  blue edges", 490, 466);
               g.setColor(Color.green);
     g.drawString("Product formation:  green edges", 490, 478);
          }

          else if(carbonFlow) {
               g.setColor(Color.magenta);
     g.drawString("Carbon flow:  magenta edges", 490, 490);
          }

          else if(phosphorousFlow) {
               g.setColor(Color.red);
     g.drawString("Phosphorous flow:  red edges", 490, 514);
          }

          else if(r) {
               g.setColor(Color.black);
               g.drawString("Flow in either direction: black
edges", 200, 526);
               g.drawString("Flow in one direction: blue to
green", 200, 538);
          }

          if(inhibByG6P) {
               g.setColor(Color.red);
               g.drawString("Enzyme(s) inhibited by G6P: red
node(s)", 490, 538);
          }
          else if(inhibByATP) {
               g.setColor(Color.red);
               g.drawString("Enzyme(s) inhibited by ATP: red
node(s)", 490, 538);
          }
          else if(inhibByCitrate) {
               g.setColor(Color.red);
               g.drawString("Enzyme(s) inhibited by Citrate:
red node(s)", 490, 538);
          }
          else if(inhibByAcetylCoA) {
               g.setColor(Color.red);
               g.drawString("Enzyme(s) inhibited by acetylCoA:
red node(s)", 490, 538);
          }
          else if(inhibByAla) {
               g.setColor(Color.red);
               g.drawString("Enzyme(s) inhibited by alanine:
red node(s)", 490, 538);
          }
```

```
          else if(inhibByFluoride) {
               g.setColor(Color.red);
               g.drawString("Enzyme(s) inhibited by fluoride:
red node(s)", 490, 538);
          }
          else if(activByAMP) {
               g.setColor(Color.green);
               g.drawString("Enzyme(s) activated by AMP: green
node(s)", 490, 538);
          }
          else if(activByF2BP) {
               g.setColor(Color.green);
               g.drawString("Enzyme(s) activated by Fructose
2,6-bisphosphate: green node(s)", 490, 538);
          }
     }

//Adjust a double value to two decimal places with rounding
     private double twoDecimalPlaces(double number) {
          final double roundingFraction = 0.005;
          double newNumber;

          number = number+roundingFraction;
          newNumber = (int)(100.0*number);
          return (double)newNumber/100;
     }
//Method to draw edges & edge labels and call paintNode()
     public synchronized void update(Graphics g) {
          Dimension d = size();
          if((offscreen == null) || (d.width !=
offscreensize.width) || (d.height != offscreensize.height)) {
               offscreen = createImage(d.width,d.height);
               offscreensize = d;
               offgraphics = offscreen.getGraphics();
               offgraphics.setFont(getFont());
          }

          offgraphics.setColor(getBackground());
          offgraphics.fillRect(0,0,d.width,d.height);
          for(int i = 0;i < nedges;i++) {
               Edge e = edges[i];
               int x1 = (int)nodes[e.from].x;
               int y1 = (int)nodes[e.from].y;
               int x2 = (int)nodes[e.to].x;
               int y2 = (int)nodes[e.to].y;
              int stoichValues = e.stoich;
              String roleType = e.role;
              int carbonValues = e.carbon;
              int phosphorousValues = e.phosphorous;
              String reversibility = e.rev;
```

```
                offgraphics.setColor(arcColor1);

                if(direction) {
                offgraphics.setColor((stoichValues < 0) ?
arcColor2 : (stoichValues > 0 ? arcColor3 : arcColor1));
                }
                if(carbonFlow) {
                offgraphics.setColor((carbonValues == 0) ?
arcColor1 : arcColor5);
                }
                if(phosphorousFlow) {
                offgraphics.setColor((phosphorousValues == 0) ?
arcColor1 : arcColor4);
                }
                if(thick) {
                offgraphics.drawLine(x1+1,y1+1,x2+1,y2+1);//draw
second line to thicken edge
                }

//Keep this line here to use the above colour changes
                offgraphics.drawLine(x1,y1,x2,y2);//draw edges

                if(stoichiometry) {
                     String lbl = String.valueOf(stoichValues);
                     if(stoichValues < 0) {

     offgraphics.setColor(stoichiometryColor1);
                          }
                     else if(stoichValues > 0) {

     offgraphics.setColor(stoichiometryColor2);
                          }
                     else offgraphics.setColor(edgeColor);
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
                }
                else if(metabRole) {
                     String lbl = String.valueOf(roleType);
                     if(stoichValues < 0) {

     offgraphics.setColor(stoichiometryColor1);
                          }
                     else if(stoichValues > 0) {

     offgraphics.setColor(stoichiometryColor2);
                          }
                     else offgraphics.setColor(edgeColor);
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
                }
                else if(carbonColour) {
                     String lbl = String.valueOf(carbonValues);
                     offgraphics.setColor(carbonValueColor);
```

```
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
                }
                else if(phosphorousColour) {
                     String lbl =
String.valueOf(phosphorousValues);

     offgraphics.setColor(phosphorousValueColor);
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
                }
                if(r) {
                     String lbl = reversibility;
                     if(reversibility.equals("y")) {

     offgraphics.setColor(arcColor1);//black
                     }
                     else if(reversibility.equals("n")) {
                          if(stoichValues < 0) {

     offgraphics.setColor(arcColor2);//blue
                          }
                          else if(stoichValues > 0) {

     offgraphics.setColor(arcColor3);//green
                          }
                     }
                     else
offgraphics.setColor(edgeColor);//black
                          offgraphics.drawLine(x1,y1,x2,y2);
                }
           }

