On the Automatic Generation of Case Libraries by Chunking Chess Games

Stephen Flinter and Mark T. Keane

Trinity College Dublin

Abstract. As a research topic computer game playing has contributed problems to AI that manifest exponential growth in the problem space. For the most part, in games such as chess and checkers these problems have been surmounted with enormous computing power on brute-force search methods using massive databases. It remains to be seen whether such techniques will extend to other games such as go and shogi. One suggestion is that these games and even chess might benefit from a knowledge-based treatment but such approaches have met with limited success. The problem, as ever from such approaches, is the characterisation of the knowledge to be used by the system. This paper deals with the Tal system, which employs case-based reasoning techniques for chess playing. In the paper, rather than focus on playing, we concentrate on the *automatic* generation of suitable case knowledge using a chunking technique on a corpus of grandmaster games.

1 Introduction

Many games like chess and checkers have been mastered by computational techniques that use extensive search with massive computing power. The failure of such techniques to deal with other games, like go and shogi, is perhaps an indication that this approach does not constitute a general solution to such problems. One alternative is to use knowledge-based techniques, like case-based reasoning, to reduce the search overhead and speed-up solution generation. After all, human chess players play quite well, with relatively little search, through the use of extensive past experience. Several knowledge-based solutions have been attempted but with limited success (e.g. [2] [10]). As always, the fundamental problem with knowledge-based approaches is the isolation of the relevant knowledge in a suitable form for use.

Case-based reasoning (CBR) seems to be a plausible technique to apply to such games. However, again, with it we face the problem of defining a suitable case library to support the technique. In this paper, we consider a chess playing system, called Tal, that uses CBR. To date, the central problem we have focused on in Tal is the development of a case library. There are two problems to be faced in doing this. First, one has to develop a good case representation. It is not obvious what form chess cases should take: for example, whether they should contain parts of a game or a full game, whether they should represent all the details of a game at the level of actual moves from play or whether some abstracted representation should be developed. In TAL, cases are represented as abstractions of actual board situations.

Second, one needs to automate the method of building cases. For an adequate level of expertise, a considerable case library is likely to be needed. Given this factor, and the availability of extensive databases of chess games, it makes sense to have a method that will generate a case library *automatically* from records of previous games. In TAL, we use a *chunking* method to do just this. The reader should note that the notion of chunking to which we refer is inspired by de Groot's ([6]) and Chase and Simon's ([3]) ideas and is quite different to the chunking method used by Laird, Newell and Rosenbloom in SOAR ([9]).

1.1 Architecture

The TAL system consists of two major components: the Library sub-system and the Game-Playing sub-system. The Library component is responsible for converting the set of example games into populated knowledge bases, as described in Section 4. The Game Playing component is responsible for applying this knowledge to novel game positions to produce candidate moves and to perform some look-ahead.

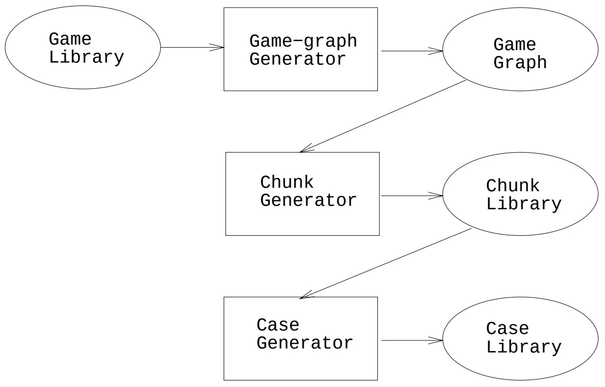

Fig. 1. TAL Library Sub-system

The Library component is shown in Figure 1. It has three main sub-components to generate the three knowledge-bases (game graph, chunk library and case library) used by TAL: the game-graph generator, the chunk generator, and the case generator. The *Game-graph Generator* takes a set of chess games and creates a game-graph based on those games. The *Chunk Generator* produces set of *candidate chunks* from the game-graph. The *Case Generator* takes the set of base chunks and recombines them into set of cases.

