Detecting Restriction Class Correspondences in Linked Open Data

A thesis submitted to the

University of Dublin, Trinity College

for the degree of

Doctor of Philosophy

Brian Walshe
Knowledge and Data Engineering Group (KDEG)
School of Computer Science
Trinity College, Dublin,
walshebr@cs.tcd.ie

Supervised by Prof. Declan O'Sullivan
Co-supervised by Dr. Rob Brennan

2014

Declaration

I, the undersigned, declare that this work has not been previously submitted as an exercise for a degree at this or any other University, and that, unless otherwise stated, it is entirely my own work.

_____

name

Date

Permission to lend or copy

I, the undersigned, agree that the Trinity College Library may lend or copy this thesis upon request.

_____

name

Date

Acknowledgements

I would like to thank my supervisor Declan O'Sullivan for his excellent planning skills, and for showing great patience with me over the years. I would like to thank my second supervisor, Rob Brennan, for talking me out of some very bad ideas I had for evaluation section.

This work is partially funded through the Science Foundation Ireland FAME Strategic Research Cluster (award No. 08/SRC/I1408), http://www.fame.ie.

This work is dedicated to the memory of my father, Chris, his HP65 program for predicting ship collisions, and his system for winning the lottery.

Abstract

The Linked Open Data (LOD) project has made a broad range of knowledge available on the World Wide Web as open datasets, using a common format and linked in such a manner that it is a simple task to explore and integrate data from different sources. The links between the datasets are one to one relationships between named URIs, but these simple links are not always sufficient to describe many relationships between data in the sets. Sometimes it is more appropriate to use a complex correspondence to describe the relationship – a correspondence which involves one or more entities in a logical formulation. This thesis discusses an approach to detecting complex correspondences between ontologies associated with LOD.

There are several challenges associated with this task. First, the datasets can be very large and the number of potential complex correspondences is a combinatorial function of their size. Secondly, many of the datasets in the LOD cloud may have large amounts of missing information or invalid entries. Our approach focuses on detecting correspondences between named classes and logically constructed restriction classes. An extensional approach – that is, one which uses instance data shared by the datasets – is employed to discover appropriate restriction classes for the correspondences. Unlike other extensional approaches, ours focuses on using methods which will perform well with only a small number of example instances – as finding matched instances is not usually a trivial task.

To evaluate the approach we first demonstrate that it can be used to detect complex correspondences between the DBpedia and YAGO2 datasets. We then show that a small sample of 15 matched instances can be as effective as using a sample in the order of 10^4 instances. Further to this we show that as the level of missing data increases, a selection metric which takes the open world assumption into account can consistently outperform the Information Gain metric, popular in the field of machine learning. Finally we compare our approach to a leading extensional approach to detecting complex correspondences, and show that our approach can produce more accurate correspondences, while using significantly less input data.

The primary contribution of this thesis is a robust, scalable approach to detecting complex correspondences between classes in ontologies. LOD ontologies were the primary motivation

behind this work, but the approach could be applied to any ontologies which contain individuals which can be mapped directly to one another with equivalence relationships.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

On the 2nd of May 1956, Richard Feynman gave an address at California Institute of Technology where he said *"In this age of specialisation men who thoroughly know one field are often incompetent to discuss another."*[1] Fifty years later, in 2006, Tim Berners-Lee published a note[2] in which he outlined four simple rules for Linked Data which promised to make the sum total of human knowledge available in a browsable, linked format which could easily be interpreted by both humans and machines. With all this knowledge available, the age of specialisation was sure to end.

Berners-Lee's rules for linked data are as follows:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards such as RDF and RDFS- XML based languages for describing ontologies – and SPARQL – a language for querying RDF
4. Include links to other URIs, so that they can discover more things.

These rules focus on making data accessible, and on making it easy to find related information. They do not however say much about *how* the information is related. Simple links cannot always describe how different fields of knowledge are related. It is still the case that a thoroughly versed expert in one field cannot easily integrate information from another field without investing time to understand the new domain of knowledge.

Ontologies provide a method of formally describing a domain of knowledge. They allow for the description of hierarchies of concepts and relationships. Understanding the correspondences between ontologies, allows us to understand how different domains of knowledge relate to one another. At a more practical level, many Linked Open Data knowledge bases have ontologies associated with them and correspondences between these ontologies can enable mediation between the datasets. Three common mediation tasks are *ontology merging*, *query answering*, and *data translation*.[1]. Ontology Merging can be used to create a single coherent ontology from information gathered from multiple independent

[1] http://calteches.library.caltech.edu/49/2/Religion.htm

[2] http://www.w3.org/DesignIssues/LinkedData.html

sources [2]. Query rewriting can be used, for example, to allow a system to chose from multiple sources to best answer a given query [3]. Instance transformation can be used by mediation services to pass the output of one process to the input of another [4].

However, heterogeneity issues such as differences in class scope or hierarchy granularity mean that simple one to one correspondences between atomic classes are not always enough to describe the relationships between ontologies. YAGO2 [5], for example, contains a rich hierarchy based on WordNet [6], and includes many professions described as classes. An instance of a person in YAGO2 who is a film director, should have this fact described using an *rdf:type* declaration. In contrast, DBpedia [7] has a shallower class hierarchy, with professions described as attribute-values, not classes. If one to one mappings between atomic classes is the only mechanism available, then we could say that *yago:Director* maps to *dbpedia:Person* with a subsumption relationship; but this is not very satisfactory. If, instead, non-atomic classes defined by restrictions can be used then it could be said that yago:*Director* corresponds with the set of instances of Person in DBpedia with the attribute *dbpedia-owl:occupation* set to *Director*.

More formally, correspondences where at least one of the linked entities is non-atomic are known as *complex correspondences* [8]. Research has been undertaken to provide a common framework to describe and use complex correspondences. This includes Correspondence Patterns [9] which provide a categorisation of the forms of complex correspondence that can exist. It provides a high level description of the correspondence problem and is analogous to design patterns [10] used in software development. On a lower level, EDOAL [11] has become the de-facto language for describing complex correspondences.

The task of discovering complex correspondences however, is still an emergent topic in the field of ontology matching. For example, 2012 was the first year that the Ontology Alignment Evaluation Initiative (OAEI) [12] campaign asked for entries capable of detecting complex correspondences. However all 16 entrants only addressed one-to-one correspondences. Similarly, in 2013 there were no entrants that addressed complex correspondences. The OAEI is held in conjunction with the Ontology Matching (OM) workshop[3]. In 2012, of the 6 technical papers accepted to the workshop, only one could be considered to address complex correspondences, discussing a method for transforming string values in Linked Open Data [13]. In 2013 a paper [14] was accepted that described how complex correspondences fitting

[3] http://om2012.ontologymatching.org/#ap

the Correspondence Pattern scheme could be used to rewrite SPARQL queries. Use of this technique requires first finding the complex correspondences, which is not a simple task. The focus on one to one correspondences means that the schemas used by LOD collections remain mostly unaligned [15].

As techniques for automating the process of discovering complex correspondences is still an emergent topic, there are relatively few published articles addressing it. Of the methods that have been discussed (see section 3.2), it is possible to broadly categorise them into structural methods [8] and common extension methods [15], [16]. Structural methods use only information about an ontology's structure to search for correspondences between classes. Common extension methods use information about the shared instances of classes to find their correspondences. Both approaches have their relative merits and the choice of which approach to use depends on the properties of the ontologies being aligned.

The LOD cloud comprises 295 individual datasets containing a total of more than 3×10^{10} triple statements, of which approximately 5×10^{8} are interlinks between the datasets[4]. Interlinks between instances in many pairs of these sets are frequent – DBpedia and YAGO2, for example, share over 2.5×10^{6} instances linked with *owl:sameAs* relationships. Other datasets which share large numbers of instances with DBpedia include Geonames and Freebase. The large amounts of matched instance data would suggest that common extension methods should be suitable for discovering complex correspondences between ontologies used by LOD datasets.

For most ontology alignment tasks it is likely that few, if any, instance matches will be provided at the start of the alignment task. Typically during the alignment process, matches must be approved by a human before they are considered valid mappings [17]. This makes it difficult to generate a large number of accurate mappings quickly. Even in cases such as the LOD, where published matches are provided, they may still need to be approved by hand as they are not reliably accurate [18]. Therefore, any common extension method which requires a large amount of matched instance data is unlikely to succeed.

Structural methods for detecting complex correspondences may appear attractive as they do not require *any* matched instances. However, in a large scale analysis of linked data [19] it was found that the utilization of RDF vocabularies by data publishers can often differ from the intended application envisioned by ontology engineers. Intuitively, there is a danger that

[4] http://lod-cloud/state/ (Version 0.3, 09/19/2011)

if complex correspondences produced using only structural information are used to perform mediation on instance data, they might not perform as expected.

Current approaches to discovering complex correspondences assume that either large amounts of the common instances shared by the ontologies are known, or that the structure of each ontology accurately reflects the data that it describes. As can be seen above, there are real world applications where these assumptions break down. Therefore, this thesis investigates methods for discovering complex correspondences which do not require large amounts of common instance data, and which are not entirely reliant on rich schema data.

1.2 Research Question

The research question addressed in this thesis is as follows: *To what extent is it possible to use small samples of accurately matched instance data to detect complex correspondences between classes in their related ontologies?*

In this context, accurately matched instance data means a set of paired URIs from the ontologies that have a high probability of referencing the same entity. The process of creating these instance matches is considered outside the scope of this thesis.

In addressing this question it is assumed that the ontologies are lightweight. That is, ontologies which contain concepts and taxonomies, the relationships and properties of the concepts; but which do not contain not axioms and constraints which clarify the intended meaning of the terms in the ontology. It is assumed that these ontologies have non-empty ABoxes, and that some elements from these ABoxes can be matched to one another. It is also assumed that the ontologies are described in RDF or OWL, and can be queried through a SPARQL endpoint.

1.2.1 Research objectives

This thesis describes an improved approach for detecting complex correspondences between ontologies using extensional information. Current methods for detecting complex correspondences rely on either an ontology with a schema which fits with a narrowly defined pattern, or which contains large amounts of shared instances between the ontologies. The new approach described in this thesis should be more tolerant to discrepancies between schema structure and instance data, while requiring less instance data than other extensional approaches. In order to address the research question the following objectives were defined:

1. Evaluate which forms of complex correspondence exist between ontologies, and therefore which forms that our detection method should focus on.

2. Study existing correspondence discovery techniques to establish the capability of state of the art matching technologies to detect the correspondences highlighted from the survey resulting from objective 1.
3. Develop improved techniques for discovering complex correspondences between classes in ontologies using common extensional information.
4. Evaluate the ability of the common extensional based techniques to detect complex correspondences in real world ontologies, using LOD as a case study.

1.2.2 Thesis Contributions

The major contribution of this thesis is the development of methods for detecting complex correspondences between ontologies that share instance data. The approach is novel in that it does not analyse the ontologies schema structure, and only requires a small sample of interlinked instance data. This allows the method to detect correspondences between ontologies which would not be detectable using current state of the art complex correspondence detection methods, which require well defined ontologies or large amounts of matched instance data.

This thesis demonstrates that the detection of complex correspondences can be seen as a feature selection task. It shows that using a commonly used feature selection metric known as information gain (IG) a sample of 20 training instances per class can produce similar detection results as using all instance data in the ontology. This thesis also shows that the beta-binomial distribution can produce more accurate results than IG when there is missing data. With 25% of triples removed from the training set, the beta-binomial based metric ranked a set of gold standard correspondences on average one place higher than when using IG. To the author's knowledge, beta-binomial based metrics have not been used to perform feature selection before. The complex correspondence detection methods developed in this thesis outperform the state of the art detectors producing more accurate correspondences in one third of the cases tested.

A minor contribution of this work is the development of methods and metrics for evaluating the quality of complex correspondences between data sources.

1.3 Technical Approach

Initially, a study of published schema and ontology matching and mapping techniques and tools was undertaken. This helped identify the capabilities of state of the art mapping and matching systems and provide general grounding in the field of correspondence detection in

ontologies and database schema.. Schemes for classifying correspondences were also analysed. The objective of this analysis was to discover the types of complex correspondence our approach should be capable of detecting.

Having carried out these studies it was necessary to develop design parameters for a system for detecting complex correspondences. This included analysing several prominent data-sets such as DBpedia [7], YAGO2 [5] and GeoNames[5] to establish which kinds of complex correspondence occur most often. It was necessary to develop a set of gold standard complex correspondences between these sets which could be used to evaluate any complex correspondence detection method and choose a metric for comparing the results of the detection method with this gold standard.

Having identified the most common correspondence patterns, methods for detecting these forms of correspondence were developed and implemented. The initial implementation, Feature Selection 1 (**FS1)**, used an off the shelf feature selection method provided by the Weka [20] machine learning toolkit. A second implementation, Feature Selection 2 (**FS2**), was also created which uses a beta binomial conjugate prior probability distribution to perform feature selection. This distribution was chosen as it can more accurately model the distribution of relationships in linked data. To the author's knowledge this distribution has not been used to perform feature selection previously. These implementations were evaluated against the gold standard described above to establish if the algorithms were producing appropriate results. The best performing of the algorithms was then compared with state of the art complex correspondence detection methods produced by Parundekar et al [15].

1.4 Evaluation Strategy

To evaluate the research in this PhD, the following strategy was used:

- **Estimate the most common complex correspondence patterns in a sample of prominent LOD data-sets.** In this step a sample of classes from two prominent LOD data-sets was selected, and complex correspondences between these classes were created by an expert. Each of these correspondences was classified by hand against the Correspondence Pattern [9] classification scheme.
- **Develop an implementation of the correspondence detection approach.** In this step an extensible implementation of the correspondence detection approach was developed. Initially this implementation used a feature selection method known as

[5] http://www.geonames.org/

Information Gain to perform the detection. This version of the implementation is referred to as FS1. A second feature selection method based on density estimation was later added to the implementation. This version of the implementation is referred to as FS2

- **Initial proof of concept.** An initial proof of concept evaluation was carried out using FS1 to perform complex correspondence detection on two prominent LOD ontologies. This evaluation is described in chapter 6.
- **Further analysis of the effects of matched instance sample size.** Following a successful proof of concept, an evaluation was carried out to more precisely determine the effects of the size of the instance data sample on the performance of FS1. This evaluation is described in chapter 7
- **Comparison of potential attribute selection measures.** This experiment compared results of using the FS1 and FS2 correspondence detection methods to perform complex correspondence detection as the level of missing attribute data increased. The experiment is described in chapter 8.
- **Comparison with state of the art Common Extension Comparison Method.** In this experiment the results of using the FS2 method to detect complex correspondences between the DBpedia and Geonames ontologies was compared with published results of a state of the art complex correspondence detector [15]. This evaluation is described in chapter 9.

1.5 Thesis Outline

The thesis is structured as follows.

1.5.1 Chapter 2: Background

This chapter provides preliminary information on the ontology matching and mapping process. It explains the relationship between ontology matching and mapping, provides a description of the EDOAL [11] format used to exchange ontology alignments, and explains the difference between *simple* and *complex* correspondences. It also provides background on several LOD data sets which are used in examples of complex correspondences throughout this text, and which are used as test cases when evaluating the complex correspondence detection methods described in this thesis.

1.5.2 Chapter 3: State of the Art

This chapter provides a description of existing correspondence detection and ontology mapping approaches, and the methods used to evaluate these approaches. In particular it discusses the Ontology Alignment Evaluation Initiative (OAEI), this is one of the main driving forces in the field of ontology mapping, but is very focused on the development of approaches for detecting one to one correspondences.

Several existing approaches to detecting complex correspondences in database schema and ontologies are then described. These include IBM InfoSphere Architect [21], Tupelo[22], COMA [23], multi-relational data mining [16], common extension comparison [15] and pattern matching [8]. These methods are compared to discover their common elements, as well as their relative strengths and weaknesses.

The state of the art in methods of describing patterns of complex correspondences is also described. Understanding the forms that correspondences may take allows us to determine which forms are most common and which we should focus our approach on. A survey of published complex correspondences is described in this chapter, comparing them with the Correspondence Pattern [9] classification framework. In addition chapter 3 discusses using Correspondence Patterns as a guide for detecting complex correspondences.

1.5.3 Chapter 4: Requirements for a system for finding complex correspondences

This chapter describes the five requirements of our approach, which were developed from the research question and the state of the art analysis. The requirements specify the type of complex correspondence that our approach will focus on detecting and the form of input data that the approach will use. Our approach will detect **Class by Attribute Value (CAV)** correspondences. It must be tolerant to incorrect data and values missing by ommision, and be capable of using a choice of scoring metrics for detecting the correspondences.

In addition, this chapter provides an *objective*, set based definition of what constitutes a "good" Class by Attribute Correspondence, which can be used when evaluating the metrics used in our approach.

1.5.4 Chapter 5: Design and implementation

The design and implementation of our approach to detecting class restriction correspondences is described in this chapter. This approach uses a sample of matched instances and simple one to one correspondences as input, and searches to find complex

correspondences which use the simple correspondences as a base. The approach converts the correspondence detection problem into a feature selection approach. Two feature selection metrics are described: Information Gain which is commonly used in the field of machine learning; and a beta-binomial probability density estimator which we developed to be more suited to the field of semantic web.

Chapter 5 includes a domain model which describes the components of the complex correspondence detection problem and its solution. It provides a use case which describes how complex correspondence detection can be used as part of the overall ontology mapping process. It then describes the process used to convert the correspondence detection task into an attribute detection task, and the metrics which our approach uses to perform attribute selection. Finally it provides a UML description of the implementation of our approach.

1.5.5 Chapters 6 - 9: Detection Method Evaluation

Chapters 6 to 9 describe four evaluations that were carried out to test our approach in different conditions. Chapter 6 describes an initial evaluation to show that it is possible to detect Class by Attribute Value correspondences in the LOD datasets YAGO2 and DBpedia.

Chapter 7 describes a further evaluation which examines the effect of the size of the set of matched instances on the detection approach's performance, and demonstrates that for a sample of 20 instances per correspondence, the performance is comparable to using *all* available matched instances.

Chapter 8 describes an evaluation which compares the use of the beta-binomial metric and the information gain metric in our approach. This evaluation shows that as the level of missing data increases the beta binomial metric shows the least drop in performance.

Chapter 9 describes an evaluation where our approach was compared with a leading restriction class correspondence detection approach [13]. Both approaches were used to detect CAV correspondences between the DBpedia and GeoNames ontologies, and their outputs were compared. This evaluation showed that while both approaches detect the same number of CAV correspondences, in one third of the cases the CAV detected by our approach was more accurate.

1.5.6 Chapter 10: Conclusions

This chapter presents the key findings of the work completed for this thesis and a description of the major and minor contribution. A list of the publications arising from this work is listed, and possible future work is outlined.

2 Background

This chapter gives a brief outline of ontology alignment and correspondence detection. Previously ontology alignment was focused on discovering one to one correspondences, but more recently complex correspondences have been recognised as necessary for some alignments. The goal of this chapter is to give precise definitions of what is meant when the words ontology, *alignment*, *mapping*, *correspondence* and *complex correspondence* are used.

This chapter also discusses the properties of some semantic web knowledge bases and ontologies that are used in the running examples of complex correspondences, and in the evaluations carried out in this thesis.

2.1 Ontologies

In a broad sense, ontologies provide a formal description of knowledge within a domain. Uschold and Jasper [24] state that *An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of the meaning. This includes definitions and an indication of how concepts are interrelated, which collectively impose a structure on the domain and constrain the possible interpretations of terms*. Commonly, ontologies are classed as either lightweight or heavyweight [25]. Lightweight ontologies describe concepts, taxonomies for these concepts, the relationships between the concepts as well as their properties. Heavyweight ontologies additionally include axioms and constraints which clarify the intended meaning of the terms in the ontology.

A further distinction can be made between ontologies described using natural language, and those described using formally defined language such as OWL [26]. Uschold and Gruninger [27] distinguish between four kinds of ontology:

- *highly informal* ontologies – expressed in a natural language
- *semi-informal* ontologies – expressed in a restricted and structured form of natural language
- *semi-formal* ontologies – expressed in an artificial and formally derived language
- *rigorously formal* ontologies, which include meticulously defined terms with formal semantics, theorems and proofs of properties such as soundness and completeness.

The majority of the ontologies in the LOD cloud can be classified as lightweight, semiformal ontologies. They are typically described in RDF and contain a taxonomy of concepts with relations and properties. Section XX provides a detailed description of a selection of prominent LOD ontologies. The techniques for detecting complex correspondences described in this thesis will focus on ontologies of this form.

2.2 Ontology alignment

Using ontologies is the preferred way to achieve interoperability among heterogeneous systems in the semantic web [11]. Ontologies are not always directly compatible with each other and often require alignment if they are to be used together. An alignment consists of a set of mappings or, more generally, correspondences between the entities, e.g. classes, objects and properties of the ontologies [28].

The OISIN framework characterises the ontology alignment process into five phases [17]:

1. *Characterisation phase*: In this phase the ontologies are selected, and analysed for their amenability to be mapped
2. *Matching phase***:** Here a set of candidate matches are identified, either manually or by some automated process.
3. *Mapping phase:* The set of candidate matches is evaluated and if a candidate is accepted is then converted to a *mapping* which may be used to execute tasks such as query rewriting
4. Management phase: This includes the sharing of mapping information with third parties, the integration of mapping into other mapping applications and can include a conflict or consistency check as well as the altering of mappings if basic preconditions have changed.

2.2.1 Ontology Matching

Mappings are discovered using a matching process. At the most basic level, ontology matching can be viewed as a black box process. Taking this viewpoint, to discover the alignment *A'* between ontologies *o* and *o'* the following inputs are used: o and o' themselves, *r* the external resources such as thesauri, *p* a set of parameters such as weights and thresholds, and A, an incomplete input alignment. These inputs and outputs are illustrated in figure 1.

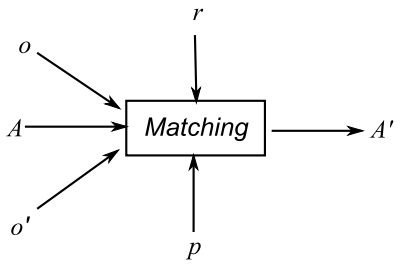


Figure 1: Ontology matching as a black-box process. The primary inputs are the ontologies o and o' which will be matched as well as A, a set of known mathces. The goal is to produce A', a more complete set of mathces. Addtional inputs to the process may be p, a set of parameters for the matcher, and r, which represents backdround information such as theasauri for eample.

There are many different techniques which may be used to automate this process of matching [29], and these can include structural, terminological and semantic based techniques.

2.2.2 Ontology Mapping

Older definitions [29] of what constituted an alignment specified that they are a collection of *mapping elements*, which are 5-tuples of the form $\langle id, e, e', n, R \rangle$, where:

- *id* is a unique identifier of a given mapping element
- *e* and *e'* are the entities (classes, objects or properties)
- *n* is a confidence measure
- *R* is a relation (e.g., equivalence (=); more general ($\sqsubseteq$); disjointness ($\perp$)) holding between the entities *e* and *e'*.

Whichever technique is used to perform alignment, the end result is to produce a set of these 5-tuple mappings, and these tuples are typically exchanged using a format known as Alignment Format.

2.2.3 Ontology Alignment Format

The defacto format for describing and sharing alignments is known as the Alignment Format [30]. Alignment Format is an XML format which was initially devised as a way of comparing the output of different ontology matching tools. It is now also used as input for matching tools that produce further alignments, as well as tools which can transform alignments into axioms and transformations.

This format consists of a header section which describes, among other things, the ontologies being aligned, the arity of the mappings, and if the alignment is injective, surjective and total or partial on both sides. This is then followed by a list of *cell* elements which describe the individual mappings in the alignment. Each cell specifies a URI for the first entity in the mapping, a URI for the second entity, a relation string (with "=" as the default), and a confidence measure between 0, and 1. The cell element may also have an identifier label.

2.2.4 Complex correspondences

Elementary correspondences, as defined in section 2.2.2, describe relationships between single named entities such as classes, objects or properties. If the correspondence involves one or more entities in logical formulation then it is called a complex correspondence [14], [31].

For example, the relationship Film $\sqsubseteq$ Work stating that work is a more general class than film, is a simple correspondence, but the relationship $\forall x$, Short Film(x) = Film(x) $\wedge$ duration(x, y) $\wedge$ y < 60, stating that short films are equivalent to films with a duration of less than sixty, is a complex correspondence.

In practice, complex correspondences can often be seen to conform to certain patterns, and these patterns are discussed in section 3.3.

2.3 Semantic web ontologies

In this thesis, semantic web ontologies are used in the running examples of complex correspondences that occur in reality. In addition, these are used in the evaluation of complex correspondence detection methods developed in the thesis. In this section DBPEDIA [7], YAGO2 [5], GEONAMES[6], FRIEND OF A FRIEND [32], VCARD[7], and THE SUGGESTED UPPER MERGED ONTOLOGY (SUMO) [33] are described.

2.3.1 DBPEDIA

DBPEDIA [7] is a multi-domain knowledge-base that is populated automatically by extracting structured information from Wikipedia[8]. This automated process is used as Wikipedia is continually being updated and expanded, and so this allows DBPEDIA to

[6] www.GEONAMES.org/ontology

[7] http://www.w3.org/TR/vcard-rdf

[8] http://wikipedia.org

continually update and expand with much less effort than for a manually created knowledge-base.

Each article in Wikipedia creates an instance in DBPEDIA with the instance name being derived from the English language article name. The properties of this instance are taken from the article and include a label based on the article title, an abstract based on the first 500 words in the article, and a classification based on the SKOS[9] category of the article. More specific properties about the instance are extracted from the info-box of the article. Info-boxes display an article's most relevant facts as a table of attribute-value pairs on the top right-hand side of the Wikipedia page. Figure 2 illustrates the info box for the Wikipedia page on the Westwood Mall Bus Terminal, which is a "station" type info-box. This info-box lists the address, coordinates, lines, structure, number of platforms, parking availability, and who owns the terminal.

Figure 2: Wikipedia info-box for the Westwood Mall Bus Terminal

DBPEDIA has an underlying information model based on triples. A partial list of the triples describing the Westwood Mall Bus Terminal, are shown in table 1.

[9] http://www.w3.org/2004/02/skos/

Table 1. A partial list of the triples describing the Westwood Mall Bus Terminal.

<dbedia:Westwood_Mall_Bus_Terminal>	<rdf:type>	<owl:Thing>
<dbedia:Westwood_Mall_Bus_Terminal>	<rdf:type>	<dbpedia-owl:Place>
<dbedia:Westwood_Mall_Bus_Terminal>	<rdfs:label>	"Westwood Mall Bus Terminal"@en
<dbedia:Westwood_Mall_Bus_Terminal>	<dbpedia-owl: parkingInformation>	"no"@en
<dbedia:Westwood_Mall_Bus_Terminal>	<dbpedia-owl:owner>	<dbpedia: Mississauga>
<dbedia:Westwood_Mall_Bus_Terminal>	<rdf:type>	<owl:Thing>
<dbedia:Westwood_Mall_Bus_Terminal>	<rdf:type>	<dbpedia-owl:Place>
<dbedia:Westwood_Mall_Bus_Terminal>	<rdfs:label>	"Westwood Mall Bus Terminal"@en
<dbedia:Westwood_Mall_Bus_Terminal>	<dbpedia-owl: parkingInformation>	"no"@en
<dbedia:Westwood_Mall_Bus_Terminal>	<dbpedia-owl:owner>	<dbpedia: Mississauga>

Several ontologies are used in parallel to classify the instances in DBPEDIA. All instances are classified using the DBPEDIA ontology, but for many instances additional classification information is available using classes from YAGO2 and SKOS. The DBPEDIA ontology is relatively shallow with 359 classes which form a subsumption hierarchy that is a directed acyclic graph. The classes were selected from the most commonly occurring types of info-box. There are 1,775 different properties, some of which have well defined domain and ranges, but many do not. New versions (whole dot iterations) of DBPEDIA are released approximately every 1-2 years, but smaller iterations are continually released which include new information that is added to Wikipedia. There is also a live version of the knowledge base which is automatically added to each time a new piece of information is added to Wikipedia.

2.3.2 YAGO2

YAGO2 [5], [34] is another ontology that uses Wikipedia as a source of information. While DBPEDIA aims to be as complete as possible, YAGO2 concentrates on accuracy. YAGO2 has a deep class hierarchy based on WordNet [6], and information taken from the GEONAMES ontology. The underlying knowledge model used by YAGO2 consists of a set of 4-tuple of information with an id element, subject, predicate and object. This means that YAGO2 allows reification or, in other words, statements about statements. For example, in

YAGO2 it is possible to state that Elvis won a Grammy Award, and additionally state that this event took place in 1967 in a very simple manner with the following two 4-tuples:

```
#1: Elvis hasWonPrize Grammy Award
#2: #1 inYear 1967
```

The same information can be expressed using triples, but it requires a more convoluted process. YAGO2 also has a greater focus on temporal information than DBpedia. The use of 4-tuple representation with an id element instead of anonymous triples, allows YAGO2 to easily express temporal properties upon facts. Creating a correspondence between the fact that Elvis won a Grammy in 1967 as expressed above and a triple representation would require some complex process where blank nodes were created.

YAGO2 has an extremely deep hierarchy, containing 224,391 classes, largely taken from WordNet [6]. This is many more than the 359 classes in DBPEDIA. Many of these classes are very specific. For example the class *People Murdered in Hawaii* is included.

YAGO2 has an iteration cycle which can last several years and therefore correspondences between YAGO2 and more rapidly changing ontologies could become incompatible and need to be repaired, or re-discovered.

2.3.3 GeoNames

The GEONAMES ontology[10], which describes geographic features, is available as a standalone ontology in RDF format in addition to being included as part of YAGO2. This ontology is interesting as it has an unusual structure. Almost all instance data in the ontology have the type *geo:Feature*. There are no classes in the ontology which would typically be associated with the field of geography. Instead two attributes, *geo:Class* and *geo:Code*, provide broad and narrow classification of the individual instances of feature. The classes in the range of these attributes do not have particularly meaningful names – "A" is used for administrative regions, and "A.ADM1" is a country. This meaning of the classes is not specified in the ontology and must be deciphered from a separate table with English language descriptions for each code.

[10] www.geonames.org/ontology

2.3.4 FOAF and the VCard Ontology

Both the Friend of a Friend (FOAF) [32] and VCard[11] ontologies provide a method for describing contact details and connections between people. The schema for these ontologies are small, with well specified domains and ranges for properties. FOAF contains 6 classes, and 12 properties, VCard has 77 classes and 67 properties. These ontologies are not subject to regular change, though VCard is still only a working draft. Unlike the previous ontologies, there is no real central repository of instance data for these ontologies. They are used to provide structure for exchanging personal data. It should be noted that while the structure of these ontologies is well defined, in practice instance data quite often does not conform to the schema [19].

2.3.5 SUMO

The Suggested Upper Merged Ontology (SUMO) is a broadly scoped ontology which can be used as a common point to integrate information from different ontologies. It has been used [33] as a "starter document" in the IEEE Standard Ontology Effort. SUMO is defined in Standard Upper Ontology Knowledge Interchange Format (SUO-KIF), a Lisp-like language which allows full propositional logic statements. Facts may be stated as n-ary tuples this means that when aligning with ontologies specified in the more common format of RDF which uses triples, it may be necessary to use complex correspondences to describe how these n-ary tuples can be converted into triples.

2.4 Summary

This chapter discussed the relationship between alignments and correspondences. Older definitions of alignments describe them as a collection of 5-tuples called mappings, which describe a relationship between named entities in two ontologies. A simple correspondence is one where elements in a mapping are classes, objects or singular relations. Complex correspondences extend mappings and allow relationships to be expressed between logically constructed entities in the ontologies.

There are many knowledge bases available on the semantic web with overlapping domains. Some of these knowledge bases are even created from the same sources. Still despite their overlapping domains and common sources, they do not model information in the same way

[11] http://www.w3.org/TR/vcard-rdf

and may require complex correspondences when integrating between them. The next chapter presents the state of the art in complex correspondence detection.

3 State of the art in Complex Correspondence detection

The objective of this chapter is to summarise the current direction of research in the field of correspondence detection, with particular focus on complex correspondence detection. Section 3.1 will show that the current focus is on developing methods which detect simple one to one correspondences between named classes or relations in ontologies. A large portion of development has been focused on scoring well in standardised tests, but section 3.1 will shows these standardised tests do not cover all possible correspondences and that complex correspondences should also be considered.

Section 3.2 gives a detailed description of the approaches which can be used for detecting and refining complex correspondences. There are a limited number of complex correspondence detection approaches currently available. As a result this section presents a result of analysing: two approaches for finding correspondences in database schemas and three approaches for detecting complex correspondences in ontologies. These are compared to see what commonalities they share and how they differ. The goal is to show that certain steps in detecting complex correspondences are fundamental, but other elements of detecting complex correspondences can be undertaken in a number of different ways.

Correspondence patterns are discussed in section 3.3. These patterns provide a classification system for describing the different forms of complex correspondences each of the approaches discussed in section 3.2 address. Examples of different correspondence patterns are provided in section 3.2.2. Section 3.3.3 discusses the forms of correspondences that have been found to occur in the semantic web. Section 3.3.4 introduces the EDOAL format which is used to describe and share complex correspondences, and section 3.3.5 outlines a correspondence detection approach which uses correspondence patterns to search for complex correspondences.

One element of detecting complex correspondences seen in many of the approaches described in section 3.2 is that they must have some form of search strategy, which includes having an idea of what constitutes a good solution and then moving towards that solution in a stepwise manner. The different solutions embody varying ideas of what constitutes a good solution. This affects both their search strategy, and the types of solution that they produce. In section 3.4, a comparison of the approaches is provided, along with a taxonomy which outlines their similarities and differences. How Correspondence Patterns can be used to inform better search strategies is discussed in section 3.4.2, and how using a classification

scheme can be used when choosing an appropriate detection method is analysed. This analysis will show that the current approaches have poor support for some of the most commonly occurring patterns of complex correspondences.

3.1 Correspondence Detection Methods

One of the primary driving forces in the field of correspondence detection is the annual Ontology Alignment Evaluation Initiative (OAEI) [12], [35]–[38]. This annual event consists of a public evaluation (as part of the Ontology Matching (OM) workshop at the International Semantic Web Conference (ISWC)) of the state of the art ontology alignment tools. Each year a set of evaluation ontologies are published and alignment tool developers submit tools which are designed to align these ontologies automatically. The tools are run on a common set of hardware, and their results are compared against a set of gold standard reference alignments. The alignment tools are assessed on their precision and recall against the correspondences contained in each of the gold standard alignments as well as on their running time.

This competition is a large driving force in the field of ontology alignment, and the specific tests performed as part of this competition have a large influence on which capabilities of ontology alignment tools developers focus upon. These tests focus almost exclusively on **one to one equivalence relationships** between named classes and relationships, and as such this means that tool developers and researchers typically focus on detecting correspondences of this kind.

Each year the competition is broken into several tracks. In 2013 these were:

Benchmark: A systematic benchmark series of tests, designed to identify the areas in which each alignment algorithm is strong and weak.

Anatomy: A real world test case based on matching the Adult Mouse Anatomy and the NCI Thesaurus describing the human anatomy.

