

Computational Chunking in Chess

By

Andrew Cook

A thesis submitted
to The University of Birmingham
for the degree of Doctor of Philosophy

School of Computer Science
The University of Birmingham

February 2011

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

ABSTRACT

Adriaan de Groot, the Dutch psychologist and chess Master, argued that “perception and memory are more important differentiators of chess expertise than the ability to look ahead in selecting a chess move” (Groot 1978). A component of expertise in chess has been attributed to the expert having knowledge of ‘chunks’ and this knowledge gives the expert the ability to focus quickly on “good moves with only moderate look-ahead search” (Gobet and Simon 1998). The effects of chunking in chess are widely reported in the literature, however papers reporting the nature of chunks are largely based on inference from psychological experimentation. This thesis reports original work resulting from extensive data mining of a large number of chessboard configurations to explore the nature of chunks within the game of chess and the associated moves played by expert chess players. The research was informed by work in the psychology of chess and explored with software engineering techniques, employing large datasets consisting of transcripts from expert players games. The thesis reports results from an analysis of chunks throughout the game of chess, explores the properties of meaningful chunks and reports effects of the application of chunk knowledge to move searching.

ACKNOWLEDGEMENTS

There are many people who have helped with this work but in particular I would like to thank my supervisor, Dr. William Edmondson for his inspiration and guidance through many relaxed and enjoyable meetings, Prof. John Barnden and Dr. Volker Sorge as members of the thesis group for keeping me on-track, Dr. Allan White for advice on statistical methods, my wife Lynda for her patience and encouragement since starting this work, and to my father for showing that age does not put a limit on positive thinking, determination and learning.

TABLE OF CONTENTS

1. INTRODUCTION	1
1.1. Chunking and other areas of artificial intelligence	4
2. THE QUESTION THIS STUDY AIMS TO ANSWER	6
3. AN INTRODUCTION TO CHUNKING IN CHESS	10
3.1. Literature review	10
3.2. Chapter conclusion	21
4. DEFINING A CHUNK	22
4.1. Chunks are learnt constellations	22
4.2. Chunks are frequently occurring configurations	23
4.3. Recall of chunks are separated by a two second boundary.....	23
4.4. Chunks can be a tool to extend short-term memory	24
4.5. Chunks contain elements that are related to each other	25
4.6. Pieces are related by proximity	26
4.7. Pieces are related by attacking/defending relationships	26
4.8. Chunks are absolutely positioned	27
4.9. Experts have a larger chunk knowledge than the novice.....	29
4.10. The relationship between chunk definitions.....	30
4.11. Chapter conclusion	31
5. INVESTIGATING THE PROPERTIES OF CHUNKS	32
5.1. What a chunk looks like	34
5.2. Chunk statistics.....	36
5.2.1 Why are there so many chunks on a chessboard?.....	36
5.2.2 The frequency of chunks in relation to player skill	38
5.2.3 Removing the most common chunks	41
5.3. Testing the skill/chunk relationship using a Pearson Correlation	44
5.4. 'Defensive' chunks	45
5.4.1 The occurrence of defensive chunks throughout a game	48
5.4.2 The persistence of defensive chunks with player skill	49
5.5. Chapter conclusion	50
6. THE DEVELOPMENT OF A PROGRAM TO INVESTIGATE CHUNKING IN CHESS	51
6.1. CLAMP - 'Chunk Learning And Move Prompting'	51
6.2. Building chunk libraries	53
6.3. Building chessboard collections	53
6.4. Move asymmetry: A case against Holding (1985)	59
6.5. Analysing chunks from collections	61

6.6.	Chunk size and memory requirements.....	64
6.7.	CLAMP design, data structures and processes.....	66
6.8.	Combining 'nodes' to build 'lists'.....	68
6.9.	Building 'lists' from collections.....	71
6.10.	The structure of a 'Trie'.....	72
6.11.	Combining 'tries' to make a 'library'.....	75
6.12.	A graphical representation of a chunk.....	76
6.13.	Hardware considerations.....	79
6.14.	Chapter conclusion.....	80
7.	USING 'CLAMPANALYSER' TO SUGGEST MOVES.....	82
7.1.	An evaluation of scores for a move to a position.....	84
7.2.	A few large chunks or many small chunks?.....	88
7.3.	Adjusting for 'move rareness'.....	89
7.4.	Changing the 'success' threshold.....	91
7.5.	The relationship between pieces within chunks.....	92
7.5.1	The number of chunks in a chunk library.....	94
7.5.2	The 'effectiveness' of a chunk.....	94
7.5.3	Analysis of the whole board area.....	95
7.5.4	Chunks and meaning.....	97
7.5.5	Analysis of chunks in small grouped areas on the chessboard.....	98
7.5.6	Analysis of chunks comprising of pieces in defensive relationships ..	102
7.6.	Changing the 'success' threshold with defensive chunks.....	106
7.7.	An analysis of de Groot's Position 'A'.....	108
7.8.	The Bratko/Kopec tests.....	112
7.8.1.	Bratko/Kopec Test 4 (best move: pawn lever).....	113
7.8.2.	Bratko/Kopec Test 5 (best move: tactical).....	115
7.8.3.	Bratko/Kopec Test 6 (best move: pawn lever).....	117
7.8.4.	Bratko/Kopec Test 7 (best move: tactical).....	119
7.8.5.	Bratko/Kopec Test 8 (best move: pawn lever).....	121
7.8.6.	Bratko/Kopec Test 9 (best move: pawn lever).....	123
7.8.7.	Bratko/Kopec Test 11 (best move: pawn lever).....	125
7.8.8.	Bratko/Kopec Test 13 (best move: pawn lever).....	127
7.8.9.	Bratko/Kopec Test 14 (best move: tactical).....	128
7.8.10.	Bratko/Kopec Test 15 (best move: tactical).....	130
7.8.11.	Bratko/Kopec Test 16 (best move: tactical).....	132
7.8.12.	Bratko/Kopec Test 20 (best move: pawn lever).....	134
7.9.	An analysis of the top moves.....	135
7.9.1.	Using 'Whole Board' chunk libraries.....	137
7.9.2.	Using 'Defensive' chunk libraries.....	139
7.9.3.	Using small grouped area chunk libraries.....	141
7.10.	Top move analysis and the number of boards scored.....	145
7.11.	Using chunks to suggest a move: a design strategy.....	147

7.12. Chapter conclusion	149
8. AN EVALUATION OF PIECES MOVED FROM A POSITION	151
8.1. 'Move From' scores by using 'Move To' analysis	153
8.2. Combining the likelihood of a <i>Move To</i> with the <i>Move From</i> score	155
8.3. Chapter conclusion	158
9. AN APPLICATION OF CHUNKING TO A CHESS PLAYING PROGRAM	159
9.1. A brief look at the MINIMAX routine	159
9.2. Optimising the MINIMAX search with alpha-beta pruning.....	162
9.3. Using CLAMP to optimise the alpha-beta search	165
9.4. Chapter conclusion	170
10. CLAMP AND CHUMP: A COMPARISON.....	171
10.1. The aims of the programs	171
10.2. Chunk acquisition.....	171
10.2.1 Analysis of the whole board.	172
10.2.2 Analysis of local areas on the board	172
10.2.3 Analysis of pieces in 'defensive' relationships to each other.	172
10.2.4 The 'eye-movement-simulator'.....	173
10.2.5 Attack, defence and proximity relationships	173
10.3 Chunk repository.....	173
10.4 Chunk Size	174
10.5 Move proposals	174
10.6 The move start and end squares.....	175
10.7 The size of the learning set.	175
10.8 Test results from the Bratko-Kopec positions	175
10.9 Test results from de Groot's Position 'A'	176
10.10 Conclusion	176
11. CONCLUSION	178
11.1 About chunking in chess	178
11.2 The question this thesis aims to answer.....	180
12. RECOMMENDATIONS FOR FURTHER WORK.....	183
13. APPENDICES	185
13.1. Appendix 1: Summary of results	185
13.2. Appendix 2: The key for the axis: 'Move to piece/position'.	187
13.3. Appendix 3: Supplementary media	188
13.4. CLAMPanalyser.....	188
13.4.1. The library naming convention for the CLAMPanalyser program	190
13.4.2. Counting the number of chunks in a chunk library.....	191
13.4.3 <i>The LibraryComparator</i>	192
13.4.4 The output from the LibraryComparator program	193

13.4.5	The 'TopMovesComparator' program.....	194
13.4.6	The output from the TopMovesComparator program	195
13.4.7	SourceCode.....	195
13.4.8	Collections_MoveTo	197
13.4.9	Collections_MoveFrom	198
14.	BIBLIOGRAPHY	199

1. INTRODUCTION

The Russian mathematician Alexander Kronrod in 1965 described chess as the “Drosophila of artificial intelligence” (McCarthy 1997). Drosophila is a genus of fruit fly, one species of which is famously known for use in genetics experimentation. The Drosophila has as a result become synonymous with scientific experiments. A sentiment similar to Kronrod’s was expressed by Saariluoma (1998) *“Chess has been, and to some extent still is, at the forefront of adversary problem-solving research. It just happens to be a well-defined task environment, which nevertheless provides information about complex problem-solving processes, and this is why it has been used for decades as the fruit fly of thought psychology”*. Saariluoma (op. cit.) described chess as a “compact and easily controllable task environment” and the game “a mental contest between two individuals with each move the result of careful thought.” Saariluoma (1995) also describes chess as *“a two player game with perfect information and no chance moves. This means that a position in chess contains all the information that is needed to make a correct choice of move.”* Chess therefore lends itself to studies in thinking and problem solving. For this reason chess has frequently been a tool for psychological research into human thinking and expertise (Charness 1981, Chase and Simon 1973a, 1973b, de Groot 1965, 1978, 1998, de Groot and Gobet 1996, Finkelstein and Markovitch 1998, McGregor and Howes 2002, Gobet and Jansen 1994, Gobet et al. 2001, Jongman 1968, Reynolds 1992, Saariluoma 1998, 2001, Simon and Barenfield 1969, Simon and Gilmarin 1973).

This thesis reports the use of the game of chess to look at an interesting aspect of cognitive functionality. Miller (1956) proposes that the human short-term memory has the capacity to remember seven, plus or minus two, items simultaneously. The finding is intriguing when considering the capacity of long-term

memory and other capabilities of the human brain (Gilhooly and Logie 1998) and Saariluoma (1998) describe the union of short term and long-term memory: “the small capacity of the working memory can be circumvented by using long-term working memory” (Ericsson and Kintsch, 1995). “This collaboration of the two main memory systems is undoubtedly a very important mechanism” (Saariluoma op. cit.). Furthermore, according to Miller (op. cit.), items are grouped in long-term memory in ‘chunks’, chunks being items that are grouped together by some common factor.

Miller (op. cit.) illustrates the above point as follows *“A man just beginning to learn radio telegraphic code hears each dit and dah as a separate chunk. Soon he is able to organise these sounds into letters and then he can deal with the letters as chunks. Then the letters organise themselves as words, which are still larger chunks, and he begins to hear whole phrases... I am simply pointing to the obvious fact that the dits and dahs are organised by learning into patterns and that as these larger chunks emerge the amount of message that the operator can remember increases correspondingly. In the terms I am proposing to use, the operator learns to increase the bits per chunk.”*

A person’s learning and expertise, in this context, is therefore an exercise in increasing chunk knowledge.

Chunking is not restricted to human cognition. Research on pigeons shows evidence that pigeons have an ability to cluster five element lists into distinct groups implying that the pigeon can ‘chunk’ items in long term memory, albeit with a short term memory limitation of five items (Chase 2000, Terrace 1987). The point is also made by Terrace that chunking in this instance is being performed in a brain that is void of ‘linguistic competence’. Rats (Cohen et al. 2001, Fountain and Benson 2006) and monkeys (Terrace 1987) also show evidence for chunking when learning.

Regarding human learning and expertise, chess has been used as the experimental 'fruit fly' for psychological study. As a medium for research chess has advantages.

The chessboard consists of just sixty-four squares with a maximum of thirty-two chess pieces comprising of six types, each type with defined rules on how the piece can move and relate with other pieces. The starting point is always the same and the objective is clearly defined: to checkmate the opposing king. "The rules of chess are sufficiently simple that children can be taught them at a very young age (four or five years old)" (Gobet and Charness, 2006), Yet despite the small number of parameters that define the game, chess provides a hugely variable system with a vast number of possible board configurations. The number of different *games* that can be played has been estimated at about 10^{120} . This number, known as the 'Shannon number' after Claude Shannon (1950) who first estimated it, is described by Shannon as 'conservative'. This hugely variable output provides a 'fine grained' system that can be precisely analysed by virtue of the small number of parameters that define the game.

Data from chess games are easy to acquire. Many hundreds of thousands of tournament games between experts, Masters and Grandmasters showing each move played have been recorded and catalogued, and are easily downloadable from the Internet in a standard file format. From this wealth of information it is possible to gain a glimpse into some of the processes that craft human thought. Or put another way, chess research provides *"good empirical evidence on some issues including the relation of memory and problem-solving which has very seldom, if at all, been researched in other task environments"* (Saariluoma 2001).

Chunking and the acquisition of expertise is an area of research that has been explored in chess (Charness 1981, Chase and Simon 1973a, 1973b, de Groot 1978,

1965, de Groot and Gobet 1996, Gobet 1998, Gobet et al. 2001, Holding 1992, Jongman 1968, Ross 2006, Saariluoma 1980, 1998, 2001, Simon and Barenfield 1969, Simon and Gilmarin 1973, McGregor and Howes 2002). The skill of players is measured and documented in tournament games, and comparisons between players is relatively simple. Experiments to look for evidence of chunking in chess are also numerous. In addition there have been a number of computer programs that search for chunks or simulate chess play with chunks (Berliner and Campbell 1984, Gobet and Jansen 1994, Walczak 1992). There is however little detail about what actually constitutes a chunk with respect to chess play.

1.1. Chunking and other areas of artificial intelligence

The term 'chunking' is used in other areas of artificial intelligence research in relation to machine learning, however the meaning of the term is not universal across all areas. The program 'Soar' for example, uses the term 'chunking' for "a learning mechanism that acquires rules from goal-based experience" with the "acquisition and use of macro-operators" (Laird, Rosenbloom and Newell 1986). The use of the term 'chunking' in this context is referring to the linking together of rules. 'Chunks' within the context of Soar are solutions to sub-problems. Within Soar's operation "when Soar does not have the knowledge to provide a solution it establishes a sub-problem. The solution to a sub-problem is saved as learned production, which can be termed 'a Soar chunk' (Kennedy and Trafton 2006). By grouping together rules chunking in Soar "leads to performance improvements" (Laird, Rosenbloom and Newell 1984).

Soar takes a 'top down' approach to problem solving. Sub-rules are added only when there is an impasse and the higher production rules do not provide a solution. CLAMP on the other hand, is bottom up, building relationships between chunks and actions. In the context of chess chunks are configuration of chess pieces

on the chessboard which is different to chunks in Soar - which are the linking together of production rules.

In other applications the term 'Chunking' can be used in the context of breaking up complex data in to smaller components, for example chunking within the HTTP protocol refers to the process of breaking a large message into smaller messages. Within the field of natural language processing the term 'chunking' refers to the breaking up of a sentence into short phrases for identification of parts of speech. Chunking is sometimes referred to as 'shallow parsing' as it can break up a sentence, for example into noun phrases, locations and names, without constructing a full parse tree (Bird, Klein and Loper 2009).

The term 'Chunking' can therefore have various meanings, depending of the application or area of research. In this thesis the term 'chunking' refers to the process whereby chess pieces are combined into groups. A 'chunk' is simply a group of some of the chess pieces that appear on a chessboard and the action of 'chunking' is the grouping together of chess pieces. The research reported in this thesis is based on the analysis of the chess piece chunks.

2. THE QUESTION THIS STUDY AIMS TO ANSWER

“While current computers search for millions of positions a second, people hardly ever generate more than a hundred. Nonetheless, the best human chess players are still as good as the best computer programs. Although this model operates excellently in computer programs, it has very little realism where human thinking is concerned. It is probabilistic and in most task environments the generation of all possibilities even to the depth of one ‘move’ is unrealistic. In making an investment decision, for example, one cannot normally generate all imaginable ways to invest and heuristically select the best: there simply exist too many ways to make the decision. This is why heuristic search models are too coarse to be realistic models of the mind. Much more sophisticated analysis is required in order to explain human problem-solving behaviour” (Saariluoma 1998).

The research reported within this thesis is a study of ‘chunking’ in the constellations of pieces on the board within the game of chess. Chase and Simon (1973b) link expertise in a domain, including expertise in chess, with knowledge of chunks as a mechanism employed by the human expert (as opposed to performing an evaluation of all possible moves), however, the question whether chunking can be linked to chess skill is open to debate. Holding (1985) argues against the notion that chunk knowledge is linked to skill but maintains that master chess players are better at chess because they are better at looking ahead. Holding’s ‘SEEK’ (**S**earch, **E**valuate, and **K**now) model attributes skill in chess to the player’s ability to efficiently negotiate search trees by using their knowledge of chess, and in their judgements in end positions (Holding op. cit.). Holding rejects what he calls ‘recognition-association’ theory as the basis of chess skill. On the other hand, Chase and Simon (1973b)

attribute knowledge of chunks as the major differentiator between the novice and the expert players.

Much of the evidence for chunking in chess is taken from psychological experiments such as de Groot's memory test on expert and novice players. In this well-known experiment de Groot tested three classes of chess player: Grandmaster plus Master, Expert and Class 'A' player, (a 'Class A player' is a good chess player, but below expert level), by showing them a chessboard configuration from an unfamiliar game with twenty-two pieces on average, for a few seconds (de Groot 1978). The subjects were then asked to reconstruct the configurations, either verbally or on another board. The experiment was repeated by Chase and Simon (1973b) but included a novice group. The results showed Masters scoring 81% correct, Class 'A' players 49% and the novices 33%. But when the positions were randomised each group only recalled only three or four pieces correctly. This dramatic result implies that advanced chess players remember pieces in structured positions, and that pieces are remembered as groups or chunks rather than the individual pieces themselves.