           FontMetrics fm = offgraphics.getFontMetrics();
           for(int i = 0;i < nnodes;i++) {
                paintNode(offgraphics,nodes[i],fm);
           }

           g.drawImage(offscreen,0,0,null);
           }
//x and y are the coordinates at which the mouse is pressed
           public synchronized boolean mouseDown(Event evt, int
x, int y) {
                double bestdist = Double.MAX_VALUE;
                for(int i = 0;i <nnodes;i++) {
                     Node n = nodes[i];
                     double dist = (n.x - x)*(n.x - x) + (n.y -
y)*(n.y - y);
                     if(dist < bestdist) {
                          pick = n;
                          bestdist = dist;
                     }
                }
```

```
                pickfixed = pick.fixed1;
                pick.fixed1 = true;
                repaint();
                return true;
            }
//Method to allow continuous smooth drag of nodes
            public synchronized boolean mouseDrag(Event evt, int
x, int y) {
                pick.x = x;//drag in x direction
                pick.y = y;//drag in y direction
                repaint();
                return true;
            }
//x and y are the coordinates at which the mouse is released
            public synchronized boolean mouseUp(Event evt, int x,
int y) {
                pick.fixed1 = pickfixed;
                if(select) {
                pick = null;
                }
                repaint();
                return true;
            }
            //Start thread
            public void start() {
                relaxer = new Thread(this);
                relaxer.start();
            }

            public void stop() {
                relaxer.stop();
            }
        }
```

Appendix A8

Shown below is the source code for an interactive Java applet to select different reactions and pathways for display in a browser. It is independent of the application. Most of the data are input to the applet via the applet argument in the html source. The applet consists of a number of inter-dependent classes, which contain methods (procedures). Single line comments are delimited with // and multiple line comments are delimited with /* and */. Both are printed in dark red font. The dot operator (.) calls a method for a particular object, which is a combination of data (variables) and actions (methods). A semi-colon terminates an instruction and two curly brackets enclose a *block* of code.

Source code for the Multi-pathway applet

```
/* This multi-pathway applet allows the option of selecting
different reactions and pathways and displaying them singly or
in combination. It also allows the selection of reactions by
cell space. Furthermore, a user-defined option for creating
reactions makes it extendable

Author: James Joseph Connolly
Submitted for Ph.D. Thesis 31/10/08

*/

//Import statements
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

/* Class to set up a GUI with buttons */
public class TrivialApplet extends Applet implements
ActionListener {

//Initialise an array with the names of the pathways
String[] pathwayName =
{"Glycolysis","Galactose_Metabolism","Pentose_Phosphate","Urea_
cycle","Regulatory_cascade","User_defined","PKI_to_PKA","DPPPK_
to_PPPK","GPPb_to_GPPa","Glycogen_to_G1P","G6P_to_Glucose"};
```

```
//Array of buttons that will denote the different pathways
     Button[] button = new Button[pathwayName.length];

     GraphPanel panel;

     private Choice choice2;
     String operation;
     boolean all;
     boolean cyto;
     boolean mito;

     //Override method init to display the group of buttons
     public void init() {

     for(int index=0;index != pathwayName.length;index++) {
               button[index] = new Button(pathwayName[index]);
               button[index].setLocation(5,25*index);
               button[index].setSize(120,20);
               button[index].addActionListener(this);
               add(button[index]);
          }
//Border layout divides applet window into different areas
          setLayout(new BorderLayout());

          panel = new GraphPanel(this);
          add("Center",panel);

          //Create a new panel p
          Panel p = new Panel();

          //Add components to panel p
          p.add(new Label("GRAPH PROPERTIES: "));
          p.add(new Button("Shake"));
          p.add(new Checkbox("Thick edge"));
          p.add(new Checkbox("Node border"));
          p.add(new Checkbox("NOP fill"));
          p.add(new Checkbox("Red font"));
          p.add(new Checkbox("Blue font"));
          p.add(new Checkbox("Maths"));
          p.add(new Checkbox("NOP movement"));

          //Add panel p to applet window
          add("South",p);

          Panel l = new Panel();
          add("North",l);
          l.add(new Label("Select LAYOUT and/or COMPARTMENT
and/or PATHWAY.  "));
          l.add(new Label("Click in WINDOW and/or click 'NOP
movement' CHECKBOX."));
```

```
          panel.add(new Label("Compartment"));
//Create a dropdown list of choices for compartment
          choice2 = new Choice();
          choice2.addItem("All");
          choice2.addItem("Cytosol");
          choice2.addItem("Mitochondrion");
          panel.add(choice2);

          //Handle events on compartment choices
          choice2.addItemListener(new ItemListener() {
               public void itemStateChanged(ItemEvent e) {

                    Choice c2 = (Choice)e.getItemSelectable();
               operation = c2.getSelectedItem();
                     if(operation.equals("Cytosol")) {
                         cyto = true;
                         mito = false;
                    }
                    else if(operation.equals("Mitochondrion"))
{
                         cyto = false;
                         mito = true;
                    }
               }
          });
     }
     /* Use an implementation of the actionPerformed method
   taken from the ActionListener abstract class */
     public void actionPerformed(ActionEvent event) {

          Graphics g = getGraphics();

          //Find the name of the button that was pressed
          Object source = event.getActionCommand();

          for(int index=0;index != pathwayName.length;index++)
{

//Inspect each pathway name in the array of pathway names
               if(source.equals(pathwayName[index])) {
                    panel.j=0;
                    panel.message2();

          String edges = getParameter(pathwayName[index]);

//Prevents extra unwanted nodes and edges if button reclicked
          button[index].removeActionListener(this);