The Game-playing component is shown in Figure 2. It uses the knowledge-bases created by the Libraries Component to analyse a given current chess position, and to select a move to play from that position. Several interesting issues

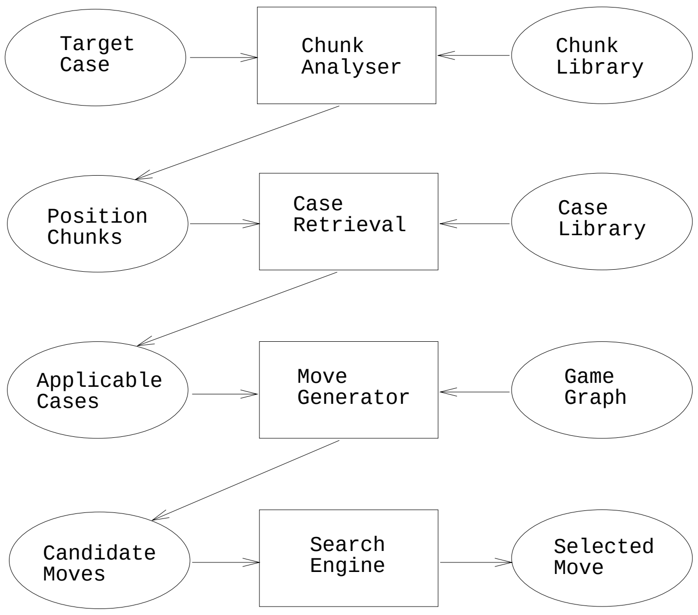

Fig. 2. TAL Game-Playing Component

arise in the design of this component, for instance, the generalisation and re-specialisation of candidate moves. However, this paper will not be concerned with the game-playing aspect of TAL, rather it focuses on the Library component and the generation of case knowledge.

1.2 Outline of Paper

In the following sections, the workings of TAL's Library component are sketched in detail. However, before doing this we consider some of the background to the ideas used in this component, in the form of psychological evidence on the use of chunks by chess players. While our idea of chunking is quite different to that used in the psychological work, and we do not make any claims for the psychological plausibility of TAL, we have been inspired by this research.

2 Psychological Evidence

According to psychological evidence, human chess players of all strengths perform what Newell and Simon called *chunking* ([6] [11] [7]). A *chunk* here is some portion of a board position represented by a meaningful grouping of pieces. Each complex board position can be broken down into a small number of chunks, each of which will contain a number of pieces. Consider the chess position shown in Figure 3. This figure is taken from [7], which itself is taken from [6]. The sets of ringed pieces represent the chunks which de Groot suggests his chess master subject used to represent the position.

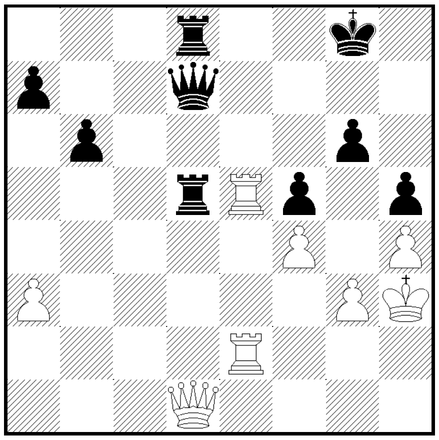

Fig. 3. Reference chunk position

As a chess player's expertise increases chunks change in two ways. First, the *size* of the chunks increase — that is, the chunks a master uses to represent a position will have more pieces than those of an intermediate level player, which in turn will have more than a rank beginner. Secondly, the *number* of chunks possessed by the player also increases with playing strength. So, not only will a master represent chess positions using richer and more complex chunks, but will also use more of them. It has been estimated that an intermediate club player will have about 1,000 chunks, while a master will have of the order of 100,000 ([6]).

TAL does not try to model this human chess playing skill — this has already been attempted a number of times, with varying success ([13] [3]) — but it does *borrow* ideas from this research. Specifically, the idea of breaking a complex situation, such as a chess position, into a number of smaller, more manageable pieces is a very useful technique for case-based reasoning; especially, if the cases can be generated automatically, using heuristic methods applied to a large data set. This process of automatically generating the required knowledge-bases is analogous to the learning which takes place as a human player gains experience.