In this thesis the link between knowledge and use of chunks, and chess skill, is investigated. Within the chess community, including the psychologists who study chess play, a question remains: *Do chunks in chess games differentiate categories of player?* The question is posed this way because experimentally this permits the underlying question to be studied - at least in part: *Do chess players use chunks in their analysis of a chessboard?* The alternative possibility is, of course, that chunking is a by-product of chess play, and not in any sense a driver of good chess play which is intentionally exploited by players. Whilst experimental differentiation of category of player, on the basis of chunks, would not demonstrate the value of chunking for the player it would offer the prospect that style of play could be shown to be linked to

chunking. This would be an indirect demonstration of the value of chunking in chess play, without the need to claim that experts use chunking explicitly.

However, in a computational environment it is possible to go further. Large numbers of chess games played by different categories of chess player can be analysed computationally and chunks identified. The deployment of chunking in a computational chess environment can be used to assess the value of chunking in new games.

Experiments similar to the Chase and Simon (1973b) experiment have been performed (and are described in the literature review section of this document) and suggest an expert chess player is equipped with knowledge of chunks. Simon and Gilmartin (1973) claim that an expert may know the order of 50,000 chunk patterns, however, the parameters that define a chunk are largely unknown. CHUMP (a 'pattern-learning move generator') uses eye movement information inspired by the gaze of chess players to extract configurations from a board based on pieces that are located at the eye fixation points (Gobet and Jansen 1994). By scanning a corpus of training games a number of eye fixation points can be compiled, with each fixation point limiting the area of the board from which chunks can be built, and with the chunks in turn being associated with a resulting move. Eye movements are *simulated* so that a large number of games can be processed, counting the frequency of occurrence of chunks associated with a move. CHUMP's extraction of chunks is based on the eye movements of chess players, but the actual reasons for the eye movements remain unknown.

Walczak (1992) introduced a system called IAM which restricts the board view to a subset of pieces in close proximity. An area of 4x4 or 5x5 pieces for example is viewed and the frequently occurring chess pieces compiled to make a chunk library. McGregor and Howes (2002) performed a series of experiments that suggest the

attack and defence relationships of pieces within chess play are more important than proximity.

This thesis will attempt to add to existing research literature by looking at chunking within a computational environment. The work reported in this thesis will:

- Look at chunks in chess play and attempt to define the properties of chunks within chess.
- Investigate potential links between the use of chunks and a player's skill.
- Isolate effective chunks.
- Incorporate chunking as part of the move generation process within a chess program.

The question this thesis will answer is:

Can the utilisation of chunks in a chess-playing program provide a plausible model for the use of chunks by human players?

3. AN INTRODUCTION TO CHUNKING IN CHESS

3.1. Literature review

Mainstream computer chess programs use search trees with a very large number of nodes, exploring every possible subsequent move and counter-move, several ply¹ ahead of the current position in order to test the consequence of a proposed move. This method, which is known as the MINIMAX routine was proposed by Claude Shannon in the 1950s (Shannon 1950). The number of positions evaluated grows

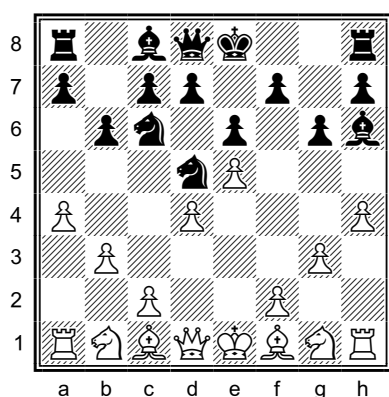


Figure 3.1: A typical chessboard

rapidly with each move ahead. For example, starting with the chessboard on the left of this page,² and looking ahead just four ply (commercial chess programs would look ahead at least eight ply) results in 2,080,734 nodes. Yet despite employing optimisations (of which the most significant is the alpha-beta search method

which is described in detail on page 159) the process requires several million-node evaluations. Human chess players however can out-perform, or at least present a serious challenge to even the most powerful chess computers. The chess computer 'Deep Blue' for example, which famously defeated the world chess champion Garry Kasparov in 1997, was capable of searching up to 40 ply, working through a potentially huge number of nodes (Campbell, Hoane and Hsiung-Hsu 2002). The methods used by a human chess player are therefore of interest as the notion that the player mentally evaluates millions of chess configurations is generally rejected. Saariluoma (1998) maintains, when describing the heuristic search of all possible

¹ A 'ply' is a move of one chess piece, either a white piece or the corresponding move of a black piece.

² The chessboard configuration shown in Figure 1 can be represented in Forsyth-Edwards Notation (or 'FEN') as: r1bqk2r/p1pp1p1p/1pn1p1pb/3nP3/P2P3P/1P4P1/2P2P2/RNBQKBNR w KQkq - 0 1

moves: “although the model operates excellently in computer programs it has very little realism where human thinking is concerned”. Adriaan de Groot, the Dutch psychologist and chess Master, argued that one of the most important aspects of skill in playing chess is not in the thought processes, searching through a tree of possible moves, but in the initial coding of relationships among the pieces on the chessboard. “It is not easy to appreciate fully the enormous effect of the expert’s reproductive completion of the perceived situation, as his perceptual advantage might be called” (de Groot 1978, pp. 307). He concluded “that perception and memory are more important differentiators of chess expertise than the ability to look ahead in selecting a chess move”.

Jongman (1968) suggested that master chess players retained the names of familiar piece configurations in ‘short term memory’ whilst remembering the piece configurations, or ‘chunks’, in ‘long term memory’. The estimate for short-term memory span of seven, plus or minus two items (Miller op. cit.) is however arguably too generous, as the span of short-term memory is dependent on the type of material being remembered (Cowan 2001). Other research suggests that even Cowan’s estimate of a maximum of four chunks being held in short term memory could be “an overestimate” (Gobet and Clarkson 2004). In any case, chunking provides a mechanism where a limitation, be it seven or less, can be compensated by remembering groups of items in long term memory thereby increasing the short term memory capacity from seven items to “seven groups of items” by “grouping primitive stimuli into larger conceptual groups” Gobet et al. (2001). The application of this technique is apparent when grouping letters into words, or even words into sentences, the memory limit of seven letters is thereby expanded considerably. With respect to chess, chunking theoretically expands the ability to remember configurations of a few pieces on the chessboard to a few *groups* of pieces. The

groups in this case being recognised configurations that occur frequently within the game.

Gobet et al. (2001) describes the cognitive function of chunking as one of the “key mechanisms of human cognition, linking the external environment and internal cognitive processes”. Chunking, despite constant cognitive limitations, explains how greater knowledge can lead to an increased ability to extract information from the environment. Expertise in the visual recognition of features in x-ray imagery has also been attributed to chunking with claims that "a chess master performs in the same manner as does a radiologist" (Wood 2009). Experiments in chunking have included non-human animals suggesting that chunking is a cognitive process that is not limited to humans. Tests on pigeons by Terrace (1987) show that lists can be memorised by pigeons with the aid of chunking, giving evidence for the association between the chunked items. Similarly rats negotiating a maze show evidence for chunking when presented with recognisable sequences (as markings on the floor) within a maze (Cohen et al. 2001). Experiments with fourteen-month-old infants show chunking as a cognitive technique for extending memory at a young age in human development (Feigenson and Helberda 2004).

A significant contribution to the debate on chunking and chess was made by de Groot (1978) and Chase and Simon (1973) with experiments involving the memory performance of chess players of varying skills. Chase and Simon claim a link between knowledge of chunks and the chess players' skill. The more skilful players are believed to have memorised a larger number of chunks.

In one experiment de Groot tested three classes of chess player: Grandmaster plus Master, Expert and Class ‘A’ player, by showing them a chessboard configuration from an unfamiliar game with twenty-two pieces, for a few seconds (de

Groot 1978). The subjects were then asked to reconstruct the configurations, either verbally or on another board. The results were as follows:

Master/Grandmaster:	93% Correct
Experts	72% Correct
Class 'A' Players	51% Correct

Table 3.1: de Groot's recall test.

Chase and Simon (1973b) repeated the experiment, but with the addition of a novice group and their test results were as follows:

Master	81% Correct
'A' Class Player	49% Correct
Novice	33% Correct

Table 3.2: Chase and Simon's test using game piece placements

But when the positions were randomised *each* group recalled only three or four pieces correctly. The conclusion of this experiment is that expert chess players remember groups of chess pieces in structured positions.

In another experiment, Gobet and Simon (1996a) compare the recall ability between chess Masters and weaker players by presenting a chessboard to each for just a few seconds. Briefly presented positions are remembered better by chess Masters compared with weaker players when the positions are meaningful. If the chess positions are random then Masters, to a large degree, lose their advantage. The small advantage that strong chess players show when remembering random boards can be attributed to small chunks appearing in the random patterns. Further investigation by Gobet and Simon (2000) reinforced this suggestion by simulating the

experiment using a large chunk database acquired by the CHREST (an acronym for “Chunk Hierarchy and REtrieval STtructures”) program (Gobet et al. 2001). Three groups of player (Class ‘A’ player, Expert and Master) were simulated and the results obtained correlated with human players, indicating that chunks are present within random data. In this experiment, the size of the chunk database varied with the expertise of the chess player, 500 chunks for a class ‘A’ player, 10,000 for the expert and 300,000 for the Master.

Chase and Simon (1973a) investigated chunking within a chess configuration with another experiment. The subject was asked to reconstruct a chessboard layout copying from one board to another. The head movements and pieces placed were recorded. It was noted that if a piece had links to other pieces (a piece being under attack from another piece would link the two pieces together for example) then the piece would be placed without delaying. If the piece being placed started a new group, then there would be a small latency before placing. A larger than normal latency was given as an indication of a chunk boundary. The suggestion from this experiment was that the subject was using his chunk knowledge to remember groups of pieces and therefore the subject possessed chunk knowledge.

In another paper Chase and Simon (1973b) conducted a number of experiments where the eye movements of the players were monitored. Similarly, Charness et al. (2001) compared the eye movements between a group of twelve intermediate skilled chess players with twelve experts, and twelve novices. When shown a board, the experts made half the number of fixations on pieces as the intermediate group. These findings were consistent with de Groot and Gobet (1996) however the experiment showed that the expert group focused on the *salient* pieces more quickly (compared with the intermediate group). The inference here is that the expert recognises chunks and quickly focuses on the salient moves. Recognition of

chunks therefore act as a trigger for an action, or as a focus to direct attention to a particular area of the board. The novice however may come up with the same move but only after examining *all* of the pieces, thereby taking a lot longer to work out his move.

Charness et al.(2001) explored the visual span of chess players measured on structured (not random) board configurations. The experts used a considerably larger visual span of the board. When testing for a check condition the experts made fewer fixations on the board compared to the novice, and reported a higher fixation *between* pieces. The suggestion with this experiment was that the expert player employs a complex perceptual encoding in his view of the board. This suggestion is consistent with theories about chunking, although this experiment implies chunking within the visual cognitive process.

Saariluoma (1991) conducted a number of experiments with blindfold chess playing. Blindfold chess is a game of chess normally conducted with the player with his back to the board and without any visual reference to the game, communicating only verbally. The player must keep track of and evaluate moves without any external reference whatsoever. Even harder than this is where the player conducts two or more games simultaneously. The record, which was set in 1985, for the number of simultaneous games (the 'blindfold player' must win 50% of the games to be considered to be playing the games) is over fifty. One experiment in particular, which used blindfolded chess players involved three groups of players, master, medium and novice. The experiment was to read five real but unknown chess games, and five randomised games, where the moves do not even follow the laws of chess. After reading all of the games the groups would recall them. The master group recalled the real games with high accuracy and better than the other groups but in the random games all groups performed the same with players unable to remember a single

position. Saariluoma argues this result is evidence for chunking mechanisms within memory as ordered games can be remembered easily by the experts (although not by the novice group), but unordered games could not be remembered by any group.

Experiments on recall of several boards by Grandmaster players show that, as the number of boards presented to the Grandmaster increases there is a percentage decrease in the number of pieces recalled (Gobet and Simon 1996). This is consistent with chunking theory as the effect of the limit of short-term memory. The result however does not exactly fit the chunking prediction but shows a slightly better recall than expected. The experimenter presented boards to the Grandmaster, gradually increasing the number in the experiment to nine boards and for as long as he can recall with 70% accuracy. Skilled players recalled more pieces than predicted by Chase and Simon's chunking theory. The improvement in recall is attributed to a phenomenon named 'templates'. Templates are attributed to a faster than expected ability to store information in *long-term* memory. Experiments with five-second presentation times, which are considered too brief for storage in long-term memory, provide evidence for a memory process in addition to chunking (Gobet and Simon 1996). The accepted time for transfer of a chunk to long-term memory is eight seconds and about a minute for seven chunks. Experts in a particular domain can store patterns related to their domain into long-term memory with a very short exposure time, whereas the novice requires more time (Gobet and Jackson 2002). Templates are described as having a 'core' pattern, which remains unchanged, with a set of 'slots', whose values can be rapidly altered. Chunks can evolve into templates through extensive experience (Gobet and Simon 1996). In the chess domain, the chessboard or a section of it can be the basis for a template. Pieces positioned on the board form patterns that are quickly encoded and stored.

The literature also reports research arguing against chunking theory. Holding (1985) argues that chess skill is not attributed to the chess player's knowledge of chunks but linked to the player's "evaluative judgments throughout the forward search". Holding conducted an experiment where sixteen chess players, consisting of eight strong and eight weak players, evaluated a board giving a score to the stronger side. Binary trees were constructed for six plies ahead from the current position and a piece was moved following each path on the search tree, evaluating each possible move. Not surprisingly the stronger players performed more accurate evaluations at the starting positions and were more consistent with the evaluations as the moves are made. However, the *reasons* for the skilled players performance in making good forward evaluations were not discussed by Holding.

The discussion so far in this chapter has considered chess skill from a psychological perspective. Chase and Simon's work is heavily cited in the literature on chunking. The reason for this is due to the fact that Chase and Simon's theory has greatly motivated research in the field. However, the dominance of their work can also "indicate a lack of originality in the field of chess research" (de Groot and Gobet 1996, pp 115). The advent of high power computers however has enabled new branches in chunking research, for example papers by Campitelli et al. (2005, 2007) report the use of fMRI brain scanning equipment to investigate brain activity, comparing Master players and the novice when viewing chess positions. Research into chunking has also included computer programs that simulate chunking. Feigenbaum and Simon (1984) developed a computer program for building chunks from random input data. EPAM (Elementary Perceiver And Memoriser) is a self-organising computer model that categorises and stores information. EPAM consists of a set of nodes (or chunks), connected by branches, forming a tree-like structure. The nodes contain tests which can check features of the input (or 'stimulus'), the

outcome of which determines which branch will be taken below the node, or if a new branch (to a new node) is to be created. If the stimulus contains *additional* information to what is held in the node then the additional information is *added* to the node. If the stimulus fails to exhibit aspects of the information within the node then a new node below the node in question is added. The discrimination net therefore grows dynamically in response to the input.

EPAM is not specialised to any one source of data but is a general-purpose tool for building a network of related information and is better known for work on associating words and spelling within words. CHREST is an enhancement of EPAM (Gobet 1998) The main difference between EPAM and CHREST is CHREST's ability to create lateral links between nodes. CHREST's enhancements make it perform better at self-organising and adaptation to complex data. CHREST was first applied to chunk analysis in chess programs, although is not limited to that domain. It has in addition an implementation of template processing. CHUMP (Gobet and Jansen 1994) is a chess-playing program, which plays only from pattern recognition. It is termed "a pattern learning move generator" based on the program CHREST with knowledge only of patterns of chess positions and no instruction on moves, goals or values of positions. CHUMP learns about moves and when to make them. The program builds two discrimination nets; one for the chunks found on the chessboard and another for the move that was made. A link between the two nets allows CHUMP to analyse a chessboard to look for chunks and then find an associated move in the 'moves' discrimination net. CHUMP's initial learning set consisted of three hundred games played by Mikhail Tal (a former world champion), however, this experience of games is small compared to the database a chess Master would acquire assuming the time needed to become expert in any domain, including chess, which is said to be "a decade of human practice for high skill in any non-trivial domain" (Simon 1981).

CHUMP is built from the following components: (1) *An eye movement simulator*, which is modelled on the eye movements of a human chess player. This gives focused attention to specific squares on the chessboard. It is intended that this filtering of pieces help generate chunks that are relevant to the configuration. (2) *A data structure based on the EPAM memory and perception model*. This structure is a 'net' rather than the 'tree' structure of EPAM. When learning, if the object is new (i.e. it does not match an existing chunk) then a node is added to the database. If an object is found then CHUMP extends the net with the new elements. (3) *A database of moves taken by the chess player in the training data*. Chunks in the net (or 'discrimination net') are linked to the database of moves so that chunks on a chessboard can be associated with moves.

The results produced by CHUMP are however rather unconvincing, presumably because of the small learning set of games. An estimate of the number of chunks memorised totalled 6710 achieving a 2.8% success rate in selecting the correct move, or 11.2% success in selecting the correct top four move suggestions. This success figure is actually not much better than choosing a move from a random selection of possible legal moves (cf. page 136), despite the fact that the training and testing data were taken from games by the same Grandmaster. This would have helped raise the success rate, as it is normal for players to stick to the same opening moves in most games. Further tests using the Bratko-Kopec positions³ showed CHUMP performing better at positional than tactical moves (Gobet and Jansen, 1994). This result is consistent with explanations of chunking because tactical moves are generally unique, whereas the same positional moves frequently occur when the

³ The Bratko-Kopec tests are described on page 99.

chessboard has a similar layout, and would therefore be captured in the chunk building process.

‘Tactical’ moves can be defined as “short term manoeuvres which have specific goals”, whereas ‘Positional’ moves are “more to do with moving pieces into advantageous positions than with direct attacks or winning of material”⁴.

Another program named ‘Chunker’ (Berliner and Campbell 1984) uses chunk knowledge to improve performance of the endgame. The program is limited to a subset of king and pawn endings. A library of chunks of all possible pawn configurations is used, each chunk having a property list, including the number of moves required to queen a pawn and the theoretical game value of the chunk. By using chunking it is claimed that the program plays with an equivalent power of a 45-ply search.

Another notable chess program that can play a complete game of chess is PARADISE (Wilkins 1980). PARADISE uses a set of about two hundred production rules to eliminate pieces from the configuration. The program narrows the search to just a few pieces and is then able to perform a deep search. PARADISE does not have knowledge of chunks but is an example of a program that uses knowledge to narrow the search tree. Another example of the use of production rules is the Capablanca program, (Linhares 2008). Capablanca is claimed to be a model of cognitive function and uses production rules to narrow the attention to an area of the chessboard when selecting a move to make. Despite the fact that PARADISE or Capablanca do not use chunking the programs are relevant to this thesis because they use a combination of techniques to make a move. PARADISE uses production rules to narrow a search (a conventional minimax type) to select the move to play.