//Create a string tokenizer and process tokens
```

```
     for(StringTokenizer t = new StringTokenizer(edges, ";");
t.hasMoreTokens();) {
            String str = t.nextToken();
            int i = str.indexOf('$');
            if(i > 0) {
               int len = 50;
               int j = str.indexOf('|');
               if(j > 0) {
                 String space = str.substring(j+1,j+2);
                 int k = str.indexOf('/');
                 if(k > 0) {
                    int stoich =
Integer.valueOf(str.substring(k+1)).intValue();
                    str = str.substring(0,j);
                    if(cyto) {
                      if(space.equals("C")) {
                      panel.addEdge(str.substring(0,i),
str.substring(i+1),len,space,stoich);
                      panel.fixedNode(str.substring(0,i));
                      }
                    }
                    else if(mito) {
                      if(space.equals("M")) {
                      panel.addEdge(str.substring(0,i),
str.substring(i+1),len,space,stoich);
                      panel.fixedNode(str.substring(0,i));
                      }
                    }
                    else if(space.equals("C") ||
space.equals("M")) {
                      panel.addEdge(str.substring(0,i),
str.substring(i+1),len,space,stoich);
                      panel.fixedNode(str.substring(0,i));
                    }
                 }
               }
            }
         }
     }
}
          Dimension d = size();
          String center = getParameter("center");
          if(center != null) {
               Node n = panel.nodes[panel.findNode(center)];
               n.x = d.width/2;
               n.y = d.height/2;
               n.fixed = true;
          }
     }
     public void start() {
          panel.start();
```

```
    }
    public void stop() {
        panel.stop();

//Change boolean value when check box or button is clicked
    public boolean action(Event evt, Object arg) {
        if(arg instanceof Boolean) {
            if(((Checkbox)evt.target).getLabel().equals("NOP
movement")) {
                panel.relaxed =
((Boolean)arg).booleanValue();
            }
            else
if(((Checkbox)evt.target).getLabel().equals("Node border")) {
                panel.showNode =
((Boolean)arg).booleanValue();
            }
            else
if(((Checkbox)evt.target).getLabel().equals("NOP fill")) {
                panel.metFill =
((Boolean)arg).booleanValue();
            }
            else
if(((Checkbox)evt.target).getLabel().equals("Blue font")) {
                panel.blueFont =
((Boolean)arg).booleanValue();
            }
            else
if(((Checkbox)evt.target).getLabel().equals("Red font")) {
                panel.redFont =
((Boolean)arg).booleanValue();
            }
            else
if(((Checkbox)evt.target).getLabel().equals("Thick edge")) {
                panel.thick =
((Boolean)arg).booleanValue();
            }
            else
if(((Checkbox)evt.target).getLabel().equals("Cell space")) {
                panel.cellSpace =
((Boolean)arg).booleanValue();
            }
            else
if(((Checkbox)evt.target).getLabel().equals("Maths")) {
                panel.math = ((Boolean)arg).booleanValue();
            }
            return true;
        }

        if("Shake".equals(arg)) {
            Dimension d = size();
```

```
                for(int i = 0;i < panel.nnodes;i++) {
                    Node n = panel.nodes[i];
                    if(!n.fixed) {
                        n.x += 80*Math.random() - 40;
                        n.y += 80*Math.random() - 40;
                    }
                }
                return true;
            }
            return false;
        }
}
/* Drawing classes: Node, Edge and GraphPanel */
class Node {

    double x;
    double y;

    double dx;
    double dy;

    boolean fixed;

    String lbl;
    int nnodes;
}

class Edge {

    int from;
    int to;
    int stoich;
    String space;

    double len;
}

class GraphPanel extends Panel implements Runnable {

    TrivialApplet graph;

    int nnodes;
    Node nodes[] = new Node[10000];

    int nedges;
    Edge edges[] = new Edge[20000];

    Thread relaxer;
    boolean stoichiometry;
    boolean direction;
    boolean redFont;
```

```
boolean blueFont;
boolean random;
boolean relaxed;
boolean thick;
boolean cellSpace;
boolean showNode;
boolean linDiag;
boolean linVert;
boolean sCircular;
boolean mCircular;
boolean lCircular;
boolean circular;
boolean math;
boolean metFill;

private Choice choice1, choice2, choice3;
String operation;

int xRStart = 300;
int yRStart = 100;
int xHStart = 260;
int yHStart = 300;

int j;
int clump;

GraphPanel(TrivialApplet graph) {
     this.graph = graph;

     add(new Label("Layout"));
//Create dropdown list of choices for layout
     choice1 = new Choice();
     choice1.addItem("Random");
     choice1.addItem("Linear Diagonal");
     choice1.addItem("Linear Vertical");
     choice1.addItem("Circular (small)");
     choice1.addItem("Circular (medium)");
     choice1.addItem("Circular (large)");
     add(choice1);

     add(new Label("Edge label"));
//Create dropdown list of choices for edge label
     choice2 = new Choice();
     choice2.addItem("None");
     choice2.addItem("Stoichiometry");
     choice2.addItem("Cell space");
     add(choice2);

     add(new Label("Edge function"));
//Create dropdown list of choices for edge function
     choice3 = new Choice();
```

```
        choice3.addItem("None");
        choice3.addItem("Edge direction");
        add(choice3);

    //Handle events on layout choices
        choice1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {

                Graphics g = getGraphics();

                Choice c1 = (Choice)e.getItemSelectable();
            operation = c1.getSelectedItem();

                if(operation.equals("Linear Diagonal")) {
                    linDiag = true;
                }
                if(operation.equals("Linear Vertical")) {
                    linVert = true;
                }
                if(operation.equals("Circular (small)")) {
                    sCircular = true;
                }
                if(operation.equals("Circular (medium)")) {
                    mCircular = true;
                }
                if(operation.equals("Circular (large)")) {
                    lCircular = true;
                }
            }
        });

        //Handle events on edge label choices
        choice2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {

                Graphics g = getGraphics();

                Choice c2 = (Choice)e.getItemSelectable();
            operation = c2.getSelectedItem();

            if(operation.equals("None")) {
                    stoichiometry = false;
                    cellSpace = false;
                }
            else if(operation.equals("Stoichiometry")) {
                    stoichiometry = true;
                    cellSpace = false;
                }
                else if(operation.equals("Cell space")) {
                    stoichiometry = false;
                    cellSpace = true;
```

```
                }
            }
        });

        //Handle events on edge function choices
        choice3.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {

                Graphics g = getGraphics();

                Choice c3 = (Choice)e.getItemSelectable();
            operation = c3.getSelectedItem();

            if(operation.equals("None")) {
                    direction = false;
                }
            else if(operation.equals("Edge direction")) {
                    direction = true;
                }
            }
        });
}
//Method to fix position of operator node
public void fixedNode(String enzyme) {
        if(enzyme != null) {
            Node n = nodes[findNode(enzyme)];
            n.fixed = true;
        }
}

int i=1;
int k=1;

//Search for node or call addNode() to create it
int findNode(String lbl) {
        for(int i = 0;i < nnodes;i++) {
            if(nodes[i].lbl.equals(lbl)) {
                return i;
            }
        }
return addNode(lbl);
}

//Method to create new node
int addNode(String lbl) {
        Node n = new Node();

        /* node layout algorithms */

        int i;
        int d;
```

```
//separation factors
int s1 = 2;
int s2 = 3;

//linear diagonal
if(linDiag) {
     for(i=-1;i<j;i++) {
          n.x = 300 + 8*s1*i;
          n.y = 100 + 8*s1*i;

          }
     }
//linear vertical
else if(linVert) {
     for(i=-1;i<j;i++) {
          n.x = 300 + 8;
          n.y = 100 + 8*s1*i;
          }
     }
//circular (small)
else if(sCircular) {
     d =120;
     for(i=-1;i<j;i++) {
          n.x = 450 + (int)(d*Math.cos(i));
          n.y = 350 + (int)(d*Math.sin(i));
          }
     }
//circular (medium)
else if(mCircular) {
     d =200;;
     for(i=-1;i<j;i++) {
          n.x = 450 + (int)(d*Math.cos(i));
          n.y = 350 + (int)(d*Math.sin(i));
          }
     }
//circular (large)
else if(lCircular) {
     d =260;;
     for(i=-1;i<j;i++) {
          n.x = 450 + (int)(d*Math.cos(i));
          n.y = 350 + (int)(d*Math.sin(i));
          }
     }

//random
else {
n.x = xRStart + 350*Math.random();
n.y = yRStart + 350*Math.random();

}
```

```
n.lbl = lbl;

//Code for repetition of side compounds
if((n.lbl).length()>3) {
     if((n.lbl).substring(0,3).equals("ATP")) {
          n.lbl ="ATP";
          }
     }
if((n.lbl).length()>3) {
     if((n.lbl).substring(0,3).equals("ADP")) {
          n.lbl ="ADP";
          }
     }
if((n.lbl).length()>3) {
     if((n.lbl).substring(0,3).equals("H2O")) {
          n.lbl ="H2O";
          }
     }
if((n.lbl).length()>2) {
     if((n.lbl).substring(0,2).equals("H+")) {
          n.lbl ="H+";
          }
     }
     if((n.lbl).length()>4) {
     if((n.lbl).substring(0,4).equals("NAD+")) {
          n.lbl ="NAD+";
          }
     }
if((n.lbl).length()>4) {
     if((n.lbl).substring(0,4).equals("NADH")) {
          n.lbl ="NADH";
          }
     }
if((n.lbl).length()>5) {
     if((n.lbl).substring(0,5).equals("NADP+")) {
          n.lbl ="NADP+";
          }
     }
if((n.lbl).length()>5) {
     if((n.lbl).substring(0,5).equals("NADPH")) {
          n.lbl ="NADPH";
          }
     }
     if((n.lbl).length()>2) {
     if((n.lbl).substring(0,2).equals("Pi")) {
          n.lbl ="Pi";
          }
     }
     j++;
nodes[nnodes] = n;
```

```
            return nnodes++;
        }
//Create new edge with labels and send node names to findNode()
        void addEdge(String from,String to,int len,String
space,int stoich) {
            Edge e = new Edge();
            e.from = findNode(from);
            e.to = findNode(to);
            e.len = len;
            e.stoich = stoich;
            e.space = space;
            edges[nedges++] = e;
        }
        //Logic for the thread
        public void run() {
            while(true) {
                if(relaxed) {
                relax();
                }
                if(random && (Math.random() < 0.03)) {
                    Node n =
nodes[(int)(Math.random()*nnodes)];
                    if(!n.fixed) {
                        n.x += 100*Math.random() - 50;
                        n.y += 100*Math.random() - 50;
                    }
                }
                try {
                    Thread.sleep(100);
                }
                catch(InterruptedException e) {
                    break;
                }
            }
        }
        //Algorithm for the motion of nodes
        synchronized void relax() {
            for(int i = 0;i < nedges;i++) {
                Edge e = edges[i];
                double vx = nodes[e.to].x - nodes[e.from].x;
                double vy = nodes[e.to].y - nodes[e.from].y;
                double len = Math.sqrt(vx * vx + vy * vy);
                double f = (edges[i].len - len)/(len*3);
                double dx = f*vx;
                double dy = f*vy;

                nodes[e.to].dx += dx;
                nodes[e.to].dy += dy;
                nodes[e.from].dx += -dx;
                nodes[e.from].dy += -dy;
            }
```

```
for(int i = 0;i < nnodes;i++) {
     Node n1 = nodes[i];
     double dx = 0;
     double dy = 0;

     for(int j = 0;j < nnodes;j++) {
          if(i == j) {
               continue;
          }

          Node n2 = nodes[j];
          double vx = n1.x - n2.x;
          double vy = n1.y - n2.y;
          double len = vx*vx + vy*vy;
          if(len == 0) {
                dx += Math.random();
                dy += Math.random();
                }
                else if(len < 100*100) {
                    dx += vx/len;
                    dy += vy/len;
                }
          }

          double dlen = dx*dx + dy*dy;
          if(dlen > 0) {
               dlen = Math.sqrt(dlen)/2;
               n1.dx += dx/dlen;
               n1.dy += dy/dlen;
          }
     }