Conference: This track is based on finding alignments within a collection of ontologies describing the domain of organising conferences. This track is notable in that it includes complex correspondences.

Multifarm: This dataset is composed of a subset of the Conference dataset translated in eight different languages (Chinese, Czech, Dutch, French, German, Portuguese, Russian, and Spanish) and the corresponding alignments between these ontologies.

Library: This track is a real-world task to match the Standard Thesaurus for Economics and the Thesaurus for the Social Sciences.

Interactive matching evaluation: This track covers matching tools that require user interaction.

Large Biomedical Ontologies: This track consists of finding alignments between the Foundational Model of Anatomy (FMA), SNOMED CT, and the National Cancer Institute Thesaurus (NCI).

Instance Matching: The instance data matching track evaluates tools able to identify similar instances among different RDF and OWL datasets.

When submitting an alignment tool for evaluation, teams are free to select which tracks they take part in. The tracks are subject to change each year, but the Benchmark, Conference, and Anatomy tracks have been a consistent feature since 2007. Instance Matching has featured since 2009, and Library was featured in 2007 to 2009, and then again in 2012 and 2013. Figure 3 shows the levels of participant tools in these tracks over the years 2007 to 2012.

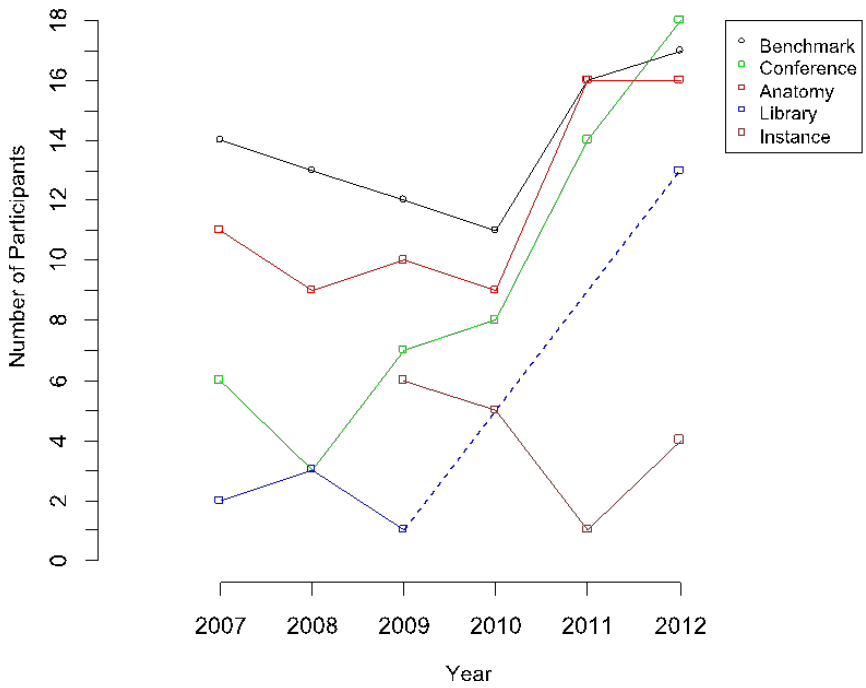


Figure 3: Participation levels for five representative OAEI tracks.

Historically, the Benchmark track was the most commonly entered track. In 2012, 17 out of 21 tools were evaluated in this track [12], and in 2013, 20 out of 23 tools participated [39]. The correspondences in the gold standard alignments for this track are all one to one

equivalence relationships between named classes and properties, and no complex correspondences are considered in the evaluation. The authors of the 2012 evaluation report [12], note that there was no significant improvement in this track over the previous year. Subsequently in 2013 there was a drop in performance of the tools in this track. The organisers of the competition theorise [39] that this was because tool developers have stopped focusing on this track to concentrate on the tracks which are based on real world problems – such as Library and Conference.

The Library track measures the alignment tools' ability to process thesauri. This section was new for 2012, but a similar track ran from 2007-2009. The thesauri used in this section are the SWT Thesaurus for Economics and the Thesaurus for Social Sciences. Both these thesauri are described in SKOS, and have been transformed to OWL for the evaluation. The reference alignment used for the evaluation was generated independently by the KoMoHe [40] project. This tool produces correspondences of the form *skos:exactMatch*, *skos:broaderMatch* and *skos:narrowerMatch*, but only *skos:exactMatch* correspondences are used for the OAEI evaluation. This shows that the evaluation track is very focused on one to one equivalences, even in a dataset where a tool has been demonstrated to be capable of finding other forms of relationships.

The Conference track is interesting as while the automated evaluation of this section only includes one to one matches, as of 2013, complex correspondences are considered for manual evaluation. Unfortunately in 2013, no tool that entered in this track actually produced any complex correspondences. As the reference alignments used to score this track are all one to one equivalences there is little incentive to detect complex correspondences which are evaluated separately in an ad-hoc manner. Previously in 2009, a pattern matching based tool [8] did demonstrate the ability to detect complex correspondences in these data-sets – though this tool was not put forward for evaluation at the OAEI.

In summary the OAEI assessments focus heavily on one to one equivalence correspondences, which in turn has led to ontology alignment tool developers and researchers focusing their efforts on detecting correspondences of this form. Performance levels in the Benchmark track have not shown significant improvement in recent years. It could be argued that diminishing returns means that these levels are unlikely to show significant improvement relative to the amount of effort applied to them. Complex correspondence detection remains a relatively un-explored avenue of research. The OAEI has begun to include assessment of

complex correspondences, but as yet none of the current entrants have attempted to demonstrate the capability to detect complex correspondences.

3.2 Approaches to detecting complex correspondences

As illustrated in section 3.1 most approaches developed for the Ontology Matching (OM) workshop focus on discovering one to one correspondences between named ontology entities, however there have been several approaches that demonstrated separately that they can address the issue of complex correspondences.

There are two complex correspondence detection methods which have been presented at OM that are particularly relevant to the work presented in this thesis. These are the Enhancement Engine of the COMA system [41] which is described in more detail in section 3.2.3 and the Pattern Matching approach [8] which is described in more detail in section 3.2.6.

The following sections describe tools which can be used to detect and refine complex correspondences. As there are so few directly comparable approaches available in the specific field of ontology matching, two schema matching tools are included. These are included as schema matching is considered to be closely related to ontology matching [29]. There are many schema matching tools which could be considered, however these Infosphere Data Architect [21] and TUPELO [22] were selected in particular as Infosphere is a leading commercial product and TUPELO is an advanced piece of work capable of automatically finding complex relationships in schemata.

These tools described in this section are compared to determine which common elements the detection process contains, and in what ways they differ. By noting the differences, it should be possible to provide insight into where possible improvements could be made.

3.2.1 Clio and IMB Infosphere Data Architect framework

IBM Infosphere Data Architect is a suite of tools which assists with discovering, modelling, relating, standardizing and integrating diverse and distributed data assets. Included in this suite are tools based on research developed for the Clio [42] project. Clio was developed to facilitate mapping between database or XML schemas and creating rules which can be used to exchange data from one site to another.

Database and XML schema are similar to ontologies used in the Semantic Web in that they all provide a vocabulary of terms that describe a domain of interest, and constrain the meaning of terms used in that vocabulary. In theory, ontologies provide additional semantics

which should allow the meaning of their contents to be inferred, however this is not always the case. In a survey of schema based mapping approaches [29] the authors note that *"In real-world applications, schemas/ontologies usually have both well defined and obscure labels (terms), and contexts [in which] they occur, therefore, solutions from both problems would be mutually beneficial"*. Therefore Infosphere Data Architect is an important piece of technology in the state of the art, as it represents a leading production level toolkit which is widely used in commercial settings to map between schemas.

Using InfoSphere Data Architect to map between schemas is essentially a four step process:

1. Discover relationships;
2. Map relationships;
3. Build expressions;
4. Generate scripts.

In the first step, relationships are detected automatically by the system. These relationships are one-to-one matches between elements in the schema. In the second step, these relationships are displayed visually and the user can add to or remove from this set of mappings. In the third step, the user has the option to add expressions, such as filter or join operations to the mappings. The mappings combined with the additional expressions are then used to automatically generate scripts which can transfer data from one schema representation to the other. Figure 4 illustrates the process.

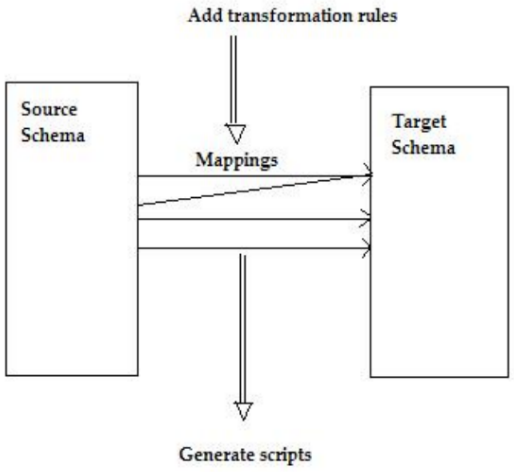


Figure 4. The process for generating executable code for exchanging data between databases using InfoSphere Data Architect [21]**.**

The expressions which can be added to a mapping are composed from elements chosen from a pre-defined set of operations. These operations include: filters – which build WHERE clauses – sort, join and transformation functions. The transformation functions include:

- Date and Time functions;
- String operations such as concatenation;
- Cast;
- Aggregate functions on columns such as MAX and SUM;
- Math;
- The predicates and operators: *, /, +, -, \ <, >, =, %, BETWEEN, NOT, LIKE, NULL, IN and EXISTS.

For example if the system were used to model the relationship between two university student databases, then the system might find that the first name field of the student information must be converted to upper case in one of the databases. Figure 5 illustrates an example of the expression builder.

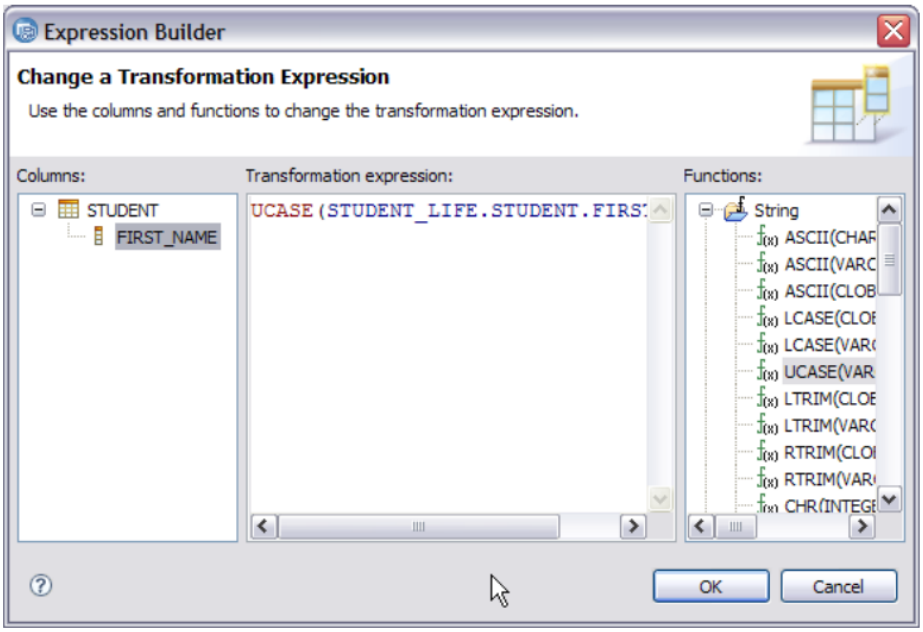


Figure 5. Adding a transformation expression to the FIRST_NAME field of the database. [21]

It is important to note that while the system is capable of finding one to one relationships between schema elements automatically; creating the expressions necessary to refine those relationships into a state suitable for generating executable scripts is a manual process.

In summary, this is a mature system which is used in commercial environments. It provides the ability to automatically detect one to one matches between schema elements, and provides a visual environment for viewing and refining these matches into confirmed mappings. This system supports the creation of complex correspondences in the form of transformation expressions, but these correspondences must be specified manually by a user.

3.2.2 Tupelo: A General Framework for Effective Solutions to the Data Mapping Problem

The Tupelo framework [22] is a more generalised analysis of the data mapping problem that takes into account the need to find transformation expressions. This framework provides a calculus for describing the common recurring problems that are encountered when mapping data and lists the solutions that can be used. An important aspect of Tupelo is that it emphasises treating mapping discovery as an *example-driven* search in a space of transformations. This approach allows it to generate queries encompassing a wide range of structural and semantic heterogeneities encountered in relational data mapping.

Tupelo combines a number of principles which create a useful perspective on the problem of detecting complex correspondences. The first of these principles is that examples of common instance data can be analysed in order to discover what transformation process is needed to convert from one representation of the instance to another. In Tupelo this is referred to as the *Rosetta Stone principle*. This principle states that there are critical instances in the databases under comparison which can be used to understand the relationship between the schemas of the databases. If the common instances which exist in both databases can be found and it can be understood exactly how these particular instances are related to each other, then this understanding can be used to generate mappings which generalise to the schema and the other instances in the database.

The second principle used in the Tupelo framework is that the set of possible representations of the data can be viewed as a vector space, which may be combined with an appropriate choice of metric to form a metric space. That is, different representations of the data can be seen as vectors and the distance between these vectors can be measured – which means that the space can be systematically searched. In practice the search space only considers the different possible representations of the Rosetta Stone instances, and not the database as a whole. This makes the search tractable.

In [22], the authors of Tupelo present a method for vectorising databases, and a selection of different metrics that may be applied to this vectorisation. They then demonstrate several search strategies based on variants of tree search. This formalism is interesting as it shows that different components may be combined to form a method for detecting complex relationships between schemas.

Two different algorithms are presented which can be used to search the space of representations of the databases. Both of these are based on tree searches.

- **Iterative Deepening A* (IDA):** A depth-bounded, depth-first search of the space where the depth is iteratively increased until a path from one representation to the other has been found.
- **Recursive Best-First Search (RBFS):** A localised, recursive best-first exploration of the state space, keeping track of a locally optimal set of transformations to get to a given state and backtracking if a more efficient path is found.

The authors present three different types of metrics which may be used to compare representations of databases. These metrics are used in the search algorithm to estimate how close a given transformation will bring the current database state to the goal state. These classes of metrics are:

Set based: These metrics count the overlap of values in the database states;

String based: Here the vectortisation of the database is viewed as a string. This allows the use of metrics such as the Levenshtein distance, which measures the number of single character insertions, deletions and substitutions required to transform one string to the other;

Euclidian distance: Here the database state is viewed as a document vector over a set of terms, and a standard Euclidian norm is used to compare them.

In the evaluation of the Tupelo framework, the authors demonstrate that it is capable of detecting complex correspondences in the Inventory and Real Estate II data sets from the Illinois Semantic Integration Archive[12]. These correspondences include scaling numerical values, converting nouns to ID numbers, and concatenation of string values.

In summary, the Tupelo framework is interesting in that it goes beyond the capability of IBM InfoSphere Data Architect. While InfoSphere could only detect one to one matches, Tupelo is capable of automatically detecting complex correspondences. Tupelo is also

[12] http://pages.cs.wisc.edu/~anhai/wisc-si-archive/

notable in the way it breaks the problem into several parts and allows for different search strategies and metrics to be combined. One issue with Tupelo is that in order for the search to be effective, a suitable set of Rosetta Stone instances are needed.

3.2.3 COMA Match System Enhancement Engine framework

The COMA Match system was first designed as a framework for combining schema matching approaches [43] and was subsequently extended to match ontologies [23]. The COMA toolset has seen continual development, and as of 2011 includes the Enhancement Engine component [41] which can take the one to one mappings produced by the schema matching approaches and use these to create complex correspondences.

The Enhancement Engine is capable of detecting correspondences which relate multiple entities together, and which use transformation operations including (but not limited to) string functions, numerical functions, date/time functions, and logical operations. COMA operates in an interactive manner and allows a user to refine the correspondences during this process. COMA uses a combination of matched instance data, analysis of the structure of the ontologies and analysis of the element names.

The search strategy it uses is to first examine possible correspondences involving sets of elements which are close together in the ontology graphs, to determine if any transformation correspondences can be found. It then uses external linguistic oracles to discover additional correspondences using the element names.

Like Tupelo, this approach allows for the selection of different components to handle different parts of the process. There is also an option for which linguistic oracles can be used.

3.2.4 Use of multi-relation data mining techniques to find complex correspondences in ontologies

The previous approaches presented are intended primarily for use on relational database and XML schema. Qin et.al. [16] proposed the use of multi relational data mining techniques to detect complex correspondences between ontologies. As was the case with InfoSphere Data Architect, the process begins by generating one to one matches between the classes and between relations in the ontologies. Object reconciliation is then used to find common instances in the ontologies. In a manner this is similar to the Rosetta Stone principle used in the Tupelo framework, but whereas in Tupelo a minimal amount of critical shared instances was sought, this system seeks to find as many shared instances as possible. In addition, there is a feedback loop where information about the shared instances discovered at this stage are

used to help find more matches between the classes and relations, which in turn are used to find more shared instances.

Figure 6 illustrates the steps used by Qin et al to convert the detection of complex correspondences to a multi relation data mining problem and generate a set of rules describing the complex correspondences. The first step is to generate one to one matches in a manner similar to a traditional ontology matcher. Then object reconciliation is used to find the common instances in the ontologies.

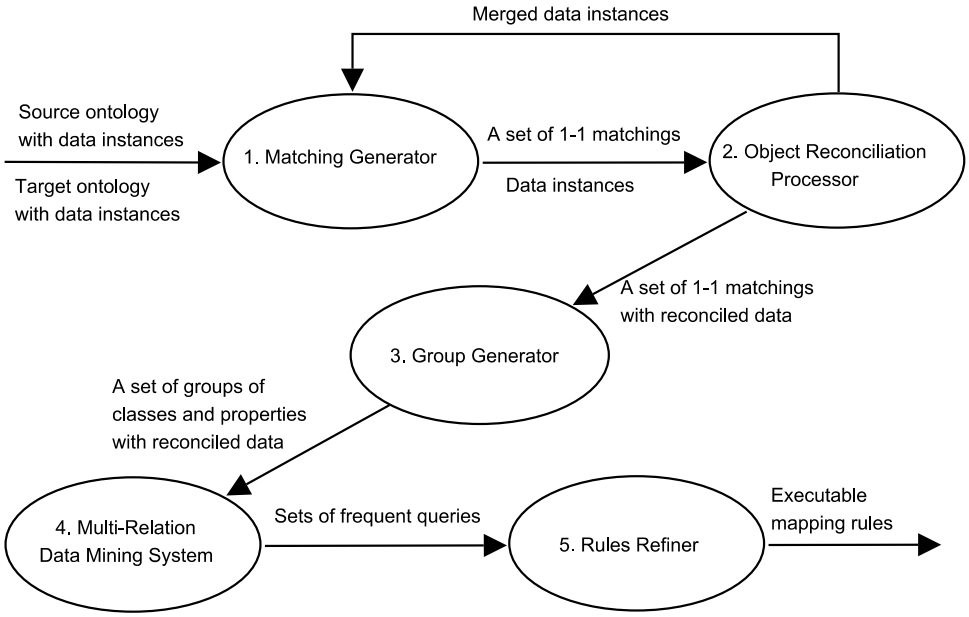


Figure 6: The steps used by Qin et al [16] **to convert the detection of complex correspondences to a multi relation data mining problem and generate a set of rules describing the complex correspondences.**

Once a sufficient number of matching candidates and reconciled instances have been detected, the matching candidates are grouped so that the classes and relations in a given group are independent of the classes and relations in any other group. Each of these groups is then sent to a multi-relational data mining system. As the matches in each group are independent of the other groups, each group can be mined independently. This data mining step searches for rules which are well supported by the data in a group, which are referred to as *frequent queries*. The final stage of the process is to accept or reject the discovered rules based on a threshold check on their support, and then generate complex mappings in the Semantic Web Rule Language (SWRL).

As stated previously, this process has many parts that are similar to the frameworks described earlier that were designed for use on database schemas which do not have support for the semantic information contained in an ontology. The only point where this semantic

information is used in the process is in finding the one to one matches. The subsequent task of finding the *complex* correspondences uses the one to one matches, not the ontologies directly. Finding initial one to one matches, whether between classes, relations or instances, is a common first step in each of these processes. As discussed in section 3.1 there has been much work in developing ontology matchers which produce one to one matches. These methods could potentially be used in place of the matching step used in this approach. Therefore to avoid duplicating work, it would be prudent to focus on the steps that take these one to one mappings as input to produce complex correspondences.

3.2.5 Common extension comparison method for complex correspondence detection

Parundekar et all. proposed a method for detecting complex correspondences which uses common extension comparison methods [15], [44]. These methods are capable of detecting complex correspondences between classes which may be named classes or derived restriction classes. This method does not analyse the structure or syntax of the ontologies to any great extent, and relies instead on statistical analysis of instance data that are known to exist in both ontologies. This method uses a tree search combined with a set of heuristics to search through the space of all possible restriction classes that could be defined in both ontologies. Each pair of restriction classes, one taken from each ontology, is compared to see how many instances they share.

Finding the number of shared instances is done by projecting the image of the restriction class from one ontology to another. If r_1 is a restriction class in ontology O_1 and r_2 is a restriction class in O_2, and a set, M, of matched instances in O_1 and O_2 is available. To find the image $l(r_1)$ of r_1 in O_2, the following process is used. First find all the instances in r_1 that appear in M and then $l(r_1)$ is made up of the instances that match to the ones in r_1. The process is illustrated in figure 7. This can be used to determine not only equivalence but also subclass and superclass relationships. As can be seen in figure 9, every instance in r_2 appears to be contained in r_1, but r_1 contains instances that are not in r_2. This would suggest that r_1 is a superclass of r_2. There is one problem with this approach though – it is reliant on having a representative set of matched instances in the set M. It could be possible for example that r_2 may have instances which are not in r_1 but which are not known as they are not in M.

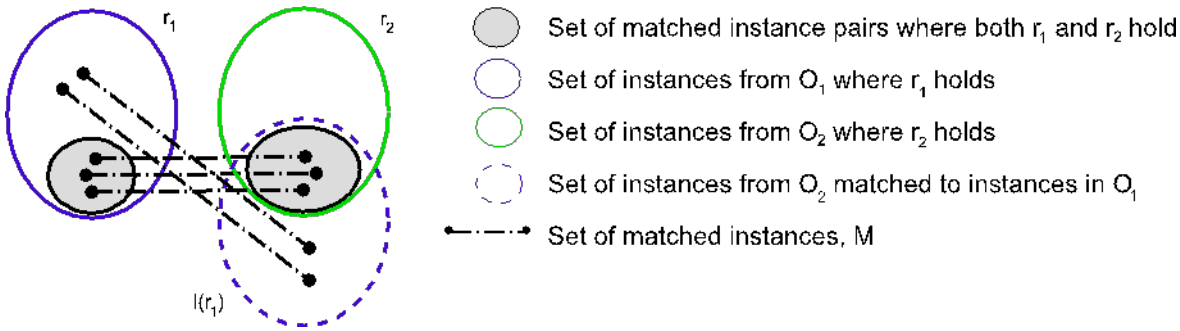


Figure 7: The level of support in the instance data for a restriction correspondence: r_1 is a restriction class in ontology O_1 and r_2 is a restriction correspondence in ontology O_2 [13]. $I(r_1)$ is the set of instances of O_2 which match with instances in r_1.

The search space of all restriction classes formed by combinations of attributes and their values is combinatorially large and a tree search method is used to navigate this space; therefore, this approach will be referred to as the **Heuristic Tree Search** (HTS) **approach** in this thesis. Each node of the tree consists of a pair of logically constructed restriction classes[13], one from each of the ontologies. These pairs are treated as candidate correspondences. The children of each node consist of the more specific restriction classes that can be created by adding an extra attribute-value constraint on one of the restriction classes in the candidate pair.

The tree is seeded by computing all possible candidate correspondences that use a single attribute-value pair from each ontology, and then explored using a depth first search. At each level attribute-value constraints are added to one of the restriction classes to evaluate all specialisations. The change in the number of instances that support the new constraint in a candidate correspondence is used to score its validity. If a new constraint causes no change in the number of supporting instances, or two few instances remain then the new candidate correspondence is rejected and no further specialisations of the correspondence are explored. The goal of this exploration is to find the most specific complex correspondences that have support from the instance data.

Figure 8 shows part of this process applied to finding complex correspondences between LinkedGeoData (namespace: lgd) and DBpedia (namespace: dbpedia). Initial candidates include a correspondence between the classes *lgd:node* and *dbpedia:BodyOfWater* and a correspondence between *lgd:node* and *dbpedia:PopulatedPlace*. One candidate

[13] http://www.w3.org/TR/owl-ref/#Restriction

correspondence which extends *lgd:node* and *dbpedia:BodyOfWater* is to restrict the class *dbpedia:BodyOfWater* to only those instances that also have the attribute *dbpedia:Place#type* set to *dbpedia:City*. This restriction produces a null set of instances so it is not considered valid. A candidate correspondence which extends *lgd:node* and *dbpedia:PopulatedPlace*, is to restrict the class *dbpedia:PopulatedPlace* to the instances that also have attribute *rdf:type* set to *owl:Thing*. This restriction produces no change in the set of supporting instances, so it is also rejected.

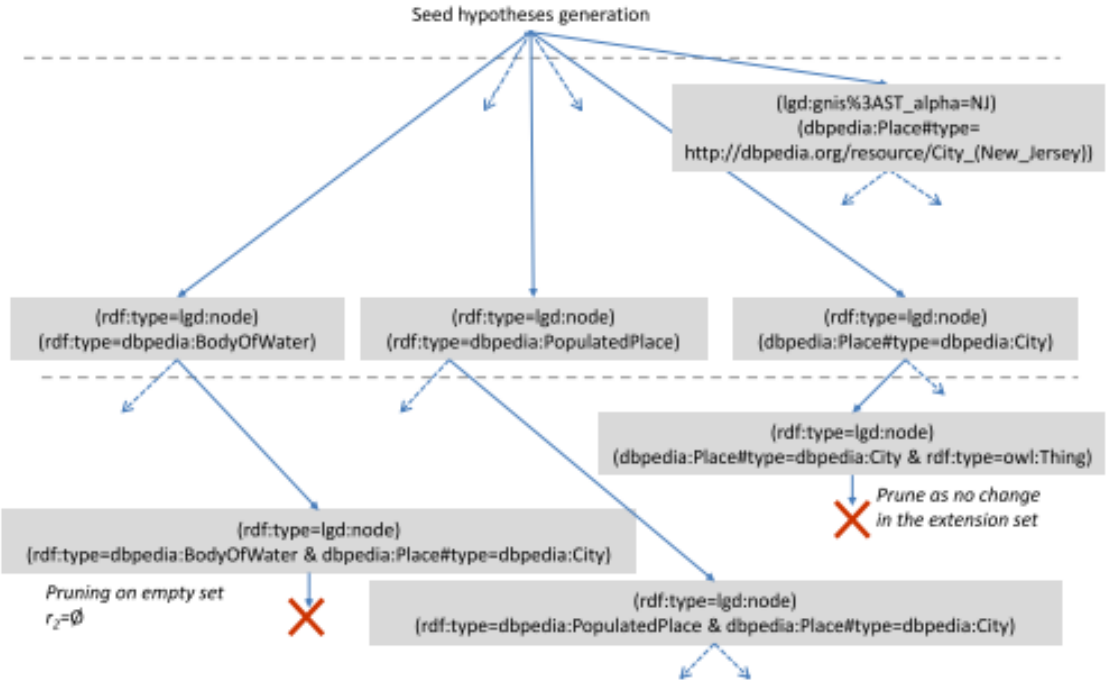


Figure 8: Exploring and pruning the space of correspondences [15].

In many ways the Tree Search approach is similar to the decision tree creation algorithms found in the field of machine learning [45] as it uses a process of recursively partitioning the data. However, while more advanced tree learning approaches use sophisticated metrics to construct a tree, the Heuristic Tree Search approach uses simple heuristics. This suggests that it may be fruitful to apply some more advanced metric to the process of finding complex correspondences. For example, the entropy measure based attribute selection used in the C4.5 decision tree construction algorithm [46].

In summary, this detection method is capable of finding complex correspondences between logical restriction classes in ontologies, and these restriction classes could potentially be very complicated. This method does not use any structural or lexical analysis of the ontology T-

Box, so it is not tied by any preconceptions such as naming conventions. The draw-back to this approach is that it needs a set of matched instance data as input, and these matched instances need to be representative of the dataset as a whole.

3.2.6 Other approaches

The CaRLA system [13] is capable of learning transformation rules between string values, though it is limited in that it cannot combine entities together in these rules, but simply manipulate the values of single properties. For example it can learn the rule name1="Jean Van Damne (actor)" → name2 ="Jean van Damne" but it cannot learn name="Jean Van Damne (actor)" → (first_name="Jean" and family_name= "van Damne"). This rule learning could be part of a complex correspondence detection method, but only addresses a narrow case.

The Silk Link Discovery Framework [47] was presented at OM in 2011. This framework is designed to find linkage rules which can be used to match objects in the semantic web. Linkage rules are used to measure how similar two given objects are. Although not exactly the same as discovering complex correspondences it shares some similarities, as linkage rules contain a transformation step before the objects are compared. Silk Link uses genetic programming to learn these transformation rules.

3.3 Pattern based complex correspondence detection

Unlike the approaches to detecting complex correspondences discussed in earlier sections, which all relied to some extent on examining common instance data in the ontologies, other approaches instead focus on analysing the terminological components of the ontologies. These approaches use Ontology Design Patterns[14] (ODPs) to find complex correspondences that match preconceived patterns. ODPs provide a method for describing modelling information in ontologies. It has been proposed [48] that if the particular ODPs used to design a given ontology are known, the ontology can be transformed into the form which would have resulted if alternative ODPs were used in the design process. Potentially, if an ontology is transformed to use the same ODPs as the ontology it is being matched to, this could make the correspondences between the ontologies less complex.

[14] http://ontologydesignpatterns.org

For example, one common anti-pattern [49] in ontology design is the *Undefined inverse relationships* pattern. Here, two independent relationships exist in an ontology which should be expressed as a single relationship and it's inverse. If an ontology of this form were being mapped to a more correctly specified ontology, it would be necessary to discover and specify the mapping for each of the relations separately. For example if ontology A contains the relationships *a:contains* and *a:containedIn*, and ontology B contains the relationship *b:in*, specifying that *a:contains* is the inverse of *a:containedIn,* then this would mean that only the mapping *a:containedIn = b:in* needs to be found, and *a:contains = b:in*$^{-1}$ will follow as a logical consequence.

Conceptually this approach of fixing the ontologies directly before mapping them appears more satisfying than the solution of using complex correspondences to cope with the underspecification that becomes apparent when comparing the ontology to a more correctly modelled one. However, to the author's knowledge this approach has not been demonstrated to work on real-world ontologies associated with the web of knowledge. The approach is thus theoretical and to the best of the author of this thesis's knowledge does not appear to have been applied yet. In any case two ontologies could be correctly modelled in terms of published best practice, and yet still have heterogeneities that will require complex correspondences.

3.3.1 Correspondence Patterns

Correspondence Patterns are a specific type of ODP designed to describe common forms of complex correspondences that can occur. Correspondence patterns provide two layered templates which model forms of complex correspondences that can occur between ontologies. These templates contain an abstract layer which describes the problem being solved using natural language terms. The second layer is known as a *grounding,* and describes a parameterised template solution to this problem expressed in a formal language – typically EDOAL (see section 3.3.4). By specifying values for the parameters in a grounding it is possible to create an instantiated grounding which can be used by a mediator to perform a task such as query rewriting or instance translation. Correspondence Patterns serve a similar function to that of Design Patterns [10] used in software engineering. Initially Correspondence Patterns were proposed as a method to assist human users when aligning ontologies [31] in the same way Design Patterns are used when creating software. However,

it has been subsequently shown that it is possible to use correspondence patterns to automatically detect complex correspondences [8].

3.3.2 Correspondence Pattern Examples

Thirty six Correspondence Pattern examples, are given in by Scharffe in his PhD dissertation, Correspondence Patterns Representation [9]. These patterns are arranged into a hierarchy which is shown in figure 9. Thirteen Correspondence Patterns are available on the Ontology Design Patterns web portal.[15] Among these patterns are *Class by Attribute Value*, *Class by Attribute Type*, *Class by Path Attribute Value*, and *Attribute Value Transformations*, which are described in detail in the following sections as these patterns occur frequently in real world datasets (see section 3.3.3), or feature in comparable state of the art complex correspondence detection approaches (see section 3.4).

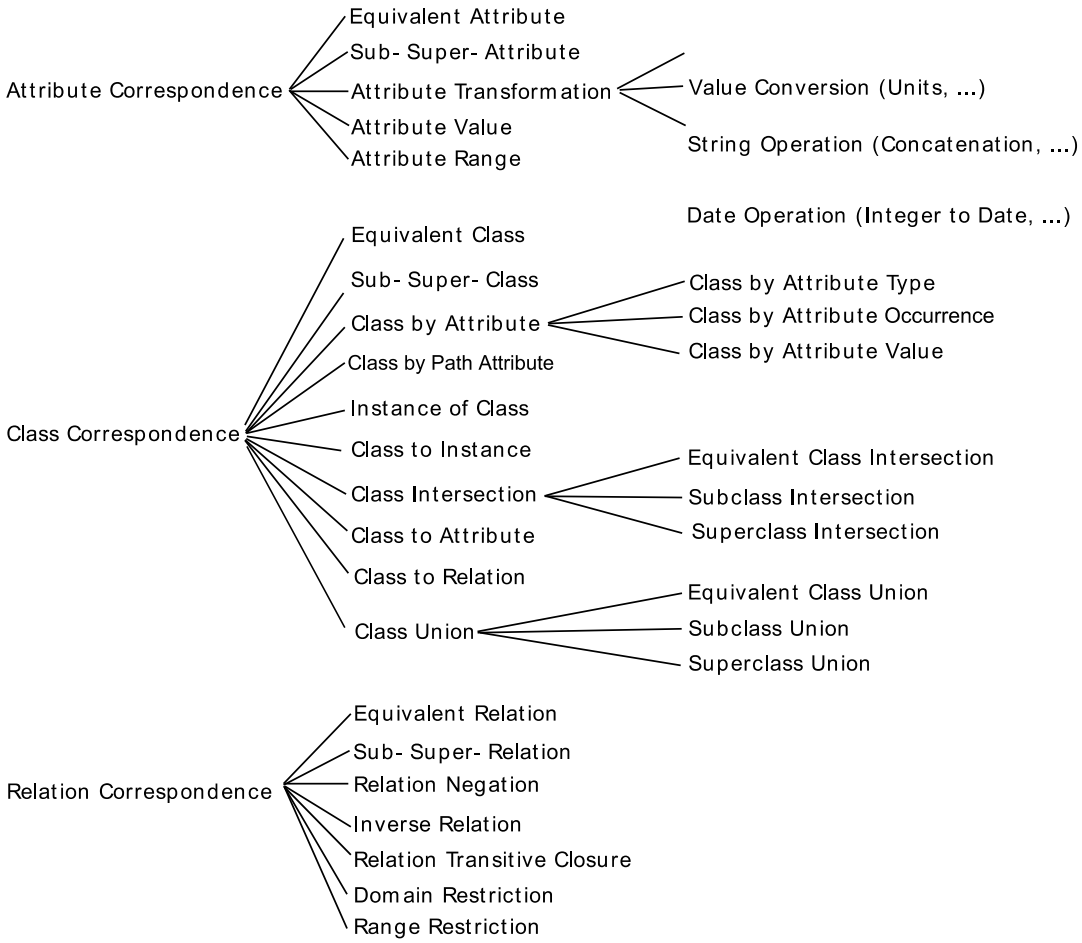


Figure 9: Partial correspondence pattern hierarchy [9].