⁴ The definition for tactical and positional moves was taken from the web page: <http://www.research.ibm.com/deepblue/reference/html/i.2.html>

The same approach is taken in the later part of this thesis whereby the research application orders an alpha-beta search based on the likelihood of a move, given by an analysis of chunks on the chessboard (cf. page 165).

3.2. Chapter conclusion

The case supporting the existence of chunks within humans and other animals based on psychological studies is reported in numerous papers. Similarly, the application of chunk knowledge in relation to learning and skill is widely reported especially with respect to skill in chess players, with quantifiable results. A considerable amount of work is reported in the literature with collection of empirical data and computational modelling of chunking systems. This thesis seeks to complement existing research by looking at chunking within a computational environment, with varying chunk parameters, such as the number of pieces that constitute a chunk and the relationship of pieces within a chunk.

4. DEFINING A CHUNK

Psychologists seem to know a chunk when they see one. A definition, however, is hard to come by. Neither the large literature on chunking by humans nor the more modest literature on chunking by animals provides an operational definition of this term. (Terrace 2001).

Although the existence of chunks within chess play has been discussed in various research papers many of the papers describe psychological experiments and these largely focus on measuring the effects of chunking on human behaviour rather than measuring the chunk properties directly (Chase and Simon 1973a, 1973b, de Groot 1965, Simon and Chase 1973). The properties of chunks are therefore deduced by inference rather than direct measurement. The remainder of this chapter looks at some of the definitions and properties of chunks given in the literature. The definitions of a chunk given in this chapter will be used later in the thesis when chunks are extracted from actual chess games.

4.1. Chunks are learnt constellations

The chess player is said to acquire his knowledge of chunks by studying previous games. Constellations of pieces become associated with moves or strategies and are stored in long-term memory. Chess masters typically study games, or more precisely, parts of a game, for several hours each day over many years (Chase and Simon (1973) estimate that it takes to “the order of ten years to achieve expertise in any domain”). Chunking theory dictates that during the study of other players’ games chunks are learnt and memorised by the subject.

4.2. Chunks are frequently occurring configurations

Programs such as IAM (Walczak 1992) EPAM (Feigenbaum and Simon 1984) and CHREST (Gobet 1998) extract chunks from games by accumulating frequently occurring patterns. Capturing frequently occurring patterns is useful as this reduces the impact of random or insignificant moves, and the repetition of a chunk that is linked to a move eliminates false associations. CHREST uses frequency of appearance as a criterion for finding chunks: “objects have to be presented several times in order to be learned” (Groot and Gobet 1996). Similarly IAM needs to detect a pattern in “at least two games” for it to be considered significant. “Patterns which have tactical significance and are known by an adversary are typically repeated in multiple games” (Walczak 1992). Frequently used chunks are often present in positional moves as opposed to novel tactical moves and for this reason systems based on CHREST favour positional as opposed to tactical moves (Gobet and Jansen 1994). From a technical perspective the selection of chunks by frequency of occurrence is an easily programmable process but selection of rare tactical moves would be more complex as this would require an understanding of the game by the program.

4.3. Recall of chunks are separated by a two second boundary

Chase and Simon (1973a) offer a definition of a chunk based on the time taken to place the chess pieces when reconstructing a chessboard. The player appeared to place the chess pieces in bursts of activity with a short latency between groups. A latency (or boundary) of two seconds or longer was considered to be the break between ending the placement of one chunk and the starting of another. The above definition of a chunk allows an estimate of the size of a chunk or the number of

pieces that constitute a chunk, although the measurement may be limited by the number of pieces that can be held in the hand (Gobet and Simon 1998).

4.4. Chunks can be a tool to extend short-term memory

According to Gobet, “a chunk is defined as long term memory (LTM) information that has been grouped in some meaningful way, such that it is remembered as a single unit. Each chunk will only take up one slot in STM⁵, in the form of a ‘label’ pointing to the chunk in LTM⁶” (Gobet and Jackson, 2002). Miller’s 1956 paper has been very influential on research that features short-term memory (Chase 1983, Chase and Simon 1973, Fenk-Oczlon and Fenk 2000, Gobet et al. 2001). Miller (op. cit.) used the term ‘chunking’ to describe the mental grouping of information from low information content items into a smaller number of high information content items. Chunking has been described as the cognitive ‘work-around’ for the short-term memory limitation by storing chunks of items in long-term memory and, in computational terms, indexing the items from the short-term memory.

The analogy works well when considering the memory task of recalling a sentence from this page. The reader has acquired a large database of words in long-term memory so that if required to write the sentence it is not necessary to recall the individual characters to spell out the words. The spelling of the words can be recalled from long-term memory. Rather than remembering the fifty-two characters in the previous sentence the reader can recall just twelve words or less (as the phrase ‘long-term memory’ is arguably remembered as a single item, or a ‘chunk’, in itself). The same reasoning for chunking is applied to chess configurations by grouping frequently used constellations of pieces and remembering these as one item. The

⁵ STM is an acronym for ‘Short Term Memory’

⁶ LTM is an acronym for ‘Long Term Memory’

number of *items* remembered in short term memory may be limited to a relatively small number, but the chunks themselves are not subject to the short-term memory limitations as they are stored in long-term memory. The number of pieces that make a chunk within the context of the game of chess is investigated and results reported in chapter 7 of this thesis.

The above view of the significance of chunking is echoed by Terrance (2001): “The basic function of a chunk is to enhance STM (short term memory)”. The research reported in this thesis shows that the meaning associated with a chunk is key to understanding the usefulness of the chunk. Chunks must therefore be organised in meaningful ways, thereby offering the potential to extend the basic function (to enhance short term memory). De Groot and Gobet (1996 pp14) describes increasing knowledge in the terms: “The more domain knowledge and experience a person has, the greater and more comprehensive are the units (clusters, patterns, configurations) in terms of which he can encode the stimulus” (de Groot and Gobet op. cit.). The organisation of chunks by incrementally adding knowledge, or building on previously learnt experience is not included in this thesis but is an area for further research.

4.5. Chunks contain elements that are related to each other

A chunk is defined as “a collection of elements having strong associations with one another, but weak associations with elements within other chunks” (Gobet and Jackson 2002), and similarly “a collection of concepts that have strong associations to one another and much weaker associations to other chunks concurrently in use” (Cowan 2001). These definitions suggest that chunks are groups of pieces, and indeed the notion that chunks are groups of pieces is implicit in papers that describe the effects of chunks. Gobet and Simon (1998) describe a chunk as a “long-term

memory symbol, having arbitrary sub parts and properties that can be used as a processing unit. Each chunk can be retrieved by a single act of recognition” (Gobet and Simon op. cit.). Some programs that extract chunks use specific relationships between pieces and these are discussed in the following sections.

4.6. Pieces are related by proximity

Papers by Gobet and Jansen⁷ (1994) and also Walczak (1992) report work done on analysis of boards with chess pieces that are in close proximity to each other. The rationale for this limitation is based on the model of human performance suggested by the ‘Principles of perceptual organization’ proposed by the early 20th-century German psychologists of the Gestalt school (Walczak 1992). One aspect of the Gestalt principle is that local objects tend to be visually grouped together. When considering a board in chess play the expert may limit their attention to small areas of the board as opposed to all sixty-four squares. An area of say four by four squares limits a chunk to a maximum size of sixteen pieces, with the pieces being in close proximity to each other.

4.7. Pieces are related by attacking/defending relationships

Wilkins (1980) writes: “Human masters, whose play is still much better than the best programs, appear to use a knowledge intensive approach to chess. They seem to have a huge number of stored ‘patterns’ and analysing a position involves matching those patterns to suggested plans for attack or defence”. Experiments with chess players by McGregor and Howes (2002) suggest that the attack/defence relationship of pieces is more significant for skilled chess players than the size of the chunk on the board. They argued against the notion that proximity of pieces is a factor in chunk

⁷ Whilst most chunks acquired by CHREST are related by proximity, CHREST also learns chunks based on attack/defence relationships, following eye movement heuristics cf. page 150

structure and suggested that experiments on chess player's ability to remember configurations provide evidence in support of the idea that the relationship of pieces is the main factor in chunk selection. That is, McGregor and Howes (op. cit.) proposed that the pieces within a chunk are related in an attacking or defending fashion. This viewpoint was earlier expressed by Simon and Barenfield (1969) who noted that attacking and defending relationships are significant factors in the players' analysis of a board: "By analysing an expert player's eye movements, it has been shown that, among other things, he is looking at how pieces attack and defend each other".

Russian psychologists Tichomirov and Poznyanskaya (1966) studied the eye movements of expert chess players and showed that the eyes did not move in a way that would be consistent with searching through a tree of moves and with the corresponding replies. The eye movements did however fixate on about twenty points on the board, abruptly moving in 'saccadic' movements between pieces that are in attacking or defending positions. The interesting point is that the eye movements are not random and the player must have known where to look next after fixating on one piece. The player would know the target square before the saccade began. The observation implies the player has knowledge of expected positions on the board based on awareness of part of the configuration. The pieces on the board are presumably positioned in recognisable groups, or frequently occurring configurations so that the chess player can expect certain pieces, or chunks, to be associated with some configurations.

4.8. Chunks are absolutely positioned

From a chess player's perspective the properties of a piece are dependent on the position the piece occupies on the board. The board is just eight by eight squares so

proximity to the edge of the board is an important factor. Generally, the centre squares are more highly prized as they command a greater influence on the board as a whole whereas the side positions provide some protection as attacks can only come from one direction. Attempts to generalise a chunk by making it independent of its position on the board (relative positioning) are not sensible (Lane and Gobet 2010). Indeed, Linhares and Freitas (2010) cite examples of chessboard configurations where there is a win for White, but white cannot win in an alternate shifted position, and another example where shifting positions changes the board from a win for White to a win for Black - assuming white plays first, see figure 4.1 below

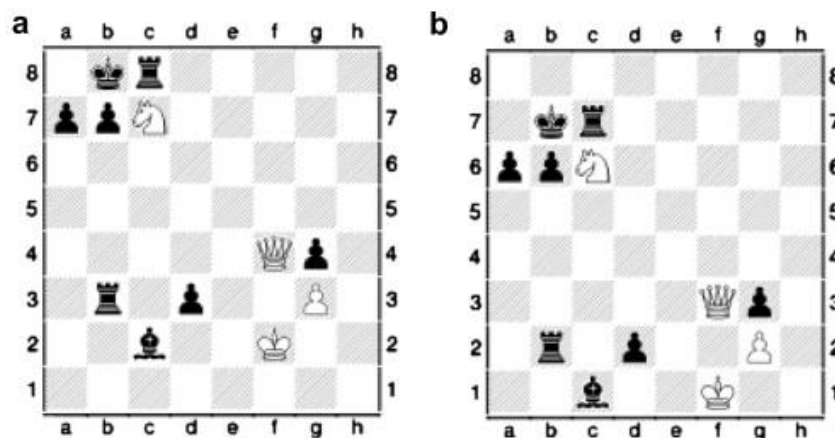


Figure 4.1: Shifting pieces from 'a' to 'b' changes winning position from White to Black (from Lane and Gobet, 2010)

Domains such as 'Go' however have a much larger playable area and in that case, the relative positioning of chunks may be more meaningful. However, Holding (1985) argued that chess chunks might be remembered in relative positions, with the relationship between the pieces of the chunk being fixed, but the position of the configuration on the chessboard being relative. If chunks were stored without fixed (absolute) positioning the number of chunks (based on Simon and Gilmartins's 1973 simulations) reduce by a factor of about twenty. Holding argues that Chase and Simon's (1973) estimate of chunks held in long-term memory is too numerous. Gobet

and Simon (1998) argued against Holding, stating “Holding’s criticisms either are not empirically founded or are based on a misunderstanding of the chunking theory and its role in a comprehensive theory of skill”. Chunks whose pieces are moved (maintaining the same relative position of the pieces with respect to each other but moving the group to a different area on the chessboard) or translated through a mirror image results in subjects having a greater difficulty recalling the configurations. Pieces with absolute positioning on the board were easier to encode and recall than pieces transposed on the chessboard (Saariluoma 1991, Gobet and Simon 1996). This result is consistent with Binet (1896/1996) in that “chess masters could not remember games unless they understood them” and changing the location of a chunk relatively on the chessboard would change the *meaning* of the chunk. In this thesis chunks are always processed with absolute positioning and the properties of chunks include meaning (the effect the chunk has within chess play). If a chunk is found with the same relationship between the pieces that make up the chunk, but in a different location on the chessboard then this constellation will be considered to be a different chunk and will be stored appropriately.

4.9. Experts have a larger chunk knowledge than the novice

Regarding the number of chunks of which an expert has knowledge, Chase and Simon (1973) stated that in order to reach expertise in any domain, including the chess domain, the training period might be long, corresponding to “a decade of human practice for high skill in any non-trivial domain”, with an estimate of the order of 50,000 chunks learned for expertise in the chess domain (Simon 1981). Simon’s estimate is based on the similarities between vocabulary of written words for highly literate individuals and on experimental analysis of chessboards by using the EPAM program (the EPAM program is described earlier in the literature review). Chase and

Simon (1973) admit that their estimate is a 'best guess' bearing in mind the data that they had available at the time of writing. Simon and Gilmartin (1973) put the number of chunks memorised by an expert player as between 10,000 and 100,000 based on results from a program 'MAPP'. MAPP (an acronym for Memory Aided Pattern Perceiver) acquired a relatively small number of chunks and from these replicated the memory recall performance of a 'Class A Player' (a 'Class A player' is a good chess player, but below expert level). The proposed number of chunks required for a chess master was extrapolated by Simon and Gilmartin (op. cit.) from the number used in the experiment at 'Class A' level to give the estimate between 10,000 and 100,000 chunks, with later estimates pointing to 300,000 chunks, even with the presence of templates (Gobet and Simon 1996).

4.10. The relationship between chunk definitions

Within the definitions of chunks described in this chapter there are several areas of overlap, for example, the definition 'Recall of chunks are separated by a two second boundary' is a consequence of the definition that 'Chunks contain elements that are related to each other', and are remembered by the subject as an item. The definition 'Chunks are frequently occurring configurations' is related to the definition 'Chunks are learnt constellations' as chunks are learnt by repeatedly seeing the chunks on chessboards, and as learnt constellations chunks would be stored in long term memory. The definition 'Chunks can be a tool to extend short-term memory' is a consequence of the properties that 'chunks are learnt constellations' and that chunks are stored in long term memory, and indeed, the property that 'Experts have a larger chunk knowledge than the novice' is a consequence that chunks have to be learned.

The property of chunks 'Pieces are related by proximity' is linked to the property: 'Pieces are related by attacking/defending relationships' as may attacking

and defending configurations exist when pieces are in close proximity to each other, and the accessibility of an attacking piece is not blocked by another piece. Many attacking and defending pieces may also depend on the chunks being 'absolutely positioned', which also is another property described in this chapter.

The properties of chunks are therefore not considered as isolated entities, but rather as interrelating and complementary definitions.

4.11. Chapter conclusion

This chapter has defined the properties of chunks that have meaning with respect to chess and the research undertaken in this thesis. Chunks are simply combinations of chess pieces. The properties described in this chapter define chunks that are significant in chess play in that the chunks may be learnt by an expert player. The properties of significant chunks are taken from the literature on chunking in chess, describing the differing and overlapping properties of meaningful chunks.

5. INVESTIGATING THE PROPERTIES OF CHUNKS

“I will suggest that chunks have at least three important dimensions, which should be systematically taken into account in the planning of training for adversary problem-solving situations. These aspects are the number of chunks, their size, and finally the relevance of their contents. If one of these dimensions is neglected, the outcome of the training will not be satisfactory” Saariluoma (1998).

The results reported in this thesis take a detailed look at the nature of chunks that have been extracted from Grandmaster games. The chunks extracted are applied to chess play in various ways explained later in this thesis. The research reported adds detail to one form of chunking, that is, one that works in a computational environment. The detailed workings of the human brain are outside the scope of this thesis but whether chunks exist, and what form they take within the domain of chess, are analysed in detail.

If chunking is significant to chess play then finding meaning within the chunk patterns is a major task. Chunks are simply constellations of chess pieces on the chessboard. Chunks that are important to this study are *frequently occurring configurations*, of which only a small subset may be significant.

Chunks may or may not have any significance in terms of the board score. Computer chess programs work by evaluating the overall board score according to the value of pieces on the board and their relative positions. It is suggested that the experienced player with an awareness of chunking can outperform a chess program (Wilkins 1980). For this reason an analysis of the value of the pieces within the chunk was not considered to be worthwhile.

This thesis reports research on the existence and properties of chunks within chess play. It should be noted that the same configuration of chess pieces positioned

on different squares on the chessboard is assumed to be a different chunk, therefore chunks will have absolute positions on the chessboard. The absolute positioning of chunks is discussed in more detail on page 59 of this thesis.

The properties of chunks will be investigated by looking at the following:

- **Chunk size.** The number of chess pieces that make up a chunk will be investigated. A human chess player would be expected to have the capacity to remember up to about nine chess pieces in a chunk (Miller 1956, Cowan 2001, Gobet and Clarkson 2004). Chunks however may exist as a property of the chessboard and the rules of the game, and their size may not be limited by cognitive function. The pieces in the chunk can include all of the pieces, including both colours, on the chessboard giving a maximum of thirty-two pieces. However, as a game advances beyond the opening moves the likelihood of finding very large chunks occurring on more than one chess game diminishes because the board layout becomes increasingly more unpredictable (cf. page 61).

Note that a chunk may consist of a single piece, however for this research chunk sizes of greater than one piece will be considered. It should be noted that a chunk in human memory *may* contain more information than simply the 'pieces on positions' on the chessboard, and consequently may require more short term memory capacity than the number of pieces constituting the chunk, however, this thesis will look at the variation in the effectiveness of chunks of different sizes in order to investigate the plausibility that knowledge of chunks consisting of a small number of pieces can be effective indicators in directing

chess play. The thesis does not attempt to define the limits of human short term memory capacity.