     Dimension d = size();
     for(int i = 0;i < nnodes;i++) {
          Node n = nodes[i];
          if(!n.fixed) {
               n.x += Math.max(-5,Math.min(5,n.dx));
               n.y += Math.max(-5,Math.min(5,n.dy));
               if(n.x < 0) {
                    n.x = 0;
               }else if(n.x > d.width) {
                    n.x = d.width;
               }
               if(n.y < 0) {
                    n.y = 0;
               }else if(n.y > d.height) {
                    n.y = d.height;
               }
          }
          n.dx /= 2;
```

```
                    n.dy /= 2;
                }
                repaint();
            }

            Node pick;
            boolean pickfixed;
            Image offscreen;
            Dimension offscreensize;
            Graphics offgraphics;

            //Node colours
            final Color fixedColor = Color.yellow;
            final Color selectColor = Color.pink;
            final Color nodeColor1 = new Color(250,220,100);
            final Color nodeColor2 = Color.white;
            final Color nodeColor3 = Color.magenta;
            final Color nodeColor4 = Color.lightGray;
            final Color nodeColor5 = Color.cyan;
            final Color nodeColor6 = new Color(255,150,50);

            //Edge colours
            final Color edgeColor = Color.black;
            final Color stoichiometryColor1 = Color.blue;
            final Color stoichiometryColor2 = Color.green;
            final Color arcColor1 = Color.black;
            final Color arcColor2 = Color.green;
            final Color arcColor3 = Color.pink;
            final Color arcColor5 = Color.blue;

    //Method to display message   1
    public void message1() {
            Graphics g = getGraphics();
            g.setFont(new Font("Serif", Font.BOLD, 30));
            g.setColor(Color.red);
            g.drawString("Ensure some reactions are present in
selected compartment.",180, 200);
            g.drawString("Then select a layout and/or click a
button.",180, 250);
            g.drawString("If no response, reload or refresh
applet, or check html source.",180, 300);
            Color backgroundColour = getBackground();
           g.setColor(backgroundColour);//clear text area
            g.fillRect(105,310,930,60);//(x,y,width,height)
    }
    //Method to display message 2
    public void message2() {
            Graphics g = getGraphics();
            g.setFont(new Font("Serif", Font.BOLD, 30));
            g.setColor(Color.black);
```

```
g.drawString("NOW CLICK WINDOW OR CLICK 'NOP movement' CHECK
BOX!",110, 350);
     Color backgroundColour = getBackground();
     g.setColor(backgroundColour);//clear text area
     g.fillRect(160,150,840,160);//(x,y,width,height)
     }

     //Method to draw nodes
     public void paintNode(Graphics g, Node n, FontMetrics fm)
{
          int x = (int)n.x;
          int y = (int)n.y;
          int op = 0;
          if(n.lbl.indexOf("MM") != -1) {
               op = 1;
          }
          if(n.lbl.indexOf("receptor") != -1) {
               op = 2;
          }

          if(n.lbl.equals("cAMP")) {
               op = 3;
          }

/* ?: is a ternary (3-way) if-then-else operator; if expression
1 is true, then expression 2 is evaluated, otherwise,
expression 3 is evaluated */
g.setColor((n == pick) ? selectColor : (op == 1 ? nodeColor3:
                                        (op == 2 ? nodeColor5:
                                        (op == 3 ? nodeColor6:
                                        (n.fixed ? fixedColor :
nodeColor2)))));


if(metFill) {
g.setColor((n == pick) ? selectColor : (op == 1 ? nodeColor3:
                                        (op == 2 ? nodeColor5:
                                        (op == 3 ? nodeColor6:
                                        (n.fixed ? fixedColor :
nodeColor4)))));
          }

          int w = fm.stringWidth(n.lbl) + 10;
          int h = fm.getHeight() + 4;

          int xCoord = x - w/2;
          int yCoord = y - h/2;
          g.fillRect(x - w/2, y - h/2, w, h);

          g.setColor(Color.black);
```

```
          if(redFont) {
               g.setColor(Color.red);
          }

          else if(blueFont) {
               g.setColor(Color.blue);
          }
          if(showNode) {
               g.drawRect(x - w/2, y - h/2, w - 1, h - 1);
               }
               g.drawString(n.lbl, x - (w-10)/2, (y - (h-4)/2)
+ fm.getAscent());

               double ratio = (double)nedges/nnodes;

//Adjust a double value to two decimal places with rounding
               if(math) {
               final double roundingFraction = 0.005;
               double newRatio;
               ratio = ratio + roundingFraction;
               newRatio = (int)(100.0*ratio);
               newRatio = (double)newRatio/100;

          g.drawString("Number of edges: "+nedges, 940, 15);
          g.drawString("Number of nodes: "+nnodes, 940, 27);
          g.drawString("Edge/node ratio: "+newRatio, 940, 39);
               }
          }
     //Draw edges & edge labels and call paintNode()
     public synchronized void update(Graphics g) {
          Dimension d = size();
          if((offscreen == null) || (d.width !=
offscreensize.width) || (d.height != offscreensize.height)) {
               offscreen = createImage(d.width,d.height);
               offscreensize = d;
               offgraphics = offscreen.getGraphics();
               offgraphics.setFont(getFont());
          }

          offgraphics.setColor(getBackground());
          offgraphics.fillRect(0,0,d.width,d.height);
          for(int i = 0;i < nedges;i++) {
               Edge e = edges[i];
               int x1 = (int)nodes[e.from].x;
               int y1 = (int)nodes[e.from].y;
               int x2 = (int)nodes[e.to].x;
               int y2 = (int)nodes[e.to].y;
               int stoichValue = e.stoich;
               String location = e.space;

               offgraphics.setColor(arcColor1);
```

```
                if(direction) {
                offgraphics.setColor((stoichValue < 0) ?
arcColor5 : (stoichValue > 0 ? arcColor2 : arcColor1));
                }
     //Keep this line here to use the above colour changes
                offgraphics.drawLine(x1,y1,x2,y2);//draw edges

