From: AAAI-83 Proceedings. Copyright ©1983, AAAI (www.aaai.org). All rights reserved.

A Chess Program That Chunks

Murray Campbell
Hans Berliner

Computer Science Department
Carnegie-Mellon University

Abstract

CHUNKER is a chess program that uses chunked knowledge to achieve success. Its domain is a subset of king and pawn endings in chess that has been studied for over 300 years. CHUNKER has a large library of chunk instances where each chunk type has a property list and each instance has a set of values for these properties. This allows CHUNKER to reason about positions that come up in the search that would otherwise have to handled by means of additional search. Thus the program is able to solve the most difficult problem of its present domain (a problem that would require 45 ply of search and on the order of 10^{13} years of CPU time to be solved by the best of present day chess programs) in 18 ply and one minute of CPU time. Further, CHUNKER is undoubtedly the world's foremost expert in its domain, and has discovered 2 mistakes in the literature and has been instrumental in discovering a new theorem about the domain that allows the assessing of positions with a new degree of ease and confidence. In this paper we describe CHUNKER's structure and performance, and discuss our plans for extending it to play the whole domain of king and pawn endings.[1]

1. Introduction

Humans are known to *chunk* chess positions [4, 5], i.e. treat logically related groups of pieces as units. They do this apparently to aid in evaluation of positions, and to suggest strategies for continuing the game, although psychological evidence does not make it clear exactly how this is done. There has been at least one program [9] that is able to recognize some chunks, but no successful chess-playing program has ever used a chunking approach.

The present work deals with a program, CHUNKER, that parses a position into chunks, and reasons about the position using information obtained from chunk libraries. There are several chunk libraries, each corresponding to a fixed group of pieces (a *chunk type*). Each chunk type has certain properties, and each *chunk instance* (a particular configuration of the chunk type) has values attached to the properties. These are used both in evaluation, by allowing reasoning with chunk property values to produce an evaluation of the whole position, and in move selection, to facilitate reaching positions where this is possible. Within its domain CHUNKER is a true expert, playing the positions with a speed and accuracy that no present human or machine can come close to matching.

2. The Domain

The domain for this study is king and three connected passed pawns (3CPP) vs. king and 3CPP,[2] for example, Figure 2-1. This type of ending has a number of advantages for our purposes:

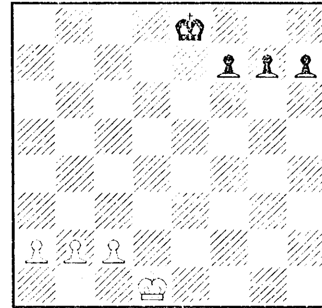

Figure 2-1: A position in the domain

- Parsing a position into chunks is relatively straightforward.
- The ending is non-trivial; errors have been found in the literature, and even strong players have difficulty with the ending's special intricacies.
- A table driven program, with a database of all positions, is impractical, requiring about 2^{29} entries.

The most important chunk type in this problem domain is a king vs. 3CPP (Kv3CPP), for example Figure 2-2. Given a position such as Figure 2-1, much can be determined by discovering propeties of the chunks in isolation and relating them to each other. The manageable size of the libraries containing properties of the individual chunks is the key ingredient in the success of this approach.

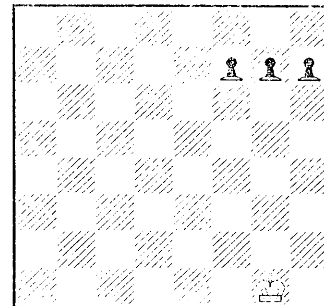

Figure 2-2: A chunk in the domain

Given that the king wins if all the pawns are captured, while the pawns win if one safely reaches the eighth rank, the battle within a chunk of Kv3CPP can be viewed under various assumptions.

- Sides must alternate moves.
- The king has the option to 'pass', i.e. make a null move.
- The pawns have the option of passing.

---

[1]This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-78-C-1551, and in part by an educational grant from the Province of Alberta, Canada.

[2]with certain restrictions. To avoid the inclusion of queen and pawn endings, the first side to promote a pawn safely is considered to have won.