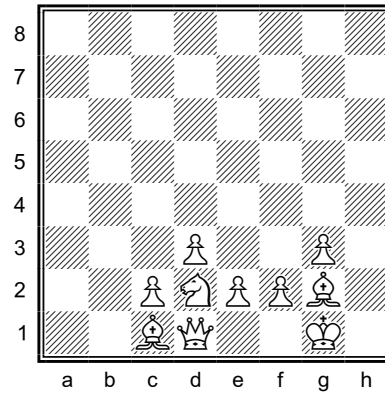
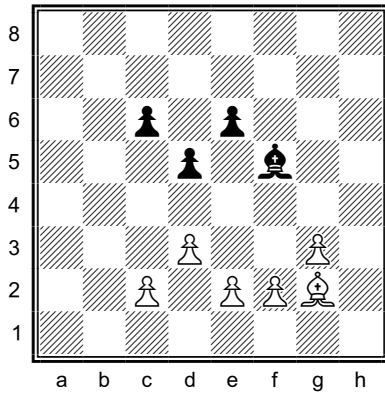
- Piece relationships. The relationships of the pieces within chunks will be investigated and reported in this thesis. In particular the properties of chunks with pieces in close proximity and pieces in defending relationships will be reported.
- Chunks and meaningfulness. Results from an investigation to what gives a chunk meaningfulness and usefulness for the chess player will be reported.
- Chunk familiarity. This research assumes an arbitrary threshold such that a chunk must appear in at least one per cent of boards from a sample of games in order to qualify as a frequently occurring chunk. If a pattern found is present on less than one per cent of the boards then the pattern is considered too infrequent and is not considered to be a chunk.
- Chunks and player's skill. The existence of chunks on the chessboard is in itself not a measure of the player's skill, as chunks may be a property of the chess pieces and the rules of the game, however, chunk knowledge and the application of this knowledge are investigated and reported in this thesis.

5.1. What a chunk looks like

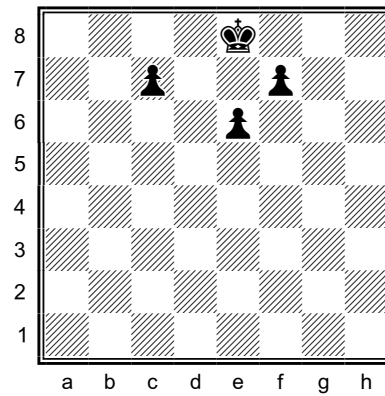
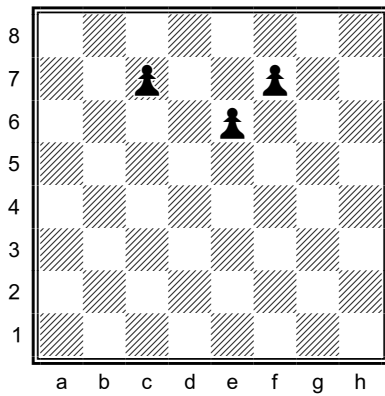
The following chessboards show a few examples of chunks. The chunks shown were extracted from Grandmaster games by using the CHREST program⁸.

Chunks may be composed of either or both colour pieces:

⁸ The chunk data were provided by Gobet.

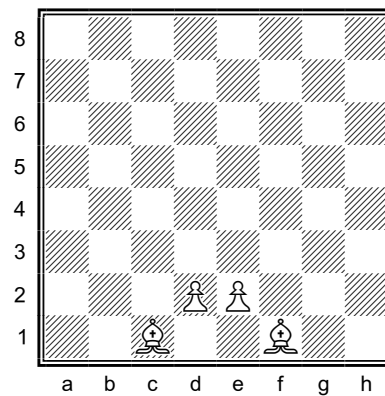
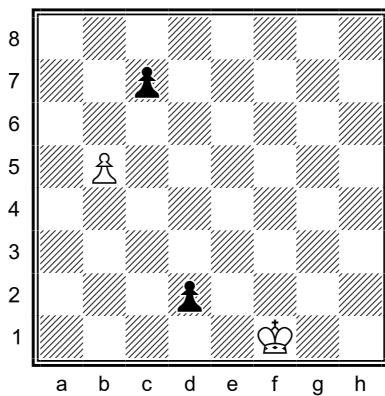


A chunk may be built from smaller chunks:



Pieces in the chunk may be unrelated (below left).

A chunk may be part of the initial board layout (below right).



5.2. Chunk statistics

From an analysis of chessboards it is clear that there are many patterns or constellations of pieces that occur frequently. The repeated constellations or chunks exist due to the properties of the chess pieces and the rules of the game. It is easy to extract chunks from chess games; the difficulty is finding meaning associated with the chunks. The existence of chunks in itself is not a measure of the player's skill as chunks are found across the whole range of skill sets. Therefore attempts to correlate the player's skill with chunks used are futile, but rather, it is the player's skill that recognises chunks to assist his chess play.

In this chapter we worked with a large number of chunks compiled from Grandmaster games using CHREST. The data were provided by Gobet and comprised of 251,735 unique chunks made up from four or more chess pieces from games starting at twenty ply from the beginning of the game.

Positions earlier than twenty ply were not considered because the beginning of a game is normally dominated by conventional opening moves. The existence of chunks within chessboards of tournament games was investigated to measure their frequency and distribution using the chunks provided by Gobet.

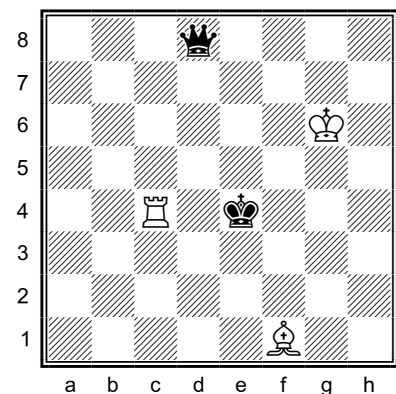


Figure 5.2: A chessboard with five pieces

5.2.1 Why are there so many chunks on a chessboard?

Before continuing with the reporting of results a few paragraphs follow to explain why so many chunks are expected to be found on chessboards. The reason why high numbers of chunks are counted is due to the way chunks are constructed. Each

chess piece can be a member of *many different chunks*. To illustrate this, consider the chessboard figure 5.2 (above), with just five pieces. The chessboard shows five pieces on squares as follows:

qd8, ke4, Kg6, Rc4, Bf1

The pieces combine to produce chunks as follows:

<qd8>	<Bf1, qd8>
<ke4>	<Bf1, ke4>
<ke4, qd8>	<Bf1, ke4, qd8>
<Kg6>	<Bf1, Kg6>
<Kg6, qd8>	<Bf1, Kg6, qd8>
<Kg6, ke4>	<Bf1, Kg6, ke4>
<Kg6, ke4, qd8>	<Bf1, Kg6, ke4, qd8>
<Rc4>	<Bf1, Rc4>
<Rc4, qd8>	<Bf1, Rc4, qd8>
<Rc4, ke4>	<Bf1, Rc4, ke4>
<Rc4, ke4, qd8>	<Bf1, Rc4, ke4, qd8>
<Rc4, Kg6>	<Bf1, Rc4, Kg6>
<Rc4, Kg6, qd8>	<Bf1, Rc4, Kg6, qd8>
<Rc4, Kg6, ke4>	<Bf1, Rc4, Kg6, ke4>
<Rc4, Kg6, ke4, qd8>	<Bf1, Rc4, Kg6, ke4, qd8>
<Bf1>	

Figure 5.3 Combining five chess pieces

A chunk is shown within chevrons and pieces separated by commas. This notation is used de Groot and Gobet (1996). The piece is denoted by the piece name (R=Rook, B=Bishop, K=Knight, Q=Queen, K=King, P=Pawn), followed by the square location on the board. If the piece name is lowercase then the piece colour is black, otherwise it is white.

The pieces combine giving a number of chunks increasing as a piece is added. With each piece added, the number of resulting chunks follow the series,

1,3,7,15, 31...

This series can be expressed as a formula:

$$\text{Combinations} = (2^n) - 1$$

Where 'n' is the number of pieces on the chessboard.

For example, a typical chessboard may have twenty-five or more pieces in play, yielding $(2^{25} - 1)$ combinations, where 2^{25} equates to 33,554,432. A large number of chunks are therefore present on the chessboard by virtue of the fact that a chunk is simply a combination of pieces. The number of chunks counted and shown on figures 5.4 and 5.5 below are the chunks generated (by combining the pieces on the chessboard) and which also appear in the list of chunks compiled by CHREST. The list compiled by CHREST, in this instance, consists of 251,735 frequently occurring⁹ chunks. A typical chessboard configuration taken from the mid-game could therefore contain a very large number of chunks with a proportion of them being found within the CHREST chunk list.

5.2.2 The frequency of chunks in relation to player skill

An analysis of games played by players of varying skill was made to see if there was any correlation between the number of chunks used within the game and player skill (the skill of the player being given as an Elo rating¹⁰). The analysis searched for the occurrence of any of the chunking patterns in the chunk database, for each board played in the game. As the length of games varied the results were normalised by taking from the game just the fifty ply prior to the conclusion of the game, therefore all game data in this analysis included at least fifty moves, with the fiftieth move being the conclusion of the game.

The results were obtained for each skill group. The number of games examined for each skill group is shown in table 5.1.

⁹ 'Frequently occurring' chunks are chunks that occur more than once within import data.

¹⁰ The letters Elo being the family name of the system creator 'Árpád Élő', a Hungarian born American physics professor.

Elo Range	Number of Games Examined
1400-1599	356
1600-1799	748
1800-1999	1021
2000-2199	2006
2200-2399	2003
2400-2599	2006
2600-2799	502

Table 5.1: The number of games used in each skill group.

The pieces on each chessboard for each game were compared with the list of chunks (from the chunk list supplied by Gobet) so that the number of chunks on each board could be counted. An average number of chunks for *each position* within the game were calculated, for all games in each skill group. The results of the analysis, which are displayed in figure 5.4, show an increase in the average number of chunks found on the board as the game progresses towards checkmate.

The most notable observation from this analysis, which is illustrated in the graph shown in figure 5.4, is the fact that *the number of chunks found is more or less the same, irrespective of the player skill*. At first sight this implies that chunking is unrelated to the skill of the player as the number of chunks on the chessboard does not change with increasing player skill. This observation does not however conflict with chunking chess skill theory, as it is the *knowledge* of the chunks that is attributed to skill and not the mere presence of chunks. The expert player may recognise patterns and so pursue certain moves as a result. The novice on the other hand would not recognise the chunks and may look ahead through many unfruitful paths.

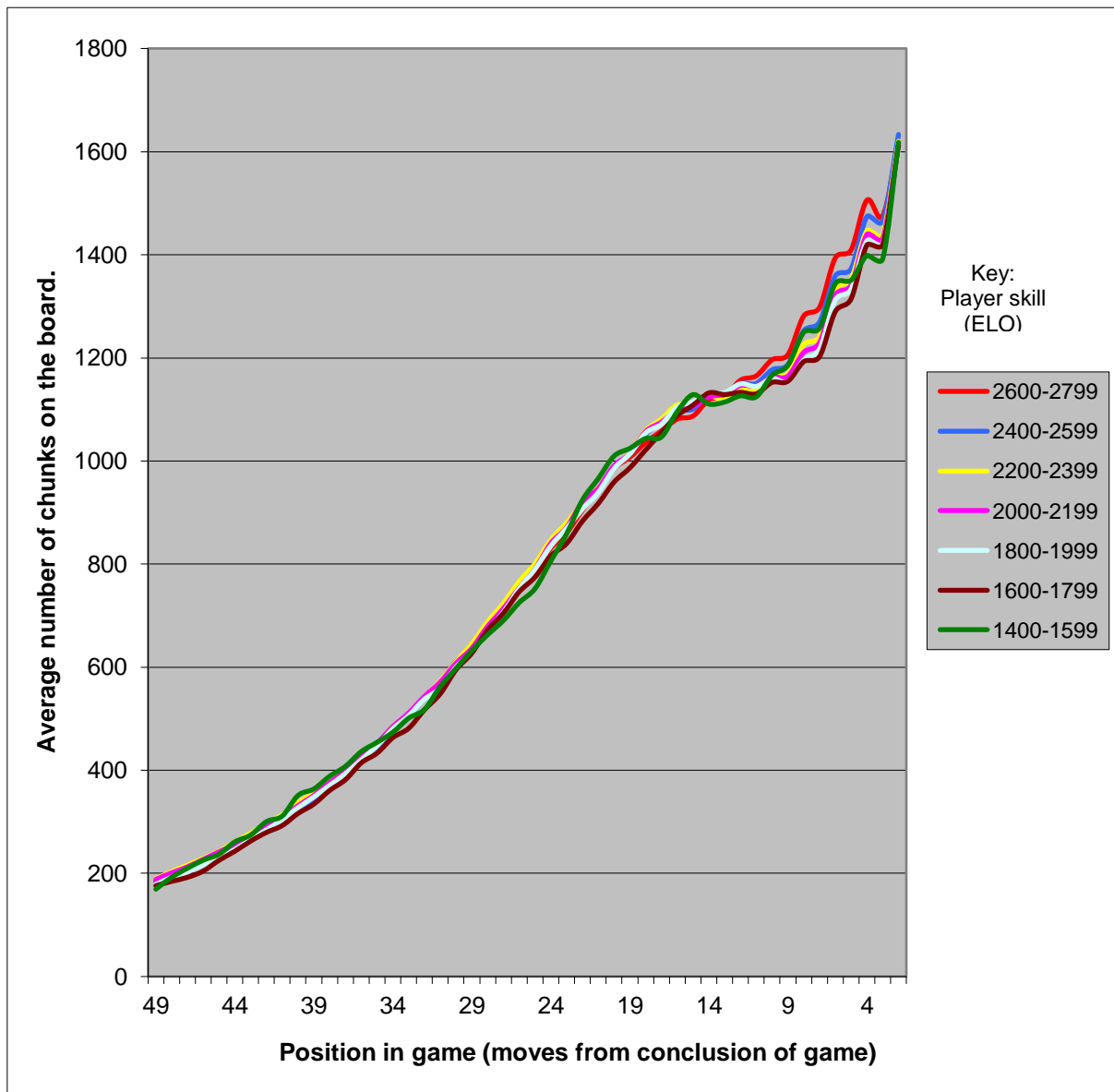


Figure 5.4 The average number of chunks found on the chessboard

The results shown in figure 5.4 show a steady increase in the number of chunks on the board as the game progresses until about fifteen moves before checkmate where the number of chunks increase at a slower rate until about ten moves before checkmate, after which the rate of chunk formation increases again. This trend is common to all skill sets. The change in the rate of formation of chunks indicates that chunk formation is significant within the game of chess and chunk formation can be an indicator of the stage of the game.

It is curious that the number of chunks increase as the game proceeds, as shown in figure 5.4 above. The reason for this increase is thought to be due to the pieces being arranged into defensive groups, and the number of such configurations increasing as the game proceeds. Indeed, as the game proceeds, pieces within small groups can become related to other pieces within other groups, in defending and attacking relationships, resulting in a network of inter-related pieces over the whole board. As the game proceeds, the network of inter-related pieces increase, and as many of these relationships are frequently occurring between different games, the number of recognised chunks increases as the game advances.

The results shown in figure 5.4 show the mean number of chunks as the game proceeds. The standard deviation for the data is approximately 180 at the point nineteen-ply before the conclusion of the game. This standard deviation is approximately the same for all skill groups, although the mean values show differences between skill groups, the differences are small compared to the 95% confidence interval of two standard deviations (360 chunks). The analysis that explores the separation of the mean values with chess skill, that follows in this chapter, is included for interest; however, it is acknowledged that the separation is not reliable.

5.2.3 Removing the most common chunks

The chunks found on the chessboard were examined in further detail to determine which chunks were used by each group of player. It was found that 186,297 out of the 251,735 chunks (74%) occur in games played by the *lowest* skill set (the lowest skill set has an Elo rating of 1200-1399). The point here is that the lowest skill-set are novices, who have supposedly not learnt a library of chunk patterns. The chunk patterns used by this group must therefore be common configurations that are

properties of the chess pieces and the rules of the game (such as advancing pawns). Many of the chunks are also insignificant, except for the property that they occur frequently within board layouts.

There is therefore a *base set* of chunk patterns present with all skill sets. These chunks may include pieces in their original 'start of game' positions for example. If we remove all of the chunk patterns used by the lowest skill set (Elo 1200-1399) the difference between the groups becomes more noticeable. The suggestion is that there may be a subset of significant chunks that are related to player skill amongst the large dataset.

By removing the base set, the number of active chunk patterns drops from 251,735 to 65,438. Analysing the game data again with the base set of chunks removed shows an increase of the 'non base chunks' with increasing player skill.

The results are plotted on figure 5.5. The graph shows the variation between the lines which were plotted in figure 5.4 and as a result the vertical scales are very different between the two graphs.