                if(thick) {
offgraphics.drawLine(x1+1,y1+1,x2+1,y2+1);//draw second line to
thicken edge
                }

                if(cellSpace) {
                     String lbl = location;
                     offgraphics.setColor(edgeColor);
                     offgraphics.drawString(lbl,x1 + (x2-x1)/2,
y1 + (y2-y1)/2);
                }

                if(stoichiometry) {
                     String lbl = String.valueOf(stoichValue);
                     if(stoichValue < 0) {

     offgraphics.setColor(stoichiometryColor1);
                          }
                     else if(stoichValue > 0) {

     offgraphics.setColor(stoichiometryColor2);
                          }
                     else offgraphics.setColor(edgeColor);
offgraphics.drawString(lbl,x1 + (x2-x1)/2, y1 + (y2-y1)/2);
                }
          }

          FontMetrics fm = offgraphics.getFontMetrics();
          for(int i = 0;i < nnodes;i++) {
               paintNode(offgraphics,nodes[i],fm);
          }

          g.drawImage(offscreen,0,0,null);
          }
//x and y are the coordinates at which the mouse is pressed
          public synchronized boolean mouseDown(Event evt, int
x, int y) {
          if(nnodes == 0) message1();
               double bestdist = Double.MAX_VALUE;
               for(int i = 0;i <nnodes;i++) {
                    Node n = nodes[i];
     double dist = (n.x - x)*(n.x - x) + (n.y - y)*(n.y - y);

                    if(dist < bestdist) {
```

```
                    pick = n;
                    bestdist = dist;
                }
            }

            pickfixed = pick.fixed;
            pick.fixed = true;
            pick.x = x;
            pick.y = y;
            repaint();
            return true;
        }
//Method to allow continuous smooth drag of nodes
public synchronized boolean mouseDrag(Event evt, int x, int y)
{
            pick.x = x;//drag in x direction
            pick.y = y;//drag in y direction
            repaint();
            return true;
        }
//x and y are the coordinates at which the mouse is released
public synchronized boolean mouseUp(Event evt, int x, int y) {
            pick.x = x;
            pick.y = y;
            pick.fixed = pickfixed;
            pick = null;
            repaint();
            return true;
        }
        //Start thread
        public void start() {
            relaxer = new Thread(this);
            relaxer.start();
        }
        public void stop() {
            relaxer.stop();
        }
    }
```

Appendix A9

Shown below is the HTML source for the Glycolysis pathway applet. Data are input via the applet argument in the html source. A semi-colon separates each n-ary relation, and a different delimiter separates each component of an n-ary relation. The following example of an n-ary relation illustrates the code for this applet:

HK$glucose/6|0&s#n!-1;

The enzyme ('HK' for hexokinase) is separated from the metabolite (glucose) by the dollar sign '$' and the metabolite is separated from the number of carbon atoms by a forward slash '/', while the latter is separated from the number of phosphorous atoms by '|', which in turn, is separated from metabolite role by the ampersand '&'. Next comes reversibility (where n = no) separated by '#', and finally, signed stoichiometry separated by the exclamation mark '!'. The data can be input manually or partially generated by the application and then modified manually. This arises from the fact that, while there are some similarities, the n-ary relations are not equivalent.

HTML source for the Glycolysis pathway applet

```
<title>TrivialApplet</title>
<hr>
<applet codebase="Java Classes" code="TrivialApplet.class"
width=1100 height=700>
<param name=glycolysis value="HK$glucose/6|0&S#n!-1;
 ;HK$ATP1/0|3&C#n!-1;HK$ADP1/0|2&C#n!1;HK$G6P/6|1&P#n!1;
 ;HK$H+1/0|0&P#n!1;PGI$G6P/6|1&S#y!-1;PGI$F6P/6|1&P#y!1;
 ;PFK$F6P/6|1&S#n!-1;PFK$ATP2/0|3&C#n!-1;
 ;PFK$ADP2/0|2&C#n!1;PFK$FBP/6|2&P#n!1;
 ;PFK$H+2/0|0&P#n!1;aldolase$FBP/6|2&S#y!-1;
 ;aldolase$G3P/3|1&P#y!1;aldolase$DHAP/3|1&P#y!1;
 ;TPI$DHAP/3|1&S#y!-1;TPI$G3P/3|1&P#y!1;
 ;G3PDH$G3P/6|2&S#y!-2;G3PDH$NAD+/0|0&F#y!-2;
 ;G3PDH$Pi/0|2&S#y!-2;G3PDH$NADH/0|0&F#y!2;
 ;G3PDH$H+3/0|0&P#y!2;G3PDH$BPG/6|4&P#y!2;
 ;PGK$BPG/6|4&S#y!-2;PGK$ADP3/0|4&C#y!-2;
 ;PGK$ATP3/0|6&C#y!2;PGK$3PG/6|2&P#y!2;
 ;PGM$3PG/6|2&S#y!-2;PGM$2PG/6|2&P#y!2;
```

```
 ;enolase$2PG/6|2&S#y!-2;enolase$PEP/6|2&P#y!2;
 ;enolase$H2O/0|0&P#y!2;PK$PEP/6|2&S#n!-2;
 ;PK$ADP4/0|4&C#n!-2;
 ;PK$H+4/0|0&S#n!-2;PK$ATP4/0|6&C#n!2;
 ;PK$pyruvate/6|0&P#n!2">
</applet>
<hr>
```

Appendix A10

Shown below is the HTML source for the Multi-pathway applet. Data are input via the applet argument in the html source. A semi-colon separates each n-ary relation, and a different delimiter separates each component of an n-ary relation. The following example of an n-ary relation illustrates the code for this applet:

glucose-6-phosphate isomerase$D-glucose 6-phosphate|C/-1;

The enzyme (glucose-6-phosphate isomerase) is separated from the metabolite (D-glucose 6-phosphate) by the dollar sign '$' and the metabolite is separated from the cell compartment (C for cytosol) by '|', and finally, signed stoichiometry is separated from the latter by a forward slash '/'. The data can be input manually or partially generated by the application and then modified manually. This arises from the fact that, while there are some similarities, the n-ary relations are not equivalent.

HTML source for the Multi-pathway applet