The passing option corresponds to moving on the other side of the board. Classifying every Kv3CPP configuration with respect to each assumption allows a great deal to be understood about the chunk. Consider the positions in Figure 2-3. In position 2-3-a, it can be shown that whichever side has the move loses (under the alternating moves assumption). Therefore, it is clear that if one side only has the passing option, that side will win no matter who is to move. In position 2-3-b, it can be shown that the pawns win, whether the king has the passing option or not. Intuitively this means that the pawns are far enough advanced to force a breakthrough. Position 2-3-c illustrates a situation where one of the pawns is lost, even with the passing option.

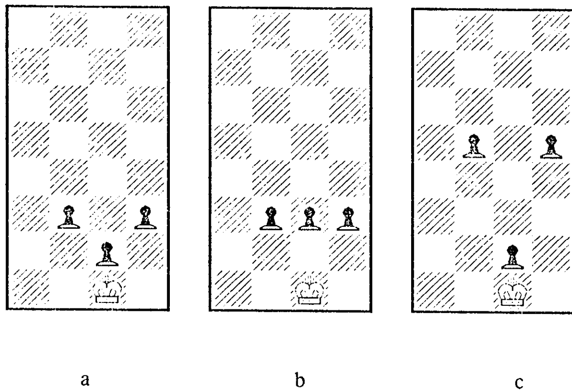

Figure 2-3: Some chunk instances

The other chunk type relevant to this domain is king and 3CPP vs. king (K3CPPvK), where the lone king has the passing option (Figure 2-4). This chunk type corresponds to the situation where a king abandons its battle against the enemy pawns and attempts to support its own pawns.

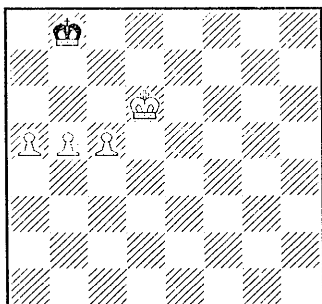

Figure 2-4: A K3CPPvK chunk

Strictly speaking there is a third chunk type in this domain, namely 3CPP unopposed. The only property of interest here is the number of moves required for a pawn to reach the eight rank. It is a simple matter to calculate this without need of a library.

It is relatively easy to produce libraries for an appropriate set of assumptions. The basic technique [6, 3] could be termed retrograde enumeration, and involves producing an entry for each possible position. The immediate wins are then labelled, and the enumeration proceeds by successively labelling those positions whose values can be determined based on positions they are known to lead to. When a side is trying to win, it is enough to know that one move leads to a win; when a side is lost, all successors must be shown to lose. One of our contributions is to use the technique, not on whole board positions (where it can have only limited use since the state space grows exponentially with the number of pieces), but on pre-defined partial board configurations. In this ending, each of the Kv3CPP libraries is 64 Kbytes. These libraries are the alternating-moves (AM) library, the pawns-can-pass (PCP) library, and the king-can-pass (KCP) library. The K3CPPvK library restricts the kings to certain areas of the board, and requires 512 Kbytes.

3. The analysis method

3.1. Level 0: The Base Program

The basis of CHUNKER is a full-width, depth-limited, alpha-beta search. Terminal nodes are those where a pawn has safely reached the eighth rank, stalemates/checkmates, or draws by repetition. Undecided positions at maximum depth are normally scored as draws. Under certain circumstances (king in check, pawn on seventh rank) positions are allowed to quiesce beyond the depth limit.

A number of techniques are used to improve searching performance:

- Moves are statically ordered to improve the chances that the best move is considered early in the search. (Alpha-beta achieves optimal performance if the best move is considered first in every position [10]).
- A hash table is used to detect positions that have occurred previously in the search, and if the position has been searched to a sufficient depth, then the stored value in the table entry is used [11, 8].
- Moves that cannot be part of the solution tree are pruned. For example, if White has a passed pawn that is out of reach of the black king, moves of the black king are useless and not considered, unless the king is attempting to support its own pawns.

The base level of CHUNKER is able to evaluate approximately 600 positions per second on a VAX 11/780. Feasible search depths range from about 15-25 ply, depending on the type of position. Since most of the standard positions in the literature require more than 25 ply to reach a conclusion (when delaying tactics are included), the base level of CHUNKER has limited usefulness. The following levels of CHUNKER are each built on the previous levels.