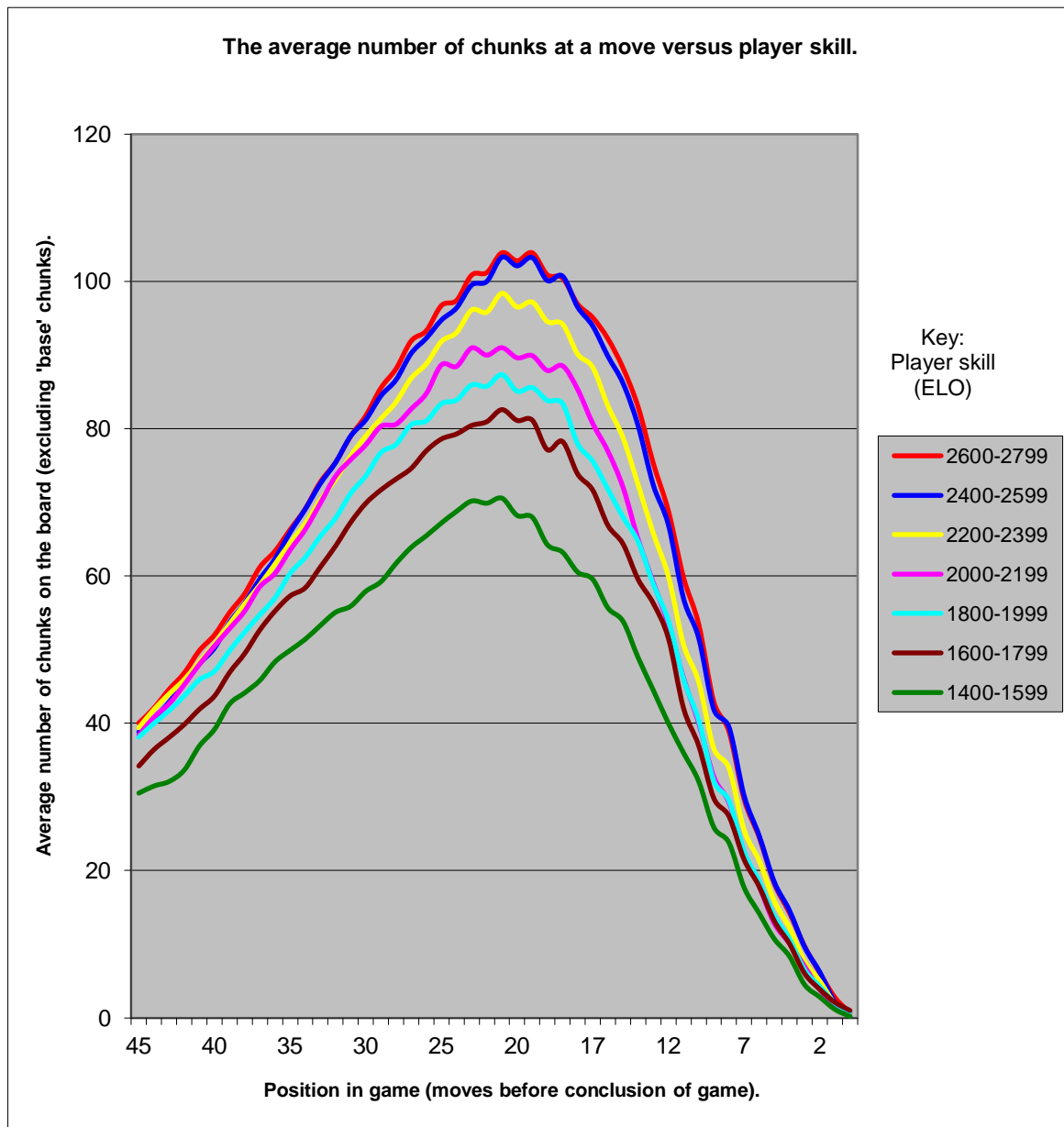


Figure 5.5: The average number of chunks found on the chessboard excluding 'base chunks'

The results displayed in figure 5.5 show an increase in the chunks used with the skill level at each point in the game. The graph shows a decline towards the game end. This decline is not a decline in the *number* of chunks present but rather a reduction in the *variety* of chunks. As the game ends the chunk patterns standardise across all skill groups and therefore become chunks used in the 'base set'. There are fewer variations in the end game and therefore by removing the chunks used by the less skilled groups the number of *additional* chunks reduces.

The results shown in figure 5.5 suggest that *chunks within chess can be a differentiator with regard to player skill* implying that at each stage in the game, players with higher ELO ratings construct more chunks on the chessboard than players with lower ELO ratings, when using a list of chunks generated by CHREST but excluding chunks used by the lowest skill group, however, this hypothesis will be investigated in more detail in the next paragraph.

5.3. Testing the skill/chunk relationship using a Pearson Correlation

The analysis in the previous sections consisted of a count of chunks as the game progressed. By removing the most common chunks (cf. page 41) there appears to be a differentiation of the number of chunks used with player skill. The differentiation is most noticeable at a position in the game that is twenty ply before the conclusion of the game, however, difference in the number of chunks between skill groups is small (only a few chunks), and the total number of chunks on the board numbers over one thousand.

To investigate the variation in number of chunks in greater detail, excluding the most common chunks, a Pearson Correlation was performed on the data. To obtain a more exact analysis and to evaluate the significance of the correlation, a number of chessboards were collated, all at a position of twenty-ply before the conclusion of the game. Each chessboard was examined and the number of chunks counted by searching the chessboards for the presence of chunks (chunks provided by Gobet). The data were correlated with the Elo skill rating for the players. The player with the lowest Elo rating in each game was recorded, together with the number of chunks, for each chessboard. A total of about two thousand chessboards from a Elo skill range between 1000 and 3000 were examined.

The results are displayed on the scatter plot below:

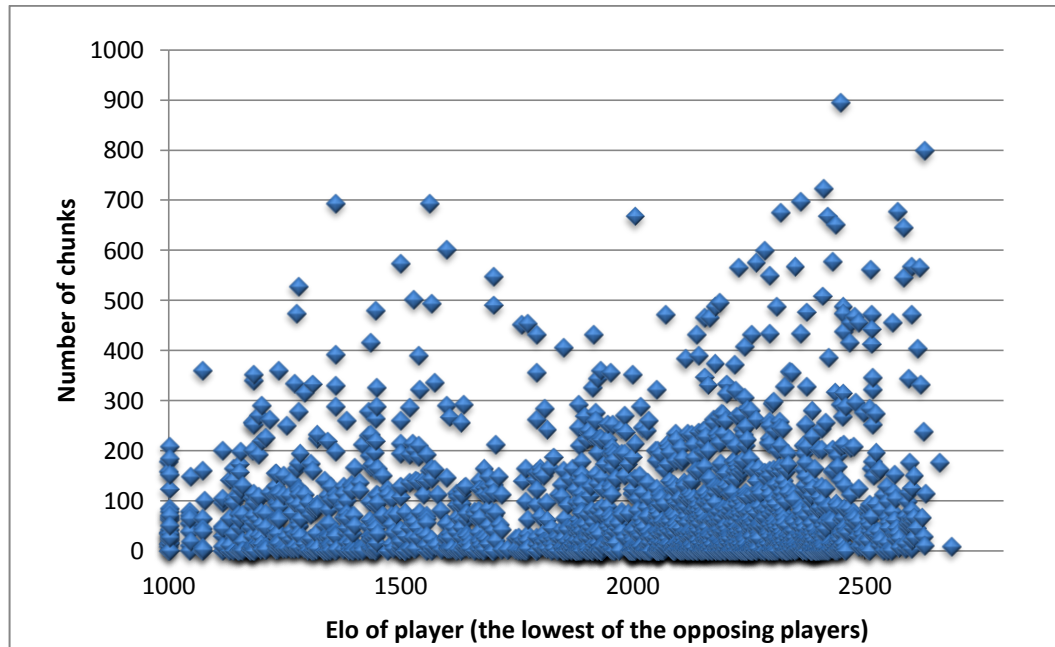


Figure 5.6: Scatter plot showing player skill and the number of chunks

A Pearson Correlation analysis was performed on the data. The result gave a correlation coefficient of 0.047, indicating that *there is little or no correlation between chess skill and the number of chunks.*

5.4. ‘Defensive’ chunks

This chapter introduces defensive configurations, as they are one of the types used in further analysis in chapter 7. As previously stated, many of the chunking patterns appear to have no meaning other than that they are frequently occurring configurations. With reference the literature on eye movements the chunk database was analysed to explore the relationship of pieces within the chunk. “By analysing an expert player’s eye movements, it has been shown that, among other things, he is looking at how pieces attack and defend each other” (Simon and Barenfield, 1969).

Using the chunk list generated by CHREST and removing all of the pieces, except those pieces that defend each other within the chunk, the number of chunks

reduces to 2,504. These chunks (*'defensive chunks'*) have each piece protecting another piece within the same chunk and in this way the group of pieces making up the chunk have an intrinsic value. Chunks in this case are therefore only composed of pieces of the same colour, yielding 972 white and 1,532 black chunks. These patterns are only chunks where the pieces *defend* each other.

It is noted that there is a large difference in the number of defensive chunks composed of black pieces and chunks composed of white pieces. The reason for this is unknown and is a topic for further research.

Examples of defensive chunks are shown below:

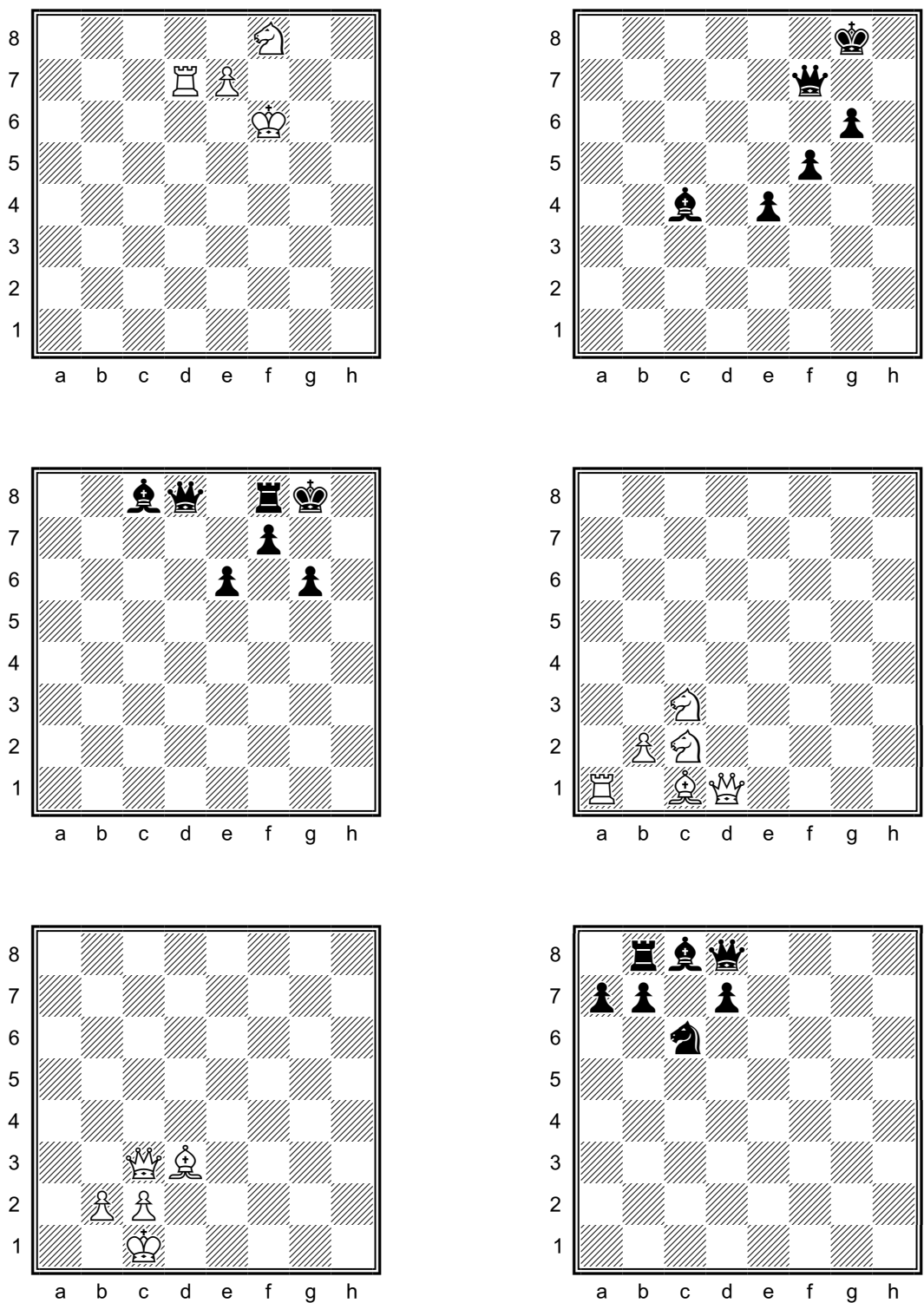


Figure 5.7: Examples of 'defensive' chunks

5.4.1 The occurrence of defensive chunks throughout a game

The graph below shows an analysis of the use of defensive chunks throughout the game. Data have been normalised so that the conclusion of the game occurs at position sixty-four. The games used for the analysis had a span sixty-four or more moves. Positions before the conclusion of the game are numbered *counting back* from sixty-four, where 'ply sixty-four' is the conclusion of the game. Where a game had more than sixty-four moves the moves before 'ply one' are ignored. The average number of chunks used at each ply from a dataset of two thousand Master players chess games are plotted on the graph below.

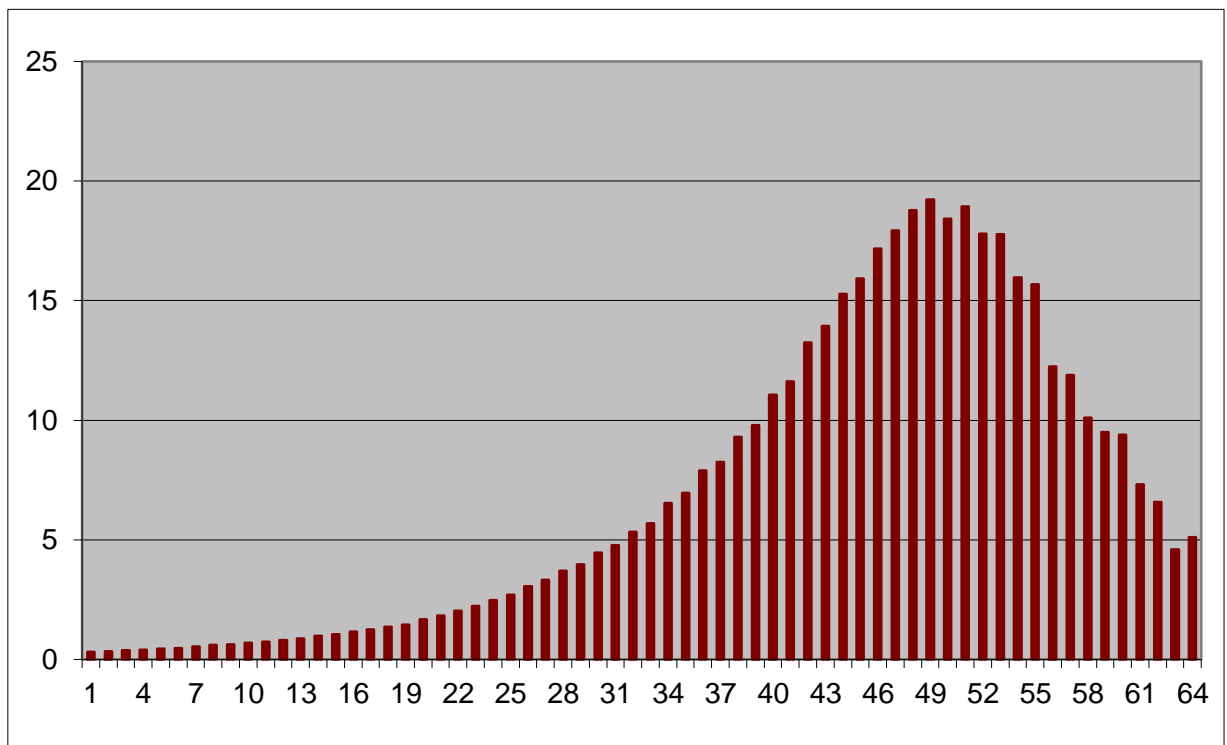


Figure 5.8: The average occurrence of defensive chunks throughout the game

The results displayed in figure 5.8 show that defensive chunks are frequent within the mid-game section. The number of defensive chunks on average increase steadily through the game until about fifteen moves before checkmate, after which the

number of defensive chunks decline. The rate of change of defensive chunks can therefore be an indicator of the progress of the game, the decline being indicative of an imminent conclusion of the game.

5.4.2 The persistence of defensive chunks with player skill

An analysis of games was undertaken to test for a correlation between the persistence of defensive chunks and the players' skill. This was achieved by measuring the average number of chunks on the board throughout the game. The analysis tests if a skilful player builds more defensive chunks into the game as part of a long-term strategy. The results show that there is no significant difference between skill groups. The lowest skill group shows a slightly higher persistence but as the sample number of games for this group is lower than the others the result is not considered reliable.

Skill Elo Range	Average Chunks throughout the game	Games Played	Number of Boards evaluated
1200 – 1399	0.323787	55	22836
1400 – 1599	0.284048	191	88890
1600 – 1799	0.295222	343	150978
1800 – 1999	0.276351	1134	539136
2000 – 2199	0.278841	8714	4113450
2200 – 2399	0.276492	18402	8827488
2400 – 2599	0.279601	18704	8795946
2600 – 2799	0.274532	4689	2290836

Table 5.2: A comparison of the use of defensive chunks with skill groups

The results of the analysis comparing the persistence of defensive chunks shows no significant differences between skill groups. It was therefore not considered necessary to perform any further statistical analysis on these data.

5.5. Chapter conclusion

A simple analysis of chess game data using chunks extracted by CHREST and provided by Gobet prompted a more detailed investigation into the relationship between the player's skill and the use of chunks. The analysis suggested that frequency of chunks throughout the game is related to the progress of the game (cf. figure 5.4) and that the number of non-base chunks present on the board is related to the skill of the player (cf. figure 5.5). However, an in-depth statistical analysis of the data found little or no correlation between the player's skill and the number of chunks used. The analysis of defensive chunks (cf. figure 5.8) also show a relationship between defensive chunks and the progress of the game. A more detailed statistical analysis using a Pearson Correlation between the player's skill rating and the number of 'non-base' chunks showed that there was in fact, little or no correlation between player skill and the selection of chunks on the board.

6. THE DEVELOPMENT OF A PROGRAM TO INVESTIGATE CHUNKING IN CHESS

The thesis so far has reported some of the statistics that accompany chunks within the game of chess. Although chunks have been found to be plentiful on the chessboard and the use of chunks are, from the literature, related to the player's skill, however, the nature of chunks within chess remains unknown. The analysis so far has shown that chunks exist in very large numbers, for example a typical chessboard with twenty-five pieces will combine to form 33,554,431 chunks.

The continuing investigation into chunking within chess will take into account the magnitude of chunk numbers but at the same time isolate chunks that are meaningful and are an indication of chess skill.

The thesis continues with a detailed description of a program developed in order to isolate and assign meaning to chunks within chess games.

6.1. CLAMP - 'Chunk Learning And Move Prompting'

"Human chess players do not perceive a position as a static entity but as a collection of potential actions" (Finkelstein & Markovitch 1998)

Extracting a database of chunks and the recognition thereof within a chessboard is in itself of little value in the same way that Ericsson and Harris's (1990) novice was trained to recall briefly presented positions to the same standard as a Master chess player, yet despite having this recall ability the novice was, not surprisingly, still a poor chess player. If however the recognition of a chunk can direct a player's attention to a move or strategy then the chunk has meaning and purpose. de Groot's

observations of Master chess players indicated that the presence of chunks had the effect of directing attention to specific chess pieces (de Groot 1978).