[15] http://ontologydesignpatterns.org

3.3.2.1 Class by Attribute Value

Class by Attribute Value (CAV) Correspondences occur when a named class in one ontology is equivalent to the subclass of a named class in a second ontology of exactly those instances which have a specified attribute-value. The parameters in the template grounding for this pattern are the *target class* the *restriction attribute* and the *restriction value*. For example, given two ontologies describing wine, it is possible that one ontology may have a specific class for *BordeauxWine*, but the other does not. If, however, the other ontology has a class for wines in general, *Vin*, and instances of this class have an attribute which describes the region the wine comes from, *terroir*, then it may be appropriate to say that the target class *BordeauxWine* corresponds with the set of instances of *Vin* with the restriction attribute *terroir* set to restriction value *Bordelais*. This pattern is illustrated in figure 10.

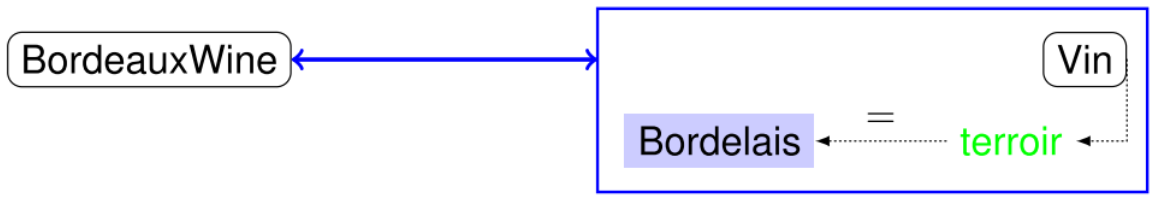


Figure 10. Class by Attribute Value Correspondence [9]. Here the class BordeauxWine corresponds with the class Vin, restricted to only those instances with the attribute terroir set to Bordelais.

3.3.2.2 Class by Attribute Type

The Class by Attribute Type (CAT) correspondence pattern is similar to the CAV pattern in that it forms a correspondence between a named class in one ontology and a restriction class in a second ontology. It differs in that the particular value of the attribute used to specify the restriction does not matter and instead the type of the value is used. For example, *WesternEuropeanCitizen* could correspond to the subset of instances of the class Person with the attribute *citizenOf* set to some country which has type *WesternEuropeanCountry*. This correspondence is illustrated in figure 11.

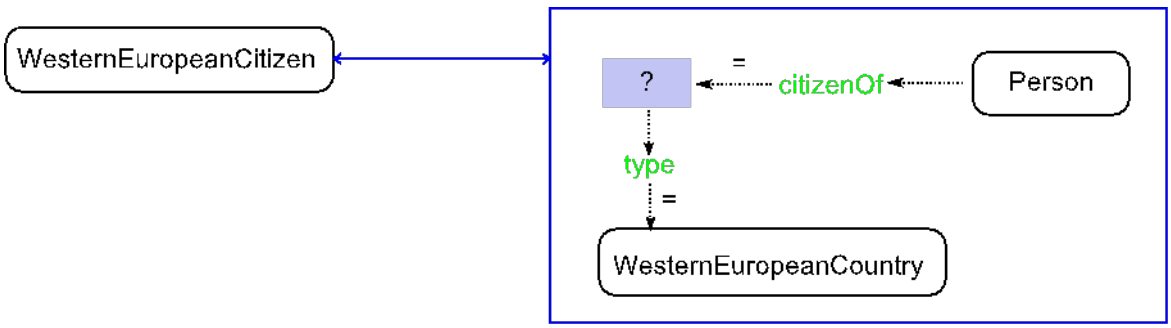


Figure 11. Class by Attribute Type Correspondence [9].

3.3.2.3 Attribute Value Transformations

Further examples of patterns are those for dealing with value transformations. For example if one ontology uses integer values to record wine in *vintage years*, but the other uses *xsd:date* values, then some form of conversion function would be required to translate between the values. This pattern is illustrated in figure 12

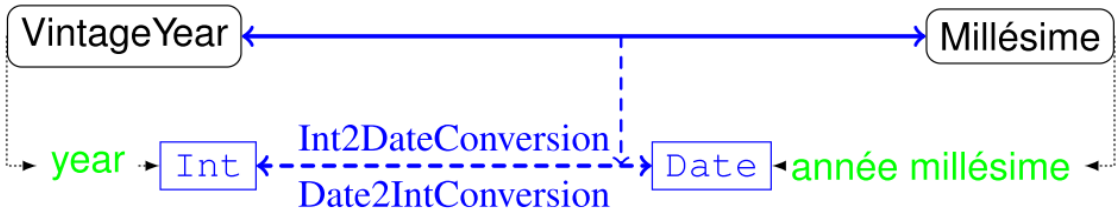


Figure 12. Attribute Value Transformation Correspondence [9].

3.3.2.4 Class by Path Attribute

Class by Path Attribute correspondences can also occur. These are similar to CAV correspondences except instead of depending on an a direct attribute of the class, a path of relationships must be followed to find the relevant attribute. An example of this is shown in figure 13. Here, the class *LocallyGrownWine* corresponds with the set of instances of a class for wine, which are sold in the same area as they were produced.

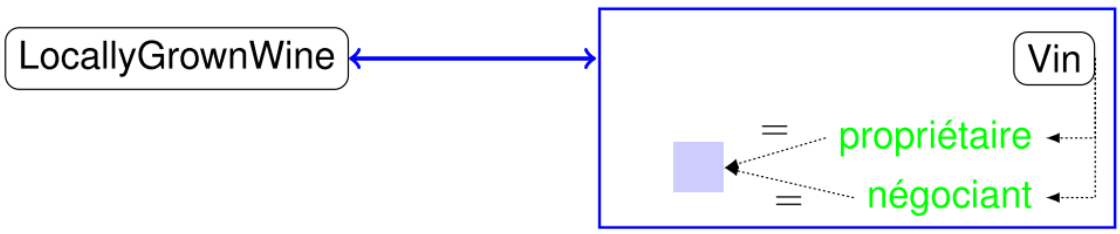


Figure 13. Path Correspondence [9]. Here LocallyGrownWine is equivalent to the restriction class created by instances of Vin that have a propriétaire attribute that is equal to their négociant attribute.

3.3.3 Complex Correspondences in Publicly available ontologies

When discussing complex correspondences it is useful to be aware of which forms of correspondence pattern are seen most often between Semantic Web ontologies. In particular, the larger ontologies such as DBpedia, YAGO2, GeoNames and Friend of a Friend are of interest. As was seen in section 3.2.6 it can be necessary to develop different algorithms for each pattern. Therefore it was considered best to concentrate research effort on the most commonly occurring patterns. In this thesis, the relative frequency of each correspondence pattern has been used to guide the development of new detection methods.

There are few complex correspondences that have been published, as the majority of focus has been on finding one to one relationships. The following sections discuss two examples of complex correspondences which have been published. Section 4.2 provides further analysis of the frequency at which these between in DBpedia and YAGO2. Appendix C lists new complex correspondences between these ontologies which were found in the course of the work described in this thesis.

3.3.3.1 Correspondences between DBpedia and GeoNames

A number of complex correspondences between GeoNames and DBpedia have been published[16] which were discovered using automated methods [15], [44]. All these complex correspondences fit the *Class by Attribute* family of patterns – that is correspondences between a restriction class and a named class. There are 351 *Class by Attribute Value* correspondences published. Additionally there are 5 complex correspondences with a restriction class on both sides of the relation.

A large number of these correspondences relate named DBPEDIA classes with restriction classes on GEONAMES which are conditioned on the properties *featureCode* and *featureClass*. These properties have short strings in their range and would be difficult to understand without outside context to translate the codes to a human readable description. For example the DBPEDIA class *Country* corresponds to the restriction class which contains the instances of GEONAMES *Feature* conditioned on the attribute featureCode set to the string "A.ADM". (See figure 14)

[16] http://www.isi.edu/integration/data/LinkedData/

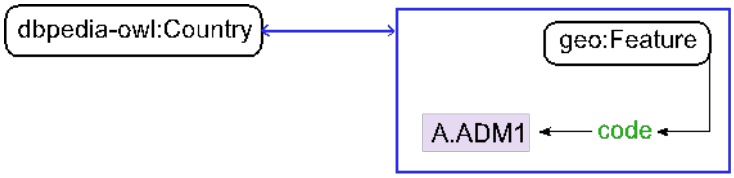


Figure 14.Complex correspondence between DBpedia class Country and a restriction class in GeoNames

One issue with these published correspondences is that many of them appear quite spurious. For example the published set of correspondences includes a relationship between the DBpedia class *Hospital* and a restriction class which contains instances of *Feature* that are located in *Great Britain*.

3.3.3.2 Complex correspondences between YAGO2 and SUMO

YAGO2 has been integrated by hand with SUMO [50]. The class hierarchy of YAGO2 and SUMO are somewhat complementary, which means that most classes in YAGO2 could be inserted into the SUMO hierarchy at the lower level with little alteration. An example of this merged hierarchy is shown in figure 15. It was found however that approximately 10% of the classes in YAGO2 contained some attribute which makes them inconsistent with SUMO. For example *Brown University* is typed as both a *building* and a *group of people* in YAGO2, but SUMO specifically states that a *building* cannot be a *person*. To resolve this some type assertions are excluded, using a set of rules.

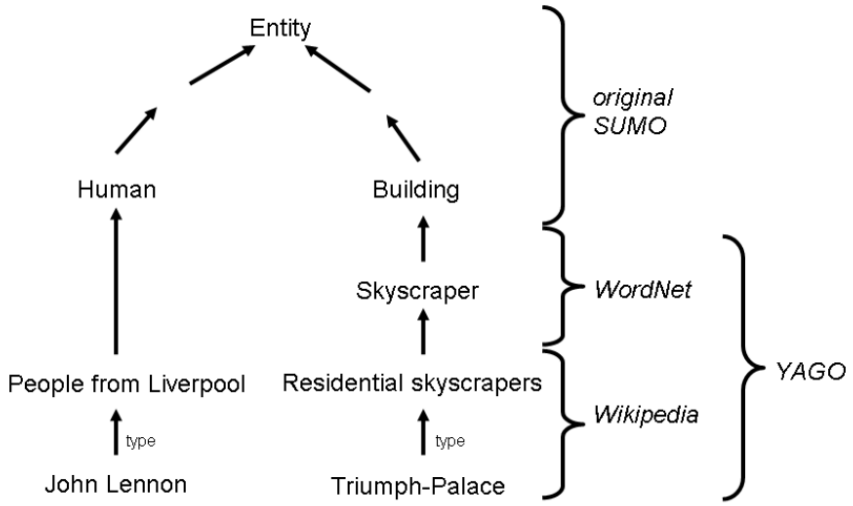


Figure 15. Merging YAGO2 classes into the SUMO hierarchy [46].

There are many *Attribute Value Transformations* that must occur. For example, YAGO2 uses strings to represent dimensioned numerical data, but SUMO has a structured method of representing these, so any numerical attribute with a dimension must be converted. Similarly

dates must be converted. In YAGO2 the date the *28th of November 1961* is represented with the string "1961-11-28", which must be rewritten as the rule `(DayFn 28 (MonthFn 11 (YearFn 1961)))` to be used with SUMO.

It was sometimes necessary to define new SUMO relations to provide an accurate correspondence with YAGO2. For example the relation *yago:establishedOnDate* corresponds with the new SUMO relationship shown in listing 1. This form of complex correspondence does not appear to fit with any published correspondence pattern. This correspondence requires creating an entirely new relationship in SUMO, and not just refining an existing one and so is unlikely to be captured in a simple pattern.

```
(instance establishedOnDate BinaryRelation)
(domain 1 establishedOnDate Agent)
(domain 2 establishedOnDate TimeInterval)
  (=>
   (establishedOnDate ?OBJ ?TIME)
     (exists (?FOUNDING) (and
      (instance ?FOUNDING Founding)
      (result ?FOUNDING ?OBJ)
      (overlapsTemporally (WhenFn ?FOUNDING) TIME))))
```

Listing 1: SUMO rule describing the YAGO2 relation foundedOnDate [50]**.**

In summary, although there are not many published complex correspondences, of those that have been published, many fit with either *Class by Attribute Value* correspondences or *Attribute Value Transformations* correspondences. Section 4.2 provides an analysis of the frequency of CAV correspondences. It was also observed that there are additional forms of complex correspondence that occur which cannot be described by the existing published set of patterns. Correspondences of this form require defining entirely new concepts or relationships in the ontologies, and not just refining existing ones, and are considered outside the scope of this thesis..

3.3.4 The Explicit Declarative Ontology Alignment Language

The Explicit Declarative Ontology Alignment Language (EDOAL) [51] provides a method for describing complex correspondences between ontologies. It was developed as an

extension of the Alignment Format, and has been a standard component of the Alignment API since version 4.0 [11]. EDOAL extends the Alignment Format by allowing complex OWL-like statements in place of the named ontology elements contained in a map element – for example allowing correspondences between unions or intersections of classes. Additionally it allows data transformations described by XQuery functions and operators[17].

This format allows very expressive correspondences being very similar to OWL, In the interests of interpretability EDOAL does not support all the features of OWL, only those necessary for describing relationships. It does not allow the defining of new named entities, for example. In addition it allows the use of variables which can be used to describe constraints and transformations. . The development of EDOAL was closely linked with the development of Correspondence Patterns [9], which describe common forms of correspondences and solutions for implementing them. Correspondence patterns provide a standardised way of describing the correspondences at a high level which aids interpretation.

Tools that use EDOAL to its full extent are still limited. For example, of the alignment tools described in the OAEI 2012 competition, none produce correspondences more complex than could be described with the Alignment Format. There are some tools available that apply complex correspondences. For example Correrdo et.al. [52] demonstrated a tool which uses EDOAL correspondences to rewrite SPARQL queries.

3.3.5 Pattern matching approach to detecting complex correspondences

One approach which uses correspondence patterns, and which has been demonstrated to at least work on test sets published by the OAEI, is the use of pattern matching to search for complex correspondences directly [8]. With this approach, a set of rules are used to compare the ontologies and evaluate if a grounding of a correspondence pattern can be used to convert from one ontology to the other. Each pattern requires a separate detection algorithm. To date only a limited number of detection algorithms have been developed as proof of concepts. In particular the approach has been demonstrated to detect several forms of Class by Attribute correspondences, as well as Path Correspondences.

The detection algorithms require a reference alignment of simple one to one correspondences as input, and use three broad criteria for detecting complex correspondences:

[17] http://www.w3.org/TR/xquery-operators/

Structural criteria: To detect complex correspondences, the pattern detection algorithms use the position of the classes in the ontology hierarchies. That is, if one class is a subclass of the other or if they are equivalent. An extended notion of subclass inclusion is used where class C_a from the ontology O_1 subsumes class C_b from the ontology O_2, if $C_a \sqsubseteq C_c$, with $C_c \in O_1$, and there is an equivalence correspondence between classes C_c and C_b.

Syntactical criteria: These criteria analyse the *local name* of classes and attributes to determine if there is any meaning in the words used to construct that name. The names are broken up into their constituent words which are then used by the algorithms. The last noun in the name is given special relevance in the correspondence detection algorithms, and is referred to as the *head noun*.

Data type Compatibility: The algorithms consider two data types compatible if there is a way of converting one to the other.

The pattern matching approach uses sets of rules to detect complex correspondences. To detect a CAT correspondence (see section 3.3.2.1 for details of CAT) between class *A* in ontology O_1 and class *B* in ontology O_2 conditioned on attribute *p* having value C, using the pattern matching approach, the following conditions must hold:

1. The string that results from removing the head noun from the label of *A* is similar to the label of *C*.
2. There exists a class C* in O_2 that is a superclass of C, which is in the range of *p* and has also a label lexically similar to *p*.
3. The domain of *p* is a superclass of A due to the reference alignment

For example if *Accepted_Paper* and *Paper* come from two separate ontologies and from an input set of reference 1 – 1 correspondences *Accepted_Paper* is known to be more specific than Paper. *Accepted_Paper* has head noun *Paper*, removing this from the class name leaves the string "Accepted", which is similar to *Acceptance*. *Acceptance* which is a subclass of the class *Decision* which is in the range of *hasDecision*. *Paper* is in the domain of *hasDecision*, and *Decision* and *hasDecision* are similar, so all the conditions of a Class by Attribute Type correspondence pattern have been met. This pattern matching process is shown in figure 16.

The Pattern Matching approach is reliant on narrowing the search space for finding complex correspondences to using only data from the T-Box of the ontologies, and then exhaustively searching this space for patterns which fit specific types of complex

correspondence. A different set of rules is needed for each type of correspondence pattern, and the pattern matching approach cannot find complex correspondences unless they exactly match the expected pattern.

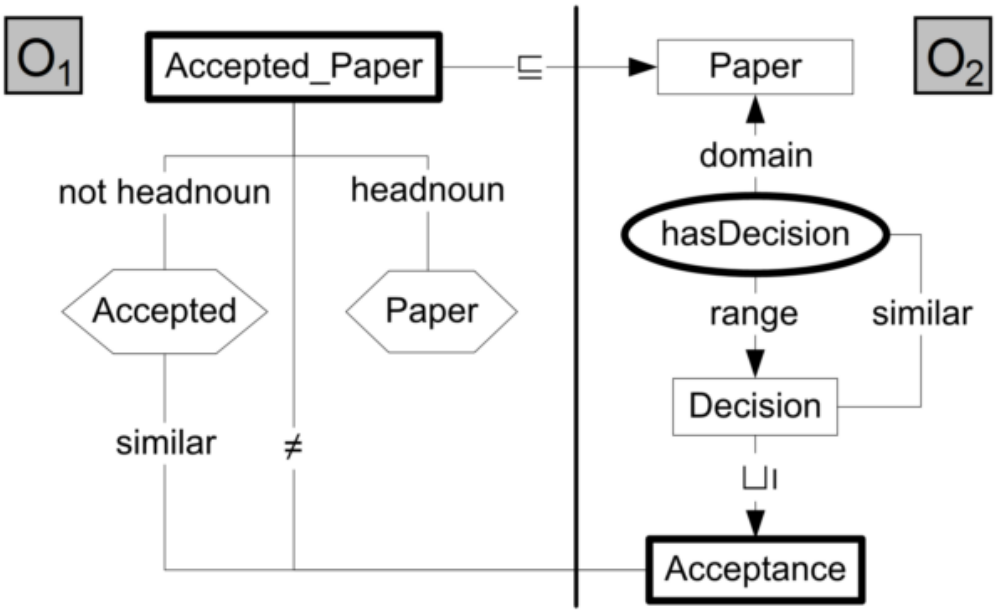


Figure 16. Detecting a Class by Attribute Type correspondence through structural and syntactic analysis [8].

As the approach does not use A-Box information it is not capable of detecting some forms of complex correspondence that are commonly occurring. For CAV (see section 3.3.2.1) it is limited to using properties that have Boolean ranges, as generally finding all values in the range of an attribute would require inspecting A-Box data. As the pattern matching approach lacks a search strategy, inspecting A-Box data would cause a combinatorial increase in the number of pattern match comparisons that would be required.

A more fundamental problem with the Pattern Matching approach is that it depends on classes having meaningful names, and there being a reliable way of separating these names into their constituent parts. There is no guarantee that this is always the case. The GeoNames ontology, for example, makes heavy use of feature class which consist of a single letter and feature classes which consist of short strings. These letters and strings have little relationship to the classes they represent. For example the feature class A represents places, and the feature code PNDN represents salt ponds.

3.4 Comparison of approaches

For all the approaches discussed above, two commonalities can be observed. The first commonality was that all approaches started by either detecting one-to-one correspondences

or using a set of one-to-one correspondences as input. Complex correspondences are not detected without first having a set of one to one correspondences.

The second commonality was that all except one approach (Pattern Matching described in section 3.2.63.4) have some form of metric which describes how close a given solution is to an ideal one and a search strategy based on this measure which they use to progress to the best solution supported by the data.

Each approach employed different search strategies. The Pattern Matching approach exhaustively searches the ontologies directly for pre-conceived patterns in their structure and naming conventions. Tupelo converts databases to a vector representation and, using some metric to measure the similarity of two vector representations, searches through the space of transformations of the vectors in a tree like manner. The Heuristic Tree Search approach searches through the space of all possible restriction classes that could exist in the ontologies. The Multi Relational Data Mining searches through grouped one to one correspondences. Not all approaches were searching for the same kind of complex correspondences. The Heuristic Tree Search and the Pattern Matching approach were looking for forms of correspondences between classes, but Tupelo and the Enhancement Engine were looking for transformations between the values of instances properties.

With respect to the need for instance data. The pattern matching approach was the only one which did not use instance data to search for complex correspondences, though the amount of instance data used varied between the approaches. Tupelo used a minimal amount of *Rosetta Stone* instances which must be carefully selected by a user to be representative of the overall dataset. The Heuristic Tree Search uses a large set of matched instances, which again must be specified by a user. The Multi Relational Data Mining approach uses entity resolution to find as many common instances as possible.

3.4.1 Taxonomy

It is possible to classify the surveyed approaches in three different ways – by the type of complex correspondences they produce, by the evidence they use to find correspondences, and by the search strategy they employ.

Figure 17 shows the taxonomy produced by looking at the types of evidence used by each approach. The primary distinction is that the detection approach can look at intentional information contained in the T-Box of the ontology, or it can look at the extensional information shared by the two ontologies. The second level of this taxonomy distinguishes

how the information is used. With intentional information, there are two possibilities either the approach seeks to highlight parts of the structure to allow a user to visually find and specify complex correspondences – as is the case with InfoSphere and Coma++ – or the approach can try to directly detect complex correspondences using a combination of structural and linguistic analysis – as is the case with the Pattern Matching approach. Taking the common extensional evidence, there are two options: the detection approach can analyse a small number of canonical instances in detail to discover their relationship, or it can examine a large number of instances to see how many support a given relationship. A further distinction can be made in the detection approaches that use levels of support: some will only use the matched instances they are supplied with at the start of the process, but others will use the complex correspondences they discover as feedback and search for additional common instances which are then used to find further correspondences. This happens in the Multi Relational Data Mining approach for example.

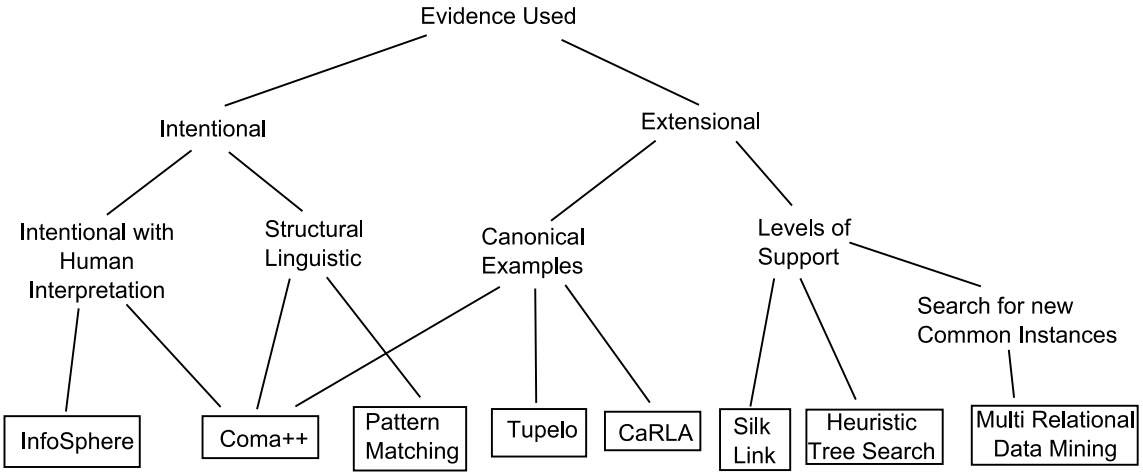


Figure 17. A taxonomy of the detection approaches surveyed from a perspective of the evidence each approach uses for detection.

Coma++ is interesting as it is the only approach which falls under multiple classifications in this taxonomy. It uses structural and linguistic techniques to find canonical examples which are then presented to a human to be approved and adjusted visually.

A second way of classifying the surveyed approaches is by the **types of relationship** they are capable of discovering. This taxonomy is shown in figure 18. Here we can see that Silk Link is different to the other approaches as it does not technically produce correspondences. Instead it produces rules that can be used to compare the similarity of linked data

information. The two most common classes of complex correspondence detected by the surveyed approaches are Value Transformations and Restriction Class correspondences. CaRLA is limited to detecting transformations of single values – for example capitalisation of a string – but Tupelo, InfoSphere and Coma++ can all handle transformations which use several values – for example concatenating two strings together. The Pattern Matching approach can detect correspondences between named classes and restriction classes, but the Heuristic Tree Search can detect correspondences with restriction classes on both ends of the relation. Multi relational data mining was the only approach which can detect path correspondences – for example it can detect person(x) ∧ worksAt(x,y) ∧ phone(y,z) ≡workPhone(x,z).

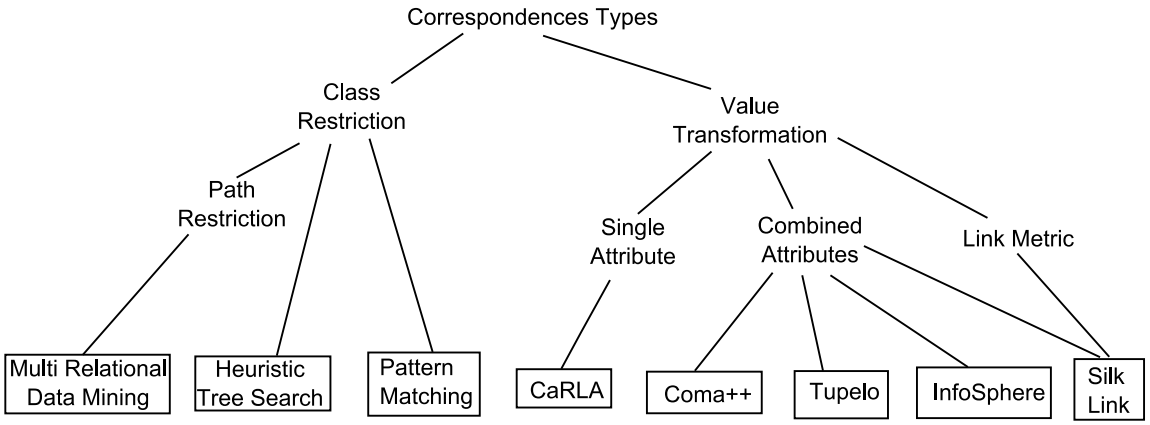


Figure 18. Type of relations detected by each of the approaches surveyed.

A third way of classifying the detection approaches is by **the search strategy** they use to detect the correspondences. This classification is shown in figure 19. The most common forms are ***heuristic searches***, which use some strategy to minimise the space. Tree Searchers being the most common. There is also the ***genetic programming*** approach, that generates a sample of random solutions and selects the best from these. It then iteratively creates new solutions based on variations of the best existing solutions and selects the best from the new solutions. In contrast, there are the exhaustive search approach. In addition there were ***interactive approaches*** such as InfoSphere and Coma++ which display possible solutions to a human user and allow them to explore these solutions visually.

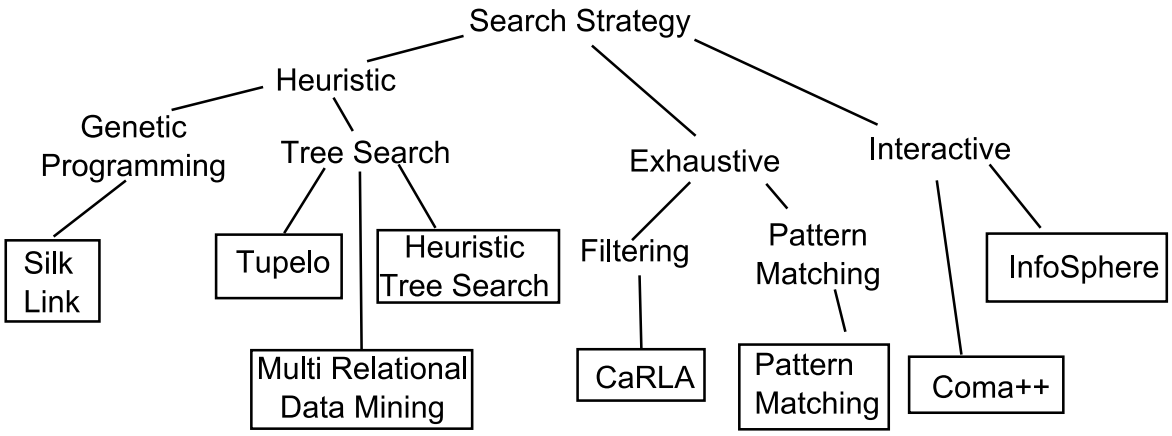


Figure 19. Search strategy used by each of the approaches surveyed.

Table 2 provides a direct comparison of the approaches discussed in this chapter in terms of the evidence they use to perform detection, the type of correspondence they detect, and the search strategy they employ. Common instances were the most widely employed form of evidence, and only the Pattern Matching approach used purely intentional evidence. Value transformations were the most common form of correspondence the approaches sought to detect, and Restriction Classes were the second most common. Many of the approaches

Table 2: A comparison of the correspondence detection approaches discussed in this chapter.

Approach	Evidence Used	Type	Search Strategy
IMB InfoSphere	Intentional + human understanding	Value transformations	Human interaction
Tupelo	Categorical extensional examples	Value transformations	Tree search
COMA++	Multiple	Value transformations	Human interaction
Multi Relational Data Mining	Level of support in common instances	Path correspondence	Tree search
Heuristic Tree Search	Level of support in common instances	Restriction Class correspondence	Tree search
Silk Link	Level of support in common instances	Link Metrics	Genetic programming
CaRLA	Categorical extensional examples	Single Value transformations	Exhaustive filtering
Pattern Matching	Intentional	Restriction Class	Exhaustive pattern

employ some form of tree search, but two search exhaustively. Two employed a search tactic where they present options to a human user who then selects how to proceed.

3.4.2 Use of Correspondence Patterns to Guide Complex Correspondence Detection

To the author's knowledge there is currently only one complex correspondence detection method that specifically uses Correspondence Patterns in its operation. This is the Pattern Matching approach, which was discussed in section 3.3.5. However there are other detection methods which use concepts similar to correspondence patterns to guide their search. Tupelo, which was discussed in section 3.2.2 searches through the space of Attribute Transformation correspondences. The Common Existential Heuristic Tree Search discussed in section 3.2.5 searches through the space correspondences between Restriction Classes. This is a broader class off correspondence than Class by Attribute Value, but in practice, it was shown in section 3.3.3 that CAT correspondences are by far the most commonly found form of restriction class.

There are several ways that knowledge of Correspondence Patterns could be used in the development of complex correspondence detection approaches and even in the improvement of existing techniques. For patterns that include a parameterised template grounding, searching for complex correspondences which fit these patterns is a matter of searching over the parameters of the grounding. This gives clear guidelines on what the search space should look like and how the search strategy should be optimised.

As seen in section 3.2.5, the Heuristic Tree Search approach is capable of finding very complex correspondences with restriction classes constrained by multiple attributes. This means it is searching over a large space of possible correspondences, and must optimise its search strategy accordingly. However, complex restriction class correspondences dependent on multiple attributes, do not appear to be very common. In section 3.3.3, it was shown that Class by Attribute Value correspondences were by far the most common Restriction Correspondence when comparing DBpedia and GeoNames. CAV correspondences only use a single attribute to form the restriction class, and so would only require evaluating a single level of the tree when searching. This means that the use of the tree does not optimise the search in any way as there is no opportunity for pruning.

The Pattern Matching approach uses an exhaustive search, but as it is searching for a smaller subset of correspondence patterns than the Tree Search, and also looking at a smaller set of evidence, it does not have as great a need to optimise its search strategy. However it is not capable of detecting CAV correspondences, which were shown to be very common. Doing so would require analysis of instance data to find the values of attributes. This would greatly increase the search space, and so an improved search strategy would be required.

3.5 Summary

As was seen in section 3.1, a large portion of the development effort for the tools in correspondence detection is concentrated on performing well in the annual Ontology Alignment Evaluation Initiative (OAEI) competition. This competition almost exclusively features on one to one equivalence correspondences, with all the automated scoring metrics being of this form. These one to one equivalences are not sufficient to describe the full relationship between the ontologies used in the tests. One track, Library, is based on a dataset which contains subsumption relationships, but these are not used in the evaluation. A second track, Conference, does feature ontologies which can have complex correspondences between them. However, while contestants in 2012 were invited for the first time to enter tools which are capable of detecting these complex correspondences, no such tools were entered in 2012, or subsequently in 2013.

There are a small number of approaches that have been published which are capable of detecting complex correspondences, or at least allowing a human to describe them in a manual step within an otherwise automated system. IBM InfoSphere is one such system. It will automatically detect one to one correspondences between schemas, but requires a human to specify transformation formula for complex correspondences, which can then automatically converted into SQL queries that can be used across databases. Other approaches, such as Tupelo can detect these correspondences automatically. These approaches vary in the type of complex correspondence they can detect, the evidence they examine, and the search strategy they employ. A taxonomy of the approaches using these three axes was developed as part of the research for this thesis and presented in section 3.4.5.

Of the approaches surveyed two common steps were seen. Most start off by detecting the one to one correspondences before using this information to find the complex ones. They mostly use the instance data that is common to both ontologies as input. Tupelo uses a small number of *Rosetta Stone* instances and examines how one instance representation can be

transformed to the other. The approaches described in sections 3.2.4 and 3.2.5 use large amounts of matched instances to examine the level of support for a given correspondence among those instances. Only the Pattern Matching method described in section 3.3.5 did not make use of instance data, and relied entirely on the T-Box portion of the ontologies. This however limits the types of correspondence patterns it can detect.

Correspondence Patterns were shown as a way to describe the forms of correspondence that the detection methods can produce. Classifying the correspondences like this is beneficial as it allows it to be seen how different parts of the detection methods can be used together. By knowing the important parameters for each correspondence pattern, it helps inform us of the search space that needs to be tackled and also what metrics to use. Additionally there are tools available which can take an instantiation of a correspondence pattern grounding and use this for example to perform query rewriting.

In this chapter too, which patterns of correspondence occur most often in semantic web ontologies has been discussed. It was seen that *Class by Attribute Value* and *Attribute Transformation* correspondences were most common. Class by Attribute Value correspondences are not handled well by current approaches, either because the search strategies they use are not optimised for this pattern, or because they simply cannot detect correspondences of this form. As an initial step it is likely be of benefit to concentrate research effort on developing approaches to detecting these forms of complex correspondence instead of the less common ones.