3.2. Level 1

If it could be shown that a particular side is winning (or losing) in *both* chunks, it would be possible to classify the overall position as a win (or a loss). The AM library provides this capability. Table 3-1 illustrates the cases that can be classified by this method. Each chunk has two associated values, corresponding to the results for

	WK/BP Chunk			
BK/WP Chunk	WTMW BTMW	WTMW BTMB	WTMB BTMW	WTMB BTMB
WTMW BTMW	W/W	W/?	W/W	?/?
WTMW BTMB	W/?	?/?	W/B	?/B
WTMB BTMW	W/W	W/B	B/W	B/B
WTMB BTMB	?/?	?/B	B/B	B/B

Table 3-1: Positions decided by AM library

White to move and Black to move. The labelling of the rows and columns indicates which side is to move and which side wins (e.g. WTMB means White to move, Black wins). The columns indicate these properties for the white-king/black-pawn chunk, and the rows for the black-king/white-pawn chunk. The first entry in a table slot is appropriate if it is White to move, and the second if it is Black to move. If there is not a '?' in the entry at the appropriate location in the table, then the value of the whole position is known. The use of this table allows CHUNKER to terminate searching a branch in many positions (approximately 18% of the total legal state space) that previously required large searches (more than 10^6 nodes) to solve (see Section 4). Of course many types of positions cannot be classified by the above approach, and so positions such as Figure 2-1 remain intractable.

3.3. Level 2

It can be demonstrated that 3CPP with the passing option always win against a king if none of the pawns can be safely captured by the king. The PCP library uses this observation to evaluate positions in which one side has lost a pawn. If Side A has lost one pawn (such that the pawns now lose in the AM library), then side B is given a win if he can keep all 3 of his pawns. This schema can classify about 16% of the total legal positions, but is not entirely disjoint from level 1.

3.4. Level 3

The KCP library allows CHUNKER to classify positions where the pawns win into two types: those where the pawns are strongly enough placed to force a breakthrough, and those that rely on *zugzwang*, i.e. the compulsion of the king to move. Only if a position appears as a win for the pawns in the KCP library do the pawns have a forced breakthrough. This knowledge allows many further positions to be terminal nodes in the search. If Player A can force a breakthrough while Player B requires zugzwang to win, Player A simply forces his pawns through; he is no longer under any compulsion to move his king. Approximately 18% of the total positions can be classified in this way, although there is some overlap with Level 1.

3.5. Level 4

Up until this point, CHUNKER has made no use of the numerical values stored in the libraries, only their 'parity'. In attempting to classify positions where both sides can force pawn breakthroughs, it is tempting to assume that each side will push its pawns forward so that the side that breaks through in the fewest ply is the winner. Unfortunately this fails due to the fact that certain pawn moves require king responses (e.g. the king is placed in check). The solution that allows this type of position to be evaluated is to keep an auxiliary library of *spare tempi* for each position, i.e. the number of free moves the king has before the pawns break through. Thus allowing the king to pass adds one to the number of spare tempi, while responding to a check does not. In positions where both sides have pawn breakthroughs, it can be shown that the side with the fewest spare tempi (accounting for side-to-move effects) will lose. This method classifies approximately 12% of the legal positions.

3.6. Level 5

A theorem has been developed about a certain class of chunks which we call 'Z' configurations. A Z configuration is one in which the pawns win, whichever side is to move (*double win*), and the win is accomplished by zugzwang, not breakthrough (as determined from the KCP library). Further, in a Z configuration the pawns are able to maintain the double win against any king move, either because the new position is still a Z configuration or by making a pawn move that establishes a new Z configuration. The theorem states: *If the pawns of one side have set up a Z configuration they cannot lose unless the other side has a breakthrough, and they will win unless the other side also has a Z configuration, in which case the position is a draw.* See [2] for the proof of this theorem, and examples of Z configurations. This schema classifies less than .1% of the total state space, but will nonetheless be shown to have a significant effect when it is applicable.

4. Results

The most authoritative book on this subject, **Pawn Endings [1]**, contains 13 positions (535:547) in this ending. Table 4-1 gives the results of CHUNKER's search of these positions. Each entry includes an estimate (in parentheses) of the search depth required for a complete solution. (Those problems that were solved at level 0 at a lesser depth achieved this due to the quiescence search.) Within the table, the first figure is nodes visited and the parenthesized number is the ply limit. Entries containing '----' are intractable (more than 10^6 nodes visited).