It has been established earlier in this thesis that chunks are present within chess configurations as a result of i) the rules of the game; ii) the properties of the pieces; and iii) the strategic play by the chess player. This chapter describes a program that uses chunks to suggest moves. If knowledge of chunks is advantageous to the chess player then the player must not only recognise chunks on the chessboard but also be able to derive an action from their presence. The previous chapters in this thesis have looked at chunks without any associated meaning.

CLAMP (which is an acronym for 'Chunk Learning And Move Prompting') extracts chunks from chess games and associates each chunk with a move. By analysing a large number of games CLAMP builds a library of chunks, with their associated 'meanings'. The library can be used to analyse a new chessboard (that is, a configuration that was not used in the process of building the library of chunks). Having found the chunks on a new chessboard, an accompanying program 'CLAMPanalyser' can compare the chunks with those held in the library and look up their associated moves. CLAMP and CLAMPanalyser are used to test the proposal that a move to be made on the chessboard is supported by a high number of chunks, where the chunks are associated with a move on previously analysed chessboards.

The method used by CLAMP and CLAMPanalyser is a simple *recognition/association* process; chunks are *recognised* by searching the chunk libraries and are *associated* to a move, based simply on which libraries the chunks are found within. Chunking (within human cognition) is "often called the recognition-association theory" (Cooke et al. 1993). We will see later that the results obtained from CLAMP and CLAMPanalyser show that chunking using a simple recognition/association is a

viable method to direct attention to salient moves, but first we need to consider the building of chunk libraries.

6.2. Building chunk libraries

CLAMP acquires chunks that exist on a chessboard just prior to a move of a piece to a square. The chessboards used for this process are taken from Grandmaster games¹¹. Grandmaster games were used as a data source for the following reasons:

- It is assumed that the moves made by Grandmasters are consistently good moves.
- Grandmaster games are plentiful and easily obtainable from the Internet giving the potential for a large data source.
- Games played by non-Grandmasters can be assumed to be absent from this dataset thereby ensuring availability of chess data *not* included within the library which can be used to test CLAMP.

6.3. Building chessboard collections

The first stage of building a chunk library requires the compilation of *collections* of chessboards giving example configurations before each possible arrival of a chess piece on each square on the board. For example, the chessboard opposite¹² shows the piece configuration with white to play from an actual

chess game. After consideration the player played Rxh7 (the rook on h3 took the

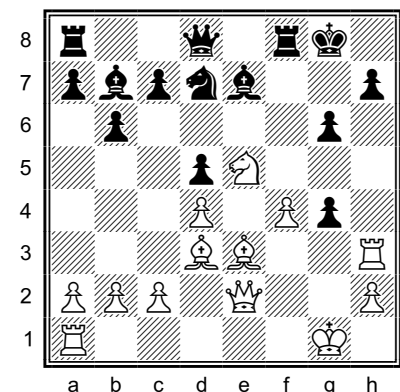


Figure 6.1: Resulting move:Rxh7

¹¹ A Grandmaster is defined as a player with an ELO skill rating between 2400 and 2799.

¹² The board FEN is: r2q1rk1/pbpnb2p/1p4p1/3pN3/3P1Pp1/3BB2R/PPP1Q2P/R5K1 w - - 0 1

pawn on h7). This chessboard could therefore be included in the collection for *rook arriving on 'h7'*

It may be that a piece arriving on a square is also a capture of the piece on the square. This additional information (whether a piece is captured or not) is not recorded explicitly. We will see however that the chunks that are generated from the board configuration (cf. paragraph 6.5) can contain any of the pieces on the chessboard before the move is made. Chunks can therefore include the captured piece. If a configuration of pieces frequently results in a move that will capture a piece then the captured piece will also exist within chunks that are associated with that move.

The chessboard below¹³ shows a configuration from another game. The move played was Ne5 making this configuration eligible for inclusion into the collection *Knight arriving on 'e5'*

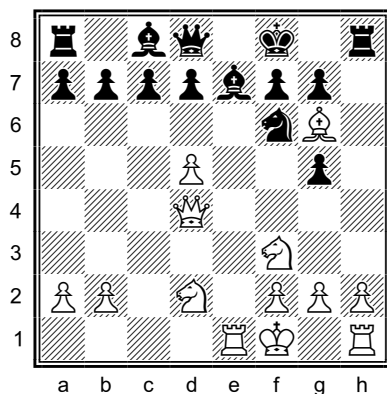


Figure 6.2: Resulting move: Ne5

A collection of one thousand chessboards was compiled with board layouts immediately prior to the arrival of a Rook on square h7. Another collection of one thousand chessboards was built giving boards prior to the arrival of the Knight on e5. Similarly, collections of one thousand chessboards for each piece (Pawn, Rook,

Bishop, Knight, King and Queen) arriving on each of the sixty-four squares of the chessboard were compiled.

¹³ The board FEN is defined r1bq1k1r/ppppbpp1/5nB1/3P2p1/3Q4/5N2/PP1N1PPP/4RK1R w - - 0 1

Building collections of chess moves in this way has the advantage that all moves have the same number of observations, but the disadvantage that information is lost about the frequency with which the moves occur in the environment. A factor for the frequency of occurrence is introduced later (cf. page 89) as the significance of each predicted move, after analysing the chunks on a chessboard, is scaled by a 'move rareness' figure.

By compiling collections of one thousand chessboards prior to each move (each piece to each of the sixty-four squares on the chessboard), each move is supported by a large number of chunks as each chessboard observation can contain a different assortment of chunks. The figure of one thousand chessboards was found to provide a sufficient variation of chunks so that at least one chunk present on a new chess configuration (not one of the chessboards used in the 'collections') can be found within the collection for a large percentage of cases (cf. page 94).

Compiling collections with equal numbers of chessboards also ensures that each move analysis can be treated equally; with the calculations for each suggested move using numbers of the same order (cf. page 82).

There are six types of piece (Pawn, Rook, Knight, Bishop, Queen and King), and sixty-four squares. According to the rules of chess, each piece can arrive onto any square on the board with the exception of the Pawn. The white Pawn starts in the second row and can only move forward and so it is impossible for a Pawn to *arrive* on the first or second row. Similarly if a Pawn arrives at row eight it is normally instantly promoted. Therefore rows one, two and eight are not viable positions for a white Pawn to arrive.

It should be noted CLAMP does not distinguish between the move of a pawn to the last row and the pawn being promoted, and with a real move of the same piece

(the piece that the pawn is promoted to) onto the same square. This could be a limitation in the operation of CLAMP when analysing moves to the last row as the chunks relevant to the two situations could be significantly different.

Both white and black pieces would need to be analysed to build a library of chunks for white and black piece moves. For this thesis, just white moves were considered; allowing CLAMP to suggest only moves when white is to play. To allow CLAMP to suggest black moves, a library of chunks would need to be built from chessboards just prior to black piece moves. The process would be identical to the process for white pieces, however, an analysis of *both* black and white pieces was considered unnecessary, as the proof of concept would be satisfied with the analysis of one colour only. For this thesis, only chessboards in the position of 'white to move' were considered.

For this research chunks are defined as frequently occurring constellations of pieces. In order to determine the frequency of occurrence, a number of boards were needed for each of the arrival squares, with a number of instances of each board extracted. Chessboards at move twenty ply or above were used for building the libraries, as the middle game is particularly interesting because it is outside the scope of standard openings and the board configurations are extremely varied. One thousand arrivals of each white piece on each square were extracted from a database of Grandmaster games (the database was supplied by ChessBase.com) requiring a total of 360,000 boards¹⁴ (of course, many more boards are examined to yield the required total). Some of the arrivals were rare, such as the white King arriving on position A8. From the collection of games used, in order to find one thousand instances of this move over fifty million boards (taken from 1,496,327

¹⁴ The number of possible moves is 6x64 (384) however as it is not legal for a pawn to arrive on rows 1,2 or 8 the number of legal moves reduce to 360.

Grandmaster games) where required. A large number of Grandmaster boards were examined to find the required number of instances for each piece on every square; each piece arrival is shown in the table below.

Move To:	Pawn	Rook	Knight	Bishop	King	Queen
A8	N/A	962602	9636907	4467304	50864607	2040445
B8	N/A	1174639	12460136	5599671	29176446	2332073
C8	N/A	820111	4095665	3056348	27958146	1984924
D8	N/A	683546	5134560	3938365	27870774	1478493
E8	N/A	935039	5050787	3800114	25795377	2066843
F8	N/A	1260767	5454715	2165152	27207363	2850746
G8	N/A	2113968	23639737	7782391	25396161	4042016
H8	N/A	1884914	23366342	20582787	46281290	2701517
A7	3521477	596916	4134549	3393952	18842186	1609270
B7	2935805	557450	2742381	1145894	9438385	1068589
C7	3242341	601982	1597288	2212306	9225011	1392916
D7	3230427	630209	1320142	1669882	9062871	1323358
E7	3582266	850656	1308897	1453806	8142512	1534606
F7	3618273	1018496	1670824	1907198	7884365	1680755
G7	4056859	1217361	3989406	889163	8756837	2482110
H7	4774298	1278239	6186154	3803997	14936412	2367307
A6	1008527	1006426	5181266	1975169	10668581	1958452
B6	635482	957923	1294400	1874056	5023890	1704808
C6	516532	788357	689251	674791	4138472	1082963
D6	800271	802506	499105	934086	3438595	1256137
E6	789274	1080382	620480	848327	3388856	1165232
F6	634253	1126181	737972	558351	3553416	1191246
G6	755318	1620143	1408503	2031708	901269	1600367
H6	1363652	1672341	3063138	1086343	6383310	1496397
A5	416995	1012265	1713427	1786855	8105232	1462195
B5	274587	979729	505148	758460	3210255	975425
C5	271800	757108	492546	705811	2304620	1030043
D5	218829	748873	269886	400474	1939944	687370
E5	201447	955100	262779	456398	1689275	836424
F5	247797	1067330	400775	953373	1673329	986576
G5	330288	1599027	431353	577783	2019553	1057355
H5	473781	1573794	1589764	1571545	3867717	696422
A4	202672	1110995	777620	2810634	7266197	568602
B4	209680	1181848	1567814	884435	2577234	1016290
C4	251143	835781	321433	392124	1470933	530568
D4	172106	734475	231347	385294	1087396	596908
E4	177802	900222	204929	374491	843087	573751
F4	139105	875703	452396	465262	910526	814563
G4	189311	1563035	1028113	745158	1114939	682095
H4	193203	1606167	620131	2422522	2099493	1050261
A3	302254	1010023	967534	516157	6448602	1452765
B3	360627	1030745	470838	999675	1704941	316647
C3	330105	787655	247661	364099	1075625	478525
D3	536731	721953	593026	372405	646436	405798
E3	498782	763272	478039	259096	433794	562873
F3	274374	620238	223837	300180	337135	392134
G3	326352	1141493	572883	1609019	523076	865603
H3	240826	1345626	1465702	683675	1034394	1537984
A2	N/A	827830	5139687	5923694	5934705	1974852
B2	N/A	824280	2325303	467965	1414601	656674
C2	N/A	491314	858240	1271097	955550	245706
D2	N/A	474966	205475	345030	524542	277968
E2	N/A	547827	343896	354055	303757	277132

F2	N/A	481116	1244209	1453267	237962	768195
G2	N/A	1229702	2116174	627078	172481	1293581
H2	N/A	1686492	2001144	10191236	307153	5005480
A1	N/A	489915	12684118	1975657	8302218	1939527
B1	N/A	204455	1776868	3904377	1222527	1141736
C1	N/A	133369	1868735	596446	651583	762673
D1	N/A	124670	1278145	1263712	933093	629093
E1	N/A	141096	880389	1529827	776908	866834
F1	N/A	138663	936311	484712	313351	1442860
G1	N/A	491533	2650237	7160748	249171	6669745
H1	N/A	980212	16876043	3893057	334234	9493423

Table 6.1: The number of chessboards required to find 1,000 instances of each piece (white only) moving to a square.

The *collections* of one thousand arrivals were saved in files with the *piece name* and *arrival square* as components of the filename (360 files in total).

6.4. Move asymmetry: A case against Holding (1985)

It is interesting to note the lack of symmetry in table 6.1 (above). Considering two points at opposite sides of the chessboard, the number of instances of a bishop arriving on H6 is substantially greater than the instances of a bishop arriving on square A6. When building the collection 1,086,343 boards were required to find 1,000 instances of the bishop arriving on A6 compared to 1,975,169 boards for one thousand instances for bishop arriving on square H6 (therefore the arrival on A6 is more frequent and easier to find than and arrival on H6). The lack of symmetry is an argument against Holding's suggestion that "chunks might be remembered in relative positions" (Holding 1985), as Holding considered Chase and Simon's (1973) estimate of chunks held in long-term memory as being too numerous (cf. page 17). The lack of symmetry in piece arrivals is a possible reinforcement of the view that chunks have different meanings depending on their *absolute* position on the chessboard. The lack of symmetry is even more pronounced when considering piece arrivals to row seven

and eight (the differences between left and right positions undoubtedly due to the initial positions of the King and Queen).

The notion that chunks are *absolutely positioned* is endorsed by Saariluoma (1994) in a series of experiments that measured the recall ability of chunks from their original position on the board and the same configuration of pieces shifted from their original positions. Even though the forms of the chunks were unchanged the recall ability decreased if the chunks were moved, implying that location information is part of the knowledge stored with the chunk in the chess players memory.

Gobet and Simon (1996) report similar results from experiments with recall of chunks using chessboards transposed to a mirror image of original configurations, using a computer simulation, with a version of CHREST. The number of chunks found on a vertical axis mirrored image chessboard is reduced compared to the recognition of chunks from the original board. Similar results were obtained, and reported in the same paper, when testing human chess players.

To conclude, the lack of symmetry in table 6.1 can be used as another argument for the notion that chunks are absolutely positioned on the chessboard. The different frequency of moves of pieces to squares on the left and right hand sides of the board show that the moves of pieces have different significances on opposite sides of the board. As the presence a piece occurs with different frequencies depending of the position on the chessboard, chunks of pieces (with the same relative piece positions within the chunk) will have different properties depending on the position of the chunk on the board. A chunk will therefore have different properties when located on different positions on the chessboard. For this reason chunks of pieces can only be considered to have the same properties when the pieces are located on fixed positions on the chessboard.

6.5. Analysing chunks from collections

Having compiled collections of boards that contextualise moves of specific pieces to specific squares, the next stage in the chunk library building process was to analyse each board within each collection to find repeated chunks. To build a 'chunk library' CLAMP systematically analysed each board in each collection, extracting all chunks in the boards and assigning them to the move that followed. Collection *Rook arriving on a7* for example, contains one thousand boards, each of which existed in a Grandmaster game immediately prior to the move of the Rook to square 'a7'. All chunks on each board are assigned to the move 'Rook to a7' and the count of how many times each chunk was found is maintained in the chunk library. As each collection was examined, chunk libraries were built and the chunks associated with the move that defined the collection.

A board can contain up to thirty-two chess pieces on sixty-four squares. All combinations of pieces (pieces with their absolute location on the board) are generated to create chunks. The order of pieces within the chunk is not considered significant and so the elements of the chunk are arranged in ascending order of value. Ordering the pieces within the chunk eliminates duplications of chunks where the pieces are arranged in a different order. As a result the number of chunks is the total number of *combinations* rather than the total *permutations* of pieces.

Using the formula described on page 37, it is therefore possible for CLAMP to find up to 4,294,967,295 chunks for *each* board, with chunk sizes ranging from one to thirty-two pieces. The theoretical maximum number of chunks when analysing one thousand boards in a *collection* could therefore be as high as 4.2 trillion. Storing and processing data of this size would present serious technical difficulties, however, as a game precedes beyond the opening moves the likelihood of finding very large *frequently occurring* chunks diminishes as the board layout becomes increasingly

diverse. This is supported by data obtained by CHREST (supplied by Gobet) from chunks extracted from Grandmaster games; the distribution of chunks by chunk size is shown in table 6.2 below:-

Chunk Size	Number found	Percentage of total
4	58,698	23.32%
5	63,457	25.21%
6	54,630	21.70%
7	36,869	14.65%
8	20,814	8.27%
9	10,036	3.99%
10	7,231	2.87%
Total:	251,735	100.00%

Table 6.2: Chunk sizes from CHREST data

CHREST was programmed to extract chunks of size between four and ten pieces. The number of chunks reduces as the chunk size increases above five pieces; the number of chunks present with a size of ten pieces reduces to less than 3% of the total chunk count. The research reported in this thesis

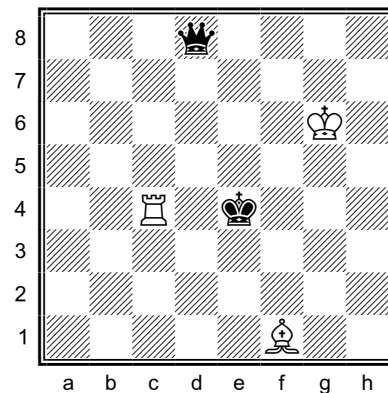


Figure 6.3: A simple chessboard

investigates chunk sizes of 2,3,4,5,6 and 7 pieces. Chunks of size greater than seven pieces account (in total) for less than sixteen per cent of all chunks extracted by CHREST and are, compared to chunk sizes of fewer than seven pieces, considered too infrequent for this study. Consequently the maximum number of chunks reduces considerably as the combination of pieces are limited to seven rather than the maximum of thirty-two.

It should be noted that, because of the huge variation between chess games each chessboard can be considered unique and therefore the chance of finding a

large group of pieces in the same configuration diminishes with increasing group size.

To illustrate how the number of chunks reduce, consider the simple chessboard shown in figure 6.3 above. The chessboard shows five pieces on squares as follows:

qd8, ke4, Kg6, Rc4, Bf1

The pieces combine to give a total of thirty-one combinations. If combinations with more than, for example, three pieces are removed, then the number of combinations reduce to twenty-four as illustrated in the table below.