```
<title>TrivialApplet</title>
  <hr>
  <applet codebase="Java Classes" code="TrivialApplet.class"
width=1100 height=700>
  <param name=Glycolysis value="hexokinase$D-glucose|C/-1;
   ;hexokinase$ATP1|C/-1;hexokinase$ADP1|C/1;
   ;hexokinase$D-glucose 6-phosphate|C/1;
   ;glucose-6-phosphate isomerase$D-glucose 6-phosphate|C/-1;
   ;glucose-6-phosphate isomerase$D-fructose 6-phosphate|C/1;
   ;6-phosphofructokinase$ATP2|C/-1;
   ;6-phosphofructokinase$D-fructose 6-phosphate|C/-1;
   ;6-phosphofructokinase$ADP2|C/1;
   ;6-phosphofructokinase$D-fructose 1;6-bisphosphate|C/1;
   ;fructose-bisphosphate aldolase$D-fructose 1;
   ;6-bisphosphate|C/-1;
   ;fructose-bisphosphate aldolase$glycerone phosphate|C/1;
   ;fructose-bisphosphate aldolase$D-glyceraldehyde 3-
phosphate|C/1;
   ;triose-phosphate isomerase$glycerone phosphate|C/-1;
   ;triose-phosphate isomerase$D-glyceraldehyde 3-
phosphate|C/1;
```

```
   ;glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)$D-glyceraldehyde 3-phosphate|C/-1;
   ;glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)$phosphate|C/-1;
   ;glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)$NAD+1|C/-1;
   ;glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)$3-phospho-D-glyceroyl phosphate|C/1;
   ;glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)$NADH1|C/1;
   ;phosphoglycerate kinase$ADP3|C/-1;
   ;phosphoglycerate kinase$3-phospho-D-glyceroyl phosphate|C/-
1;
   ;phosphoglycerate kinase$ATP3|C/1;
   ;glyceraldehyde-3-phosphate dehydrogenase
(phosphorylating)$H+1|C/1;
   ;phosphoglycerate kinase$3-phospho-D-glycerate|C/1;
   ;phosphoglycerate mutase$3-phospho-D-glycerate|C/-1;
   ;phosphoglycerate mutase$2-phospho-D-glycerate|C/1;
   ;phosphopyruvate hydratase$2-phospho-D-glycerate|C/-1;
   ;phosphopyruvate hydratase$phosphoenolpyruvate|C/1;
   ;phosphopyruvate hydratase$H2O1|C/1;pyruvate kinase$ADP4|C/-
1;
   ;pyruvate kinase$phosphoenolpyruvate|C/-1;pyruvate
kinase$ATP4|C/1;
   ;pyruvate
kinase$enolpyruvate|C/1;tautomerisation$enolpyruvate|C/-1;
   ;tautomerisation$pyruvate|C/1">
  <param name=Pentose_Phosphate value="glucose-6-phosphate 1-
dehydrogenase$D-glucose 6-phosphate|C/-1;
   ;glucose-6-phosphate 1-dehydrogenase$NADP+1|C/-1;
   ;glucose-6-phosphate 1-dehydrogenase$D-glucono-1,5-lactone
6-phosphate|C/1;
   ;glucose-6-phosphate 1-dehydrogenase$NADPH1|C/1;
   ;glucose-6-phosphate 1-dehydrogenase$H+2|C/1;
   ;6-phosphogluconolactonase$D-glucono-1,5-lactone 6-
phosphate|C/-1;
   ;6-phosphogluconolactonase$H2O2|C/-1;
   ;6-phosphogluconolactonase$6-phospho-D-gluconate|C/1;
   ;6-phosphogluconolactonase$H+3|C/1;phosphogluconate
dehydrogenase (decarboxylating)$6-phospho-D-gluconate|C/-1;
   ;phosphogluconate dehydrogenase (decarboxylating)$NADP+2|C/-
1;
   ;phosphogluconate dehydrogenase (decarboxylating)$D-ribulose
5-phosphate|C/1;
   ;phosphogluconate dehydrogenase (decarboxylating)$CO2|C/1;
   ;phosphogluconate dehydrogenase
(decarboxylating)$NADPH2|C/1;
   ;ribulose-phosphate 3-epimerase$D-ribulose 5-phosphate|C/-1;
   ;ribulose-phosphate 3-epimerase$D-xylulose 5-phosphate|C/1;
   ;ribose-5-phosphate isomerase$D-ribulose 5-phosphate|C/-1;
```