The monotonic improvement across rows is striking.[3] The addition of a single new schema for classifying positions can produce speed-ups of 6 orders of magnitude or more by avoiding the searches that would be required without that schema. In many positions, adding a schema produces little or no improvement in the performance. This could mean that the solution was already minimal, or that the new knowledge item had limited relevance to the given position. When a new schema is relevant though, search improvements can be spectacular: there are many instances of improvements of over three orders of magnitude.

[3]Two exceptions occur in the table. Paradoxically, it is possible for alpha-beta search to perform more poorly when knowledge is added. Consider the case where the first move examined in a position loses. Suppose a search with a certain schema finds this loss, while the search alone cannot (and scores the position as a draw). In the latter case, the improved alpha value of 0 (draw) up from the starting value of -1 (assumed loss), allows more cutoffs in the remainder of the search.

Position	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
535: (33)	----	9879 (13)	6438 (13)	1944 (13)	108 (6)	108 (6)
536: (33)	----	10381 (13)	10497 (13)	6002 (13)	185 (8)	185 (8)
537: (25)	----	2678 (9)	2498 (9)	1677 (9)	60 (4)	60 (4)
538: (35)	----	3636 (8)	3572 (8)	1917 (8)	223 (8)	2 (1)
539: (23)	307749 (21)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)
540: (27)	----	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)
541: (31)	----	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)
542: (33)	----	13 (2)	13 (2)	13 (2)	13 (2)	1 (0)
543: (17)	9051 (15)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)
544: (30)	----	8778 (11)	8829 (11)	1232 (10)	509 (10)	509 (10)
545: (27)	940534 (23)	3804 (7)	3804 (7)	25 (4)	25 (4)	1 (0)
546: (33)	----	110468 (17)	102421 (17)	9905 (14)	3332 (14)	1 (0)
547: (45)	----	----	----	----	211513 (18)	23962 (18)

Table 4-1: Performance on Test Positions

The effect of the hash table on these searches should not be discounted. For example, position 547, searched at level 5 without the hash table, required 420,507 nodes, a factor of 17 slower. Other positions also show considerable performance benefits, especially those requiring large searches.

To gain a feeling for the importance of the respective schemas, the percentage of terminal nodes classified by each are listed below. The data is derived from the level 5 search of position 547 (Figure 2-1):

- terminal nodes - 6.6%
- hash table lookups - 11.8%
- level 1 - 66.8%
- level 2 - 5.3%
- level 3 - 6.6%
- level 4 - 1.7%
- level 5 - 1.2%

Clearly the level 1 schema is dominant in terms of positions classified, but this is partly due to the fact that it is the first to be tried. Although level 5 accounts for only 1.2% of the positions classified, it produced an almost nine-fold speedup in the search. A similar improvement also occurs at level 4 with an additional classification of only 1.7% of the positions. Thus, the ability to classify even a small additional percentage of positions can produce very large savings in search. This must be dependent to some degree on how near the root of the tree the knowledge can be applied. We can only assume this is random; however, each problem clearly evinces a point where it begins to become tractable, and from then on the addition of knowledge produces dramatic improvements until what appears to be a minimal search is hit.

The above results were obtained using the 'automatic' parse of positions into two chunks of Kv3CPP, which is the intended theme for this set of problems. Since we wanted to investigate alternate position parsings, we examined the possibility of decomposing a position into one chunk of K3CPPvK, and another of 3CPP unopposed. The K3CPPvK library contains the number of spare tempi the lone king has before the pawns force promotion. If the king and pawns in cooperation can force a pawn through before an unopposed pawn promotes in the other chunk, the position is classified as a win. The choice of this alternate chunk parsing is made based on both the result of the position under the usual decomposition, and an estimate of the probability of success of such a strategy determined from the piece placement. For example, if the position in Figure 4-1 is interpreted in the usual way, it is clear that White is lost (see Level 2). Realizing that the white pawns are far advanced gives rise to the possibility of the alternate parse, which leads to the conclusion after a short investigation that White can win by joining his king to the pawns in an attack on the black king.

Since the problem set was solved correctly when chunk interaction is ignored, adding interaction to the Level 5 version of CHUNKER does not significantly affect any of the results or searches of the test positions. Only positions 536 (about 5% more nodes) and 547 (about 1% more nodes) showed any effect at all. Of course in some positions (that are not in the text set, for example Figure 4-1), the possibility of an alternate parsing is essential in determining the correct solution.