4 Requirements

The following chapter describes the requirements for a restriction class correspondence detector. In addition this chapter describes an analysis carried out to determine the most common patterns of complex correspondence, and the criteria that will be used to determine if the correspondence produced by a detector is correct. Section 4.1 describes the requirements, and section 4.2 describes the analysis of common correspondence patterns, and section 4.3 describes the criteria for assessing the correctness of a correspondence.

4.1 Requirements List

Through a combination of the research question, the state of the art analysis, and a further case study which is detailed in section 4.2 the following list of requirements was developed.

R1. Detecting Complex Correspondences

The analysis of the state of the art undertaken in section 3 revealed that currently, while there are many approaches available for detecting one to one correspondences in ontologies, there are few that are capable of detecting complex correspondences. The approach developed should address this imbalance by providing a method for detecting complex correspondences. Therefore the first requirement, **R1**, is that the approach be capable of detecting complex correspondences. That is, correspondences containing multiple elements in a logical formulation.

Refinement to R1:

R1.1 Detecting the Class by Attribute Value correspondence

In section 3.3.3 of the state of the art analysis, Class Restriction correspondence patterns such as Class by Attribute Value (see section 3.3.2.1) correspondences were among the most common seen in real world semantic web ontologies. A further deep analysis of correspondences between DBpedia and YAGO2, which will be described later in section 4.2 will demonstrate that Class by Attribute Value correspondences appear to be the most commonly occurring complex correspondence in these ontologies. Therefore the refined requirement **R1.1**, the approach should be capable of detecting **Class by Attribute Value** Correspondences was specified.

R2. Compatible with Open World Assumption and tolerant to incorrect data

The approach developed in this thesis is primarily focused on ontologies available in RDF where *anyone can say anything*. The RDF standard specifically states that designers of applications that use RDF should design their applications to tolerate incomplete or inconsistent sources of information. Therefore this requirement needs to apply to the complex correspondence approach developed.

R3. Choice of metrics

In section 3.2.2 it was seen that Tupelo, which is a general framework for solving data mapping problems, allows a choice from multiple metrics to be searched for solutions. This was seen as a desirable feature as it allows flexibility, and the ability to experiment with different search strategies while still using the same framework. Therefore requirement **R3** for the approach developed should be that it be possible to select from multiple metrics when detecting complex correspondences. Furthermore the implementation of this approach should allow the addition of new metrics in a pluggable manner.

R4. Use of simple correspondences and instance matches to detect complex correspondences

In section 3.2 it was seen that many of the existing approaches to detecting complex correspondences require an input set of one to one correspondences, as well as a set of matched instances. Section 3.1 showed that there are many approaches available which can produce these one to one correspondences Therefore requirement **R4** of our approach is that one to one correspondences and matched instances be used as input to detecting complex correspondences.

Refinement to R4:

R4.1. Minimisation of instance matches required

The Pattern Matching approach to detecting complex correspondences seen in section 3.3.2 uses no matched instances as the developers of this approach believe that there are not always sufficient matched instances available to be used for complex correspondence detection. However it was seen that using no instance data prevented the approach from detecting CAV correspondences, amongst others. Therefore requirement R3.1 for our approach is that it is capable of detecting complex correspondences when only a small number of matched instances are available.

R5. Scalability

Datasets published on the web as linked open data and their ontologies can potentially be very large, both in terms of their A-Box and T-Box. The DBpedia ontology for example contains information on 4.0 million *things* with 470 million *facts*. The class hierarchy of the YAGO2 ontology contains 224,391 classes. Requirement R5 of our approach is that it the detection approach is capable of scaling to ontologies of this size.

4.2 Analysis of Class Restriction correspondences in YAGO and DBpedia

Section 3.3.3 in the state of the art chapter described published complex correspondences in linked open data sources. This information was used as a guide for the types of complex correspondence that could be expected to occur most often and which our approach should be capable of detecting. To gain further insight into the nature of the complex correspondences that occur in linked open data, an analysis of complex correspondences between YAGO2 and DBpedia was carried out. This analysis is intended to show which types of complex correspondence occur most often as well as the nature of the data available when searching for complex correspondences.

DBpedia and YAGO were selected as they represent two of the largest general knowledge data sets in the semantic web. They were both generated from the contents of the Wikipedia website, and so they should have similar contents. The differences in the ontologies should largely be due to different choices in the way the information should be modelled, and imperfections in the process used to extract information from Wikipedia.

DBpedia has a taxonomy with 170 classes, while YAGO2 has a much richer taxonomy with approximately 15,000 classes. Many of the YAGO2 classes have very narrow scope with no equivalent named class in DBpedia. For some of these it is possible to find an equivalent restriction class – in other words a CPV or CPE correspondence.

Version 3.7 of DBpedia was used, excluding article abstract data, as abstracts are unique to each instance and so can never be used in a class restriction correspondence. The DBpedia dataset includes class information that is not specified in the DBpedia ontology, so we removed these. The 09/01/2012 version of YAGO2 core was used. Transitive closure on class membership applied to make all class membership statements explicit in both datasets.

As the correspondences were created by hand, only 50 YAGO2 classes were examined to find the best class – named or logically constructed – in DBpedia which they should

correspond with. The choice of a "best" correspondence is somewhat subjective. A more rigorous examination of what should constitute a good restriction class correspondence is discussed in later in section 4.3.

To create this sample of 50 YAGO2 classes, a SPARQL select query was used to find all YAGO2 classes that have at least one instance and then randomly selecting 50 classes from this list. Figure 20 shows the number of different patterns of correspondence found when this process was complete. Twelve YAGO2 classes had direct one to one equivalence correspondences with DBpedia ontology entities, though only 6 of these were class to class equivalences and the other 6 were class to instance correspondences. Thirty four YAGO2 classes corresponded with DBpedia classes that were much broader in scope – subsumption correspondences – and 6 YAGO2 classes did not correspond with anything in DBpedia. Of the 23 subsumption correspondences, 11 could be made into equivalence correspondences by using the Class by Attribute Value (CAV) correspondence pattern.

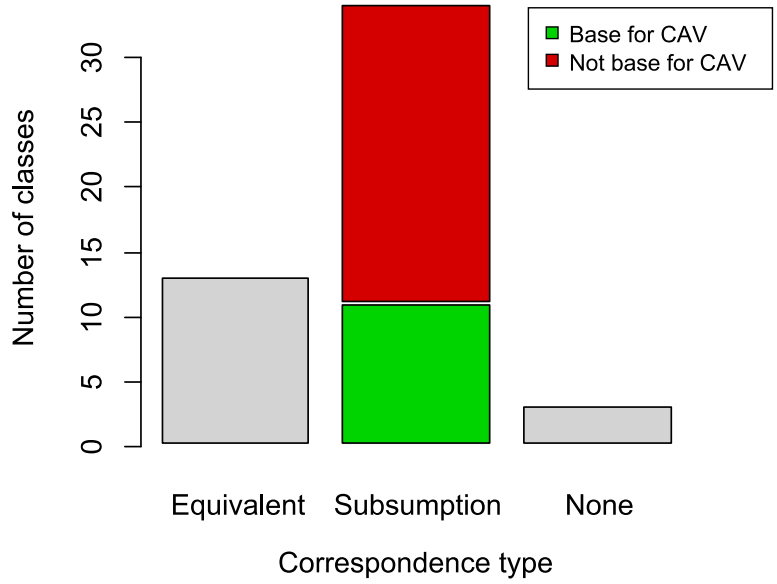


Figure 20.Types of correspondence found between from YAGO2 classes to DBpedia.

Although this case study was not extremely rigorous, it does show that CAV correspondences are relatively frequent between YAGO2 and DBpedia. In the sample of YAGO2 classes examined, CAV correspondences were almost as frequent as direct equivalences.

As stated previously the choice of a "best" restriction class correspondence was still something of a subjective decision in this investigation. The guiding method used was to pick attributes and values with similar meanings as the class name being investigated. This did not

always produce good results as there was often a choice of very similar attribute names – *occupation* vs. *profession* for example.

4.3 Defining an *optimal* Class Restriction correspondence

As the field of complex correspondence detection is relatively unexplored, there is a lack of consensus on how complex correspondences should be judged. What does it mean to say a particular detection approach produces "good" results? With no wide spread set of test cases, it was left to the author of the thesis to design one. What constitutes an optimal complex correspondence is not a clear decision. For example should the semantic meaning of the attribute and value used in the restriction be the guiding principal, or should consideration be given to other factors such as how well defined the attribute is taking into account its consistency in the instance data?

As correspondences are used in tasks such as query rewriting and gathering information contained in instance data, the author of this thesis chose instead to use instance set size as the guiding factor. We consider a class restriction correspondence optimal if it maximises the similarity of the sets of instances of the two classes in the correspondence. The Jaccard measure [53] is used to measure set similarity.

Due to the open world assumption it is not always possible to measure the true Jaccard for a class. We can count the *known* instances, but as more information becomes available the score may change greatly. Intuitively, using *well-defined* properties – ones which are unlikely to change as more information becomes available – to define the restriction class will give a known Jaccard that is more likely to be close to the true Jaccard.

The following sections (4.3.1 – 4.3.3) provide a formal description of what constitutes the intuitive idea of a well-defined class restriction attribute, and how this relates to picking an optimal class restriction correspondence.

4.3.1 Graph Representation of Linked Data Knowledge Bases

Let $\mathbb{K} = \{N, P, R\}$ be the graph of all knowledge which can be described as triples, with N being the set of all classes and instances of these classes, P being the set of all binary relations with domain and range N, and $R \subseteq N \times P \times N$ being the set of all relations between the elements of N. (With $\times$ being the Cartesian product.)

For a given Linked Data Knowledge Base, KB we assume that KB can be described as

$$KB = \{N_{KB}, P_{KB}, R_{KB}\}$$

with

$$\begin{array}{lll} N_{KB} & \subseteq & N \\ P_{KB} & \subseteq & P \\ R_{KB} & \subseteq & N_{KB} \times P_{KB} \times N_{KB} \end{array}$$

As a shorthand, in this paper we will write $\langle s, p, o \rangle \in KB$ to mean that $s, o \in N_{KB}$, $p \in P_{KB}$ and $(s, p, o) \in R_{KB}$

This viewpoint assumes that each KB does not contain any incorrect triples, but it does allow that triples can be missing. The issue of missing triples is known as the *Open World Assumption*. For any given KB

$$\langle s, p, o \rangle \in KB \quad \Rightarrow \quad \langle s, p, o \rangle \in \mathbb{K}$$

but

$$\langle s, p, o \rangle \notin KB \quad \not\Rightarrow \quad \langle s, p, o \rangle \notin \mathbb{K}$$

This assumption is necessary due to the dispersed nature of knowledge on the Semantic Web, which means that logical clashes would occur when integrating information from multiple sources if we make a closed world assumption.

4.3.2 Local Completeness and Well-defined Classes

The open world assumption presents us some challenges when working with linked data knowledge bases. It is not possible, in general, to select instances which do *not* have a given attribute. In general this makes sense, as we would not expect a knowledge base to know everything about any given instance. Often however, this behaviour is less desirable. If the KB contains large amounts of data, then we might expect that if the KB knows about entities s and o and the predicate p, but does not contain the relationship $\langle s, p, o \rangle$ then that relationship is unlikely to hold. We define this as *Local Completeness*

More formally, we say a knowledge base KB is Locally Complete if $\forall s, o \in E_{KB}, p \in P_{KB}$ then $\langle s, p, o \rangle \in KB \equiv \langle s, p, o \rangle \in \mathbb{K}$

I addition we also define a more narrow concept related to local completeness. We say that a class C is *well defined for attribute* p in knowledge base KB if $\forall o \in N_{KB}, i \in C, \langle i, p, o \rangle \notin KB \Rightarrow \langle i, p, o \rangle \notin \mathbb{K}$

4.3.3 Set size based optimal class by attribute value correspondence

What constitutes the "best" complex correspondence can be a subjective decision. Without being able to say that one correspondence is better than another it is impossible to decide if a detector for these correspondences is performing well. In this section the author of this thesis

defines what will be considered as an optimal solution to a Class by Attribute Value correspondence.

We say a resource i is an instance of a class C if $\langle i, \texttt{rdf:type}, C\rangle \in \mathbb{K}$, and define the function instances(C) which gives the set of all instances in $\mathbb{K}$ which are known to have type C.

$$\text{instances}(C) = \{i \mid \langle i, \texttt{rdf:type}, C\rangle \in \mathbb{K}\}$$

When inspecting the contents of any knowledge base KB, it is not safe to assume that $\langle i, \texttt{rdf:type}, C\rangle \notin KB \Rightarrow i \notin \text{instances}(C)$, *even if KB is locally complete*, as it is possible that i or $C \notin R_{KB}$. That is if the knowledge base does not specifically state that i is an instance of C, local completeness is not a strong enough condition to guarantee that i is not an instance of C.

In the case where $C \notin N_{KB}$, the *Class By Attribute Value* correspondence pattern may be used to approximate C, by specifying class membership in terms of a class $C' \in N_{KB}, C' \sqsupseteq C$ which has its scope restricted by predicate and resource available in KB. We use the notation $C\,|_{p=o}$ to denote the instances of class C with attribute p taking value o.

$$C\,|_{p=o} = \{s \mid \langle s, p, o\rangle, \langle s, \texttt{rdf:type}, C\rangle \in \mathbb{K}\}$$

Class by Attribute Value is a design pattern, and as such is usually only seen as a guideline. It is difficult to say what the *best* CAV for a give class is but for our purposes we define it as

$$\text{CPV}_{\text{MAX}}(C, KB) = \underset{p,o,C' \in N_{KB}}{\text{argmax}}\, J(C'\,|_{p=o}, instances(C))$$

Where J is the Jaccard measure. CPV maximizes the similarity between the set of instances defined by restriction of class C' on a predicate p and object o known to knowledge base KB, and the true set of instances of class C.

It is important to note that we wish to maximise the Jaccard index of the set of instances selected from $\mathbb{K}$ the set of all knowledge, not the set of instances selected from KB. The reason for this is that we wish to find a solution that will generalise well, and will not fail as more instances are added to KB.

4.4 Summary

Five requirements for the approach that needs to be developed were defined in this chapter: The approach should be capable of detecting class by attribute value correspondences. It should be tolerant to the levels of incomplete and incorrect data such as those seen in LOD

datasets. It should provide a choice of metrics for detecting the correspondences. It should use one to one correspondences and instance matches as input, and it should scale to datasets as large as DBpedia and YAGO.

Section 4.2 provided an analysis of the process of manually finding class correspondences in DBpedia and YAGO, and discussed the types of correspondences found and some of the choices that had to be made when selecting appropriate properties and values to define class restriction correspondences.

It was seen that there can be an element of subjectivity in selecting the "best" class restriction correspondence. Section 4.3 provided a set-based definition of a best class restriction correspondence which removes the subjectivity of the choice. However it was also shown that due to the open world assumption it is not always possible to find the best correspondence directly, and instead it must be estimated in some manner.

5 Design and Implementation

This chapter describes the design and implementation of an approach to detecting class restriction correspondences which meets the requirements outlined in chapter 4. First a domain model is presented which describes the problem being tackled and the various components to the solution. A use case is then described in section 5.2 which shows how this complex correspondence approach would be used as part of a general ontology mapping task.

The proposed approach to detecting complex correspondences is to convert the detection problem to a feature selection problem. Section 5.3 describes how this conversion can be carried out. Section 5.3.2 describes an existing feature selection method which will be applied in our approach, and section 5.3.3 describes a new feature selection metric that has developed during this research which should improve performance when working with an open world assumption. Section 5.4 describes an implementation of the approach.

5.1 Domain Model

In order to understand the problem more clearly a domain model was created describing the aspects relevant to a complex correspondence detection approach. This domain model is presented in this section in several parts to make it easier to explain. The goal of this model is to examine the structure of the data available and precise nature of the problem – what does it mean to detect a class by attribute value correspondence. On examination it can be seen in the model that the complex correspondence detection problem is similar to the machine learning and statistical problem of feature selection.

The first part of the domain model shown in figure 21 describes complex correspondences and their components using UML notation.

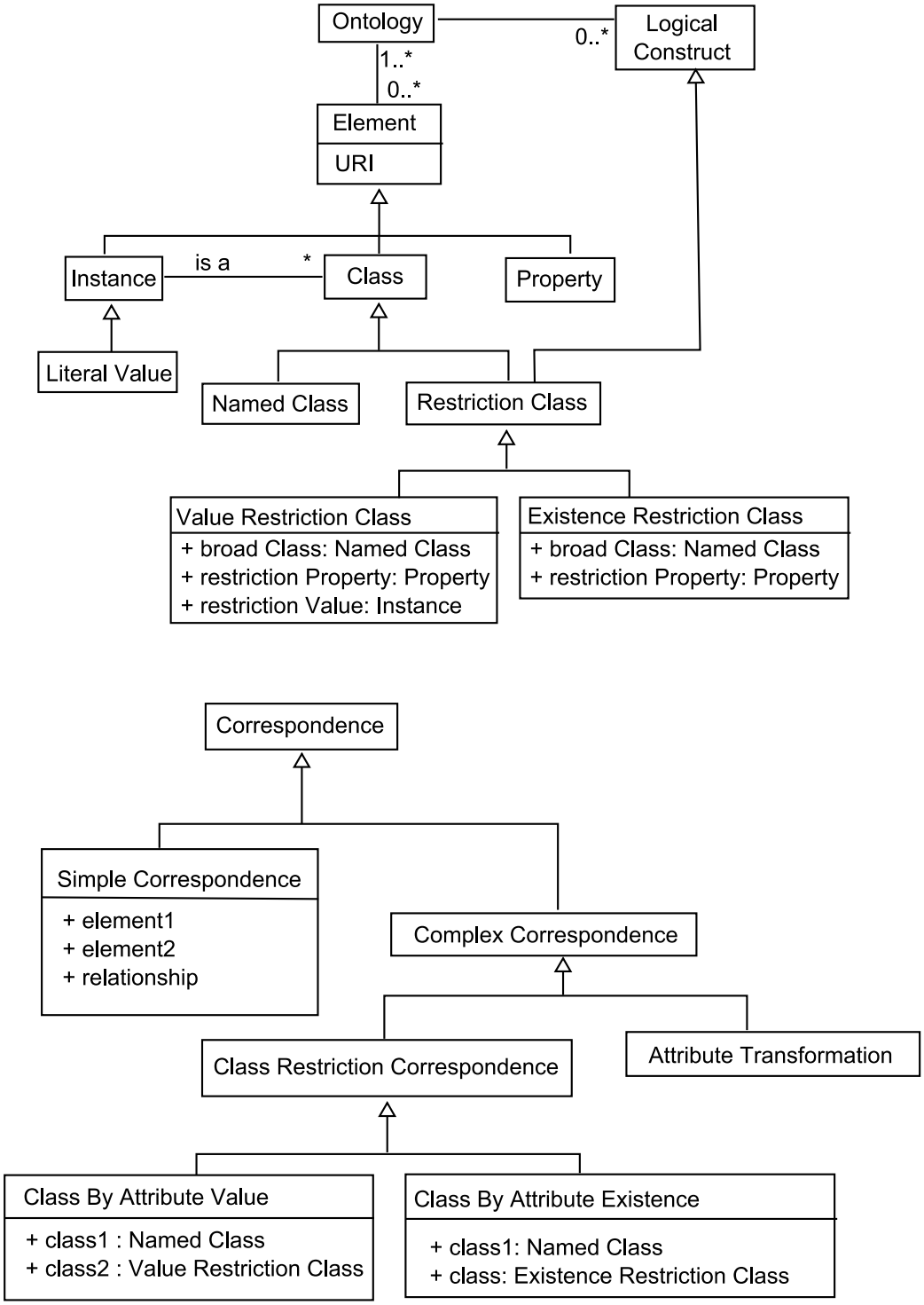


Figure 21. A portion of the domain model describing restriction classes, and class restriction correspondences.

A simple one to one correspondence describes a relationship between two named elements in two ontologies. However one of the most powerful uses of ontologies is that we can describe not only the explicitly declared elements of a given ontology, but also logical

constructions within it. One of the most common logical constructions are Restriction Classes, which are formed by restricting the scope of some named class by placing a condition on an attribute and a value of that class. The Class by Attribute Value correspondence pattern is a correspondence between a named class and a restriction class. Detecting such a correspondence therefore requires searching in the space of logically constructed classes in one ontology to find a correspondence with a named class in a second ontology.

The OISIN framework [17] provides an integrated software and process framework designed to minimise the amount of design time work involved in mapping ontologies, devolve as much work as possible to a runtime algorithm, and share the mappings as much as possible such that human involvement. Within the OISIN framework, there are four phases when aligning ontologies. The first is the characterisation phase where the ontologies are examined and a strategy for aligning the ontologies is selected. The second phase is the matching phase, where an ontology matching tool (or some other means) is used to produce a set of candidate matches, and in third phase these matches are examined and used to produce a set of mappings which consist of confirmed correspondences between the ontologies. The final phase is the management phase where the mappings are put to use or archived for use in the future.

In the existing OISIN framework, converting a candidate match to a committed mapping typically consists of presenting it to a human user who may accept or reject it, possibly with the assistance of some form of consistency checking tool to assist in the decision. Candidates are thought of as being either *right* or *wrong*. When dealing with the possibility of complex correspondences, instead of thinking of a candidate match being either right or wrong, instead there is the possibility that the match represents an *underspecified correspondence*. With the addition of some extra refinement these candidate matches may form the basis of a complex correspondence. Figure 22 describes a domain model for the OISIN framework extended to include underspecified correspondences and complex correspondence detection.

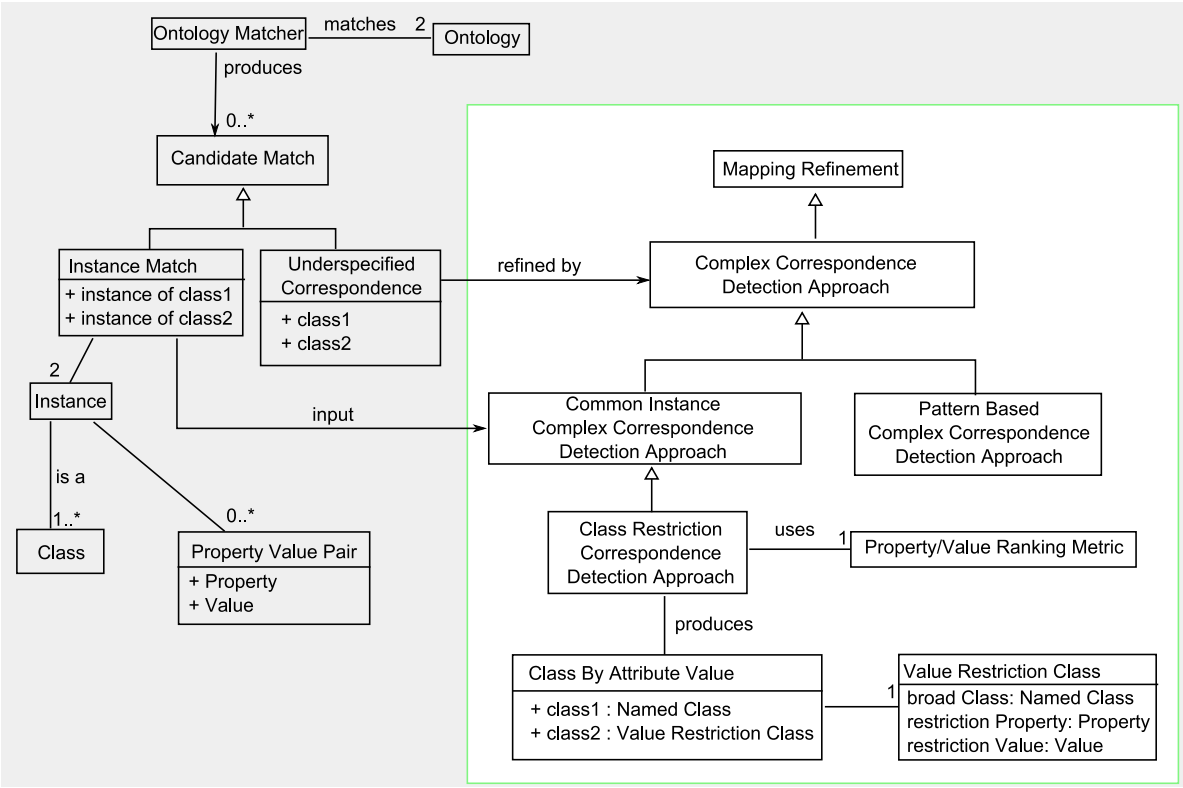


Figure 22. Domain model of the relationship between complex correspondence detection and more general ontology mapping.

Thinking of the complex correspondence detection problem in this manner shows how it could fit into a wider ontology mapping scenario. Existing matching approaches could be used to produce a set of candidate matches. Among these matches would be included matches between classes in the ontologies and matches between instances of these classes. Some of the class matches might be completely correct, some may be irrelevant and others may require further refinement to become correct.

As seen in section 3.2 many of the existing approaches to detecting complex correspondences use common instances – records of individuals that exist in both ontologies – to help detect complex correspondences between classes. In the approach presented in this thesis, it is assumed that these instance matches will be produced as part of the ontology matching process. We accept however that, as these are only candidate matches, it is possible that there may only be a limited number of these matches that are correct. Thus the approach must be robust enough to deal with mistakes and low numbers of matches in the set of instance matches.

Inspecting the input data to be used for detecting complex correspondences, it can be seen that each instance match consists of a pairing of instances, and each instance has at least one

class, and possibly a set of other properties. For each individual instance an attribute can be assigned multiple values simultaneously. Therefore we combine the attribute and value together as a pair to produce a unique feature. A value restriction class is a logical construction defined by a base class and attribute and value which restrict the scope of the base class. Searching for features which can be used to classify individuals is a well known problem in the fields of machine learning and statistics, and is known as feature selection. Section 5.3.1 shows in detail how complex correspondence detection can be seen as an feature selection problem. It discusses how existing feature selection approaches can be applied to the problem, and describes a new metric for selecting features which accommodates the open world assumption which applies in the field of the Semantic Web, but which is not typical of the field of machine learning.

The following section describes a use case for complex correspondence detection methods and an extension of the mapping phase of the OISIN framework which allows for the detection of complex correspondences.

5.2 Complex correspondence detection use case

Figure 23 shows the use case for detecting complex correspondences in context of an ontology mapping system. The user must first examine the ontologies to assess their amenability for mapping and make the decision to map. The user then uses an ontology matcher to generate candidate matches between the ontologies. In the OISIN framework, they would then be required to evaluate the candidate matches to confirm which ones should become mappings. The *Detect Complex Correspondences* use case is an extension of this process. Here instead of just confirming the correspondence in the mapping, some correspondence detection approach is used to find if there are additional conditions which can be used to express a more accurate complex correspondence.

Confirm Mappings is carried out by the user, but Detect Complex Correspondences is carried out jointly by the user and a Complex Correspondence Detector actor. Detecting complex correspondences is a difficult task, and not one we expect a machine agent to be capable of carrying out unassisted. Instead, in this use case the complex correspondence detector actor narrows down the possible complex correspondences to a smaller number of choices and then the user chooses from those options.

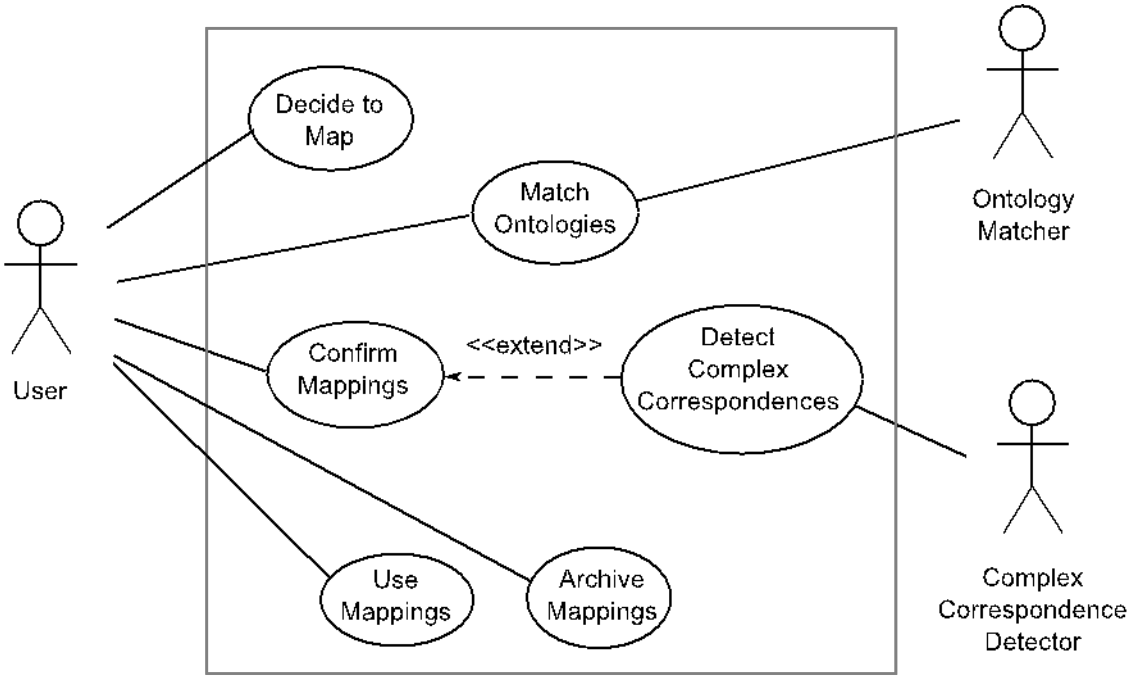


Figure 23. The Detect Complex Correspondences use case.

The Detect Complex Correspondences use case can be extended into the more specific cases of Detect Class by Attribute Value and Detect Class by Attribute Existence cases. In Detect CAV the detection agent is given an underspecified correspondence between class C_1 and class C_2 and provides a list of properties and values which can be used to produce a value restriction class with C_1 as a base class and which is equivalent to C_2. Detect CPE is similar but here the detection agent only provides a list of properties which can be used to form the existence restriction class. Figure 24 shows these steps added into the mapping phase of the OISIN process.

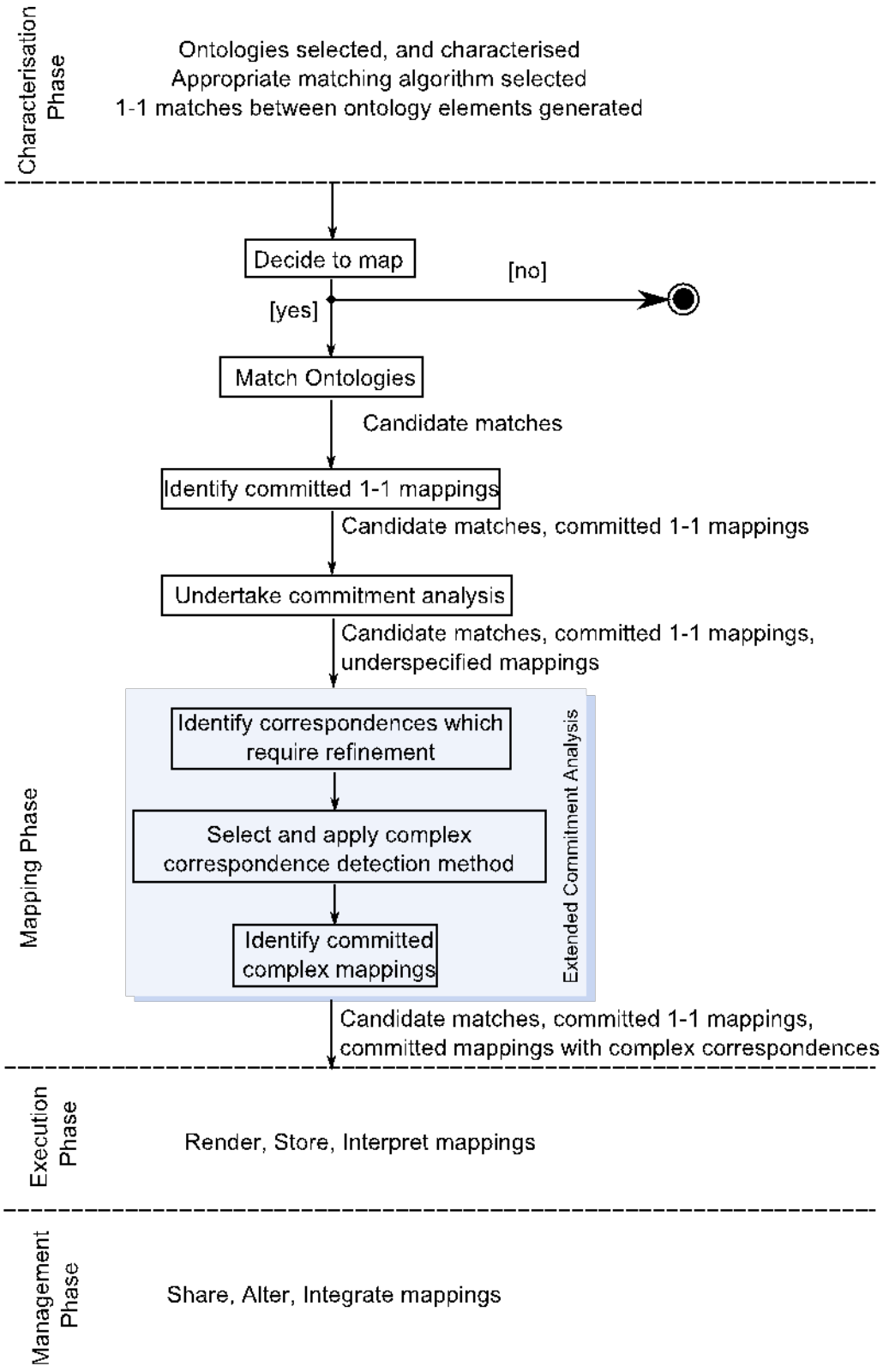


Figure 24. OISIN Process extended to include complex correspondence detection.

5.3 Complex correspondence detection as an feature selection problem

As seen in the analysis of the domain model for complex correspondence detection, when a common instance approach is used, inferences are made using a set of example instances, with each instance having a set of associated attribute-value pairs including a class-type attribute. Class by attribute value (CAV) correspondences consist of a correspondence from a named class to a restriction class defined by an attribute and value. Therefore detecting a CAV correspondence requires finding an attribute value pair – from among possibly thousands of possibilities – in the sample of instances which provide a good indication that a given instance has the class value that is of interest. In our use case, it is not expected that the system directly finds the best attribute-value pair to define a CAV correspondence. Instead it must reduce the search space to a smaller number of possible pairs that the user will then decide from. This can be done by ranking all attribute-value pairs and then selecting the top N.

This is a well-known problem in the fields of machine learning and statistics, and is known as feature selection [53]. With feature selection, there is an assumption that many features in the data are irrelevant to the classification process, and the goal is to return the subset of the features which *are* relevant.