<qd8>	<Bf1, qd8>
<ke4>	<Bf1, ke4>
<ke4, qd8>	<Bf1, ke4, qd8>
<Kg6>	<Bf1, Kg6>
<Kg6, qd8>	<Bf1, Kg6, qd8>
<Kg6, ke4>	<Bf1, Kg6, ke4>
<Kg6, ke4, qd8>	<<Bf1, Kg6, ke4, qd8>
<Rc4>	<Bf1, Rc4>
<Rc4, qd8>	<Bf1, Rc4, qd8>
<Rc4, ke4>	<Bf1, Rc4, ke4>
<Rc4, ke4, qd8>	<<Bf1, Rc4, ke4, qd8>
<Rc4, Kg6>	<Bf1, Rc4, Kg6>
<Rc4, Kg6, qd8>	<<Bf1, Rc4, Kg6, qd8>
<Rc4, Kg6, ke4>	<<Bf1, Rc4, Kg6, ke4>
<<Rc4, Kg6, ke4, qd8>	<<Bf1, Rc4, Kg6, ke4, qd8>
<Bf1>	

Table 6.3: Piece combinations with chunks of size greater than three marked with a strikethrough

In order to make comparisons between chunk sizes, CLAMP processed the collections to build chunk libraries for *distinct* chunk sizes; for example, a library of chunks was built with four piece chunks and another library with five piece chunks. Building libraries for each chunk size in isolation also reduces the maximum number of chunks processed and stored. In the above example the number of three-piece chunks resulting from the combinations equate to ten. Note that chunks are made from both black and white pieces.

6.6. Chunk size and memory requirements

The following table shows the computer memory required against chunk size:

Chunk Size (Max)	Number of chunks after combining the pieces (Total)	Number of chunks of selected chunk size	Memory required for combinations (bytes)
2	528	496	1552
3	5,488	4,960	21,392
4	41,448	35,960	201,192
5	242,824	201,376	1,409,448
6	1,149,016	906,192	7,752,792
7	4,514,872	3,365,856	34,679,640

Table 6.4: Chunk Statistics and memory requirements

Table 6.4 shows the maximum numbers of chunks for a chessboard with thirty-two pieces when limiting the chunk size to between two and seven. The table shows the number of chunks increases with an increasing chunks size, with a corresponding increase in memory required for processing and storing. Limiting the number of pieces in the chunk reduces the number of potential chunks that can be extracted from a chessboard to more manageable proportions. The potential number of chunks processed is the sum of the chunks extracted from each chessboard in a collection (a collection consists of one thousand boards). As each chunk occupies space in memory or on disk, careful consideration was given to the software design to maximise processing speed and minimise memory requirements while processing data.

The following code snippet, which is written in the C++ language, shows how pieces are combined to produce all combinations with a size 'CHUNKSIZE' pieces. The function is called, passing each piece and position to construct all combinations on the 'List' array variable (the memory size required for the List variable is shown in table 6.4).


```

#define BLOCKCELLS 20000000

BOOL CGenLibs::Combination(char piece, char pos)
{
    int size;
    short px;
    px = piece;
    px *=64;
    px += pos;                                //px is the 'piece-position object'
    long TheEnd = end;
    List[end++] = px;
    List[end++] = -1;                          //add terminator (-1)
    start = 0;
    for(;;) {                                  //add px to all existing cells
        if( start == TheEnd ) break;
        for(size=0;size<32;size++) {
            if(List[start+size] == -1) break; //break out with size set to the chunk size+1
        }
        size++;                                //include the spacer character
        start+=size;                           //move start on if we need to skip this chunk
        if(size > CHUNKSIZE ) continue;       //only consider chunks of chunk size or less
        start=-size;                           //revert back to chunk start to continue

        for(;;) {                               // Add piece to chunk - all
            List[end] = List[start++];         //copy this sequence to new sequence for adding
            if( List[end] == -1 ) break;
            end++;
        }
        List[end++] = px;                       //append this Piece to the sequence
        List[end++] = -1;                       //mark end of sequence

        //Reallocate memory if the buffer is too small
        if( end*2 > (BLOCKCELLS*SizeMem)-1280 ) { // Memory Buffer too small!
            SizeMem++;                          //increase allocation
            List = (short *)realloc( List, BLOCKCELLS*SizeMem);
            if( List == NULL ) return true;      //return on error
        }
    }
    return false;                              //exit 'ok'
}

```

Figure 6.4: Code Snippet – Making chunks by combining pieces

Combinations of pieces are assembled in memory pointed to by the variable 'List'.

The memory used is allocated from the system heap as required.

6.7. CLAMP design, data structures and processes

The CLAMP program processes a large amount of data when analysing piece constellations from many Grandmaster games. Careful consideration was given to the design of data structures and processes for efficient processing and data storage.

In order to conserve memory space, a 'piece-position' is represented by CLAMP as a single integer comprising of sixteen bits. The lower six bits define the square on the chessboard that the piece occupies (six bits have the range capacity for 2^6 or decimal sixty-four numbers). The upper ten bits of the integer define the chess piece. As chunks comprise of both black and white pieces there are twelve possible piece types (six black and six white) that need to be defined, needing at least four bits. The 'piece' part of the integer and the 'position' part require ten bits in total. A sixteen-bit integer was chosen, as it is the smallest integer that the compiler supports that accommodates ten or more bits.

Piece-positions within chunks are sorted into ascending order, for example:

< Nc3, Bb3, Ra2, Pa1 >

... is considered to be the same chunk as:

< Bb3, Nc3, Pa1, Ra2 >

Sorting piece-positions into ascending order will:

- Eliminate multiple representations of the same chunk where pieces within the chunk are combined in a different order (a similar technique is adopted by CHREST as a means to remove 'redundant links' between chunks (Groot and Gobet 1996).
- Facilitate forward only searching through chunk library lists.

Forward only searching is a fast and efficient method to search for chunks within the chunk libraries. The 'cursor' (the position within the file currently being looked at) starts at the beginning of the library file and progresses to the end of the file, without the need to re-start at any point. The library is organised so that the components of a chunk are found in ascending order, for example Nc3 will always be positioned after Bb3. When searching through the library Nc3, will always succeed Bb3 so that, provided we search for piece-positions in the same order, we need only one pass through the library to find all components of the chunk.

Piece-positions are stored in a structure called a 'node', containing a counter, to record the frequency of the piece-position, and a pointer to index another node. When building a chunk library the pieces on a chessboard are combined to make chunks. Chunks are added to the library by adding piece-positions after sorting the piece-positions into order, and advancing through the library file from the beginning to the end. Chunks are stored within a library by linking nodes to construct a *linked list data structure*.

The 'node structure' is defined as follows:

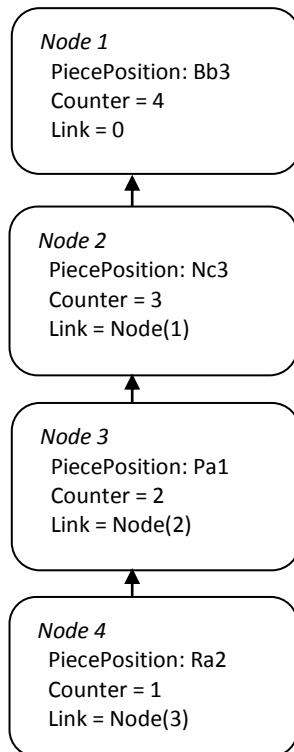
```
struct _Node {
    short PiecePosition;           //the Piece-Position
    short Counter;                //count of times this node was searched
    long Link;                    //address of preceding node
};
```

Figure 6.5: Code Snippet - The 'node' structure

6.8. Combining 'nodes' to build 'lists'.

Chunks are represented by a linked list of nodes, for example, a four-piece chunk will comprise of four nodes, with the link in each node pointing to the previous node.

The chunk < Bb3, Nc3, Pa1, Ra2 > would have nodes as follows:

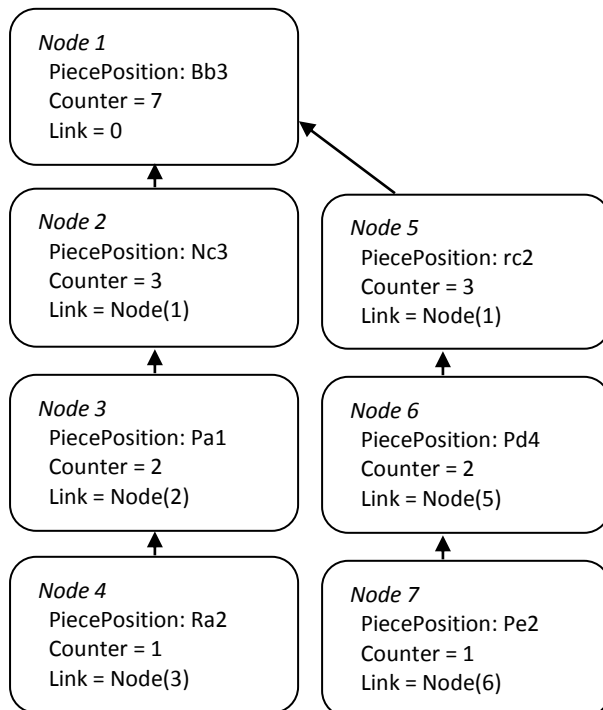


The first 'piece-position' is always the starting node. Subsequent piece-positions are added to the library by adding a node with the 'Link' pointer indexing the previous node (the node that contains the previous 'piece-position'). Similarly, the third piece-position node will be added with the link indexing the second node and so on.

A second chunk that is to be added to the library that also starts with Bb3 will *add* to the linked list. The chunk <Bb3, rc2, Pd4, Pe2> will join, omitting the first node, but incrementing the counter in the first node. The counter keeps account of how many times the node has been referenced (this count will later be used to

eliminate infrequently occurring nodes from the library so that only frequently occurring chunks are contained within the chunk library).

The list of nodes, after adding the chunk <Bb3, rc2, Pd4, Pe2> will be as follows:



The following example shows how five chunks are represented in the list structure.

Chunks to add:

< Bb3, Nc3, Pa2, Ra1 >

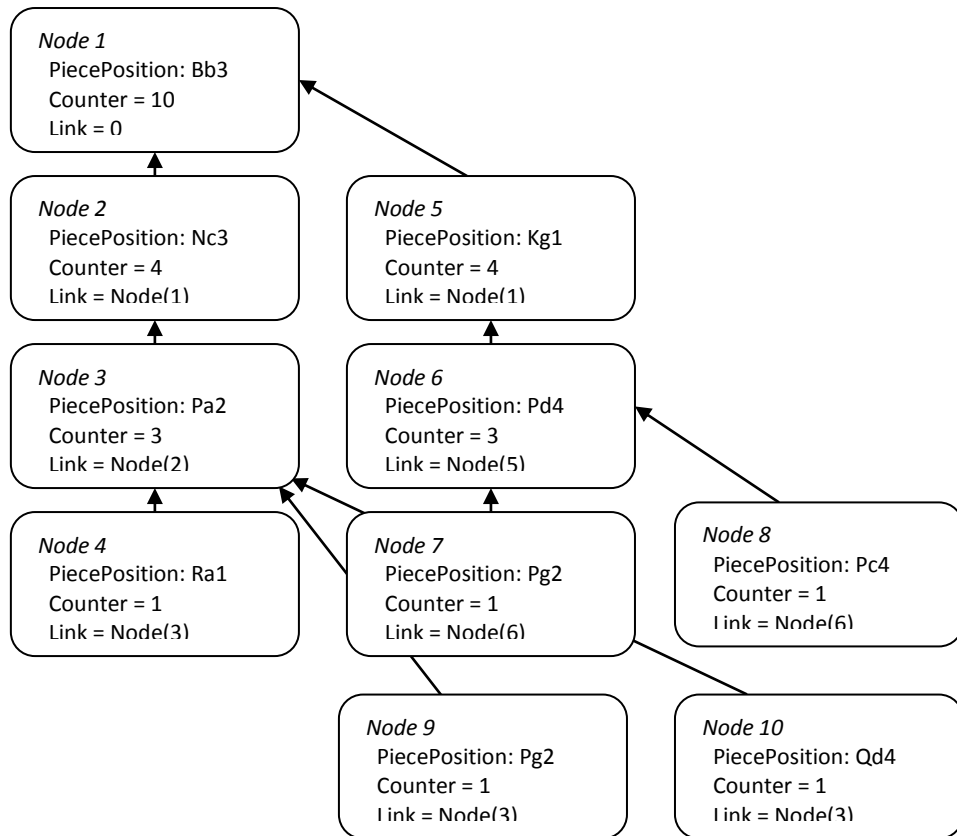
< Bb3, Nc3, Pa2, Pg2 >

< Bb3, Nc3, Pa2, Qd4 >

< Bb3, Kg1, Pd4, Pg2 >

< Bb3, Kg1, Pd4, Pc4 >

The chunks are represented in the list as follows:



The representation of chunks in the list structure is necessary for two reasons.

- The list will hold in a compact form a large collection of chunks. Duplicate piece-position sequences are eliminated, as are duplications of the same chunk. The example above has, for example, reduced six chunks consisting of thirty piece-positions to just ten nodes.
- Each node in the list has a counter to record the frequency of the node. For this thesis one of the properties of a significant chunk is that it is frequently

occurring. The node counter can be used as a measure of the significance of the chunks within a list.

6.9. Building 'lists' from collections

When building chunk libraries, CLAMP will take a collection (a collection is one thousand chessboards showing the configuration immediately prior to a specific move of a piece to a square), combine the pieces on each board in the collection to construct chunks, and save the chunks in list structures. The elements of the list consist of the piece (Pawn, Bishop, King Etcetera), which is represented as a numeric code, plus the position the piece occupies on the chessboard. The combination of piece and position gives a unique numeric value that represents the piece on a square. Chunks are 'pieces on squares' and are sorted in order of the numeric representative values. Because the chunks are sorted into ascending order the first element in the chunk is always the root node of a list. When building the chunk libraries there may be up to 768 lists in construction, each list having a different root node for each of the twelve chess pieces (chunks are composed of black and white Pawn, Rook, Knight, Bishop, King and Queen pieces) on each of the sixty-four squares of the board. CLAMP will first find the list with the root node as the beginning piece-position of the chunk. CLAMP then searches through the list to find the node that matches the second piece-position *and* where the 'Link' element in the node points to the previously matched node. If a node is found then the counter element is incremented, otherwise a new node is appended to the list. CLAMP continues searching from the current position for the next node and so on until all nodes in the chunk have been found or added.

As piece-positions within chunks are ordered in ascending value, higher value piece-positions will always be added to the list after lower value piece-positions. This

property of the list ensures *forward only* searching through the list, by virtue that higher value elements will always succeed lower value elements in position within the list. Adding a chunk to a list is therefore efficient and fast as it involves, at most, just one search through the list.

All chunks from each board form a collection stored in lists using the process described above. Due to the large number of chunks that may be present on each board, and as one thousand boards in the collection are to be assimilated, the process of consolidation of chunks into lists needs to be efficient. When processing all of the data from a collection, the number of nodes stored can be large, for example the actual number of chunks contained in a library consisting of five piece chunks amounts to 45,646,773 with each *node* comprising of eight bytes, giving a potential file size of 1,825,870,929 bytes. The actual size of the library reduces to 599,966,829 bytes (a 67% reduction in size) as a result of the elimination of duplicate sequences.

The reduction in storage size by using the list is important because the lists are constructed in the computer's physical memory. If the memory required for the construction of lists exceeds the available physical memory then the processing time would be greatly extended due to the need to swap physical memory with disk storage and thereby incurring an associated operating system overhead.

6.10. The structure of a 'Trie'

The 'list' data structure described above has the elements of the list sorted in order. As chunks are added, the nodes may link in ways representing 'branches' from the 'trunk' like branches in a tree. As the elements within the structure are such that the first node (or root) is shared by all chunks within the structure, and the elements of the chunks are stored in the contents of each node in the path from the root to the

node, rather than the node itself (refer to section 6.8 for more a more detailed explanation of this structure). In this thesis 'lists' are contained in a structure named a 'trie'. The 'trie' structure is the list prefixed with a header represented as a 'C' program structure as follows:

```
struct _Trie {
    short PeicePos;           //Root Piece-Position
    long NodeMax;            //Number of nodes in this trie
    long NodeLimit;          //Maximum nodes allocated to this trie
    long Address;            //Address of starting node
    struct _Node
        short PiecePosition; //Piece-Position of this node
        short Count           //node frequency
        long Link              //this contains address of preceding node
    } [NodeLimit];           //there are 'NodeLimit' node structures
}
```

Figure 6.6: Code Snippet - The 'trie' structure

The trie structure has the piece-position, which is the root of the trie, as the first parameter. The root is the first element in the chunks that are stored in the trie. The 'NodeMax' value is the number of nodes that are used in the trie. The 'NodeLimit' is the memory space allocated during construction of the trie. When adding a node, if the NodeLimit has been exceeded then CLAMP will allocate more memory and increase the NodeLimit value in the trie structure.

The following code snippet from the CLAMP program¹⁵ shows how the trie structure is constructed when a chunk is added to a list:

¹⁵ The full source code for CLAMP is provided on the disks that are attached to the back cover of this thesis.

```

int CGenLibs::AddNode(int size, short Tchunk[])
{
    int Base;
    int exists = 0;
    int LinkPos;
    char buff[80];
    struct _Node * Node;
    int PrevLink = -1;                                // set for first Piece (no previous links)

    for(Base=0;Base<MAXTREES;Base++) {
        if( TreeStart[Base].PeicePosition == -1) {      // make a new tree if necessary
            TreeStart[Base].PeicePosition = Tchunk[0];
            TreeStart[Base].chunklimit = 1000;
            TreeStart[Base].Tree =
                (struct _Node *)malloc( sizeof(struct _Node) * TreeStart[Base].chunklimit );
            if( TreeStart[Base].Tree == NULL) return;   // Insufficient memory!
            TreeStart[Base].chunkmax = 0;
        }                                              // Have we seen this before? If so add to the tree
        if( TreeStart[Base].PeicePosition == Tchunk[0]) break;
    }

    for(int p=0;p<size;p++) {
        LinkPos = 0;
        Node = TreeStart[Base].Tree;
        for(;;) {
            if( LinkPos == TreeStart[Base].chunkmax ) {
                if( TreeStart[Base].chunkmax == TreeStart[Base].chunklimit ) {
                    TreeStart[Base].chunklimit += 1000;

                    TreeStart[Base].Tree = (struct _Node *)realloc( TreeStart[Base].Tree,
                        sizeof(struct _Node) * TreeStart[Base].chunklimit);           //increase allocated memory
                    if( TreeStart[Base].Tree == NULL ) return -1;
                }
                Node = TreeStart[Base].Tree;
                Node += LinkPos;
            }
            Node->Piece = Tchunk[p];
            Node->Count = 1;  ///#
            Node->Link = PrevLink;
            PrevLink = LinkPos++;
            TreeStart[Base].chunkmax++;
            Node++;
            //always add to the new node after adding a node
            if( ++p < size ) continue;
            break;
        }
        if( Node->Piece == Tchunk[p] ) {                //most unlikely first
            if( Node->Link == PrevLink ) {

                // just for information on chunks
                Node->Count++;                          //found again
                if(p == size-1) {                       //
                    if( Node->Count > THRESHOLD ) exists = 1;
                }
                PrevLink = LinkPos;
                break;
            }
        }
        LinkPos++;
        Node++;
    }
}
return exists;                                     //return the number of times this chunk exists
}

```

Figure 6.7: Code Snippet – CLAMP function to construct ‘trees’

6.11. Combining 'tries' to make a 'library'

A chunk library consists of a group of files, each file being composed of the tries compiled from the processing of one collection and associated with the move that generated the collection. Each file is given a name that is linked to the move of the piece to the square that was the basis for the collection.