3 Theory of Chunking Applied to TAL

As it appears in TAL, the *chunking hypothesis* could be stated as the belief that complex situations or episodes, such as chess positions, can be better understood in terms of a number of simpler component structures. Specifically, given a set of representative chunks (representative of the entire set of chess games, that is), one can break a previously unseen chess position into a set of such chunks. The composition, arrangement and relationships of the set of component chunks can then be used to retrieve similar previously seen positions (reminding). The hypothesis goes on further to postulate that a move made in a similar position to the one under examination is worthy of consideration as a move for the current position, once it has undergone the appropriate transformation or adaptation, which will express it in terms of the current position. Such a move is called a *candidate move.*

It is not necessary that every candidate move be relevant to the current position — in fact, it is important that *not* every move be relevant: otherwise, we return to the problem of exponential growth in our search which we are trying to avoid with this method. Once this set of candidate moves has been generated, it is then possible to search the game-tree, like any ordinary chess program, but using the candidate moves to guide the search. In this manner, the hope is to reduce the average number of moves to be searched at a given chess position from 35 down to about three or four. However, there are certain problems which must be resolved in implementing this chunking hypothesis. We should have:

- a, preferably, automatic method for refining the initial set of example games to a set of base chunks (achieved by our *chunking heuristic*).
- a means of filtering out unwanted and useless chunks from the set of candidate chunks, leaving only useful and meaningful base chunks.
- a way of creating a set of base cases from the set of base chunks, where these cases represent situations in which one or more chunks are present. Here a base case, will be a position expressed in terms of chunks from the chunk library rather than in terms of atomic pieces.
- a method for characterising novel, unseen positions into a collection of chunks from our chunk library.
- a retrieval method to find a set of relevant similar positions from the game graph to this characterisation of a target position
- a set of methods for generalising and re-specialising moves made from similar positions, to put them in terms of the new position. In the instances, where this is impossible, the candidate move is simply discarded. Once we have a set of candidate moves, we can then begin to search from them, and repeat the above process for as long as necessary.

4 The Inner Workings of Tal's Library Component

In the next three sections we will explain in more detail the nature and composition of each of Tal's knowledge bases.

4.1 Game-graph Generator and Game Graphs

The *Game-graph Generator* takes a set of chess games, in Portable Game Notation (PGN — a standard notation for chess games), and creates a game-graph based on those games. The game graph is a cyclic-directed graph which stores raw chess games. Each node in the graph represents a single, unique chess position and is stored in Forsythe-Edwards Notation (FEN — another standard notation, this time for chess *positions*). Each arc from one node to another represents a legal move made from one chess position to another. The graph is directed because the effects of a move are not (always) reversible, and cyclic because positions can repeat. The graph is tree-like in that it has a single root which is the initial chess position. Finally, both the nodes and the arcs have *counter tags*, which specify the number of times a given node or arc appears across the set of games submitted to the graph.

A complete chess game is represented as a path or *thread* through the graph, linking the positions occurring in the game by the moves made from position to position. Each unique position can be represented by one and only one node. If two or more games share the same position (which is inevitable), they must share the same node. The graph can have only one root — the initial chess position — and will have one or more leaves; with each leaf representing the termination of a game, corresponding to a win, draw or loss for white (or black).

4.2 Generating Chunks for the Chunk Library

The chunk library is a repository of base chunks. Each chunk is composed of a *salient piece* and one or more other pieces. The salient piece of a chunk is the most important piece: the one from which other pieces hang. For example, recall the position shown in Figure 3. If we consider a white pawn chunk, with the three pawns on f4, h4, and g3 (note that, this does not correspond exactly to the master's chunks shown in the diagram, and is used simply for the sake of argument). Together, these pawns make up a cohesive chess unit. The salient piece in this chunk is the pawn on g3, as it is the piece which relates all three — without this pawn the other two would just be single, separate, unrelated units.

The purpose of the Chunk Generator is to traverse the game-graph produced by the Game-graph Generator and to produce a set of *candidate chunks*. A candidate chunk is defined to be a chunk created according to some chunking heuristic, which will be either discarded or added to the chunk library according to some *chunk selection criterion*. So, there are two issues which the Chunk Generator must tackle: firstly, how should the chunks themselves be composed; and secondly, once they are composed, how can the most relevant and useful ones be selected, for inclusion in the chunk library.