The chess machine Belle [7] was run on positions 535, 539, 543 and 545. Belle is capable of searching 130,000-160,000 nodes a second. In the 4 positions tested, Belle, taking on the order of 2.5 hours per position, played a correct move in each case. However only in one of the positions was Belle able to actually see that the selected move forced a win (or a draw). In fact, we estimate that Belle would require on the order of 10^{13} years to completely solve Figure 2-1. For a more detailed discussion of Belle's performance on these positions, see [2]. It should be noted that, though Belle would probably be able to play correctly the 4 positions presented to it, it is extremely doubtful if it could correctly evaluate them if they occurred deep in its search tree. Further, in position 547 (Figure 2-1) where a small misstep such as initial P-QN3 (b3) turns a winning position into a losing one, it would seem that only a "theory" of what the ending is all about, such as gained by our schemas, would suffice to play correctly.

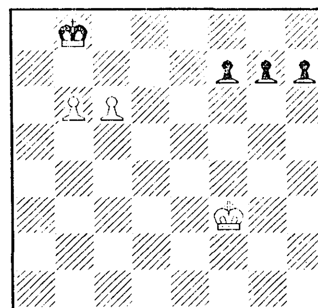

Figure 4-1: A position that requires an alternate chunk parse

During the course of this research, CHUNKER discovered errors in the solutions [1] to problems 546 and 547. In position 546 the kings do not have to oscillate between the designated squares (f3/f4, c5/c6) to draw; both sides have alternate methods of drawing. In position 547 1. P-QR4 (a4) does not win as stated in the book. It actually allows the clever setting up of a Z configuration, the importance of which is clearly not appreciated by the authors.

5. Generalization

5.1. The analysis method

While the scheme of chunked knowledge with properties, exploited by search, works extremely well in the given domain, the question should be asked as to how general this scheme is. In the given domain there are only three chunk types, and the problem of parsing a board position into chunks is simplified due to the fact that there are only three possible position decompositions.

We expect to generalize the basic scheme to the whole domain of king and pawn endings. To do this many more chunk libraries will be required, and these will require many new properties. New reasoning schemas that correspond to the needs of particular positions will be developed to use the new properties. We consider that a moderate set of such schemas will suffice for this domain so that it will not be necessary to have a general reasoning engine. The ultimate analysis method in CHUNKER will be a search through a set of alternative parsings of a position, rather than the standard search through a set of states of the domain. When a parsing does not lead to an immediate evaluation, further searching must be done, as in the present work. Typically both players will have the option of enforcing certain parsings on a position. The search through alternate parsings terminates when it is shown that one player can win (or draw) against any position decomposition chosen by the opponent. A detailed account of analysis/search methods and libraries can be found in [2].

5.2. Libraries

We expect to build a permanent set of libraries dealing with configurations that are frequently encountered. These would include typical battles between pawn formations, for example king and pawn vs. king, or two pawns facing two. Relations other than those used in the present work would included number of moves to establish a passed pawn, spare moves available without losing material, invasion points for opposing king, etc.

Some types of chunks that could occur are so complex and rare that to build libraries for all possibilities would require unrealistically large memories. For such chunks (containing doubled pawns for instance) it will be necessary to do the analysis to produce properties on the fly. However, once such an analysis is done, these properties can be retained in a temporary chunk library for the duration of the solution process.

5.3. Search

In the present domain CHUNKER performs adequately using a depth-limited depth-first alpha-beta search with hash-table support. In a more general case, the search may be oriented to specific purposes, started from various sub-trees, and restricted in format. It is expected that some form of search will be necessary for most positions. However, positions very likely fall into two classes: 1) Those where it appears possible to make a determination of the exact game-theoretic value of the position, and 2) those where general principles have to be invoked to find a likely value and a most-likely-best move. In the first case, chunk schemas and search should be able to handle the problem. In the second case an evaluation polynomial in the properties, in conjunction with a search will be needed to give a direction to the solution process when it is not expected to reach any known goal. This method is also necessary for defensive play in losing positions, since if all losing positions are classified as equal there will be no criterion for the defense to put up a fight.

6. Summary and Conclusions

We have presented a new methodology; one for allowing a program to manipulate properties of chunks found in chunk libraries in order to evaluate whole chess positions. While it has been known that humans use chunked knowledge to come to grips with the complexities of a chess position, no effective method for doing this has previously been demonstrated. This is also apparently true for other domains of even moderate complexity.