Selecting an appropriate subset of features requires searching through a combinitorially large space, so typically an feature selection approach will use some heuristic to pick individual features and then add more using a greedy search process. Applying feature selection methods to the problem of detecting complex correspondences therefore requires that there is a systematic way of converting the graph like data found in the semantic web into a table of features as is commonly used in machine learning, and then selecting an appropriate measure to select from the features.

Information Gain (IG) is one of the most popular metrics for performing feature selection. It is discussed in section 5.3.2. One issue with IG however, is that it is intended for use in an environment where the Closed World assumption applies. Requirement R6 states that our approach needs to be compatible with an Open World assumption. Therefore a second metric had to be developed in the thesis, described in section 5.3.3, which is designed to accommodate this assumption.

5.3.1 Representing triple data as feature vectors

Typically, when fitting a model, the data takes the form $D = \{\mathbf{x}(1), \ldots, \mathbf{x}(n)\}$, where $\mathbf{x}(i)$ is a p-dimensional vector, of measurements of each variable that are of interest. When dealing with data described as a set of triples with p predicates, it is not generally possible to represent the data with p-dimensional vectors, as each instance can simultaneously take multiple values per predicate. For example it is perfectly valid for a person to have the occupation actor and director at the same time.

Several methods have been described for converting triples to a form more suitable for analysis. Parundekar et. al. [15] split each instance into multiple p-dimensional vectors which cover all permutations of the values the instance has for each predicate. Nickel et.al.[54] represent the data as a three-way tensor $\mathcal{X}$ of size $n \times n \times m$ where the entry $\mathcal{X}_{i,j,k} = 1$ denotes that the triple consisting of the ith entity, kth predicate and jth entity exists.

For our purposes, the data is vectorised as follows. Each predicate and object pair is treated as a single feature numbered 1 to k, and for each instance, s_i, which appears as a subject in a triple is represented by a feature vector $\mathbf{x}(i)$, with $x(i)_k = 1$ if the triple with subject s_i and the kth subject-object pair exists. It should be noted that these vectors are *sparse* with most of the entries being 0. Vectors of this form can be stored in memory using significantly less space than non-sparse vectors. *The* notation $P(\langle p, o \rangle)$ is used for the probability that a given feature vector has the entry that corresponds to the attribute-value pair p and o equal to one.

5.3.2 Information Gain

Information Gain [55] (also known as Mutual Information or the expectation of the Kullback–Leibler divergence) is one of the most common metrics used to perform feature selection. IG is a form of entropy measure – a measure of the uncertainty of a variable. Essentially it allows estimation of the certainty of one variable when another is known. It is a powerful metric and forms the basis of one of the most popular decision tree learning algorithms, C4.5 [46] as well as the J4.8 algorithm [20]. For each feature, A, Information Gain measures the average decrease in the number of bits required to Huffman encode the class, C, of an instance if the value of A is known. It is defined as follows:

$$IG(C|A) = H(C) - H(C|A)$$

Where $H(C)$ is the entropy of C – the number of bits, on average, required to transmit a stream of values drawn from C's distribution:

$$H(C) = -\sum_{c} p(c) \log_2 p(c)$$

And *H(C|A)* is the conditional entropy of *C* given *A*

$$H(C|A) = \sum_{a} p(a) H(C|A = a)$$

$$= \sum_{a} p(a) \left(\sum_{c} p(c|a) \log_2 p(c|a) \right)$$

Using IG as the scoring metric requires a *closed world assumption* to be made– the assumption that any statement that is not explicitly said to be true is false. This is in contrast to the *open world assumption* used in the semantic web, which requires statements to be considered as simply unknown if they are not explicitly said to be false. It is possible that if additional information were known then this assumption that the statement is false could prove to be inaccurate.

A second problem with the IG metric is that while it will score features highly if their presence is a good indicator of an instance's class, it will also score a feature highly if its *absence* in the training set is a good indicator of the instance's class. Correspondences based on a negation are unlikely to generalise well in situations where a reasoner is used.

5.3.3 Beta-binomial class prediction metric

As IG is intended for use with a closed world assumption, in the research for this thesis a new metric was designed to operate with an open world assumption. This metric measures the probability that a properly value pair will provide the best CPV correspondence for a given class, *C* to a restriction class in the knowledge base KB.

In general, it is not possible to evaluate $\mathrm{CPV}_{\mathrm{MAX}}(C, KB)$ directly as it relies on the intersection and union of two sets that cannot be known completely - the instances of C and $C'' = C' |_{p=o}$.

We wish to maximize

$$J(C, C'') = \frac{| C \cap C'' |}{| C \cup C'' |} = \frac{| C \cap C'' |}{| C | + | C'' \backslash C |}$$

If C and C' are fixed, $J(C, C'')$ is maximised selecting p and o which maximizes $| C \cap C'' |$ and minimizes $| C'' \backslash C |$ - that is selecting the pair which maximize the size of $\{s \mid \langle s, p, o \rangle, \langle s, \texttt{rdf:type}, C \rangle \in \mathbb{K}\}$ and minimize the size of $\{s \mid \langle s, p, o \rangle, \langle s, \texttt{rdf:type}, C' \rangle \notin \mathbb{K}\}$

Looking at the problem from a different perspective, we wish to estimate $\hat{p}, \hat{o}$ which maximize the probability that given $i \in instances(C')$, then $i \in \text{instances}(C)$ if i has attribute $\hat{p}$ set to value $\hat{o}$ and $i \notin \text{instances}(C)$ if i does not not have $\hat{p}$ set to value $\hat{o}$.

$$\hat{p}, \hat{o} = \underset{p,o}{\text{argmax}} P(C \mid \langle p, o\rangle, C') * P(\neg C \mid \neg\langle p, o\rangle, C') \quad (1)$$

Where $P(C)$ is the probability that a given instance is a member of class C, and $P(\langle p, o\rangle)$ is the probability that for a given instance, s, the triple $\langle s, p, o\rangle \in \mathbb{K}$.

In practice, as C and $\langle p, o\rangle$ are defined in separate knowledge bases, there will be relatively few instances of C for which the value of $\langle p, o\rangle$ is known and $P(C \mid \langle p, o\rangle, C')$ can be more easily modelled by using the Bayesian transformations:

$$\begin{aligned} P(C \mid \langle p, o\rangle, C') &= \frac{P(\langle p, o\rangle \mid C, C')P(C)}{P(\langle p, o\rangle, C')} \\ &= \frac{P(\langle p, o\rangle \mid C, C')P(C)}{P(\langle p, o\rangle \mid C')P(C')} \\ &\propto \frac{P(\langle p, o\rangle \mid C)}{P(\langle p, o\rangle \mid C')} \end{aligned}$$

As $P(C)$ and $P(C')$ are constants Also, $P(\langle p, o\rangle \mid C, C') = P(\langle p, o\rangle \mid C)$ because $C \sqsubseteq C'$ which means all instances of C are instances of C'.

Similarly

$$\begin{aligned} P(\neg C \mid \neg\langle p, o\rangle, C') &= 1 - \frac{(1 - P(\langle o, p\rangle \mid C))P(C)}{(1 - P(\langle o, p\rangle \mid C'))P(C')} \\ &\propto 1 - \frac{(1 - P(\langle o, p\rangle \mid C))}{(1 - P(\langle o, p\rangle \mid C'))} \end{aligned}$$

Therefore, finding the solution to equation 1 is dependent on finding a suitable estimate for $P(\langle p, o\rangle \mid C)$ and $P(\langle p, o\rangle \mid C')$.

If $P(\langle p, o\rangle|C) = \theta_C$, then for a sample n instances, the likelihood of seeing k instances with $\langle p, o\rangle = 1$, can be estimated with the distribution $p(k \mid n, \theta_c) = {\theta_c}^k(1 - \theta_c)^{n-k}$. If we assume that θ is drawn from a $\text{Beta}(\alpha, \beta)$ prior then

$$\begin{aligned} p(\theta_c \mid n, k, \alpha, \beta) &= p(k \mid n, \theta_c)p(\theta_c \mid \alpha, \beta) \\ &= {\theta_c}^{k+\alpha+1}(1 - \theta_c)^{n-k+\beta-1} \\ p(\langle o, p\rangle|\mathrm{C}) = \mathbb{E}[\theta_c \mid k, n, \alpha, \beta] &= \frac{\alpha + k}{\alpha + \beta + n} \end{aligned}$$

This leaves a choice of parameters α and β. As it can be believed *apriori* that most possible triples do not exist, $\alpha = 1$ and $\beta = 2$ can be chosen which provides a prior distribution weighted towards 0.

5.4 Implementation

In order to evaluate the approach, an implementation was created in the Java programming language. Java SE 6 was chosen as there are mature libraries available for extracting information from RDF datasets, and for performing feature selection. The Jena API (version 2.6) was used to perform feature extraction from the RDF sources, and version 3.6 of the Weka data mining API was used to perform information gain feature ranking.

The basic steps which the implementation must perform are:

1. Convert the graph based data in the ontologies to a table with a row for each instance, and columns representing the features of these instances – in this case a Boolean value indicating if the instance is known to have a particular attribute and value
2. Rank the features
3. Convert the top N features back to attribute-value pairs and return them to a user.

One of the main requirements of our implementation is that there be multiple metrics available to perform correspondence detection. As we will be evaluating and comparing the different metrics it is desirable that our implementation not duplicate any functionality as this could potentially lead to differences in performance between the detection metrics due to bugs. Additionally from an object orientated design perspective it would be preferable to shield clients from the fact that the correspondence detection problem is being converted to a feature selection one.

To solve these issues the *Strategy* [10] design pattern was used. The Strategy pattern is used primarily when implementing a family of algorithms which are interchangeable. A secondary use of the strategy pattern is that it can be used to hide data that clients shouldn't know about. There are three participants in the strategy pattern:

Strategy - an interface common to all supported algorithms.

ConcreteStrategy - an implementation of the Strategy interface

Context - this object is configured with a ConcreteStrategy object and maintains a reference to it. It provides the Strategy with data and calls its methods when required.

In this implementation, the *CAVDetector* class acts as our context, and the interface *FeatureRanker* is our strategy. The CAVDetector class is described in detail in section 5.4.1, but in brief, a CAVDetector object contains a reference to a concrete instantiation the FeatureRanker interface. When called to perform correspondence detection, it converts graph based ontology data to a tabular form that the FeatureRanker can understand, and calls the

FeatureRanker's *rankFeatures()* method. The relationship of these classes is shown in figure 25. Two concrete classes were implemented with the FeatureRanker interface, IgFeatureRanker, which uses the information gain metric described in section 5.3.1, and BetaBinomialFeatureRanker, which uses the beta binomial metric described in section 5.3.3. The implementations of IgFeatureRanker and BetaBinomealFeatureRanker are described in more detail in sections 5.3.2 and 5.3.3 respectively.

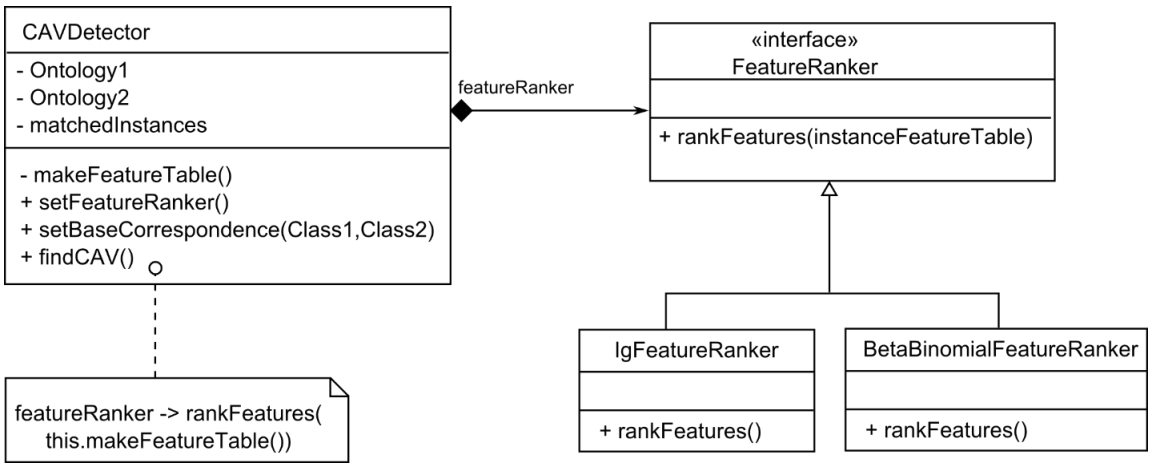


Figure 25. The CAVDetector and classes implementing the FeatureRanker interface.

In this implementation it is assumed that data contained in the ontologies can be accessed through SPARQL end points. The message sequence between a client, CAVDetector, Feature Ranker, and the ontology endpoint is shown in figure 26. Pseudo code for this sequence is as follows:

1. Instantiate CAVDetector object cavd,
2. Provide it with a reference to an OntologyEndpoint as well as a set of matched instances which will be used as evidence for detecting the CAV correspondences.
3. The client then instantiates an object implementing the FeatureRanker interface and passes a reference to the CAVDetector object.
4. For each underspecified correspondence – a one to one correspondence that the user suspects may form the base of a complex correspondence – the client passes the base correspondence to the CAVDetector.
   4.1. CAVDetector generates a feature table based on the instances of the classes in the base correspondence
5. The client calls the function findCAV()

5.1. The CAVDetector in turn passes the feature table to the FeatureRanker which ranks the features

6. The CAVDetector returns the attribute-value pairs that correspond to the top N features.

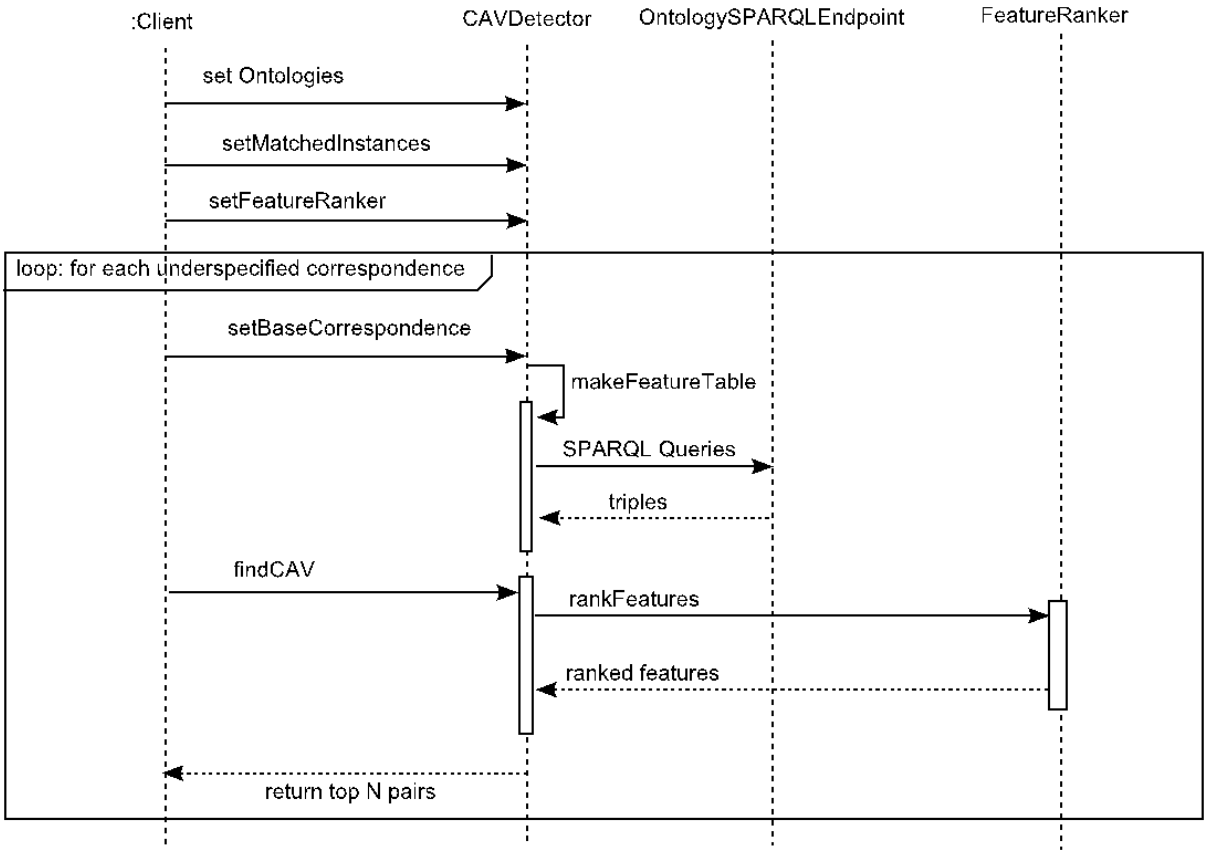


Figure 26. Messages passed between a client, the CAVDetector, FeatureRanker and ontology SPARQL endpoint.

5.4.1 CAVDetector

The CAVDetector class acts as the context participant in a *strategy* design pattern. This means that once a client has passed a reference to a FeatureRanker object to the CAVDetector, all the clients requests are received through this CAVDetector and are forwarded to the FeatureRanker.

The CAVDetector is responsible for gathering data from the ontology SPARQL end-point and converting it to a form that can be used by a FeatureRanker. It does this by creating a FeatureTable object for each base correspondence it is supplied with. The basic steps for

creating a FeatureTable for the base correspondence $C_1 \sqsubseteq C_2$ and the set **M** of matched instances, are as follows:

```
Find the training set T of all instances of C1 and C2 in M
Find all parameter value pairs in T and label them pv1..n
for-each instance t in T:
  add row v, where:
        if (t has PVi): vi = 1
        else: vi = 0
```

This produces a table with *n* columns representing the attribute-value pairs, and a row for each instance in the training set. To aid scalability, rows of feature tables are implemented as sparse binary vectors. These vectors only contain the indexes of non-zero entries. The task of ranking the attribute-value pairs is delegated to a FeatureRanker object which the CAVDetector holds a reference to. Once these pairs have been ranked, the CAVDetector returns the top N to the client which called it.

5.4.2 Feature Ranker

The FeatureRanker interface acts as the strategy participant in the strategy design pattern. This interface defines a single method – rankFeatures() Concrete implementations of this interface are responsible for performing the ranking of a feature table which has been created by a CAVDetector object. Two concrete implementations of the FeatureRanker were created: IgFeatureRanker and BetaBinomialFeatureRanker

5.4.2.1 IgFeatureRanker

IgFeatureRanker is a concrete implementation of the FeatureRanker interface. It acts as a ConcreteStrategy participant in the Strategy design pattern. This class ranks features representing attribute-value pairs using the Information Gain metric described in section 5.3.2. This ranking was implemented using the InfoGainAttributeEval class which is part of the Weka API. As shorthand, the use of IgFeatureRanker with the CAVDetector object will be referred to as FS1

5.4.2.2 BetaBinomealFeatureRanker

BetaBinomialFeatureRanker is a second concrete implementation of the FeatureRanker interface. This class ranks features using the method described in section 5.3.3. More

specifically it scores each column, $\boldsymbol{f_i}$ of the feature table by estimating $c * P(C_1 \mid f_i, C_2)P(\neg C_1 \mid \neg f_i, C_2)$. The value $c = P(C_1)/P(C_2)$ is unknown but constant across all features so does not affect rankings.

As a shorthand, the use of BetaBinomialFeatureRanker with a CAVDetector object will be referred to as FS2.

5.5 Summary of relationship between requirements, implementation and evaluation

This section explicitly states the relationship between the requirements, implementation and evaluation of the complex correspondence detection method. The first requirement **R1.1** was that the method be capable of detecting CAV correspondences. This was addressed in the implementation by developing a Java class, CAVDetector, capable of converting the CAV detection problem into a feature selection one, and then using a feature selection algorithm. This requirement was assessed in the evaluation described in chapter 6, where CAVDetector was used to detect CAV correspondences in DBpedia and YAGO2.

Requirement **R2** is that the method must be tolerant to missing data. This requirement was addressed in the design by using a common extensional approach, which analyses instance data statistically instead of using hard rules. This requirement is assessed first in the evaluation in chapter 6 which shows that the approach can tolerate levels of missing data found in real world LOD ontologies, and then in the evaluation described in chapter 8 which analyses the performance of the approach as a function of the level of missing data.

Requirement **R3** is that there be a choice of metrics for performing CAV detection. This was addressed in the implementation by using the Strategy design pattern. This design pattern allows different metrics to be implemented and used without having to make changes to other classes. Two different metrics were implemented, one based on information gain (see section 5.4.2.1) and one based on density estimation (see section 5.4.2.2). These metrics were tested in the evaluation described in chapter 8.

Requirement **R4** was that the detection approach should minimise the amount of matched instance data needed to perform detection. In the implementation this was addressed by using the information gain metric which is a popular metric in applications where small amounts of data are available. For example in tree building where leaf nodes contain small numbers of training data. This requirement was assessed in chapter 7 where it is shown that similar

performance could be achieved using a small sample of matched instances as could be achieved using all available instances.

Finally requirement R5, that the approach be scalable to large datasets such as DBpedia and YAGO2 was addressed in the implementation by using sparse vectors to store data, and is reinforced by the ability of the approach to perform well when only small sets of instance data are used. This requirement was not assessed directly, however in the evaluation carried out in chapter 6 the approach had no difficulty scaling to use on version 3.6 of DBpedia and the September 2012 version of YAGO2.

The relationship between requirements, implementation and evaluation are shown in table 3.

Table 3. Relationship between requirements, design, and evaluation.

Requirement	Implementation decision	Evaluation chapter
R1.1	Convert problem to feature selection, use feature selection algorithm.	6
R2	Use of statistical inference provides greater tolerance to missing or incorrect data than use of hard rules	6, 8
R3	Use of the Strategy design pattern means metrics used in the implementation can be changed without altering other parts of the code.	7
R4.1	Use of Information Gain metric. Second metric developed was shown to be comparable or outperform IG for similar numbers of instance data	7, 8
R5	Use of sparse vectors to represent data, combined with ability to perform well with only small sets of instance data	6[18]

In summary, the implementation of this approach follows the strategy design pattern, which allows several different metrics to be used to perform CAV correspondence detection. The CAVDetector class acts as the context participant in the design. Its responsibility is to act as

[18] The purpose of the evaluation in chapter 6 was not expressly to test the scalability of the approach, however the datasets used are among the largest of the real world ontologies.

the interface to a client and insulate them from the method used to solve the problem. When asked to perform CAV detection, it converts the ontological data from a graph form to a tabular form which is then processed by an object implementing the FeatureRanker interface. Two concrete classes implementing the FeatureRanker were developed, one based on the information gain metric which makes a closed world assumption about the data it uses, and a second class BetaBinomialFeatureRanker, which uses density estimation to perform ranking and makes an open world assumption about the data it uses.

Chapters 6-9 describe a series of evaluations carried out using this implementation. As a shorthand in these evaluations use of the IgFeatureRanker with CAVDetector will be referred to as FS1, and use of the BetaBinomialFeatureRanker with CAVDetector will be referred to as FS2.

6 Detecting Class Restriction Correspondences in DBpedia and YAGO2

The research question for this thesis is: *To what extent is it possible to use samples of accurately matched instance data to detect complex correspondences between classes in their related ontologies*. Published results of state of the art methods using matched instances [15] use large volumes of instance, but it is not the case that such large volumes of matches are available. The purpose of this evaluation is to show that it is possible to find meaningful Class by Attribute Value and Class by Attribute Existence correspondences using only small sets of matched instances. There are two ways that a complex correspondence detection method can help, either by detecting the correspondence directly, or by narrowing the search space down to a small number of possibilities. When detecting correspondences directly it is necessary to determine both the most likely correspondence and that this correspondence is really occurring. Narrowing the search space down requires that the correspondence be ranked highly, but can defer to a human to determine if the correspondence is real.

This evaluation will test the ability of the FS1 method – which uses the Information Gain (IG) metric to rank possible complex correspondences – to detect correspondences directly by not only ranking the correct correspondence in 1st place, but also giving it a sufficiently high IG score to be confident that the correct correspondence has been selected. The ability of the FS1 method to narrow the search space down is assessed by its ability to rank the correct correspondence in the top five, with no concern for the magnitude of the IG scores.

This evaluation serves as a proof of concept, and was carried out relatively early in the development of the thesis. It demonstrates the feasibility of the FS1 process of converting ontological data to a set of features in tabular form and then using the Information Gain metric to rank these features. The choice of 15 matched instances in the training sets is somewhat arbitrary, and the effects of training set size are explored in subsequent experiments.

5.6 Hypothesis

This experiment tests the following hypotheses:

- Given an initial elementary correspondence where one class is more broad then the other, and a small sample of instances of the classes, the FS1 correspondence detection method

can rank the attributes and values that can be used to narrow the scope of the broader class to more closely correspond with the more narrow class.

- By using a threshold on the information gain score, it is possible to automatically detect complex correspondences by selecting the highest ranked attribute-value pair if they score above the threshold.

5.7 Methodology

For this experiment the FS1 complex correspondence detection method is required to refine a selection of underspecified, elementary correspondences between a class in a target ontology and a class in a source ontology. To do so it must rank the attributes of the source class named in each underspecified correspondence for attribute's ability to specify a Class by Attribute Value correspondence which more accurately describes the relationship between the source and target classes.

This evaluation assumes that underspecified 1-1 correspondences are known between a source and target class, the Source class is more broadly scoped than the Target class, and that it is necessary to chose an attribute and corresponding value which can be used to restrict the scope of the Source class to make it equivalent to the scope of the Target class.

As the entire population of matched instances is not used to estimate the correct attribute and value, this estimation is subject to vary depending on the particular instances that are used. Therefore in this evaluation, the correspondence detection process is performed several times with randomly chosen sets of matched instances, and the variation in performance is observed. Two scores are recorded: the Information Gain (IG) score given to each attribute-value pairing, and the ranking of the gold standard attribute-value pair when they are ordered by IG. Ideally, the IG score should remain consistent for each attribute-value pair regardless of the matched instance sample used, and the gold standard attribute-value pair should be consistently ranked highly.

In this evaluation matched instance sets consist of 20 instances taken from the intersection of the Source and Target classes. As stated previously, this is a somewhat arbitrary number, and was chosen because if given two ontologies with over-lapping instance data, but no previously matched instances, it would not be unreasonable to ask a human performing ontology alignment to manually find 15 examples individuals of a class which exist in both ontologies.

There are five Test Cases, each case consisting of an underspecified correspondence between a Source and Target Class, and a gold standard Class Restriction Correspondence which describes the relationship between the classes. In the underspecified correspondences the Source Class has a broader scope than the Target Class. In four of the test cases, the gold standard correspondence is a Class by Attribute Value which specifies an attribute and corresponding value that will restrict the scope of the Source class to be equivalent to the scope of the Target class. The fifth test case has a Class by Attribute Existence gold standard correspondence which only specifies an attribute which restricts scope but no specific value for the restriction.

Important issues when carrying out the evaluation are the metrics used to score the selection process, and the creation of a test set of underspecified correspondences with corresponding gold standard CPV correspondences.

5.7.1 Procedure

The procedure followed for this evaluation is as follows: For each test class the Source and Target class are identified, and these are used to create 100 input matched instance sets, each with 15 randomly chosen individuals which are instances of both the Source and Target class (the positive entries) and 50 individuals which are instances of just the Source Class (the negative entries).

As DBpedia contains YAGO2 classifications for each of its individuals, selecting positives is achieved by using the following SPARQL query, and copying the results into an array.

```
SELECT ?thing
WHERE {
  ?thing a dbpedia-owl:SourceClass
  ?thing a yago:TargetClass
}
```

Random samples of the instances can then be generated by shuffling the array and selecting the top 15 entries. The negative entries for the training sets are generated in a similar manner using the following SPARQL query

```
SELECT ?thing

WHERE {  ?thing a dbpedia-owl:SourceClass }
```

This set could potentially include positive entries, however in a real world application the set of negatives could also potentially contain positives, and out detection method should be able to tolerate this.

Once each training set has been created, it is used to estimate the best attribute-value pairs to complete the underspecified correspondence in the test case using the FS1 method. The scores of every attribute-value pair and the rank of the gold standard attribute-value pair are recorded.

When this has been completed 100 times the mean and variance of the attribute-value pairs and the mean reciprocal rank (MRR, see section 6.2.2) of the gold standard attribute-value pair are calculated. This procedure is illustrated in Figure 27.

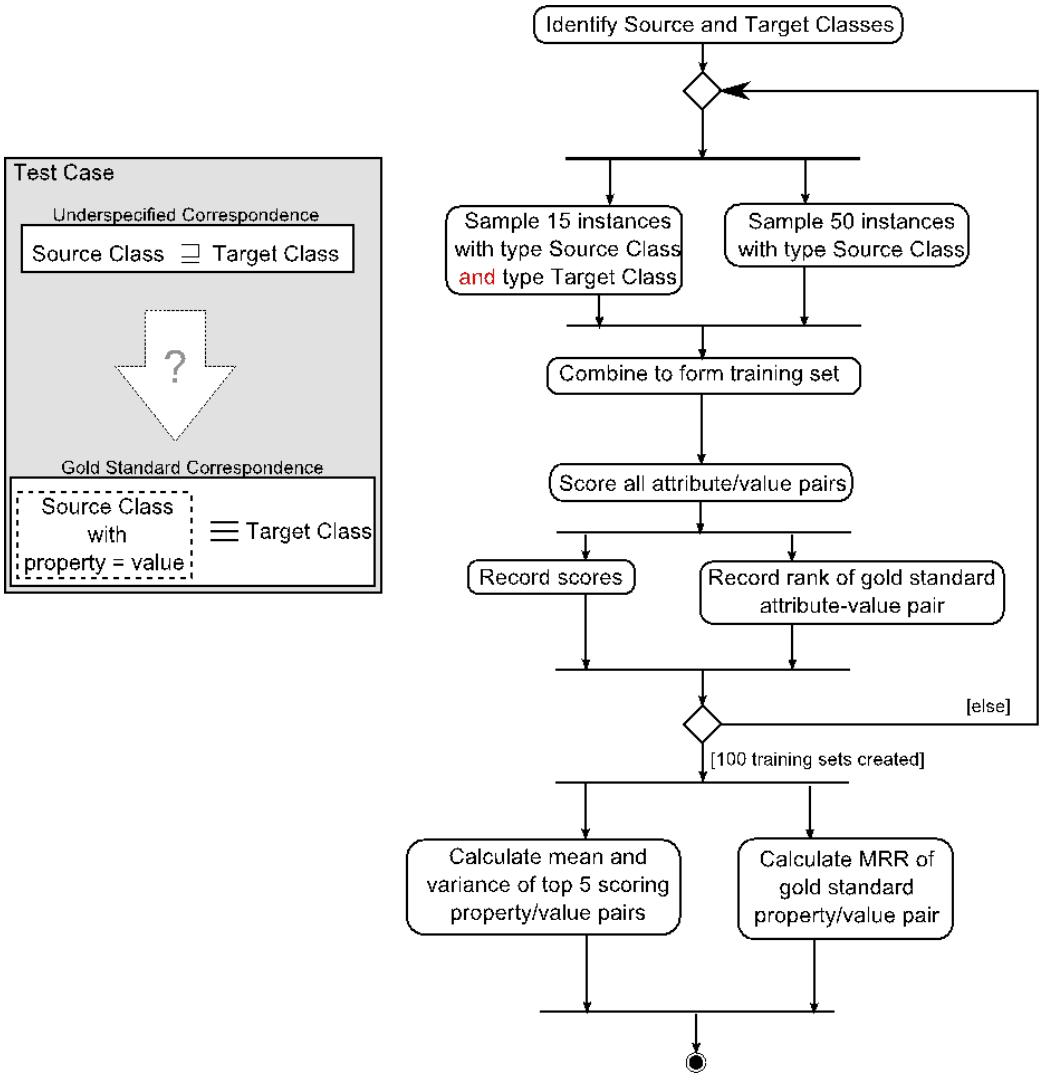


Figure 27. An activity diagram of the process followed in experiment 1.

5.7.2 Metrics

The purpose of this evaluation is to measure the sensitivity of the FS1 method to training set size. To do this, *random* samples of training data are used. As these training sets are generated randomly, the performance of FS1 will vary with each training set. Therefore the metric used must be able to measure mean performance over several training sets.

The metric used is the Mean Reciprocal Rank, which is the mean of the multiplicative inverse of the rank of the gold standard. For a set of training sets Q_i, $i=1..n$, each of which produces $rank_i$,

$$MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{rank_i}$$

With this metric, a perfect detector which ranks the gold standard in first position every time would receive an MMR score of 1. A detector which ranks the gold standard in first place half the time and in second place otherwise would score 0.75. As performance degrades, MMR score will approach zero.

The effectiveness of the FS1 method to detect correspondences directly is measured using a Receiver Operating Characteristic (ROC) curve. This curve compares the true positive versus the false positive rate for determining that the highest ranked attribute-value pair is the gold standard, as the threshold on the IG score is varied. The area under this curve is used to determine how well the differentiation works. Purely random results would produce an area of 0.5 and a perfect classifier would produce an area of 1.

5.7.3 Test Case Selection

5.7.3.1 Ontology selection

These cases are based on correspondences between the YAGO2 and DBpedia ontologies. In these correspondences DBpedia acts as the source and YAGO2 as the target. These ontologies were selected as both been created from the same source, Wikipedia, and therefore cover overlapping domains. This similarity means reasonable to assume that there are many correspondences between the ontologies.

Despite being created from the same source have differing structures. YAGO2 features a rich hierarchy with many narrowly defined classes. DBPEDIA has a much more shallow hierarchy, with broad classes. As YAGO2 has many more classes than DBpedia, it can be

expected that there are many classes in YAGO2 which do not correspond directly with a class in DBpedia. Much of the information that is modelled in the class hierarchy of YAGO2 is modelled using attributes in DBpedia. Therefore, it can be expected that there are many Class by Attribute Value correspondences occurring between a broad DBpedia class and a more narrowly scoped YAGO2 class.

Another important property of DBpedia and YAGO2 is that in both ontologies, the URIs for instance data have been formed using the URL of the Wikipedia entry on that instance. This means that matching instances between the ontologies is trivial.

1.1.1.1. Selection of test case correspondences

Five test cases were created, which represent complex correspondence detection tasks of varying difficulty. Each test case correspondence consists of a narrowly scoped YAGO2 class and a more broad DBpedia class. For each correspondence there is a gold standard attribute and value pair which specify a CPV correspondence. This attribute-value pair was selected so that the Jaccard index of the set of instances of the target class and the set of instances of the source class restricted by the attribute and value is maximised. . The Jaccard index of two sets, A and B, is defined as $J(A,B) = |A \cap B| / |A \cup B|$. Two identical sets will have a Jaccard index of 1, and sets which share no elements will have a Jaccard of 0. It should be noted that the Jaccard index cannot usually be used to detect complex correspondences as it requires knowledge of *all* instances which are common to the classes which is not typically known.