In Tal candidate chunks are generated according to the principle of *direct interaction*. Two pieces are said to directly interact in situations where one defends the other (in the case where both pieces are of the same colour), or where one attacks the other (in the case where the pieces are a different colour). Specifically, each candidate chunk consists of a salient piece, and the enumeration of all direct interactions with that piece. The algorithm for the generation of the set of candidate chunks follows.

```
For each piece in the position:
let the piece be the salient piece;
```

```
if the piece is black, normalise the position to white;
determine all the direct interaction between the salient piece and the remainder
of the pieces in the position;
enumerate each possible permutation of salient piece and directly interacting
piece(s), and store as a candidate chunk.
```

The second issue facing the Chunk Generator is that of selecting a set of base chunks based on the set of candidate chunks. This task must yield a small set of chunks (relative to the set of candidate chunks) which are representative of the total set of chunks required to express each position. Obviously, there will be a large amount of duplication across the set of candidate chunks, and it is this factor which is used as the chunk selection criterion. Specifically, the frequency of each chunk is determined (by frequency here we mean the number of times it occurs across the entire game set), and a selection made on the basis of this frequency. Refer to Section 5 for more details on the results to date.

4.3 Case Library

Just as the chunk library is a repository of chunks, so the case library is a repository of cases. In the Tal system, a case is a representation of a board position in terms of a number of base chunks found in the chunk library. Specifically, a case consists of the set of base chunks which appear in the examined position. Thus, when a novel position is encountered, it is decomposed into its constituent chunks. Its similarity to other positions can then be measured by a distance metric between the current position and the other positions encoded in the case library.

5 Results

At the moment, the implementation of Tal's Library Component is complete, and work on the Game-Playing component is under-way. We have conducted experiments using a set of 350 games from the late grandmaster and ex-world-champion, Mikhail Tal. This game set yields a total of 43,592 unique candidate chunks. An analysis of this data reveals an L-shaped distribution. Those chunks which have a low frequency tend to have a very large count. For example, there are 16,352 different chunks with a frequency of 1; that is, from the set of 43,592 chunks from, approximately 38% of those appear only once. Conversely, those chunks which have a very high frequency have very low counts — typically only one or two.

Since we are looking for a representative set of chunks from the entire set of possible chunks, those points lying on the extremities of the distribution are uninteresting. Those with a low frequency and high count are not useful because they occur too infrequently to be of assistance in retrieving relevant positions. That is, the cost of storing them exceeds the benefit gained from greater coverage. Equally, those chunks with a high frequency and low count are not useful because they are not discriminating enough. Storing them would direct the system to retrieve too many cases to handle. For the current system, we use a chunk library approximately 10% the size of the complete set of possible chunks. From the candidate chunk set, we have pruned those chunks with a frequency 10 or less, and a frequency 500 or more. This selection criterion leaves a chunk library with a population of 4,533 base chunks. A graph of the distribution of the remaining chunks can be seen in Figure 4.