One of the difficulties that "practical" AI systems that reason have had is that the "facts" that they reason from are almost always *ad hoc* collections input by the system designers. This creates great difficulties both in maintaining the consistency of a set of "facts" and in producing adequate coverage of the domain in the face of the difficulties in maintaining consistency. Our method avoids both these problems. By generating our facts from an exhaustive search, we guarantee accuracy for the facts. This makes the conclusions derived from them completely trustworthy, and would, in principle, allow the building of several levels of reasoning upon such a structure.[4]

We demonstrate a simple instance of our method on a subset of king and pawn endings in chess. The higher level of abstraction due to chunking provides a framework for powerful reasoning schemas in the domain. It is quite remarkable to see the effect of the additional concentration of knowledge on the tractability of the problems in the test set. As additional facts and reasoning schemas are invoked, even the extremely complex base position recedes to a point where one minute of CPU time reveals all its mysteries, an 18 order of magnitude speed-up.

We have shed some light on the role that chunks have in problem solving. It was previously known that chunks are a way of breaking up images into component parts. In a large domain, it is more reasonable to catalog parts than total images which will probably never be encountered again. However, the computational utility of chunking was not clear. In our method, chunks and the relations among them serve as data for a problem solver. While this combination may use more time than looking up of images, in a large domain there is no meaningful alternative. The nature of a chunk is apparently determined by *functional* considerations.[5] These functional considerations define the information needed for the problem solving process. In the end, a chunk name becomes a slot to which properties and their values can be attached.

The method can be extended to more complicated pawn endings [2]. This involves the creation of additional chunk libraries with new properties, and the extension of the search and reasoning methods to take advantage of these. We estimate that the domain can be covered with on the order of 60 to 100 reasoning schemas and approximately 30 properties that would be employed in the schemas.

The program that has been developed in the course of this research is the world's foremost expert in its restricted domain and has found two corrections in the existing literature, made it possible to develop a new theorem about its domain, and allowed the composition of a number of worthwhile additions to the existing literature. Despite the fact that both present authors are expert chess players and have had quite a bit of exposure to the domain of this study both from books and from the results of computing experiments, CHUNKER regularly outperforms its creators in novel situations. This augurs well for the promise of the technique.

References

1. Averbakh, Y. and Maizelis, I. *Pawn Endings*. B.T. Batsford Ltd., 1974.

2. Berliner, H. and Campbell, M. Using chunking to solve chess pawn endgames. Carnegie-Mellon University, April, 1983.

3. Bramer, M. A. Computer-generated databases for the endgame in chess. The Open University, October, 1978.

4. Chase, W. G. and Simon, H. A. "Perception in Chess." *Cognitive Psychology 4*, 1 (January 1973).

5. Chase, W. G. and Simon, H. A. The Minds Eye in Chess. In *8th Annual Psychology Symposium Volume: Visual Information Processing*, Chase, W. G., Ed.,Academic Press, 1973, ch. 8, pp. 215-281.

6. Clarke, M. R. B. A Quantitative Study of King and Pawn against King. In *Advances in Computer Chess 1*, Clarke, M. R. B., Ed.,Edinburgh University Press, 1977.

7. Condon, J. H. and Thompson, K. Belle Chess Hardware. In *Advances in Computer Chess 3*, Clarke, M. R. B., Ed.,Pergammon Press, 1982.

8. Marsland, T. A. and Campbell, M. "Parallel Search of Strongly Ordered Game Trees." *Computing Surveys 14*, 4 (December 1982), 533-551.

9. Simon, H. A., and Gilmartin, K. "A Simulation of Memory for Chess Positions." *Cognitive Psychology 5* (1974), 29-46.

10. Slagle, J. and Dixon, J. "Experiments with some programs that search game trees." *JACM 2* (April 1969), 189-207.

11. Slate, D. and Atkin, L. CHESS 4.5 - The Northwestern University chess program. In *Chess Skill in Man and Machine*, Frey, P., Ed.,Springer-Verlag, 1977, ch. 4.

---

[4]While this method appears to be quite general, we make no claim that it is ready for implementation in more than a few select domains. Clearly, there must be a way of defining useful chunk boundaries and a method of generating property values before the enterprise can be undertaken.

[5]In the present case, a chunk is an assemblage required to assess the outcome of a battle of pawns (possibly supported by king) versus king.