Five classes from YAGO2 (*Actor*, *PeopleFromToronto*, *Musician*, *Politician*, and *Director*) were selected. Each of these have an underspecified correspondence with the class *Person* in DBpedia. These YAGO2 classes were selected so that the CAV correspondences for some are more difficult to detect than others. Two factors were considered when assessing the difficulty of detecting a correspondence. Firstly, if the Jaccard index of the set of instances of the source class and the set of instances of the restricted class is low then the correspondence is considered difficult to detect. Secondly the detection can be confused if there are several combinations of attribute and value which produce a similar Jaccard score.

In addition, classes were selected so that there would be sufficient instances of each class to select multiple samples of 15 instances the class and be sure to a reasonable certainty that each sample does not share a large number of instances with the others. The YAGO2 classes selected and their instance counts are summarised in table 4.

.

Table 4. Test case classes.

Class	Instances
yago:Actor	29599
yago: PeopleFromToronto	1058
yago:Musician	1585
yago:Politician	1044
yago:Director	619

Two classes with easily detected complex correspondences were selected, describing actors and people from Toronto. The YAGO2 class *PeopleFromToronto* corresponds with the DBpedia class of *Person* scoped by the attribute *dbpedia-owl:birthplace* set to *dbpedia:Toronto*. This is the easiest to detect test case, as these sets of instances overlap perfectly, and have a Jaccard index of 1. The class Actor exists in both YAGO2 and DBPedia, which is a special case of scoping *dbpedia-owl:Person* using the attribute *rdf:type* set to *dbpedia-owl:Actor*. These sets overlap reasonably well with a Jaccard index of 0.62.

Two more difficult to detect correspondences are provided by the YAGO2 classes *Musician* and *Politician*. This is because the Jaccard measures for the classes in their respective correspondences are low. Both of these classes correspond to the class *dbpedia-owl:Person* scoped by the attribute *dbpedia-owl:occupation* set to the values *dbpedia:Musicain* and *dbpedia:Politician* respectively. The Jacard measure for musicians is 0.13, and the measure for politicians is 0.1. In addition, the choice of correspondence for musicians is made more difficult by the ability to alternatively define a complex correspondence using *rdf:type* set to *dbpedia-owl:MusicalArtist*. While these choices may appear very similar from a semantic point of view, conditioning *on rdf:type=dbpedia:MusicalArtist* produces a set of instances which only have a Jaccard index of 0.04 with *yago:Musician*.

The YAGO2 class *Director* was selected as the fifth test case. Unlike the previous complex correspondences which are examples of Class by Attribute Value, *yago:Director* corresponds with *dbpedia-owl:Person* scoped by the attribute *dbpedia-owl:directorOf* set to *any* value, which is an example of the Class by Attribute Existence correspondence pattern. The Jaccard index for this correspondence is 0.05. The conditions for each of these complex correspondences are summarised in table 5.

Table 5. Correspondence Conditions.

	YAGO Class	Condition on dbpedia-owl:person			Jaccard
1	yago:PeopleFromToronto	dbpedia-owl:birthPlace	=	dbpedia:Toronto	1
2	yago:Actor	rdf:type	=	dbpedia-owl:Actor	0.62
3	yago:Musician	dbpedia-owl:occupation	=	dbpedia:Musician	0.13
4	yago:Politician	dbpedia-owl:occupation	=	dbpedia:Politician	0.1
5	yago:Director	dbpedia-owl:director	=	[any value]	0.05

5.8 Results

On running the evaluation, test cases 1 and 2, (people from Toronto and Actors) with the gold standard attribute-value pairs for each case receiving an MRR of 1. Test cases 3, 4 and 5 (Musicians, Politicians and Directors) scored an MRR of 0.66, 0.27 and 0.67 respectively. These results are summarized in table 6.

Table 6. Mean reciprocal rank for each test case.

Test Case	Mean Reciprocal Rank
1	*1*
2	*1*
3	*0.66*
4	*0.27*
5	*0.67*

For test case 1 (people from Toronto) the highest ranked attribute value pairs make a clear progression on the attribute *dbpedia-owl:bornIn* with values Toronto, Ontario and Canada. The mean and variance of the attribute-value pair scores are shown in figure 28. Test case 2 (Actors) shows a less logical progression in the attribute-value ranks, but there is a clear distinction between the score of the highest ranked pair, and the lower ranked pairs. The scores for this test case are illustrated in **Error! Reference source not found.**figure 29. The variance of the IG scores is low enough that the gold standard can be expected to score higher than the other possible attribute-value pairs.

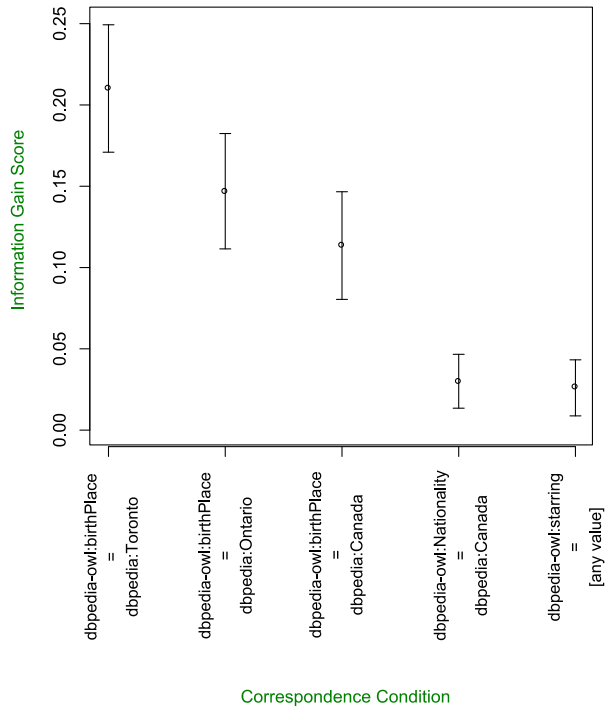


Figure 28. Top 5 ranked conditions for refining yago:PersonFromToronto ⊑ dbpedia-owl:Person. Bars show mean Information Gain score to one standard deviation.

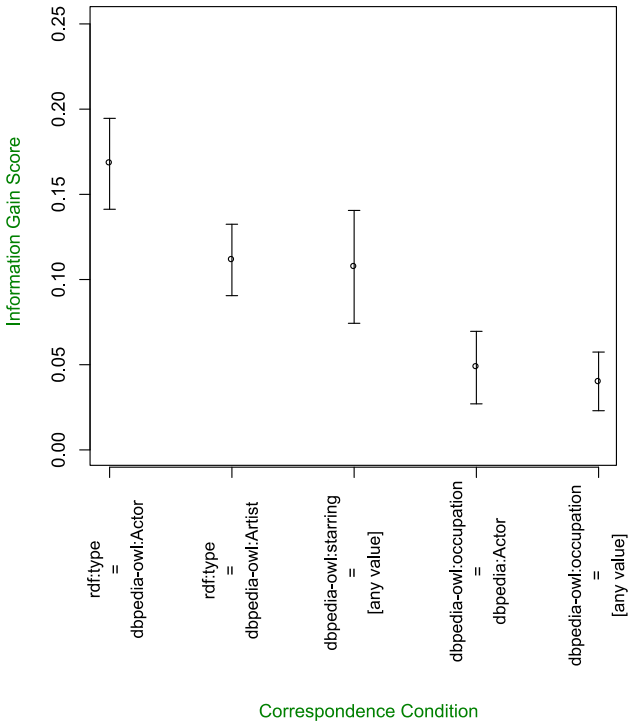


Figure 29. Top 5 ranked conditions for refining yago:Actor ⊑ dbpedia-owl:Person. Bars show mean Information Gain score to one standard deviation.

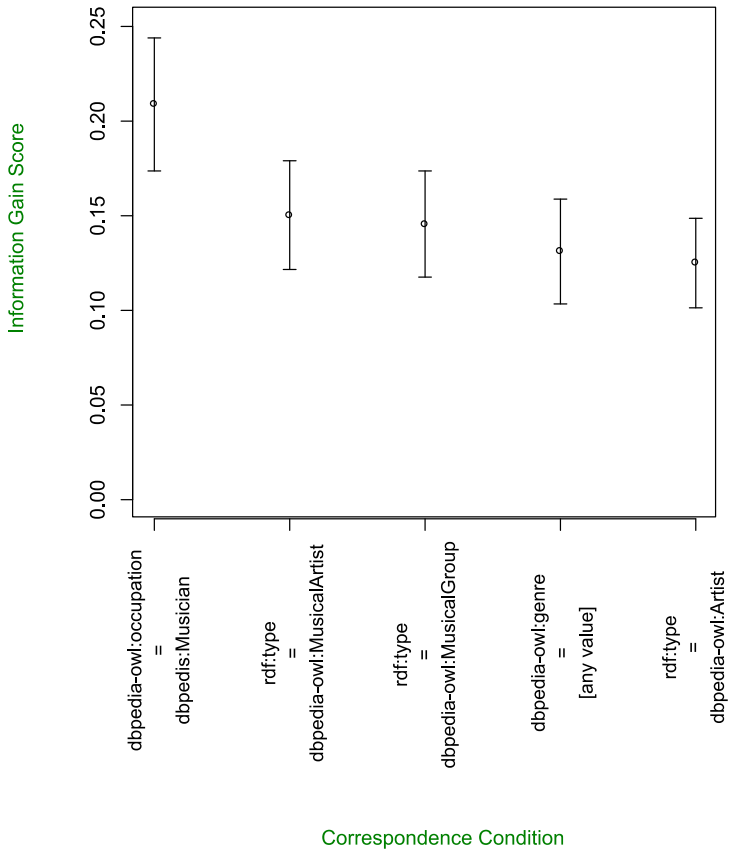


Figure 30. Top 5 ranked conditions for refining yago:Musician ⊑ dbpedia-owl:Person. Bars show mean Information Gain score to one standard deviation.

As can be seen in figure 30, the attribute-value pair *dbpedia-owl:occupation = dbpedia:Musician* scores highest on average. The pairs *rdf:type = dbpedia-owl:MusicalArtist* and *rdf:type = dbpedia-owl:MusicalGroup* also score highly, but there is a clear difference in the scores.

Figure **Error! Reference source not found.**31 shows the mean score for each of the attribute-value pairs in test case 5 (Directors). Here the gold standard scored highest on average, but the difference in the scores is less obvious, but there does appear to be a somewhat of a downward trend in the mean score for the attribute-value pairs.

Figure 32 shows the mean score for each of the attribute-value pairs in test case 4 (politicians) while the gold standard attribute-value pair for this test case did score highest on average, there is no significant difference between the scores of the top five ranked pairs. This means that for a random sample of *yago:Politician* instances, the probability of the gold standard pair being ranked highest is lower than for the other test cases. Automated detection of this correspondence is not feasible using this method, but it is still useful for informing a human who would make a decision on the correct correspondence.

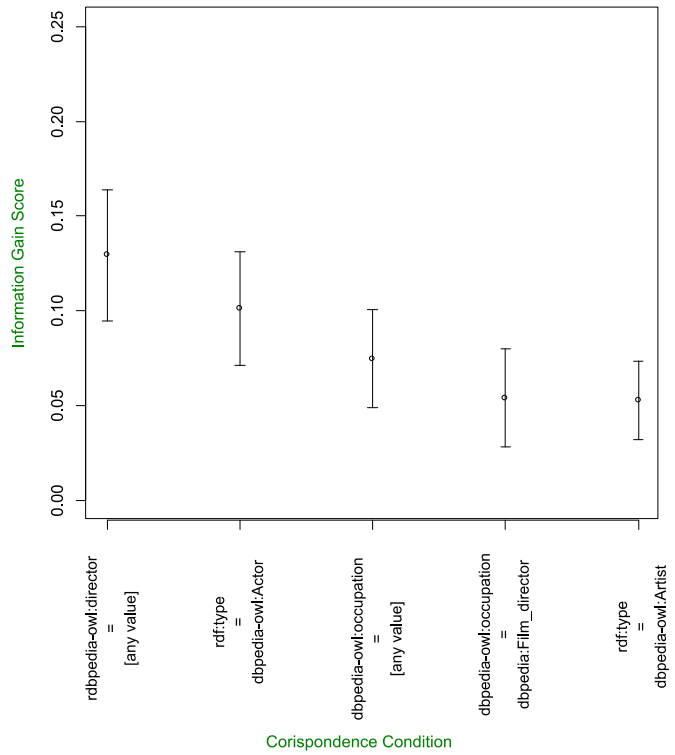


Figure 31. Top 5 ranked conditions for refining yago:Director ⊑ dbpedia-owl:Person.

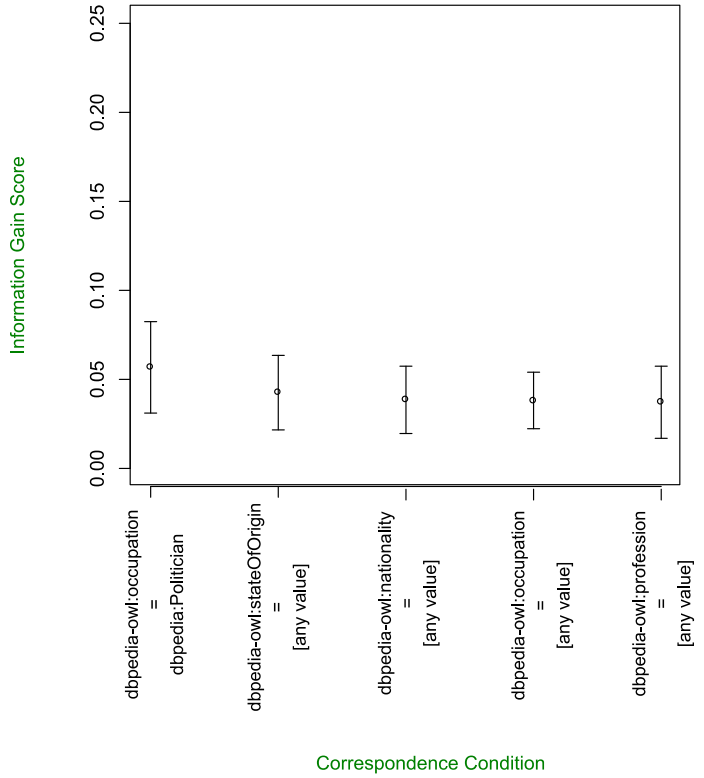


Figure 32. Top 5 ranked conditions for refining yago:Politician ⊑ dbpedia-owl:Person.

Figure 33 shows a ROC curve for using the IG score of the top ranked attribute-value pair to determine if the ranking is correct. The area under this curve is 0.78.

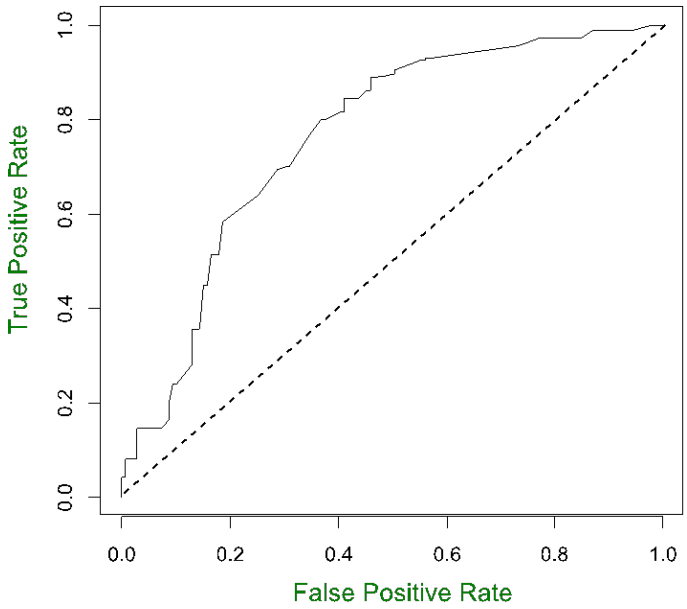


Figure 33: ROC curve for using the IG score of the top ranked attribute-value pair to determine if the ranking is correct.

5.9 Analysis

For test case 1 (people from Toronto) the highest ranked attribute-value pairs make a clear progression on the attribute *dbpedia-owl:bornIn* with values Toronto, Ontario and Canada – covering the city, state and country respectively, that a person from Toronto would be born in. There is then a clear separation between these scores and the score for the next highest attribute-value pair – *dbpedia-owl:nationality = dbpedia:Canada*. The mean and variance of the attribute-value pair scores are shown in figure **Error! Reference source not found.**28. Test case 2 (Actors) shows a less logical progression in the attribute-value ranks, but there is a clear distinction between the score of the highest ranked pair, and the lower ranked pairs.

The attribute-value pair *dbpedia-owl:occupation = dbpedia:Musician* scores highest on average. The pairs *rdf:type = dbpedia-owl:MusicalArtist* and *rdf:type = dbpedia-owl:MusicalGroup* also score highly, but there is a clear difference in the scores.

In test case 4 (Politicians), while the gold standard attribute-value pair for this test case did score highest on average, there is no significant difference between the scores of the top five ranked pairs. This means that for a random sample of *yago:Politician* instances, the probability of the gold standard pair being ranked highest is lower than for the other test

cases. Automated detection of this correspondence is not feasible using this method, but it is still useful for informing a human who would make a decision on the correct correspondence.

For the overall problem of determining if a top ranked correspondence is the gold standard, the area under the ROC curve of 0.78 shows that the FS1 method does perform significantly better than random. Still, this performance is not excellent. One problem is that a simple cut on the IG score of the highest ranked attribute-value pair, no consideration is given to how highly other pairs scored. A more advanced test might show improved results by comparing the scores of the top N ranked attribute-value pairs.

5.10 Conclusions

This evaluation shows that for cases where the data clearly supports a Class by Attribute Value restriction correspondence, then a small sample of 15 matched instances can provide enough information to detect the correct class restriction correspondence between the ontologies. In cases where the data supports the correspondence less clearly, performance is still good, but the FS1 method is more suited to reducing the search space for a human user than for detecting the complex correspondence directly.

This evaluation serves as an initial proof of concept showing that small training sets can still produce meaningful results. Further evaluations in this thesis determine more precisely the effects of training sample size. In addition one metric (IG) was used to perform ranking in this evaluaton; chapter 8 will describe an evaluation where a second metric was compared with IG.

6 Detection of complex correspondences with samples of instance data

The previous evaluation served as a proof of concept, demonstrating that for a small selection of underspecified correspondences between the DBpedia and YAGO2 ontologies, the FS1 correspondence detection method could be used to narrow down the potential complex correspondences to selecting from a list of five or less. In that evaluation the use of the word "small" to describe the training set size was somewhat arbitrary, with 15 matched instances in each set. The purpose of the evaluation described in this section is to establish more clearly what effect training set size has on the detection results.

6.1 Hypothesis

When using the FS1 method to detect that a Class by Attribute Value (CAV) correspondence exists between two classes, there is some sample size of shared instances that will produce the same true positive rate ($\varepsilon \pm 0.01$) as if provided all shared instances.

6.2 Methodology

This evaluation assumes somewhat idealised conditions - that for each class, C_s, in the first ontology a Class by Attribute Value complex correspondence exists which maps the scope of C_s perfectly to a class C_t in the second ontology. This correspondence takes the form $C_s|_{p=x} \equiv C_t$ – in other words: every instance of class C_s with attribute p set to value x, is also an instance of class C_t. It is also assumed that the subsumption relation $C_t \sqsubseteq C_s$ is known *a priori*. For each of the classes in the first ontology the correspondence detection method should be capable of finding the values of p and x necessary for each correspondence.

The purpose of this evaluation is to test if in the case where only small sets of shared instances are known it is still possible to detect correspondences almost as well as when all shared instances are known. The feature selection method used to pick the attribute and value is dependent on having a sample of instances which belong to both classes in the correspondence. Often however, knowledge of the entire set of shared instances will be incomplete, and it is possible that relatively few of the instances will be known – in this case it is important that the feature selection method still work.

6.2.1 Procedure

The procedure followed for this evaluation is as follows: For each test class the *Source* and *Target* class are identified, and these are used to create 100 training sets, each with N randomly chosen individuals which are instances of both the Source and Target class (the positive entries) and 300 individuals which are instances of just the Source Class (the negative entries). As with the evaluation described in chapter 6, these individuals are selected by first using a SPARQL SELECT query to find all individuals which meet the criteria, and then sampling at random from these individuals.

For each training set the FS1 detection method is used to rank all attribute-value pairs, if the gold standard attribute-value pair for the test case is ranked first, this is counted as a true positive (TP), otherwise a false positive (FP) is recorded. Once this procedure has been carried out 100 times for each test case the TP and FP rate at N is calculated. N is then reduced by 5 and the TP and FP rates re-evaluated. This is repeated with N ranging from 50 to 5. The procedure is illustrated in figure 34. To provide additional resolution the process was carried out a second time with N ranging from 5 to 1, decremented by 1 each time.

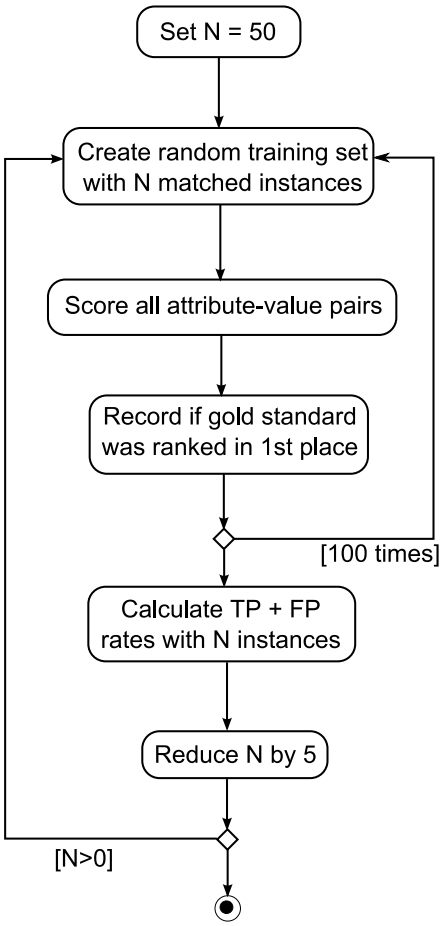


Figure 34. Evaluation procedure.

6.2.2 Metrics

The purpose of this evaluation is to measure the ability to select the optimal attribute-value pair needed to specify a Class by Attribute Value correspondence as the training size used is varied. The primary metrics used to measure this are the true positive (TP) and false positive (FP) rates. The TP rate applies to underspecified correspondences which can refined to CAV correspondences. It measures the proportion of times an attribute-value pair matching the gold standard is selected. The false positive rate applies to underspecified correspondences which *cannot* be refined to CAV correspondences. It measures the proportion of times that the detection method spuriously reports that a score for an attribute value pair for one of these correspondences that is higher than the lowest scoring TP case. These rates are bounded below by 0 and above by 1. A perfect classifier will have a TP rate of 1 and a FP rate of 0. A special case of the TP rate, denoted TP_{lim}, is also used. This is the True positive rate for the detection method when a sample of *all* matched instances are used.

Additionally the Jaccard measure is used to evaluate similarity of two classes, C_a and C_b.

$$J(C_a, C_b) = \frac{|instances\ with\ type\ C_a\ and\ C_b\ |}{|instances\ with\ type\ C_a\ or\ C_b\ |}$$

6.2.3 Test Case Selection

This evaluation requires a set of gold standard complex correspondences to evaluate the effectiveness of the detection method. As was the case in the evaluation in chapter 6, these correspondences consist of pairs of classes – which form an underspecified correspondence – and an attribute value pair which fully describes the complex correspondence between the classes.

There are several potential sources for a set of gold standard complex correspondences. A promising option might seem to be to select two ontologies with overlapping domains of interest, such as DBpedia and YAGO2, and construct complex correspondences between them by hand. There are several reasons why this approach would not be appropriate. Firstly, these ontologies are large and it would be difficult to find all the possible complex correspondences by hand. There is often a choice of several properties or values to use in a complex correspondence that can *appear* to be equally valid, even to an experienced ontology matching expert. From a semantic point of view the words used in the possible CAV correspondences may be similar, but a restriction using one of these properties may include

different instance data. For example to reduce the scope of the class *dbpedia-owl:Person* to that of *yago:PeopleFromPhiladelphia,Pennsylvania* it is not obvious if the attribute *dbpedia-owl:birthplace* or *dbpedia-owl:hometown* should be used. When similar attributes like this exist, it makes it difficult to objectively say which one should be used in a gold standard complex correspondence. Thirdly, without performing an exhaustive search, it is not possible to say that *no* complex correspondence exists for a given class, it is only possible to say that it is unknown.

In order to avoid these issues, an artificial target ontology was created with 50 classes which are constructed to have a known, best CAV correspondence with a class in the DBpedia ontology, and 50 classes which are designed to have *no* CAV correspondence with *any* class in DBpedia. The target ontology contains a flat hierarchy with a class *t:Person* which is defined to be equivalent to *dbpedia-owl:Person*, and 100 additional classes which are all direct OWL subclasses of t:Person. The hierarchy of this ontology is shown in figure 35. The 50:50 split between positive and negative cases might not accurately reflect the true split, but as the correct ratio is unknown this ratio was selected for simplicity. A higher proportion of negative cases would likely increase the level of false positives. As the use case for the detection approach is for a human user to approve correspondences suggested by the detection process, false positive rate is not as great a concern so it should not affect the conclusions that can be drawn from the evaluation.

Fifty of the classes were defined in the following manner which ensures CPV correspondences with DBPedia: For each new class, a attribute-value pair, *(p,x)*, was selected at random from the DBPedia ontology. If there was a minimum of 50 instances of *dbpedia-owl:Person* with attribute *p* set to value *x* then these instances are considered to form a new class. Fifty instances of each class are required as otherwise there would be insufficient numbers available to create a training set.

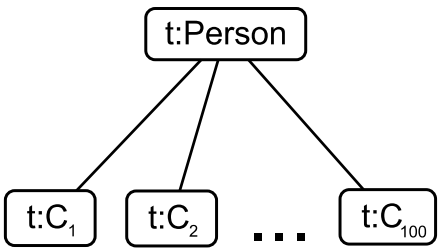


Figure 35: The hierarchy of the target ontology

Additionally when creating new classes, ones which were too similar to existing ones were rejected. Any new class with a Jaccard measure over 0.9 with an existing class was rejected, and a new atttribute value pair was selected instead. This step was performed to ensure that there was more diversity in the test set. This should not bias results, in favour of the FS1 detection method, as each test case is assessed by the detection algorithm separately and confusing test cases is not an issue. This step is to help create a wider variety in the test cases.

As each of these 50 classes were defined using an attribute-value pair to narrow the scope of the DBpedia class Person, by definition there is a CAV correspondence from dbpedia-owl:Person to the class. The attribute-value pair used to define the class is, obviously, the pair that defines the best CPV for the correspondence. The full process for creating these classes is illustrated in figure 36.

A further 50 classes were defined which have no CAV correspondence. These classes were not defined using an attribute-value pair. Instead, random samples of instances of *dbpedia-owl:Person* were selected, and a class was defined for each sample as that which contained the sample. Therefore, there should be no attribute-value pair which can be used to define a CAV correspondence for this class.

Version 3.7 of the DBPEDIA ontology was used for the evaluation. This was hosted on a Virtuoso Open Source server. Sampling of instances was performed using SPARQL queries to return all appropriate instances, storing the URI for each instance in a list, randomising the list and then selecting the top N. Where N is the number of instances required for the sample.

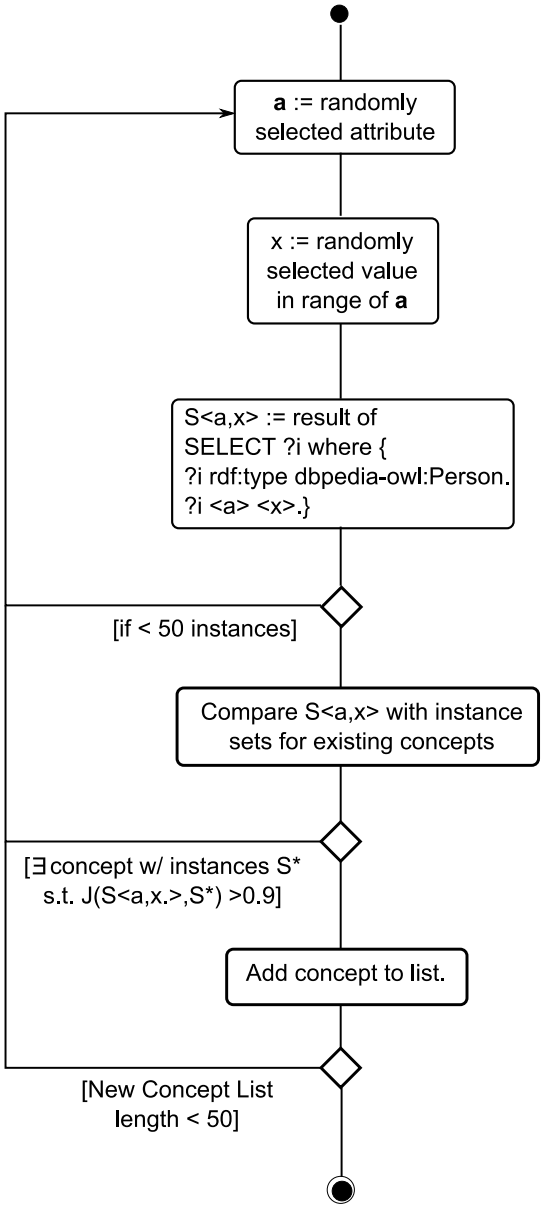


Figure 36. The process used to create the 50 classes with known CPV correspondences.

6.3 Results

Error! Reference source not found. Figure 37 summarises the results of the evaluation, showing the TP and FP rate as a function of N, the number of matched instances used. The range of this graph has been limited to $1 \leq N \leq 20$ as there is little insight to be gained from the values of N greater than 20.

As can be seen, the FP rate – detecting CAV correspondences where there are none – very quickly converges to zero when given three or more training instances. The TP – the rate at which the FS1 detection method finds the correct attribute-value pair – also improves very quickly as a function of training set size. Five instances is sufficient for a TP rate of 0.86, and

increasing the training set size only provides gradual improvements. Twenty instances provides a TP rate of 0.9 with a rate of 0.91 being achieved when all 440711 available instances are used in the training set.

6.4 Analysis

As expected, the detection rate increased as a function of the training set size, and there were demising returns on the increase in performance as the training set size grew. The surprising result was how quickly the performance converged and the strict monotonic increase that was observed. A training sample of only five instances produced a true positive rate of 0.86 and no false positives.

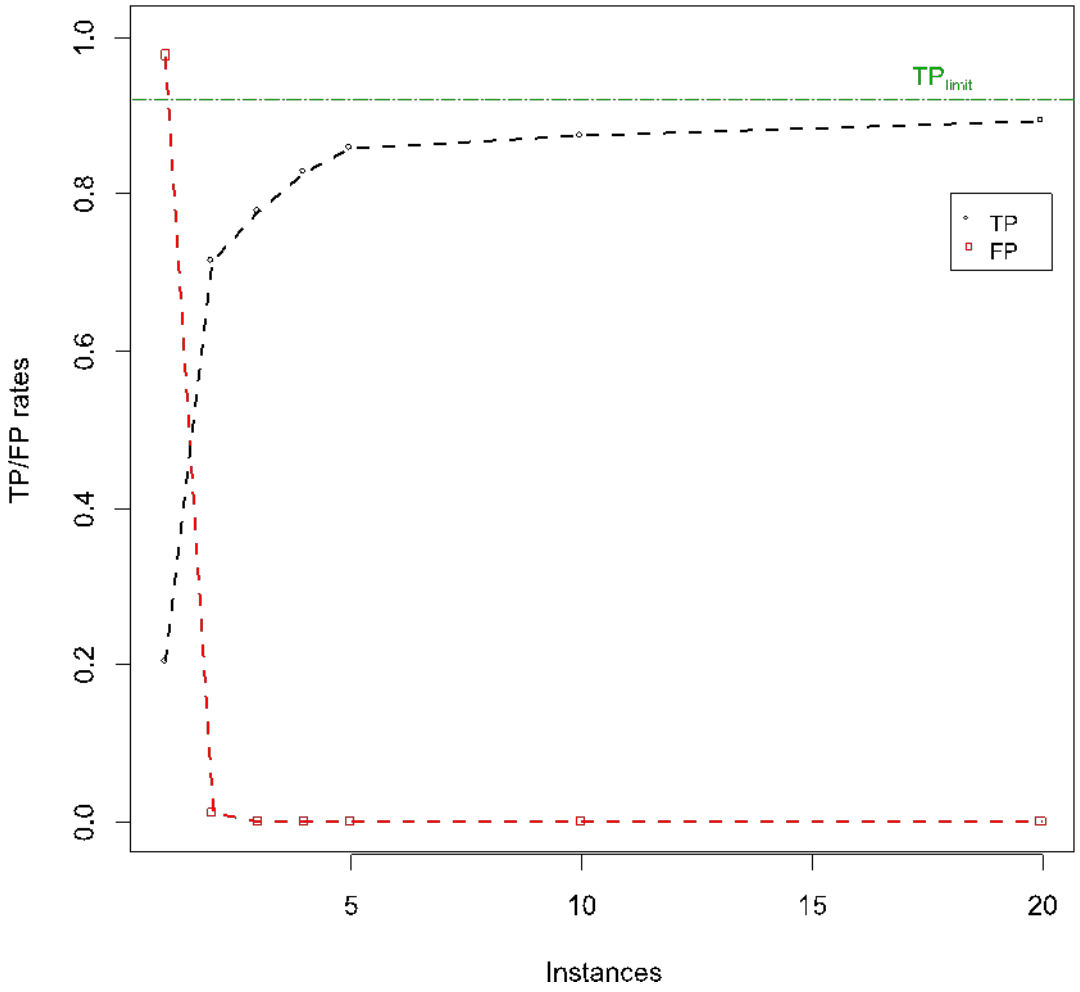


Figure 37. The TP and FP rates for the FS1 detection method. The upper limit TP_{limit} value found when using all available instances is displayed as a green dashed line. For clarity, points for the TP/FP rates are only plotted at 1-5, 10 and 20 instances.

It is difficult to say that performance like this could be expected in real world application of the FS1 detection method. It is important to consider one important factor of this assessment which is inflating the performance. In this assessment, an artificially created ontology is used. This ontology was generated by sampling entries from an existing ontology, and this sampling has introduced some bias. For the instances in the synthetic ontology, most of their properties are subject to missing or incorrect values. The properties and values which are used in the test cases CAV correspondences, however, are *not* subject to missing or incorrect values. This could not be expected in the real world.

That said, it should be noted that 20 instances per class with 150 classes is a maximum of 3000 instances, and may be considerably less as classes may share instances. This is less than 0.7% of the total number of instances available. Even if missing data were to cause a significant derogation in performance, FS1 could still be expected to only require a fraction of the available data to produce good results.

6.5 Conclusions

This evaluation set out to show that using the FS1 method to detect CAV correspondences between two ontologies, there is some sample size of shared instances that will produce the same true positive rate ($\varepsilon \pm 0.01$) as if provided all shared instances. In the ontologies that were used in this evaluation a sample of 20 instances per class was sufficient to detect correspondences with a 0.9 true positive rate, which was within 0.01 of the true positive rate of 0.91 achieved when *all 440711* shared instances in the ontologies were used.