```
   ;ribose-5-phosphate isomerase$D-ribose 5-phosphate|C/1;
   ;transketolase1$D-ribose 5-phosphate|C/-1;
   ;transketolase1$D-xylulose 5-phosphate|C/-1;
   ;transketolase1$sedoheptulose 7-phosphate|C/1;
   ;transketolase1$D-glyceraldehyde 3-phosphate|C/1;
   ;transaldolase$sedoheptulose 7-phosphate|C/-1;
   ;transaldolase$D-glyceraldehyde 3-phosphate|C/-1;
   ;transaldolase$D-erythrose 4-phosphate|C/1;
   ;transaldolase$D-fructose 6-phosphate|C/1;
   ;transketolase2$D-erythrose 4-phosphate|C/-1;
   ;transketolase2$D-xylulose 5-phosphate|C/-1;
   ;transketolase2$D-fructose 6-phosphate|C/1;
   ;transketolase2$D-glyceraldehyde 3-phosphate|C/1;">
  <param name=Urea_cycle value="MM1(in)$c-CO2|C/-1;
   ;MM1(in)$m-CO2|M/1;MM2(in)$c-NH4|C/-1;MM2(in)$m-NH4|M/1;
   ;carbamoyl-phosphate synthase (ammonia)$ATP5|M/-2;
   ;carbamoyl-phosphate synthase (ammonia)$m-NH4|M/-1;
   ;carbamoyl-phosphate synthase (ammonia)$m-CO2|M/-1;
   ;carbamoyl-phosphate synthase (ammonia)$H2O3|M/-1;
   ;carbamoyl-phosphate synthase (ammonia)$ADP5|M/2;
   ;carbamoyl-phosphate synthase (ammonia)$phosphate|M/1;
   ;carbamoyl-phosphate synthase (ammonia)$carbamoyl
phosphate|M/1;
   ;ornithine carbamoyltransferase$carbamoyl phosphate|M/-1;
   ;ornithine carbamoyltransferase$m-L-ornithine|M/-1;
   ;ornithine carbamoyltransferase$phosphate|M/1;
   ;ornithine carbamoyltransferase$m-L-citrulline|M/1;
   ;MM(out)$m-L-citrulline|M/-1;MM(out)$c-L-citrulline|C/1;
   ;argininosuccinate synthase$ATP6|C/-1;
   ;argininosuccinate synthase$c-L-citrulline|C/-1;
   ;argininosuccinate synthase$L-aspartate|C/-1;
   ;argininosuccinate synthase$AMP|C/1;
   ;argininosuccinate synthase$diphosphate|C/1;
   ;argininosuccinate synthase$N-(L-arginino)succinate|C/1;
   ;argininosuccinate lyase$N-(L-arginino)succinate|C/-1;
   ;argininosuccinate lyase$fumarate|C/1;
   ;argininosuccinate lyase$L-arginine|C/1;
   ;arginase$L-arginine|C/-1;
   ;arginase$H2O4|C/-1;arginase$c-L-ornithine|C/1;
   ;arginase$urea|C/1;MM3(in)$c-L-ornithine|C/-1;
   ;MM3(in)$m-L-ornithine|M/1">
  <param name=Regulatory_cascade value="adrenergic
receptor$adrenaline|C/-1;
   ;adrenergic receptor$adrenaline-(adrenergic receptor)|C/1;
   ;adrenaline-(adrenergic receptor)$adenylate cyclase I|C/-1;
   ;adrenaline-(adrenergic receptor)$adenylate cyclase A|C/1;
   ;adenylate cyclase A$ATP7|C/-1;adenylate cyclase A$cAMP|C/1;
   ;adenylate cyclase A$PPi|C/1;cAMP$protein kinase I|C/-1;
   ;cAMP$protein kinase A|C/1;
   ;protein kinase A$dephospho-phosphorylase kinase|C/-1;
   ;protein kinase A$ATP8|C/-1;
```

```
    ;protein kinase A$phospho-phosphorylase kinase|C/1;
    ;protein kinase A$ADP6|C/1;
    ;phospho-phosphorylase kinase$glycogen phosphorylase b|C/-1;
    ;phospho-phosphorylase kinase$ATP9|C/-1;
    ;phospho-phosphorylase kinase$glycogen phosphorylase a|C/1;
    ;phospho-phosphorylase kinase$ADP7|C/1;
    ;glycogen phosphorylase a$glycogen|C/-1;
    ;glycogen phosphorylase a$Pi1|C/-1;
    ;glycogen phosphorylase a$glucose I-phosphate|C/1">
   <param name=Galactose_Metabolism
value="galactokinase$galactose|C/-1;
    ;galactokinase$ATP10|C/-1;galactokinase$ADP8|C/1;
    ;galactokinase$galactose 1-phosphate|C/1;
    ;galactose 1 phosphate uridylyl transferase$galactose 1-
phosphate|C/-1;
    ;galactose 1 phosphate uridylyl transferase$glucose 1-
phosphate|C/1;
    ;galactose 1 phosphate uridylyl transferase$UDP-glucose|C/-
1;
    ;galactose 1 phosphate uridylyl transferase$UDP-
galactose|C/1;
    ;UDP-galactose 4-epimerase$UDP-galactose|C/-1;
    ;UDP-galactose 4-epimerase$UDP-glucose|C/1;
    ;phosphoglucomutase$glucose 1-phosphate|C/-1;
    ;phosphoglucomutase$D-glucose 6-phosphate|C/1">
   <param name=PKI_to_PKA value="cAMP$protein kinase I|C/-1;
    ;cAMP$protein kinase A|C/1">
   <param name=DPPPK_to_PPPK value="protein kinase A$dephospho-
phosphorylase kinase|C/-1;
    ;protein kinase A$ATP11|C/-1;
    ;protein kinase A$phospho-phosphorylase kinase|C/1;
    ;protein kinase A$ADP9|C/1">
   <param name=GPPb_to_GPPa value="phospho-phosphorylase
kinase$glycogen phosphorylase b|C/-1;
    ;phospho-phosphorylase kinase$ATP12|C/-1;
    ;phospho-phosphorylase kinase$glycogen phosphorylase a|C/1;
    ;phospho-phosphorylase kinase$ADP10|C/1">
   <param name=Glycogen_to_G1P value="glycogen phosphorylase
a$glycogen|C/-1;
    ;glycogen phosphorylase a$Pi2|C/-1;glycogen phosphorylase
a$glucose 1-phosphate|C/1">
   <param name=G6P_to_Glucose value="glucose-6-phosphatase$D-
glucose 6-phosphate|C/-1;
    ;glucose-6-phosphatase$H2O5|C/-1;glucose-6-
phosphatase$glucose|C/1;
    ;glucose-6-phosphatase$Pi3|C/1">
   <param name=User_defined value="operator$non-operator1|C/-1;
    ;operator$non-operator2|C/1">
   </applet>
   <hr>
```