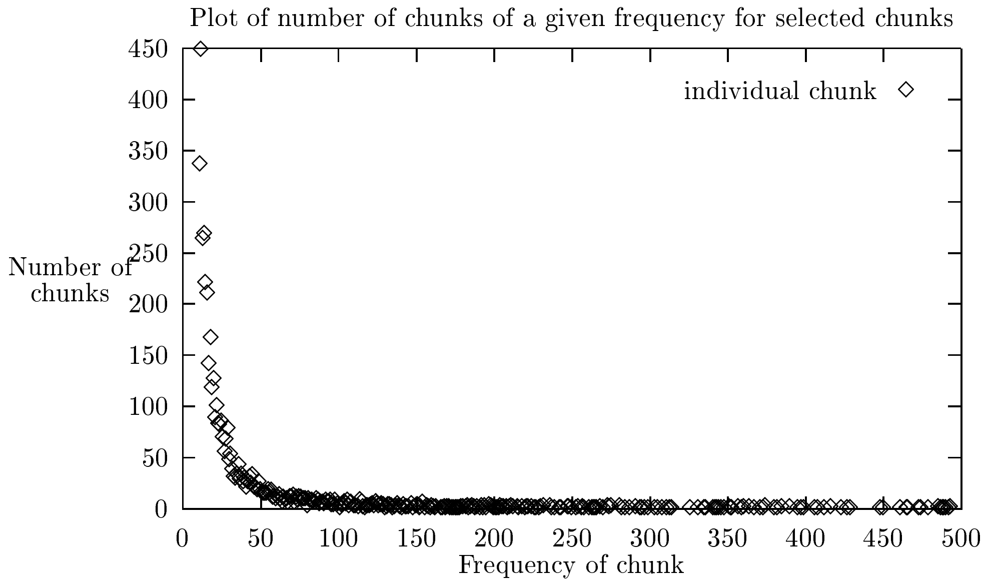

Fig. 4. Plot of selected chunks

When we chunk the reference position found in Figure 3 using Tal we get the *automatically generated* chunks shown in Figure 5. As can be seen, there is some similarity between the two sets of chunks, without them being identical. This is not a critical factor: we are not trying to reproduce the same chunks of a human master player, but simply using these chunks as aids to retrieval.

6 Future Work

The work described in this paper mainly revolves around the generation of the knowledge bases. In the next set of experiments we will focus on case retrieval, and in particular, on generalisation and re-specialisation of retrieved candidate moves. The primary issue here is to correctly map the move made from the retrieved case into an appropriate move for the target case, and to recognise the situations where no such meaningful maps exists.

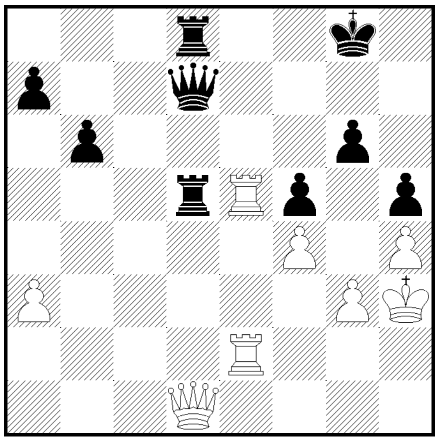

Fig. 5. Base chunks chosen by TAL

In this paper we have described TAL's current chunking heuristic and chunk selection criterion. These are the two most important factors in determining the quality of the system's chunk library, and hence ultimately its case library. In future experiments, we intend to investigate different chunking heuristics and selection criteria.

In the long term, while much of TAL is closely tied to the game of chess, we believe that the approach of automatically chunking complex situations and building cases out of such chunks is generally applicable and useful. One of the biggest problems with current CBR technology is the dependence on hand-crafted case libraries. While TAL does not completely solve this problem — we still had to encode the chunk representation and chunking and selection criteria — it does go a long way towards alleviating the time-consuming and pain-staking task of building the individual base cases.

7 Related Work

There is a large project underway in the Institute of Learning Sciences at Northwestern University named CASTLE. This project takes a Model-Based Reasoning (MBR) approach to computer chess ([4] [5] [1]). The ILS approach is a very knowledge intensive one, and involves planning, learning and high-level reasoning. Kerner has recently published work on a CBR approach to chess ([8]). However, his work focuses on developing a case-base *evaluation function* for a standard searching chess system, and so differs significantly from TAL. Pell's work on METAGAME ([12]) deals primarily with forcing programs to learn, and in particular, to create their own evaluation function.

8 Summary and Conclusions

In this paper, we have described the Tal CBR system, and in particular, its Library Component. We have demonstrated a method for automatically generating a case library from expert data. The hypothesis advanced by this work that positions composed of similar base chunks will tend to have a similar chess 'meaning', and that reasoning performed on one can usefully be applied to the other has so far been supported. Further work is necessary to refine the generation of base chunks and cases, but experiments indicate that this approach is a useful and fruitful one. Further, we propose that the approach to such problems, that of "chunking" complex situations, is one which could be generally applicable to case-based reasoning.

This article was processed using the LaTeX macro package with LLNCS style