As discussed in the analysis section (**Error! Reference source not found.**) such rapid convergence to the maximum detection rate cannot be expected in real life, as one of the ontologies used in the evaluation has properties that were not subject to missing data. The issue of missing data will be addressed in the following evaluation. Even without performance converging quite as rapidly, FS1 could safely be expected to perform well with only a small fraction of the instance data contained in large ontologies.

7 Comparison of feature selection measures for Linked Open Data.

The evaluation described in section 6 was carried out in somewhat idealised circumstances. It was based on a simplifying assumption that for each instance involved in a CAV correspondence, the attribute-value pair defining the correspondence would be present. An assumption was by the detection method used that for a given instance, that all implicit attributes of the instance had been made explicit and that there were no missing properties for the instance. Therefore if an attribute-value pair was not present then it was assumed that the instance *should not* have that parameter set to that value – this is known as the closed world assumption. In ontology engineering however, the Open World Assumption usually applies. This assumption means that if an attribute-value pair is not present, it cannot be assumed to be untrue, it is only unknown. The evaluation described in this chapter relaxes the assumption that missing data is untrue while maintaining the assumption that all implicit attribute-value pairs have been made explicit.

The complex correspondence detection method used in the evaluation described in Chapter 5, section 6 – referred to as FS1 for short – uses the Information Gain (IG) metric to perform feature selection. The IG metric is reliant on the closed world assumption to operate correctly. For small amounts of missing attribute-value pairs, the IG metric can be expected to perform as normal, but higher levels of missing values, it will break down. This evaluation will test how this performance degrades as a function of missing data levels.

The FS2 complex correspondence method was developed as an alternative to FS1. In theory it should more accurately measure the fact that missing attribute-value pairs represent *unknown* values. This method is described in detail in section 5.3.3. When ranking attribute-value pair candidates for a complex correspondence, the FS2 is expected to be less sensitive to missing values in instance data. This sensitivity is investigated in the experiment described in this chapter.

7.1 Hypothesis

As the closed world assumption breaks down, use of a beta-binomial distribution based measure (the FS2 method) will rank the correct attribute-value pair for a CAV correspondence higher than when using an information gain measure (the FS1 method). That is, when the percentage of missing data increases, FS2 will on average rank the correct parameter/value pair specifying a complex correspondence higher than FS1 does.

7.2 Methodology

This evaluation is similar to the evaluation described in chapter 7, with the introduction of the possibility that some values may be missing from the instance data sets. This is done by creating a set of synthetic ontologies which are constructed to have known levels of missing values. This evaluation compares the performance of two correspondence detection methods, FS1 and FS2 on a set of test cases for each of these ontologies. The mean reciprocal rank for each detector is calculated for the cases in the test set.

In addition, the performances of the detectors are compared pair wise or each test case and a likelihood ratio test is used to show that this difference in rankings is not simply due to random chance.

7.2.1 Hypothesis test

The distribution for the differences in rankings produced by the two detection methods is expected to be highly non normal. The detection methods are expected to agree regularly, however if FS2 performs as expected there should be a significant number of cases where FS2 outperforms FS1 by a high degree, and only a few cases where FS1 performs best. This should produce a skewed distribution with a peak near 0 or 1 and a heavy tail on one side.

To test this hypothesis a Bayesian approach is used where the likelihood measures of two models, one for each hypothesis, are compared against some collected data. The null hypothesis is that the difference in rankings is due to random chance. If this hypothesis is true then the differences should be normally distributed around 0, and can be modelled with one free parameter for standard deviation.

The alternative hypothesis is that the differences follow a skewed distribution with a heavy tails in favour of FS2. For this evaluation a Skewed T distribution is used. The model used for this hypothesis is produced by fitting a Skewed T distribution to a set of rankings

produced from a set of test cases (described in section 8.2.2) using maximum likelihood estimation.

The Skewed T distribution model has 4 free parameters and the model used for the Null Hypothesis has 1 free parameter. This means that the log of the ratio of the likelihoods of the ranking data for the Null Model and the Alternative Model should approximately follow a χ_3 distribution. The alternative model will be assumed to hold if $p<0.01$.

This approach differs from frequentist methods such as a paired t-test in an important way. If we were to use a t-test we would be testing to see if the difference in the scores follows a normal distribution and would accept our alternative hypothesis if it does not. This action of rejecting the null hypothesis does not tell us much about the alternative. With the approach used in this we are fitting two models which we can then compare. The model we fit for the alternative hypothesis will serve a descriptive purpose. In addition the decision to accept/reject the null is not as critical. The null serves as something we can compare with our alternative model to test its validity. This also means that sample size is not as critical. With a frequentist approach too large a sample would lead to a bias towards rejecting the null. This is not as important when using the Bayesian approach as we do not outright reject any model, merely compare them.

7.2.2 Test correspondence creation

The test sets used in this evaluation consist of a set of 50 Class by Attribute Value complex correspondences from classes in DBpedia to a synthetically generated ontology, as described in section 8.2.3. This synthetically created ontology was created so that there would be a definitive gold standard attribute-value pair, *p* and *v*, which would define each complex correspondence in the test set.

7.2.3 Test ontology creation

The base synthetic ontology used to create the test set of correspondences is assumed to contain no missing triples. To simulate increasing levels of missing data, new synthetic ontologies were created by removing triples which include the gold standard attribute-value pairs as their subject and object from the base ontology. Ten such test sets were created with the level of missing triples ranging from 0 to 50 percent, increasing in increments of 0.05%. This process is illustrated in **Error! Reference source not found.**40(a)

7.2.4 Metrics

This evaluation uses the Mean Reciprocal Rank (MRR) score. This score measures the inverse of the average rank given to the gold standard attribute-value pair by the detection algorithm. A perfect detector should give an MRR of 1, and the lower bound for the score is 0. For a detailed explanation of this metric, please see section 6.1.2.2.

7.2.5 Procedure

The procedure for this evaluation was as follows:

An instance training set size was selected through trial and error which consistently produced an MRR of over 0.9 for both FS1 and FS2 on the test set with no missing data. Thirty instances were found to produce these results. The particular training set size is not critical to the evaluation as it is simply intended to produce a reasonable starting point to compare the detection methods. Furthermore, the effects of training set size have previously been investigated in the evaluation described in chapter 7.

Having established the training set size to use, for each level of missing data the MMR for both FS1 and FS2 was calculated over all correspondences in the test set. In addition the pair-wise difference in ranking scores was recorded. To do this, for each test case, 50 samples of instances were drawn from the test ontology with the appropriate level of missing data, and each sample used to train a separate correspondence detector. This means that for each of the 50 test cases 50 rankings were produced, producing 2500 rankings each for FS1 and FS2 for each level of missing data.

Once all test cases for a given level of missing data were evaluated, the Null Model and Alternative Model were fitted to the rankings produced using maximum likelihood estimation and the likelihood ratio recorded. This procedure is illustrated in figure 38(b).

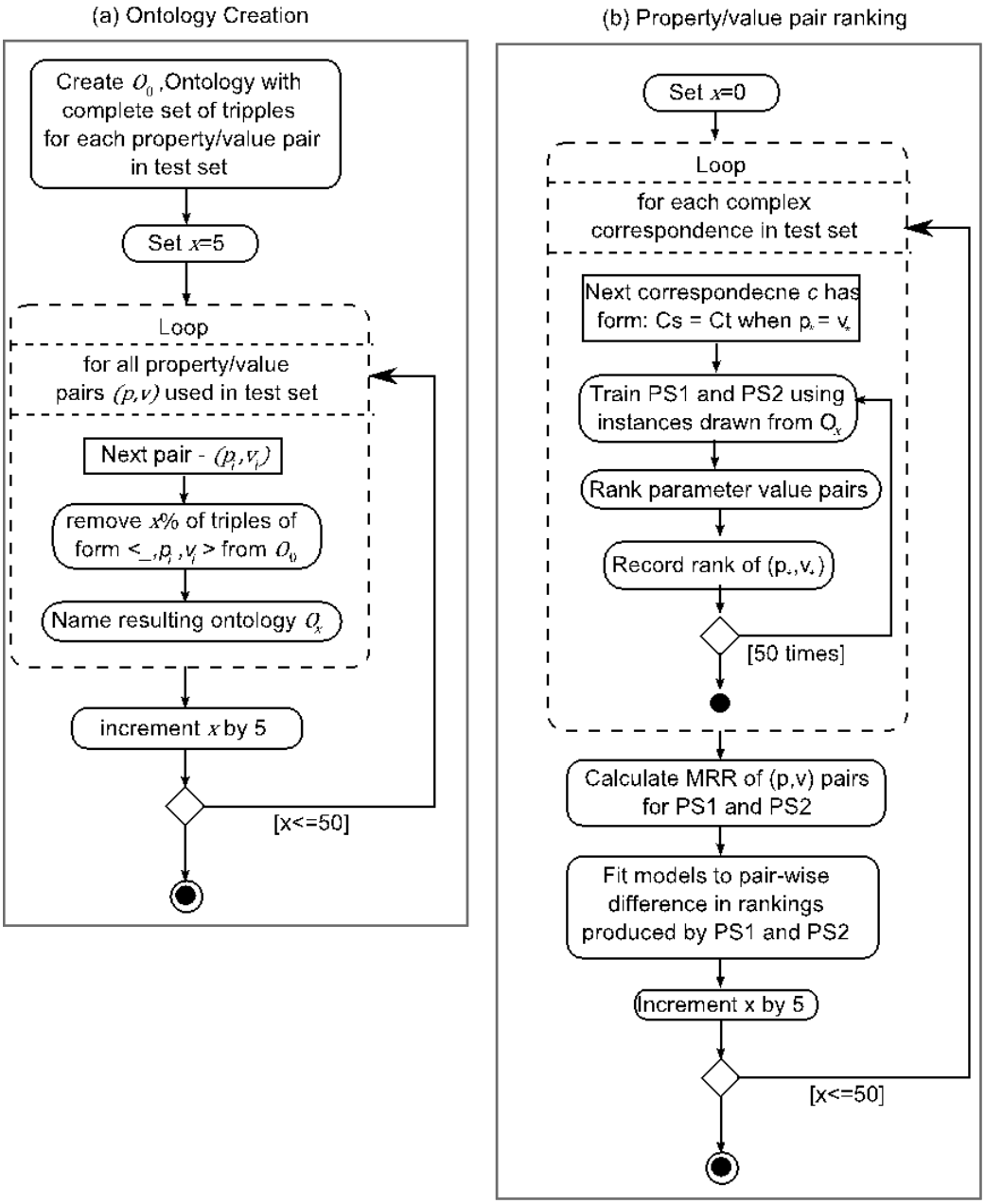


Figure 38. Procedure for Evaluation 3.

7.3 Results

The MMR score for each of the detectors as a function of missing data is shown in figure 39. As can be seen the performance of both FS1 and FS2 degrade as the level of missing data increases, though FS2 degrades at a slower rate. The mean pair-wise difference in the ranking produced by both methods is shown in figure 40.

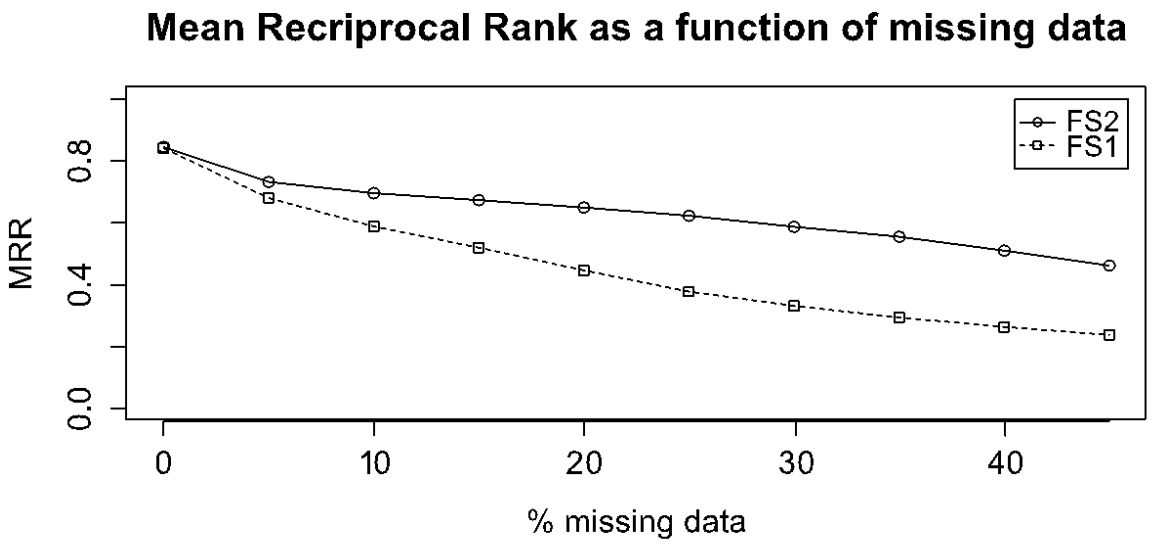


Figure 39. Mean Reciprocal Rank using FS1 and FS2 as a function of missing data.

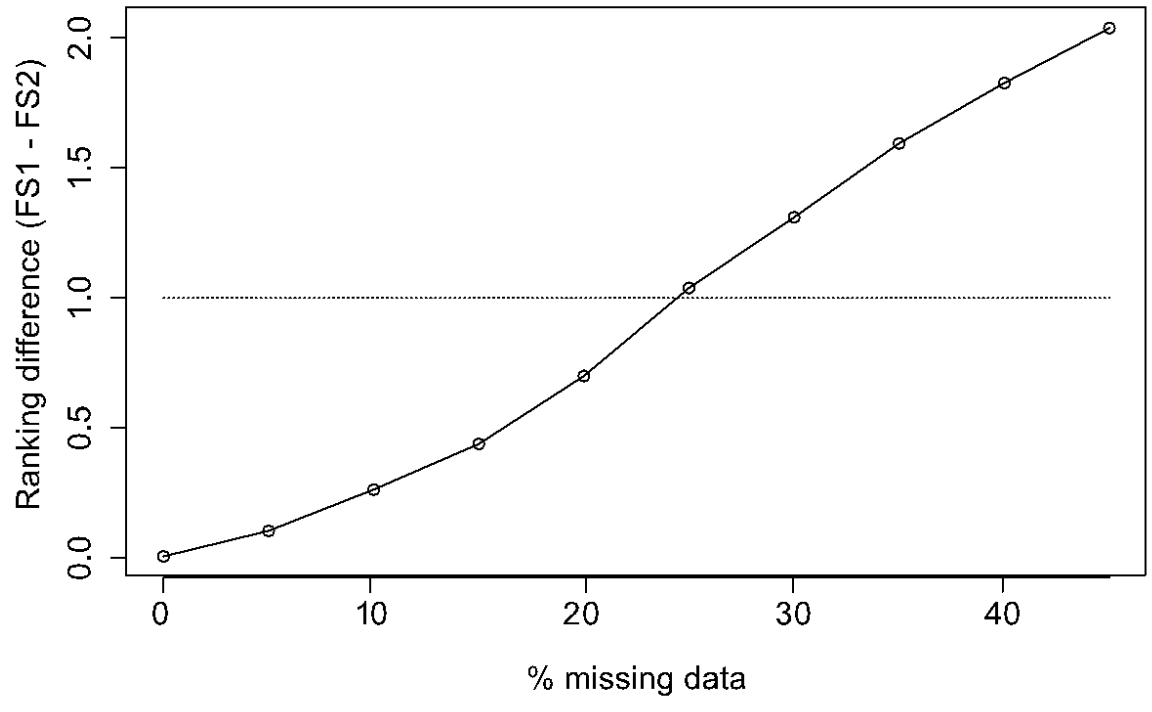


Figure 40. Pair-wise difference in FS1 and FS2 rankings. Error bars are not presented as the exact distribution of the rankings is discussed in section 8.4

7.4 Analysis

Both figure 39 and figure 40 are presented without error bars as the skewed nature of the ranking scores causes them to look slightly misleading. Discussion of the spread of the

difference in rankings is left for later. The pair-wise rank mean difference is strongly monotonic increasing as a function of the level of missing data. (That is FS2 ranks the gold standard higher on average in comparison with FS1 as the level of missing data increases.) With approximately 25% missing data, FS2 appears to rank the gold standard one position higher on average than FS1.

Error! Reference source not found. Figure 41 shows the distribution in pair-wise rank differences at 0, 30 and 50% missing data. In these graphs the sample distribution is displayed as grey bars, overlaid with best fits of Null Hypothesis Model – a normal distribution with a mean of 0 and a free standard deviation parameter – and the Alternative Model – a Skew T distribution with free position, skew, scale and degrees of freedom parameters.

As can be seen in figure 41(a), with 0% missing data, both detection methods are in strong agreement, with a relatively even split between which method performs best when they disagree. Visually the differences in rank appear to be normally distributed, and the best fit Alternative Hypothesis Model looks very similar to the Null Hypothesis Model. A likelihood ratio tests however gives a p-value of 5.5e-15, which would suggest the difference in these models is significant. The 90% inter-quartile range of the Alternative Model however is approximately {-0.73, 0.01, 0.7}, meaning that in practical terms it is not possible to say that FS2 provides improved performance over FS1.

Error! Reference source not found.3(b), with 30% missing data in the training sets used, shows quite a different picture. Here the two rankings are often in agreement, however there are a large number of cases where they disagree. The distribution of these disagreements is quite skewed with no cases where FS1 outperformed FS2. Performing a likelihood ratio test returned a p-value of 0 (due to rounding) so the null hypothesis can very certainly be rejected. The 90% inter-quartile range for the Alternative Hypothesis Model is approximately {-0.18, 0.95, 3.25} showing that there is a strong chance that FS2 will outperform FS1 by a whole rank, possibly more, and a low chance that FS1 will perform better.

Error! Reference source not found.3(c), shows the dank difference distribution at 50% missing data in the training sets. Here the peak of the distribution appears to have moved to 1, with a very strong tail in favour of FS2. The 90% inter-quartile range of the alternative hypothesis model in this case is approximately {0.02, 1.5, 6.04}, which would suggest that FS2 will very regularly outperform FS1 by a significant margin.

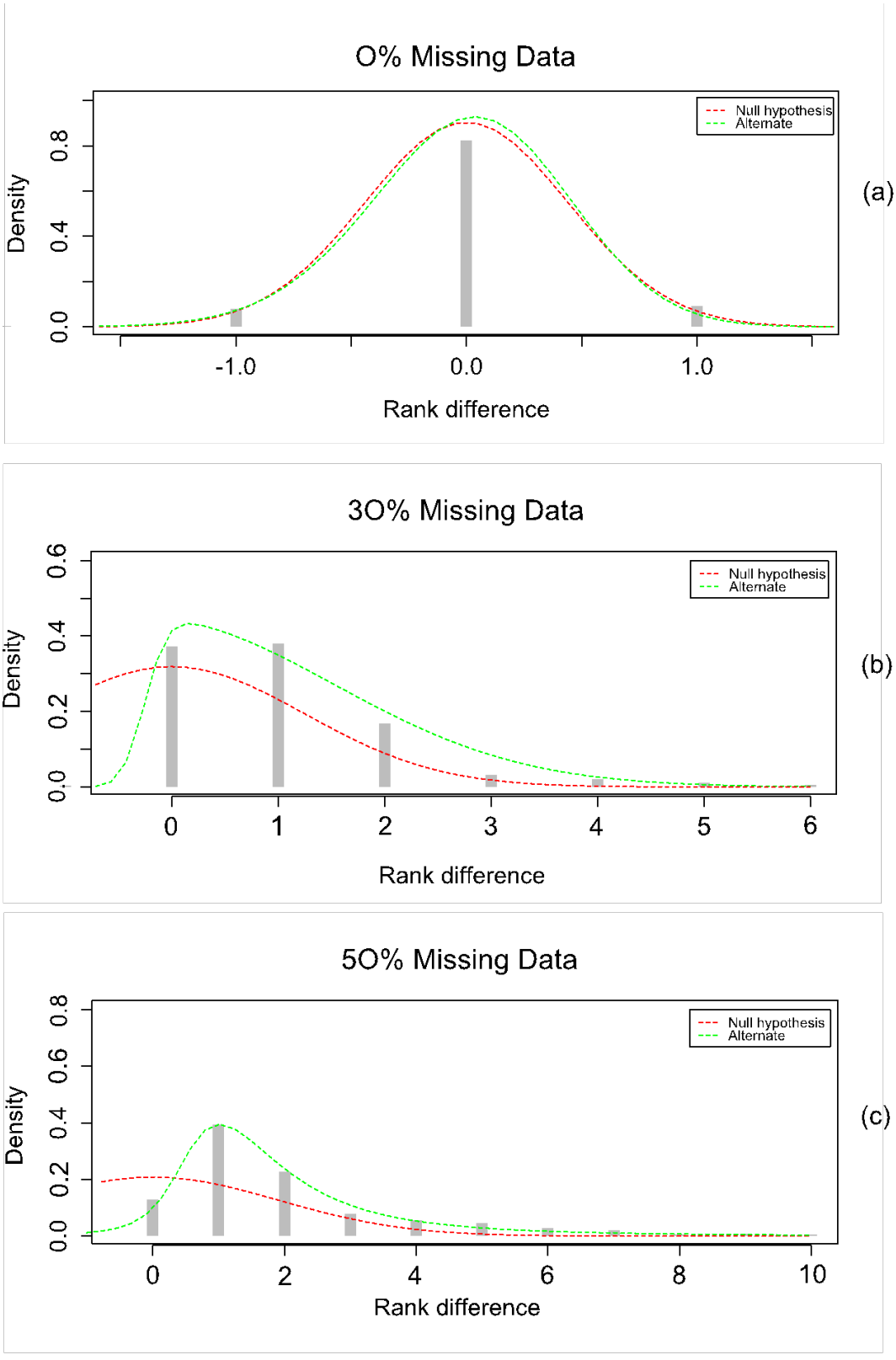


Figure 41. Distribution of ranking differences at 0, 30 and 50% missing data

7.5 Conclusions

This evaluation has shown that when there are no missing attribute values in the instances used to train the FS1 and FS2 correspondence detection methods, there is little to separate their performance. Technically, as there is statistically significant evidence that the difference in rankings produced by FS2 and FS1 is non-normal and favours FS2, it could be said that FS2 is slightly better. In practical terms, however, this improvement is not worth noting, as the model fitted to the sample data suggests that the mean difference in rankings is almost zero, that 90% of the rankings will be the same using both methods, and that there are almost as many cases where FS1 outperforms FS2 as the opposite happens.

When the level of missing values in the instances increases, the difference in the rankings produced by FS1 and FS2 becomes apparent. This is as hypothesised. The mean ranking difference appears to be monotonically increasing as the level of missing data ranges from 0 to 50%. At 30% the mean rank difference is close to 1. The Skew T model fitted to the sample of rank differences at this level has a median of 0.95 suggesting that in almost 50% of cases, FS2 should produce a rank one place higher than FS1. The 90% inter-quartile range of (-0.18, 3.25) would suggest that it is very rare for FS1 to outperform FS2, but FS2 may regularly outperform FS1 by several ranking positions.

As the level of missing data increases this difference becomes more pronounced. It is debateable how useful it is to work with data where large amounts of information are missing, however this evaluation would suggest that for all levels of missing data, FS1 very rarely outperforms FS2, but there are many cases where FS2 can be expected to outperform FS1.

8 Comparison of FS2 with a state of the art common extension based CAV detection method

As seen in the state of the art analysis, techniques for automating the process of discovering complex correspondences is still an emergent topic, with relatively few published detection methods. One prominent published method is that described in Parundekar et. al. [15], [44]. Like the FS1 and FS2 methods, Parundekar et. al's method is capable of detecting Class by Attribute Value correspondences and also like FS1 and FS2, does this by analysing instances of the classes in the correspondences. Methods such as this are known as common extension comparison methods [29].

Unlike the FS1 and FS2 methods, Paranundekar et. al.'s method does not use samples of instance data but instead uses *all* instances available. An important aspect of the research question of this thesis was to assess if the use of samples of instances could be used to detect complex correspondences. Therefore this evaluation compares Paranundekar et. al.'s method with the FS2 method to determine what difference in performance sampling produces. FS1 is not considered here, as in the previous evaluation, it was shown that FS2 will almost always outperform FS1.

Paranundekar et. al.'s method uses a tree based search to evaluate combinations of classes, properties and values to find complex correspondences. The search is aided by a set of heuristics on when to prune the tree and which branches to follow. It will be referred to as Heuristic Tree Search (HTS) method for this evaluation. It is not clear if the heuristics used in HTS apply equally to classes with greatly differing numbers of instances. The HTS method also lacks an ability to model missing data in a manner FS2 does. Therefore it is possible that FS2 may *outperform* the HTS method even though the FS2 method uses much less input data. This evaluation will test this possibility.

8.1 Hypothesis

The FS2 complex correspondence detection can detect more Class by Attribute Value correspondences between ontologies than the Heuristic Tree Search (HTS) method. The correspondences detected by the FS2 will be as or more correct than those found by HTS.

In the context of this evaluation the term *more correct* is evaluated by using the text descriptions of the attribute and values to determine if the most appropriate attribute was used

to scope the context of the classes in the complex correspondence. This allows for some subjectivity, but any cases where this occurs will be clearly stated in the results section of this evaluation.

8.2 Methodology

This evaluation compares the ability of the FS2 and HTS methods to detect Class by Attribute Value correspondences in two ontologies. HTS has previously been used to discover complex correspondences between the GEONAMES and DBPEDIA ontologies, and with the results being made available publicly. Therefore this evaluation performs complex correspondence detection using the FS2 method and compares the results.

This evaluation is concerned with Class by Attribute Value correspondences. Correspondences of this form consist of a Restriction Class – defined by an attribute and value pair – in one ontology, a named class in a second ontology and a relationship between the classes, usually equivalence or subsumption. As GeoNames only provides a single class[19] *geo:Feature,* all CAV correspondences between GeoNames and DBpedia can be expected to consist of a restriction class in GeoNames and a named class in DBpedia. Furthermore as *geo:Feature* is assumed to be broadly equivalent to *dbpedia-owl:Place*, this evaluation assumes that all named DBpedia classes in the correspondences will be sub classes of *dbpedia-owl:Place*.

The procedure for this evaluation is to first find all CAV correspondences fitting the description above in the set of published correspondences which were discovered using HTS. This process is described in more detail in section 8.2.1.2. Then, for each sub class of *dbpedia-owl:Place*, the FS2 method was used to discover if there was an attribute-value pair in GeoNames which could specify a CAV correspondence between a restriction class in GeoNames and the given sub-class. Twenty instances of each sub-class of dbpedia-owl:Place were used to train the FS2 detector. This number was selected as in the evaluation described in chapter 7 no significant improvement in performance was shown after 20 instances.

The results of using FS2 are then compared with the CAV correspondences found using HTS. The most important metric is the number of valid correspondences each approach found. In addition, in the case where both methods find a CAV correspondence to a given DBpedia class, a record is made of which method produces a restriction class that more closely resembles the DBpedia class.

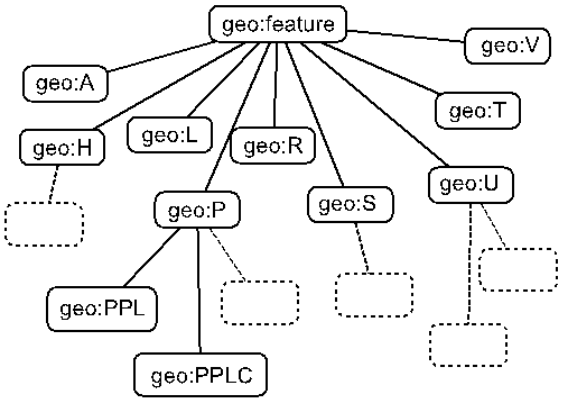
[19] See section 8.2.1.1 for details.

8.2.1 Data Sets

The data sets used in this evaluation consist of the two ontologies between which the complex correspondences must be detected – namely DBpedia and GeoNames; a set of complex correspondences between these ontologies which have previously been detected between the ontologies using the HTS method and made public; and the training set of instances that were used by the HTS method to produce the published correspondences.

8.2.1.1 Ontologies Used

DBpedia is a source of linked data extracted automatically form Wikipedia containing approximately 1.5 million instances and a taxonomy of 170 classes. GeoNames contains approximately 8 million instances, in a flat structure. That is, GeoNames does not contain a detailed taxonomy, and almost all instances are of type *geo:Feature*. It does however, contain the attributes *geo:featureCode* and *geo:featureClass* which together behave somewhat like *rdf:type*, forming a two level taxonomy. For example, it could be viewed that the values of *geo:featureClass* are direct sub-classes of *geo:Feature* and for each feature class, the *geo:featureCode* valuesbelow form a second level of sub-classes. A sketch of part of the hierarchy restriction classes produced using the *geo:featureClass* and *geo:featureCode* properties is displayed in figure 42. The names of the feature classes and codes say little about their meaning. The class *A* refers to countries, states and regions, the class *P* denotes a city or village. The code "PPL" refers to a populated place and "PPLC" a capital of a political entity.



Geoname Feature Classes	
Class	Meaning
A	country, state, region,...
H	stream, lake, ...
L	parks,area, ...
P	city, village,...
R	road, railroad
S	spot, building, farm
T	mountain,hill,rock,...
U	undersea
V	forest,heath,...

Geoname Feature Codes	
Code	Meaning
PPL	populated place
PPLC	capital of a political entity

Figure 42. An incomplete diagram of the class hierarchy that results from treating GeoNames feature class and feature code values as classes.

8.2.1.2 Published Complex Correspondences

Parundekar et.al. have previously performed complex correspondence detection on DBpedia and GeoNames using the HTS method, and published the results publicly[20]. These complex correspondences cover more patterns than Class by Attribute Value. Where as a CAV correspondence consists of a correspondence between a restriction class defined by a single attribute-value pair, and a named class, the results produced by the HTS method can include correspondences between restriction classes with multiple attribute-value pairs, and additionally the correspondence may have restriction classes on both sides of the relationship.

These results are described in comma separated variable (CSV) tables. These tables list:

- the two restriction classes in the correspondence (R1 and R2)
- the support score for the alignment from the first source (*|Img(R1) int R2|* / *|Img(R1)|*, where Img(R1) is the number of instances of R1 that are contained in the second source);
- the support score for the alignment from the first source (*|Img(R1) int R2|* / *|R2|*)
- the type of relationship the correspondence describes
- The size of the intersection of R1 and R2
- The size of R1
- The size of R2

[20] http://www.isi.edu/integration/data/LinkedData

As these correspondences cover many cases that do not fit the Class by Attribute Value pattern it was necessary to filter these out. The only cases that are considered relevant to this evaluation are correspondences between restriction classes in GeoNames, and named classes in DBpedia. The following procedure was used to find correspondences of this form.

1. All correspondences which contain restriction classes in DBpedia were removed.
2. In the case where there were multiple correspondences to a single DBpedia class, the correspondence with the highest $|Img(R1)\ int\ R2| / |R2|$ score was kept.

After following this procedure, 16 correspondences remained. All restriction GeoNames classes featured a single attribute-value pair. These correspondences are listed table 7.

Table 7. Class by Attribute Value correspondences found in published complex correspondences detected using the HTS detection method.

DBpedia Class	**Restriction**	Restriction Meaning[21]
HistoricPlace	inCountry=US	Places in the United States of America
City	featureCode=P.PPL	A city, town, village, or other agglomeration of buildings where people live and work
Bridge	featureCode=S.BDG	A structure erected across an obstacle such as a stream, road, etc., in order to carry roads, railroads, and pedestrians across
River	featureCode=H.STM	A body of running water moving to a lower level in a channel on land
Lake	featureCode=H.LK	A large inland body of standing water
PopulatedPlace	featureClsss=P	City, village,...
Island	featureClass=T	Mountain,hill,rock,...
ProtectedArea	inCountry=CA	Places in the country Canada
Skyscraper	featureClass=S	Spot, building, farm
Airport	featureCode=S.AIRP	A place where aircraft regularly land and take off, with runways, navigational aids, and major facilities for the commercial handling of passengers and cargo
Stadium	parentFeature = 2635167	A sub-feature of the United Kingdom of Great Britain and Northern Ireland.
Country	featureCode=A.PCLI	Independent political entity
Building	inCountry=GB	A location in the country Great Britain.
Mountain	featureClass=T	Mountain, hill, rock,...
Hospital	featureClass=S	Spot, building, farm
BodyOfWater	featureClass=H	Stream, lake, ...

[21] See http://www.geonames.org/export/codes.html for further details of GeoNames feature codes and feature classes.

Of these correspondences, the ones featuring *HistoricPlace*, *Stadium*, *ProtectedArea* and *Building* are clearly wrong – as they state that the class of the instance is dependent on its location. Better correspondences exist for *Island, Hospital*, and *Skyscraper* which are matched to the Feature Classes T, S and S again respectively. These are not wrong, but more specific feature codes are available – *T.ISL*, *S.HSP* and *S.BDG*. Additionally the DBPedia class City is reported as corresponding with the restriction class defined by *featureCode = P.PPL*, but the feature code *P.PPL* can represent populated places that are not cities, such as villages. There is however, no better code than *P.PPL* available. Therefore 12 of the correspondences are considered to be valid, and of these 12 there are three which while valid could be improved.

8.2.1.3 Training Sets

To ensure that only the search methods – and not the issues of matching or pre-processing instance data – are compared, we re-use Parandukar's training set of DBpedia/Geonames instance pairs. This data set consists of 369287 instances that have been found to exist in both DBpedia and GeoNames. The number of instances of each DBpedia class in this set varied greatly, with many having zero instances in the training set. Of the non-zero entries, the rarest was *dbpedia-owl:WineReigon*, with 3 instances, and the most common was *dbpedia-owl:PopulatedPlace* with 60197 instances. The instance counts for each DBpedia class found in the training set are shown in figure 43, which uses a logarithmic scale.

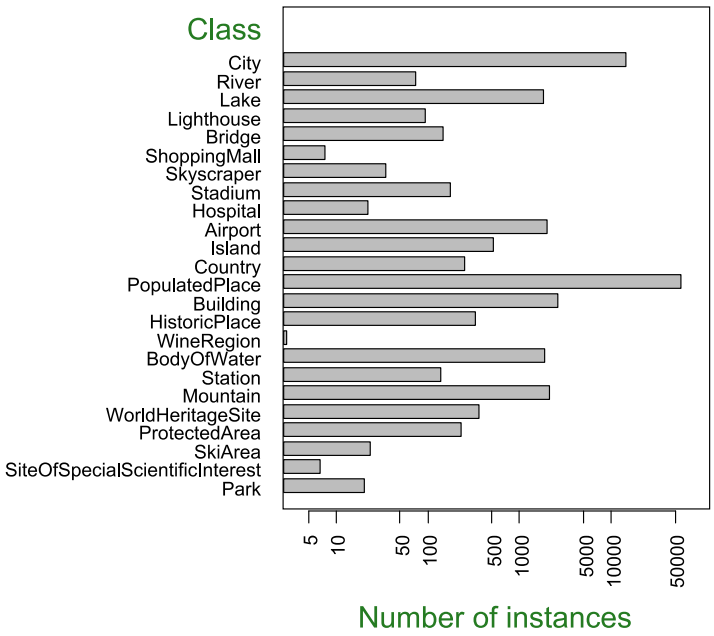


Figure 43. The number of instances of each class present in the training set.

8.2.2 Results

Using FS2 to detect correspondences between GeoNames and DBpedia produced 13 CPV correspondences. These correspondences are listed in table 8. Of the 13 correspondences found with the FS2 method, only one is clearly incorrect. The DBpedia class *Stadium* is reported as corresponding with the restriction class defined by the feature code L.PRK which represents parks. The class *Skyscraper* is reported as corresponding to GeoNames features with the featureCode *S.BDG*, which represents buildings, this correspondence is not entirely semantically correct, but there are no better featureCode values available to represent skyscrapers. Also the correspondence between DBPEDIA *ProtectedArea* and the restriction classed defined by *featureCode=L.PRK* – parks – is problematic as these two things are not strictly equivalent, nor is one a sub-classes of the other, but they are strongly correlated, and this does represent the best correspondence that could be found between *ProtectedArea* and any restriction class in GeoNames.

8.3 Analysis

Both methods produced similar numbers of acceptable correspondences between GEONAMES and DBPEDIA with each producing 12. FS2 however, produced only one incorrect correspondence:

- *Stadium* ≡ *featureCode = L.PRK*

While the HTS method produced four incorrect correspondences:

- *HistoricPlace* ≡ *inCountry = US*
- *Stadium* ≡ *parentFeature = 2635167*
- *ProtectedArea* ≡ *inCountry = CA*
- *Building* ≡ *inCountry = GB*

This gives a false positive rate of 0.25 for the HTS method versus 0.08 for SP2.

The nature of these incorrect correspondences is interesting, in that for HTS, the attribute-value pairs used in the restriction class for the correspondence refer to the location of the feature and are very clearly inappropriate - being located in the country Great Britain is very much not a defining characteristic of being a building for example. The mistake produced by FS2 is much less obvious. Stadia are often referred to as parks, it is conceivable that the training data contained many examples of a stadium that were mistakenly labelled as a park.

Table 8. Class by Attribute Value correspondences between DBpedia and Geonames, discovered using the FS2 detection method.

Named Class	Restriction Class	Restriction Meaning
Park	featureCode = L.PRK	Park: an area, often of forested land, maintained as a place of beauty, or for recreation
ProtectedArea	featureCode = L.PRK	Park:
Mountain	featureCode = T	Mountain, hill, rock, ...
BodyOfWater	featureClass = H	Stream, lake, ...
Country	featureCode = A.PCLI	Independent political entity
Island	featureCode = T.ISL	Island: a tract of land, smaller than a continent, surrounded by water at high water
Airport	featureCode = S.AIRP	Airport: a place where aircraft regularly land and take off, with runways, navigational aids, and major facilities for the commercial handling of passengers and cargo
Hospital	featureCode = S.HSP	Hospital: a building in which sick or injured, especially those confined to bed, are medically treated
Stadium	featureCode = L.PRK	Park:
Skyscraper	featureCode = S.BLDG	Building: a structure built for permanent use, as a house, factory, etc.
Bridge	featureCode = S.BDG	Bridge: a structure erected across an obstacle such as a stream, road, etc., in order to carry roads, railroads, and pedestrians across
Lake	featureClass = H.LK	A large inland body of standing water
River	featureCode = H.STM	A body of running water moving to a lower level in a channel on land

There were three cases where HTS produced correspondences that, while not incorrect, could be improved on. These were:

- *Island* ≡ *featureClass=T*
- *Hospital* ≡ *featureClass=S*
- *Skyscraper* ≡ *featureClass=S*

For all of these cases, SP2 produced the more accurate correspondences

- *Island* ≡ *featureCode = T.ISL*
- *Hospital* ≡ *featureCode = S.HSP*
- *Skyscraper* ≡ *featureCode = S.BDG*

There were no cases where HTS produced correspondences that were more specific than those produced by FS2.

FS2 was shown to be more capable of finding correspondences for classes with very few instances. For example the class Park with 21 instances, which FS2 produced a correct correspondence, and HTS produced no correspondence. Also, two out of the three cases where FS2 produced a more precisely defined correspondence were for classes with few instances – Hospital which had 23 instances in the training set and Skyscraper which had 36.

One further observation that should be noted, is that while the HTS detection method is capable of finding complex correspondences with restriction classes defined by multiple properties, no correspondences of this form were found between GeoNames restriction classes and named DBpedia classes. FS2 is only capable of detecting correspondences between restriction classes defined by a single attribute, but this was sufficient for this task.

8.4 Conclusions

FS2 and HTS were both capable of finding the same number of valid CAV correspondences between DBpedia and GeoNames – 12 each. In one quarter of these cases however, FS2 produced more accurate complex correspondences – that is ones where the restriction class in the correspondence produced by FS2 was more closely related to the named class in the correspondence. It would also appear that FS2 performed better in relation to HTS when there were less training instances available.

One of the main limitations of this evaluation is that with only 14 CAV correspondences in total found between the two search methods, it is difficult to say with great certainty that FS2 will generally find more accurate CAV correspondences than HTS. A further limitation of this evaluation is that the training sets used for the FS2 detection method were drawn at random, and it is possible that different sampling could produce different results. This is especially true in the case where only a small number of matched instances are available – it is very possible that there is a systematic reason why those particular instances have been matched, which in turn will mean that the available instances are not representative of the population as a whole.

The goal of this evaluation was to show that FS2 can outperform HTS while requiring less training data. It is difficult to conclude strongly that this is the case given the evidence provided, but it is clear that both methods produced very similar results.

It *is* relatively safe to conclude from this evaluation at least there was nothing gained by using much a larger set of training data. This in itself is an important result. Often when performing ontology mapping the number of matched instances available will be relatively low, and in such cases, by using the FS2 method it should be possible to begin searching for correspondences between the taxonomic portions of the ontologies, without first searching for more matches between the instances of the ontologies. Furthermore, using smaller training sets places less strain on available storage and computing resources.

9 Conclusions

In this chapter, the research objectives of the thesis are presented with discussion on the degree to which they have been achieved. This is then followed by a discussion of the contributions of this work, with the core contribution of this work being the development of methods for detecting Class Restriction correspondences of type Class by Attribute Value and Class by Attribute Existence between ontologies which may contain missing or incorrectly labelled data using only small samples of matched instance data in the ontologies. Further to this the implications this work will have for further research are discussed.

9.1 Research Objectives

In chapter 1, four research objectives were listed. These objectives focused on first identifying the forms of Complex Correspondence that most commonly occur between ontologies, surveying existing methods for detecting complex correspondences, developing new techniques for detecting the most common complex correspondences and then evaluating these techniques. The extent to which these research objectives were achieved are discussed in the following sections.

9.1.1 Identifying common patterns of complex correspondence

The first research objective O1, was to stated as follows: *Evaluate which forms of complex correspondence exist between ontologies, and therefore which forms our detection method should focus on*. To do this, a review of published correspondences between the LOD associated ontologies YAGO, DBpedia, GeoNames, FoaF, VCard and SUMO was carried out, with these correspondences being classified against the Correspondence Patterns classification scheme.

YAGO and DBpedia are based on Wikipedia entries. Finding correspondences between the instance data in the ontologies is trivial as the URI of the Wikipedia entry used to generate each instance can be used to identify the instance in each ontology. Identifying correspondences between the classes in the ontologies is not as easy, and for many classes there is no direct equivalent in the other ontologies. There is no widely agreed upon set of correspondences between the classes in the ontology. In contrast class restriction correspondences between DBpedia and GeoNames, *have* been published. In this set of

correspondences, aside from one to one mappings, Class by Attribute Value correspondences were most common

The correspondences between FOAF and VCARD were more complex, often requiring operations such as string splitting and concatenation. Published correspondences between YAGO2 and SUMO were the most complex, often requiring full lisp-like descriptions. It was felt that correspondences of this form would be too difficult to detect as this was potentially an AI complete problem. Class Restriction correspondences present a much more tractable problem, and appear to be occurring in sufficiently large quantities that it would be beneficial to automate their discovery. To estimate how frequently these correspondences occur, a sample of classes from YAGO2 was selected, and these were corresponded by hand with DBpedia classes. This found that out of 50 YAGO2 classes, while 10 corresponded directly with DBpedia classes, 14 required a Class by Attribute Value correspondence.

The research objective O1 has been answered to the extent that Class by Attribute Value correspondences were identified as commonly occurring correspondences, and within DBpedia and YAGO2 it was shown that CAV was the most commonly occurring complex correspondence. CAV also occurs between DBpedia and GeoNames. YAGO2 and DBpedia are among most complex structured ontologies in LOD, while GeoNames is much simpler. While YAGO2 and DBpedia instances have many attributes which could be used to form a restriction, GeoNames only have a few. Still it was seen that CAV can occur in this simple ontology, so we expect that it is possible in other ontologies regardless of their complexity.

9.1.2 Survey of existing complex correspondence detection techniques

The second research objective, O2, was stated as follows: *Study existing correspondence discovery techniques to establish the capability of state of the art matching technologies to detect the correspondences highlighted from the survey resulting from objective 1.*

The primary venue for discussing correspondence detection is the Ontology Matching workshop (OM). This workshop, however, is heavily concentrated on discussing one to one correspondence detection. Each year the workshop runs a competition, the Ontology Alignment Evaluation Initiative (OAEI). This competition features several tracts, all of which require a set of one to one correspondences to be detected which are compared against a gold standard. Complex correspondence detection is very much an emergent area, and 2012 was the first year the competition call included an invitation for entries that could detect complex correspondences. Of the 16 entrants for the competition in 2012, none could detect complex

correspondences. Again in 2013 there were no entrants to the competition that could detect complex correspondences. One paper at OM2013 demonstrated how complex correspondences could be used to perform automated query-rewriting, however their approach still requires the complex correspondences to be discovered manually.

Two families of approaches to detecting Class Restriction correspondences were found in the literature. One is based on pattern matching [8], this uses a combination of reasoning and analysing the language used in the names of elements in the ontologies to detect how they may be related. The advantage of this approach is that it does not require any training data to detect the correspondences. The disadvantage is that it requires the properties in the ontologies to have well defined ranges and domains, and that all the elements follow a consistent and meaningful naming scheme.

A second, more robust, method uses common extension information – knowledge of shared instances data – to detect complex relations between the classes in ontologies [15] [16] . This method is more robust, as unlike the pattern based approach which uses hard rules, this approach uses heuristics to search through the set of all possible Class Restriction correspondences that could exist between the classes in the ontologies and identify ones which are supported by the instance data. The disadvantage of this approach is that it requires large amounts of accurately matched instance data in the ontologies, and this is not always available.

9.1.3 Improved techniques for detecting complex correspondences

The third research objective, O3, was as follows: *Develop improved techniques for discovering complex correspondences between classes in ontologies*. As a result of research objective **O1**, It was decided that only the Class by Attribute Value pattern of complex correspondence would be addressed. This pattern also covers the Class by Attribute Existence (CAE) pattern which is a special case of CAV, but CAE performance was not evaluated directly.

Use cases for the detection techniques were developed, and these were used to define a sequence of actions which would extract the necessary information from the ontologies, and convert the detection problem to a feature selection one, where the properties associated with each class are scored for their ability to define Class Restriction correspondences between the

classes. The choice of scoring metric is an important factor in the performance of detection methods using this process.

Two metrics were evaluated. The first was Information Gain (IG), which is a commonly used metric for feature selection in machine learning. This process of selecting properties using IG was labelled FS1. As IG was developed to be used in Closed World Assumption environments, and the Semantic Web operated under an Open World Assumption, a new metric based on a beta-binomial distribution (see section 5.3.3) was developed which was designed to compensate for this assumption. This process was labelled FS2.

Several evaluations were carried out to compare the abilities of FS1, FS2, and a leading Restriction Class correspondence detector from the state of the art, which was labelled as the Heuristic Tree Search (HTS) method. The evaluation described in chapter 7 demonstrated that the FS1 detection method was capable of producing results with small training samples of (20 instances) which were comparable to results produced when thousands of training instances were available. The evaluation described in chapter 8 demonstrated that FS2 could outperform FS1 when triples describing instances in the training set were missing at random, ranking the gold standard on average one place higher when 25% of the triples were missing. Finally, the evaluation described in chapter 9 demonstrated that FS2 could outperform HTS, even while using less training data. To detect complex correspondences between DBpedia and GeoNames, HTS used all available matched instances in these ontologies (up to $5{\cdot}10^4$ instances per class), and produced 12 Class by Attribute Value correspondences. FS2 also produced 12 CAV correspondences, but only required a training set of twenty instances per class. Of these 12 CAV correspondences, 4 of those produced by FS2 were judged to use attribute-value pairs which more accurately described the relationship between the classes in the correspondence. These results are very promising as they suggest that FS2 is outperforming HTS, which is held in high regard, previously being awarded a spotlight at ISWC 2012.

9.1.4 Evaluate the ability of the techniques to detect complex correspondences in real world ontologies, using LOD as a case study.

The fourth research objective, **O4**, was as follows: *Evaluate the ability of the techniques to detect complex correspondences in real world ontologies, using LOD as a case study*.

There were two evaluations which addressed this research objective. the evaluation described in chapter 6 served as a proof of concept that the FS1 method could be used to detect Class by Attribute Value correspondences between DBpedia and YAGO2. Though it is difficult to draw any hard conclusions from that particular evaluation, it served well to inform the design of the FS2 detection method, and to provide a guide on how to perform subsequent evaluations. This evaluation demonstrated that it was possible to reliably detect the gold standard correspondence in 3 out of 5 the test-cases using FS1, even with small training sets of 15 instances. It also highlighted that missing or incorrect values in the datasets are a problem. It can be the case that while there is a attribute-value pair available which could describe a CAV correspondence between two classes, this information is missing in a large number of instances of these classes, meaning that a CAV correspondence based on the attribute and value would be of little practical value for use in an ontology mediation task.

It was observed that if the FS1 method was used to *rank* all attribute-value pairs, the correct one would almost always appear in the top 3. This result reinforces the conceived use case, where the detection method is used to assist a human user – reducing the choice they need to make from a selection of thousands to selecting the best option from a small number of choices.

DBpedia and YAGO2 are large ontologies which can make it difficult for a human to understand how they are related. Finding complex relationships between them is not a trivial task, but with the use of FS1 or FS2 this task is much easier. Instead of having to select form tens of thousands of options, the user only has to pick from a handful.

9.2 Contributions

The primary contribution of this thesis is the development of methods for detecting Class by Attribute Value and Class by Attribute Existence correspondences between LOD ontologies containing incorrect information. These methods improve on previous ones in that they can operate on ontologies which are not strictly well formed, and they do not require large amounts of training data.

Specifically, this thesis has demonstrated that the task of detecting complex correspondences can be converted to a feature selection task. It has demonstrated that using a standard feature selection metric – information gain (IG) – a sample of 20 training instances per correspondence can produce results comparable to using *all* available instances. The implication of this is that it is possible to detect complex correspondences in ontologies with

small amounts of instance data, and also when dealing with large ontologies, scalability can be improved by using samples of instance data instead of the entire ontology. This thesis demonstrated that a beta-binomial distribution could be used instead of information gain to perform feature selection, with an improvement in performance in the presence of missing data. When 25% of the triples were removed from the training sets, the beta-binomial metric ranked the gold standard complex correspondences on average one place higher than using IG. This difference was often much greater than one position, and IG never outperformed the beta-binomial. The detection method developed in this thesis outperformed the leading detection method from the state of the art, with one third of the complex correspondences generated by FS2 being judged to be more accurate than those produced by Parundekar et.al. [15]. A minor contribution of this work is the development of methods and metrics for evaluating the quality of complex correspondences between data sources.

The approach developed in this thesis has several application areas. While the primary application is data from a consumer's perspective, it could also be of great benefit in the publication of new data to the LOD cloud. There are many information silos that exist in the world where vast amounts of data are stored in a manner which makes it difficult to use. Simply making this data accessible is not usually a particularly difficult task – making its meaning *understandable* is where the difficulty lies. As the approach in developed in this thesis is extensional, it can be used to understand how different attributes in the data relate to existing concepts used in LOD as well as assisting in the process of forming federations of autonomic networks.

The approach was is not strictly limited to LOD ontologies, it can be used in any application where ontologies are used to share information between independent organisations – for example in the federation of autonomic networks, or in the field of biomedicine.

Papers that have resulted from this work include:

Brian Walshe, Rob Brennan, Declan O'Sullivan, "Correspondence Pattern Attribute Selection for Consumption of Federated Data Sources", ***Distributed Autonomous Network Management Systems/Network Operation and Management Symposium, Maui, Hawaii,. USA, April 16-20, 2012, pp1234 – 1240***

This presented an initial evaluation which showed that it is possible to use information gain to detect Class by Attribute Value (CAV) and Class by Attribute Existence (CAE) correspondences between DBpedia and YAGO2 ontologies, and proposed that the approach could be of use in quickly integrating information in federated networks.

Brian Walshe, "Identifying Complex Semantic Matches", *Extended Semantic Web, Heraklion, Crete, Grece, May 27-31, 2012*

This paper was presented as part of the PhD symposium. It provides an analysis of the state of the art in correspondence detection at the time of publication and notes that most approaches are only concerned with detecting simple one to one equivalence correspondences. There are few approaches that can detect one to one subsumption correspondences and only a handful that can detect Class Restriction correspondences such as CAV or CAE. This paper touches on the findings of the "Correspondence Pattern Attribute Selection for Consumption of Federated Data Sources" paper and concludes that machine learning techniques should be investigated to see if they can be used to detect complex correspondences.

Brian Walshe, Rob Brennan, Declan O'Sullivan, "A comparison of Complex Correspondences Detection Techniques", *International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012*

This paper contains a comparison of the methods developed in the papers above with two state of the art complex correspondence detection methods, providing a table of the patterns of correspondence each is capable of detecting. In this paper a sample of 50 classes from YAGO2 were analysed by hand to see if complex correspondences existed between them and DBpedia. CAV was found to be the most common form of complex correspondence in the sample. It was also noted that one of the state of the art complex correspondence detectors, the Pattern Matching approach, was not capable of detecting CAV correspondences.

Rob Brennan, Brian Walshe, Declan O'Sullivan, "Managed Semantic Interoperability for Federations", *Journal of the Network and Systems Management, special issue on MFCN, 2013*

This was is a collaborative paper where the author of this thesis contributed a further analysis of the applicability of complex correspondence detection techniques for automating the process of network federation. In this paper the possibility of fully automating the detection of complex correspondences was investigated, however a ROC curve analysis did not show to be feasible. This reinforced the view that when searching for complex correspondences, the most feasible approach is to use automation to reduce the search space to a ranked set of possibilities, and have a human user decide from these possibilities.

9.3 Further Work

This work focused on the development and evaluation of metrics which could be used in detecting Class Restrictions correspondences. A running assumption of the work was that, barring some quantum leap in artificial intelligence research, a fully automated method of detecting complex correspondences will never be possible, and that instead the technology developed for this task should help *guide* a human expert to make more informed decisions. Although it is outside the scope of this thesis, an important future consideration should be how the results produced by the detection methods developed here should be presented to human users so that they can make decisions, and how these decisions can be fed back into the detection process to iteratively produce better results.

Another issue that was made apparent during the development of this thesis, is that unlike one to one correspondences, there is no agreed upon test set for evaluating complex correspondences. The publication of standardised tests for correspondence detection has greatly influenced the direction of research in the field, with such a large number of papers published each year discussing advances in the detectors performance in these tests. Therefore if comparable improvement is to be seen in complex correspondence detection it is conceivable that the wide publication of some similar target would help greatly.

9.4 Final Remarks

James Clerk Maxwell once said that *"In every branch of knowledge the progress is proportional to the amount of facts on which to build, and therefore to the facility of obtaining data"* [56]. With the arrival of the Semantic Web and Linked Open Data we have the means to access huge volumes of data on any subject imaginable with little effort. Yet we have still not seen the end of Feynman's age of specialisation. Access to data is not the same as understanding data. It is not enough to know that two things are linked. We need to understand *how* they are linked.

References

[1] P. Shvaiko and J. Euzenat, "Ontology matching: state of the art and future challenges," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 158 – 176, 2013.

[2] N. F. Noy and M. A. Musen, "PROMPT : Algorithm and Tool for Automated Ontology Merging and Alignment," in *Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000, pp. 450–455.

[3] E. Vidal, L. Raschid, and M. Natalia, "Query Rewriting in the Semantic Web," in *International Conference on Data Engineering*, 2006.

[4] G. Li, L. Li, J. Zhang, and Z. Luo, "Mapping Based Automatic Instance Transformation," *Second Int. Symp. Intell. Inf. Technol. Appl.*, pp. 799–803, Dec. 2008.

[5] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A large ontology from wikipedia and wordnet," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 6, no. 3, pp. 203–217, Sep. 2008.

[6] G. A. Miller, "WordNet : A Lexical Database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[7] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia - A crystallization point for the Web of Data," *Web Semant. Sci. Serv. Agents World Wide Web*, no. 7, pp. 154–165, Sep. 2009.

[8] D. Ritze, C. Meilicke, O. Sváb-Zamazal, and H. Stuckenschmidt, "A pattern-based ontology matching approach for detecting complex correspondences," in *Proc. of Int. Workshop on Ontology Matching (OM)*, 2009.

[9] F. Scharffe, "Correspondence patterns representation. PhD thesis," University of Innsbruck, 2009.

[10] Gamma, Helm, Johnson, and Vlissides, *Design Patterns*. Addison-Wesley, 1994.

[11] J. David, J. Euzenat, F. Scharffe, and C. Trojahn dos Santos, "The alignment API 4.0," *Semant. Web*, vol. 2, no. 1, pp. 3–10, 2011.

[12] K. Eckert, A. Ferrara, L. Hollink, C. Meilicke, A. Nikolov, D. Ritze, P. Shvaiko, B. C. Grau, and B. Zapilko, "Results of the Ontology Alignment Evaluation Initiative 2012," in *International Semantic Web Conference*, 2012.

[13] A. Ngomo Ngonga, "Learning Conformation Rules for Linked Data Integration," in *Ontology Matchign Workshop*, 2012.

[14] P. Gillet, C. Trojahn, O. Haemmerlé, and C. Pradel, "Complex Correspondences for Query Patterns Rewriting," *Ontol. Matching*, 2013.

[15] R. Parundekar, C. A. Knoblock, and L. Ambite, "Linking and Building Ontologies of Linked Data," in *9th International Semantic Web Conference*, 2010.

[16] H. Qin, D. Dou, and P. Lependu, "Discovering Executable Semantic Mappings Between Ontologies," in *2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS*, 2007, pp. 832–849.

[17] D. O'Sullivan, "The OISIN framework: ontology interoperability in support of semantic interoperability. PhD thesis," Trinity College Dublin, 2006.

[18] H. Halpin, P. J. Hayes, J. P. Mccusker, D. L. Mcguinness, and H. S. Thompson, "When owl : sameAs isn ' t the Same : An Analysis of Identity in Linked Data," in *ISWC*, 2011.

[19] J. Lorey, Z. Abedjan, F. Naumann, and C. Böhm, "RDF Ontology (Re-) Engineering through Large-scale Data Mining," in *ISWC*, 2011.

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[21] E. Wilson, S. Vibhute, C. Bhatia, R. Jain, L. Perniu, S. Raveendramurthy, and R. Samuel, *Getting Started with InfoSphere Data Architecht*, 1st ed. IBM, 2011.

[22] G. H. L. Fletcher and C. M. Wyss, "Towards a General Framework for Effective Solutions to the Data Mapping Problem," *Lect. Notes Comput. Sci.*, vol. 5880, pp. 37–73, 2009.

[23] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with COMA++," *Proc. 2005 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '05*, p. 906, 2005.

[24] M. Uschold and R. Jasper, "A Framework for Understanding and Classifying Ontology Applications," pp. 1–12.

[25] A. Gómez-Pérez, M. Fernández-López, and O. Corcho, *Advanced Information and Knowledge Processing*. .

[26] W3C, "OWL Web Ontology Language Reference," 2011. [Online]. Available: http://www.w3.org/TR/owl-ref/. [Accessed: 08-Mar-1BC].

[27] M. Uschold and M. Gruniger, "Ontologies: Principals, Methods and Applications."

[28] J. Euzenat and P. Shvaiko, *Ontology Matching*. Berlin: Springer-Verlag, 2007.

[29] P. Shvaiko and J. Euzenat, "A Survey of Schema-Based Matching Approaches," *J. Data Semant.*, vol. 4, pp. 146 – 171, 2005.

[30] J. Euzenat, "An API for Ontology Alignment," in *3rd International Semantic Web Conference (ISWC)*, 2004, pp. 698–712.

[31] F. Scharffe and D. Fensel, "Correspondence patterns for ontology alignment," *Knowl. Eng. Pract. Patterns*, pp. 83–92, 2008.

[32] D. Brickley and L. Miller, "Friend of a Friend Specification," *http://xmlns.com/foaf/spec/*, 2010. [Online]. Available: http://xmlns.com/foaf/spec/. [Accessed: 17-Dec-2010].

[33] A. Pease, I. Niles, and J. Li, "The suggested upper merged ontology: A large ontology for the semantic web and its applications," in *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, 2002.

[34] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. De Melo, and G. Weikum, "YAGO2: exploring and querying world knowledge in time, space, context, and many languages," in *Proceedings of the 20th international conference companion on World wide web*, 2011, pp. 229–232.

[35] C. Caracciolo, J. Euzenat, L. Hollink, R. Ichise, V. Malaisé, C. Meilicke, J. Pane, and P. Shvaiko, "Results of the Ontology Alignment Evaluation Initiative 2008," in *ISWC workshop on Ontology Matching*, 2008.

[36] J. Euzenat, A. Ferrara, L. Hollink, A. Isaac, C. Joslyn, V. Malaisé, C. Meilicke, A. Nikolov, J. Pane, F. Scharffe, P. Shvaiko, and V. Spiliopoulos, "Results of the Ontology Alignment Evaluation Initiative 2009," *ISWC Work. Ontol. Matching*, vol. 1, 2009.

[37] A. Ferrara, C. Meilicke, A. Nikolov, J. Pane, P. Shvaiko, and H. Stuckenschmidt, "Results of the Ontology Alignment Evaluation Initiative 2010," in *ISWC workshop on Ontology Matching*, 2010.

[38] J. Euzenat, A. Ferrara, W. R. Van Hage, L. Hollink, C. Meilicke, A. Nikolov, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Svab-Zamazal, and C. Torjahn, "Final results of the Ontology Alignment Evaluation Initiative 2011," in *ISWC workshop on Ontology Matching*, 2011.

[39] B. C. Grau, Z. Dragisic, K. Eckert, A. Ferrara, R. Granada, V. Ivanova, E. Jim, and D. Ritze, "Results of the Ontology Alignment Evaluation Initiative 2013," 2013.

[40] P. Mayr, "Building a terminology network for search : the KoMoHe project Philipp Mayr Vivien Petras," in *International Conference on Dublin Core and Metadata Applications*, 2008, pp. 117–182.

[41] S. Massmann, S. Raunich, and D. Aumüller, “Evolution of the coma match system,” in *International Workshop on Ontology Matching*, 2011.

[42] R. Fagin, L. M. Haas, M. Hern, R. J. Miller, L. Popa, and Y. Velegrakis, “Clio : Schema Mapping Creation and Data Exchange,” in *Conceptual Modeling: Foundations and Applications*, vol. 5600/2009, 2009, pp. 198–236.

[43] H. Do and E. Rahm, “COMA: a system for flexible combination of schema matching approaches,” in *28th Intl. Conference on Very Large Databases (VLDB)*, 2002.

[44] R. Parundekar, C. A. Knoblock, and L. Ambite, “Discovering Concept Coverings in Ontologies of Linked Data Sources,” in *11th International Semantic Web Conference*, 2012, pp. 427–443.

[45] L. Rokach and O. Maimon, *Data mining with decision trees: theory and applications*. World Scientific, 2008.

[46] R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[47] R. Isele and C. Bizer, “Learning Linkage Rules using Genetic Programming,” *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 23, p. preprint, 2013.

[48] O. Sváb-Zamazal, E. Daga, and M. Dudás, “Tools for Pattern-Based Transformation of OWL Ontologies,” in *Proc. of Int. Semantic Web Conference*, 2011.

[49] M. Poveda, “Common pitfalls in ontology development,” in *Workshop on Ontology Quality, Co-located with EKAW*, 2010, no. 1.

[50] G. De Melo, F. Suchanek, and A. Pease, “Integrating YAGO into the Suggested Upper Merged Ontology,” *2008 20th IEEE Int. Conf. Tools with Artif. Intell.*, pp. 190–193, Nov. 2008.

[51] J. Euzenat, F. Scharffe, and A. Zimmermann, “Expressive alignment language and implementation (working paper),” 2007.

[52] G. Correndo and N. Shadbolt, “Translating expressive ontology mappings into rewriting rules to implement query rewriting,” in *International Workshop on Ontology Matching*, 2011.

[53] D. Hand, H. Mannila, and P. Smyth, *Data Mining*. Cambridge: MIT Press, 2001, p. 344.

[54] M. Nickel and O. Ring, “Factorizing YAGO Scalable Machine Learning for Linked Data,” in *21st International World Wide Web Conference*, 2012, pp. 271–280.

[55] H. Föllmer, “On entropy and information gain in random fields,” *Probab. Theory Relat. Fields*, vol. 22, no. 2, pp. 207 – 217, 1973.

[56] L. Campbell and W. Garnett, *The Life Of James Clerk Maxwell: With Selections from His Correspondence and Occasional Writings*. Merchant Books, 2008.

Appendix

Appendix A. Notation Glossary

In this thesis a selection of mathematical and visual notation are used which are not widely used standards. These include the following.

- **Equivalence correspondence:** $C_1 \equiv C_2$

  Class C_1 corresponds with class C_2 and they can be considered as equivalent. From an extensional point of view, every instance of C_1 can be considered an instance of C_2 and vice versa

- **Subsumption correspondence:** $C_1 \sqsubseteq C_2$

  Class C_1 corresponds with class C_2 but C_2 is has broader scope. From an extensional point of view, every instance of C_1 can be considered an instance of C_2, but not every instance of C_2 is an instance of C_1.

- **Restriction class:** $C|_{s=o}$ or $C|_{s=*}$

  $C|_{s=o}$ refers to the restriction class defined as all instances of class C which have attribute s set to value o.

  $C|_{s=*}$ refers to the restriction class defined as all instances of class C which have attribute s set to any value.

Appendix B. Evaluation data

The test sets and output logs for the evaluations described in chapters 7 and 8 are included in the DVD which accompanies this manuscript. This section provides a brief synopsis of this data.

Test sets

The test sets used in both these evaluations consist of a synthetically created ontology based on sub classes of dbpedia-owl:Person. Fifty such classes are included in this ontology. For each class there is a gold standard restriction class based on DBpedia attributes which forms a complex correspondence with DBpedia. The folder /`testsets` in the root of the DVD lists 50 files labelled `set_`*`i`*`.txt` for i = 1..50. with the ith file describing the gold standard restriction class for the ith test class, with the DBpedia URIs of the instances of the ith class.

For example test class C_1 is described in `/testsets/set_1.txt.` The gold standard correspondence between this class and DBpedia is:

$$C_1 \equiv \text{dbpedia-owl:Person}|_{\text{dbpedia-owl:stateOfOrigin= dbpedia:Scotland}}$$

The first 5 instances this class shares with dbpedia-owl:Person are:

`http://dbpedia.org/resource/Alexander_Selkirk`

`http://dbpedia.org/resource/John_Brown_%28servant%29`

`http://dbpedia.org/resource/Flora_MacDonald_%28Scottish_Jacobite%29`

`http://dbpedia.org/resource/Robert_Baillie`

`http://dbpedia.org/resource/John_Witherspoon`

Evaluation Log for Chapter 7

A log from the evaluation described in chapter 7 is provided in the file `/matched_set_size_eval.log`. This file lists the mean rank given to the gold standard correspondence for each class in the test set using FS1 and differing sizes of matched instances.

For example the log entry

```
GS = http://www.w3.org/2002/07/owl#Person,
http://dbpedia.org/ontology/stateOfOrigin,
http://dbpedia.org/resource/Scotland
Using 3 instances
FS1 rank: m=1.6, var=0.2400
```

States that the gold standard cotrrespondence is :

$$C_1 \equiv \text{dbpedia-owl:Person}|_{\text{dbpedia-owl:stateOfOrigin= dbpedia:Scotland}}$$

A sample of 3 matched instances are being used. The mean score of the gold standard using FS1 was 1.6 and the varience was 0.024

Evaluation Log for Chapter 8

The file `/FS1_FS2_differences.csv lists` the output of the evaluation described in chapter 8. The columns in this file record the difference in the score for the golden score for FS1 and FS2 for 50 different instance samples for each of the 50 test classes. A positive score indicates FS1 performed better and a negative score indicates FS2 performed best. A score of 0 shows they performed equally well.

Each row of the file describes the scores on datasets with increasing levels of missing data ranging from 0% to 50% in increments of 5%.

Appendix C. Inventory of Complex Correspondences

This section provides an inventory of the complex correspondences discovered between ontologies in the course of this thesis.

Complex correspondences between DBPedia and YAGO2

Table 9 describes class by Attribute Value correspondences were found between DBpedia and YAGO2.

Table 9. Class by Attribute Value correspondences between DBpedia and YAGO2

Named Class	Restriction Class
yago:Actor	rdf:type (X, dbpedia:Person) ∧ dbpedia-owl:occupation (X, Actor)
yago: *PeopleFromToronto*	rdf:type (X, dbpedia:Person) ∧ dbpedia-owl:birthplace (X, dbpedia:Toronto)
yago:Musician	rdf:type (X, dbpedia:Person) ∧ dbpedia-owl:occupation(X, Musician)
yago:Politician	rdf:type (X, dbpedia:Person) ∧ dbpedia-owl:occupation (X, Politician)

Table 10 describes Class by Attribute Existence correspondences which were found between DBpedia and YAGO2:

Table 10. Class by Attribute Existence correspondences between DBpedia and YAGO2

Named Class	Restriction Class
yago:Director	rdf:type (X, dbpedia:Person) ∧ dbpedia-owl:director(_,X)
yago:Actor	rdf:type (X, dbpedia:Person) ∧ dbpedia-owl:starring (_,X)

Complex correspondences between DBpedia and GeoNames

Table 11 describes Class by Attribute Value correspondences found between DBpedia and GeoNames.

Table 11. Class by Attribute Value correspondences between DBpedia and GeoNames

Named Class	Restriction Class
dbpedia:Park	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "L.PRK")
dbpedia:Mountain	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "T")
dbpedia:BodyOfWater	rdf:type (X, geo:feature) ∧ geo:featureClass(X, "H")
dbpedia:Country	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "A.PCLI")
dbpedia:Island	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "T.ISL")
dbpedia:Airport	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "S.AIRP")
dbpedia:Hospital	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "S.HSP")
dbpedia:Bridge	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "S.BDG")
dbpedia:Lake	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "H.LK")
dbpedia:River	rdf:type (X, geo:feature) ∧ geo:featureCode(X, "H.STM")