One of the definitions of a chunk (cf. page 23) is that chunks are frequently occurring configurations. In order to remove infrequent chunks from the libraries, infrequent nodes are removed from the tries before saving in the library. The number of times a node has been found is recorded in the counter (an element of the node structure). An arbitrary threshold of a chunk existing on one per cent of the total boards within the collection was assumed so, as one thousand boards were processed in each collection, all nodes with a counter value less than ten were removed from the tries before copying to the library file.

Rather than having each trie as an individual file (there are 768 tries resulting from each collection), which would generate a total of 276,480 files for all collections, the tries from a collection are concatenated into a single 'library' file. The library therefore consists of 360 files, corresponding to one file for each white piece arriving on each of the sixty-four squares of the chessboard. One file results from each collection. The library file contains chunks that are associated with a move of a piece to a square on the chessboard.

The library file has a structure shown in the 'C' code snippet below:

```
long IndexSize           //this contains the number of index items
long ChunkSize;         //this contains the chunk size
long TreeSize           //this contains the size of the tree data

struct _index {         //index structure
    short PiecePos;     //this contains the tree root item
    long StartPos;     //this contains the address of the tree structure
    long Size;         //this contains the size of the tree structure
} [IndexSize];         //there are 'IndexSize' items, sorted by 'PiecePos'

struct _Tree {
    short PeicePos;     //Root Piece-Position
    long NodeMax;      //Number of nodes in this tree
    long NodeLimit;    //Maximum nodes allocated to this tree
    long Address;      //Address of starting node
    struct _Node
        short PiecePosition; //Piece-Position of this node
        short Count         //node frequency
        long Link           //this contains address of preceding node
    } [NodeLimit];        //there are 'NodeLimit' node structures
} [TreeSize];           //there are 'TreeSize' tree structures
```

Figure 6.8: Code snippet - The Library file format

The library file is associated with an arrival of one chess piece type to a specific square on the chessboard. The file contains all of the tries associated with the move, with an index to facilitate rapid access to the part of the file containing the trie of interest. A header section records the *size of chunks* that were processed to make the library, the *number of items* that are indexed and the *number of tries* within the file. The index includes the starting piece-position of the chunk that is the root of the trie, with the starting position as a byte offset to the start of the trie, and the number of nodes that make up the trie. The index is sorted in ascending order of starting piece-positions.

6.12. A graphical representation of a chunk

This chapter has described how chunk libraries were built. A chunk library encapsulates domain knowledge by associating chunks with moves. A chunk can be associated with a number of moves. Each move that the chunk is associated to is

'scored' with the number of occurrences that the chunk is present within a sample collection of chessboards that precede the move. The 'move' is a composite of the *piece* and the *square* it is moved to. The number of possible moves is therefore three-hundred and sixty as there are sixty-four squares and six piece types but minus twenty-four illegal moves which include a pawn arriving on the first, second and eighth rows (cf. page 56). A chunk can be represented in graphical form with the 'x' axis labelled 1-360, representing the piece/position combination of the move and the 'y' axis as the score for the move.

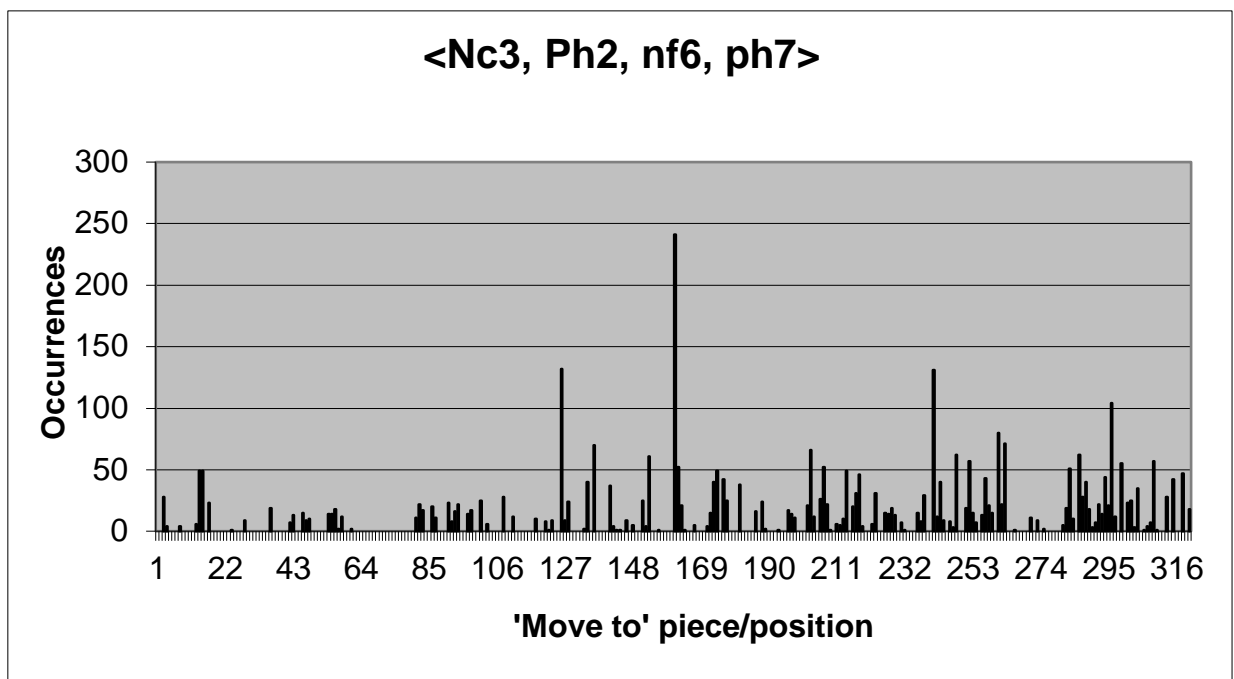


Figure 6.9: Graphical representation of chunk < Nc3, Ph2, nf6, ph7 >

The key for the axis 'Move To piece/position' for the graphs shown in figures 6.9, 6.10 and 6.11 is given in appendix 13.2 on page 187.

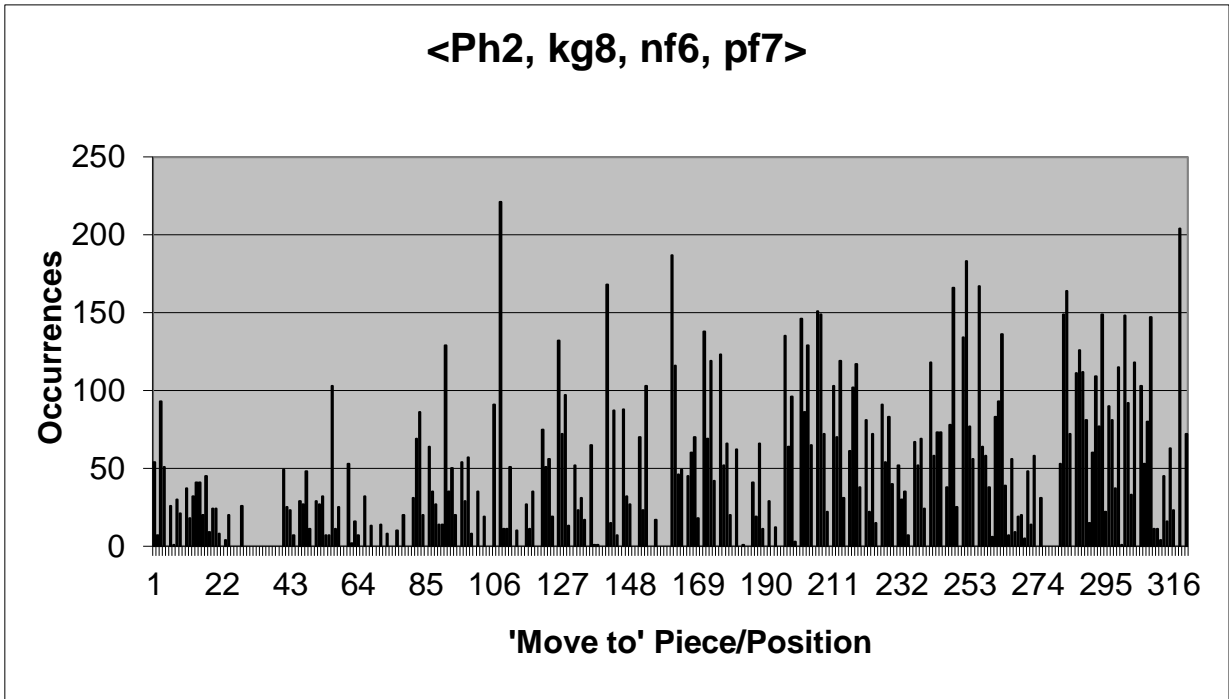


Figure 6.10: Graphical representation of chunk < Ph2, kg8, nf6, pf7 >

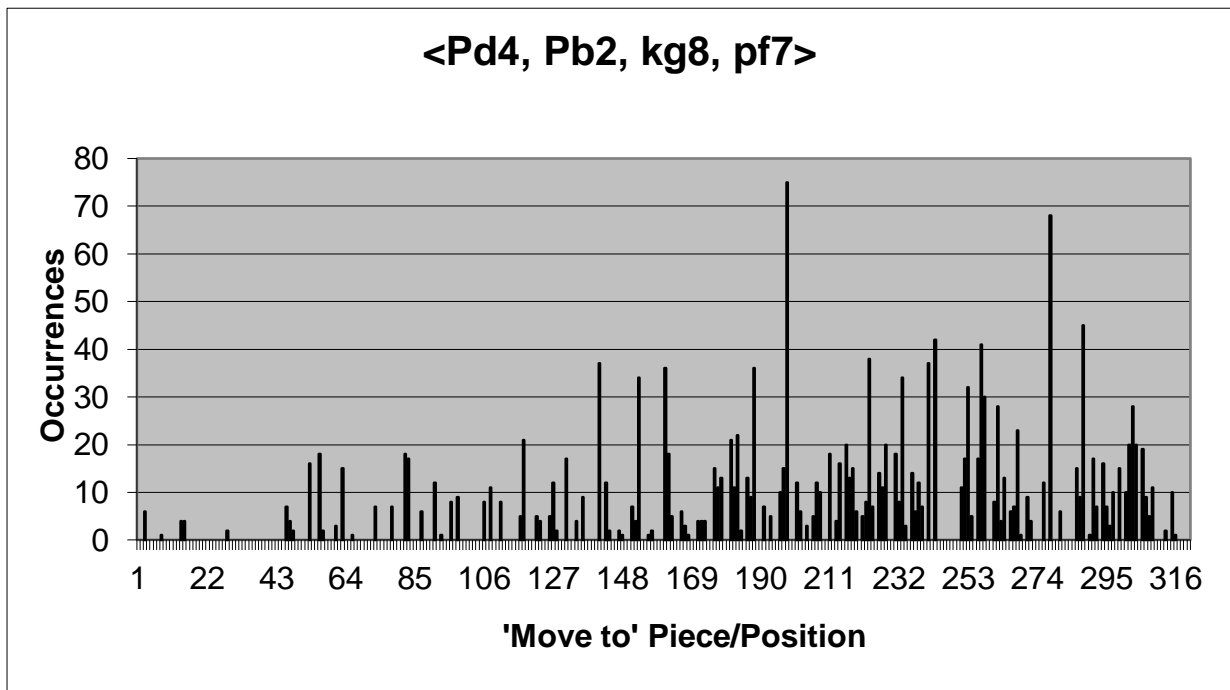


Figure 6.11: Graphical representation of chunk < Pd4, Pb2, kg8, pf7 >

A four-piece chunk compiled from pieces on the whole board area and consisting of pieces < Nc3, Ph2, nf6, ph7 > is represented in the graph shown in figure 6.9 (above). A typical chessboard would contain approximately 36,000 chunks (cf. page 64) each with a unique profile. Figures 6.10 and 6.11 show the profiles for two more four-piece chunks(compiled from pieces on the whole board area). Chunk < Ph2, kg8, nf6, pf7 > was found on chessboard configurations prior to a number of moves. Chunk < Nc3, Ph2, nf6, ph7 > however was found on fewer configurations.

6.13. Hardware considerations

The building of library files requires a large amount of processing despite careful consideration being given to the relevance of data, the size of the chunks, data structures and searching techniques. Several variations of libraries were built (the variants are discussed later in the thesis cf. page 92) which required CLAMP to run numerous times. The software was written in C++ and designed to exploit physical available memory on the processor to minimise operating system overheads. The process was run on a cluster computer (The Birmingham University BlueBEAR cluster¹⁶) employing 128 processors for parallel processing under MPI (Message Passing Interface), the program being written to exploit the parallel processing properties of the cluster and as a result the program execution, compared with running CLAMP on a PC, was increased by a factor of about one hundred times. CLAMP is a compiled C++ program using efficient techniques to maximise the speed of execution, however, some library builds required a lot of processing, for example building a chunk library from pieces from the entire board space, for a chunk size of five pieces would take approximately five months on a high performance stand-alone

¹⁶ BlueBEAR is equipped with 1536 cores with 8GB memory per node (each node has four cores) and 150 TB of disk space.

P.C. Using 128 processors on BlueBEAR the processing time reduced to twenty-eight hours.

The research required for this thesis necessitated the compilation of thirty chunk libraries.



Figure 6.12: The BlueBEAR Cluster computer.¹⁷

6.14. Chapter conclusion

This chapter has described the program ‘CLAMP’, a software program designed to compile ‘libraries’ of chunks that can be associated with chess piece moves within the game of chess. CLAMP is designed so that it can compile libraries according to various specifications, to build libraries made from chunks with specified numbers of

¹⁷ This photograph is shown with permission from the IT Services, The University of Birmingham.

chess pieces, or with specific relationships between the pieces of the chunk (such as pieces in close proximity, or pieces in defensive relationships). The chunk libraries link chunks that are present on a chessboard to moves that are commonly made by a chess player. Chunk libraries therefore encapsulate the knowledge of commonly played chess moves following the presence of specific chunks on a chessboard. The next chapter describes the program 'CLAMPanalyser' which will use the knowledge contained within the chunk libraries to suggest chess moves.

7. USING 'CLAMPANALYSER' TO SUGGEST MOVES

“Human masters, whose play is still much better than the best programs, appear to use a knowledge intensive approach to chess. They seem to have a huge number of stored ‘patterns’ and analysing a position involves matching those patterns to suggested plans for attack or defence” (Wilkins 1980).

The previous chapter described how CLAMP built ‘chunk libraries’ by analysing Grandmaster games. This chapter will describe how the program ‘CLAMPAnalysers’ uses the chunk library to select a move. CLAMPAnalysers will analyse a chessboard (a chessboard that was not used in the process to build the libraries) by extracting chunks in the same manner as used for the ‘library building’ process (cf. page 61). Each possible move on the test board is examined and the chunks searched for, in each of the appropriate libraries. A library file is associated with a move of a piece to a square. If the chunk from the chessboard being analysed is found within the library then the score for this move is increased. The *move score* is the total of the number of chunks from the chessboard that are found to exist in the appropriate library file for that move.

In order to suggest a move CLAMPAnalysers will combine all of the pieces on a test chessboard to build chunks. The pieces on the test chessboard are combined to make chunks with the same chunk size as the chunks within the library, the ‘compatible chunk size’ being read from the first byte in the library file. The pieces and their positions on the chessboard are combined and sorted into order using the same method that was used when building the library files.

Considering each piece on the chessboard and all possible moves for that piece, CLAMPAnalysers will compile a list of all of the pieces and their arriving squares. The chunk library consists of a set of files with each file corresponding to

each piece arriving on each square. To calculate a score for the move CLAMPanalyser will count how many of the chunks are present within each of the corresponding library files. For example, if the test board can make the move:

‘Bishop to square A5’

CLAMPanalyser will examine the library file associated with this move (*Bishop to square A5*). CLAMPanalyser will search for each chunk that has been generated from the test board, to see if it exists in this library file. Each time a chunk from the test board is found in the library file the score for that move (in this case *Bishop to square A5*) is incremented.

Using the same procedure, every possible move for each chess piece on the chessboard can be scored. If the analysis of a move gives a high score then a lot of the chunks on the test board match the chunks within the collections of boards that are associated with that move. A high score is therefore an indication that the move was frequently played in the set of Grandmaster games (the games that were used to build the library) when similar chunks were present on the chessboards. Assuming that Grandmasters make good moves the score can be an indication of likelihood that the move is also a good move.

CLAMPanalyser does not possess any knowledge of the relative value of pieces or tactics that could determine which out of all the possible moves would be a better move. The program has no knowledge of the effect of a move, for example, if moving one piece would expose another (break a defensive relationship), or even if a move onto a square occupied by an opponent’s piece would result in the opponent’s piece being taken. The score assigned to each piece is based purely on the

comparison of chunk patterns from the test chessboard with those stored in the library files. CLAMPAnalyser performs an assessment of the board without look-ahead or the sophisticated scoring methods that are common in conventional chess programs.

The results that follow in this chapter show a correlation between the score assigned from CLAMPAnalyser and the score calculated by a conventional chess program. Moves which are assigned a high score by CLAMPAnalyser are therefore suggested as likely candidates for the move to be made by the player.

7.1. An evaluation of scores for a move to a position.

The chessboard¹⁸ shown opposite is taken from a tournament game between two expert players, Kenneth Coates (white) and James Parkin (black), on 5th May 2001. White is to play the next move.

CLAMPAnalyser calculated a score for each piece move on the chessboard by combining the pieces to make chunks and then, by searching the chunk library, counting the frequency of each chunk in

each of the library files. On the configuration opposite white has fourteen pieces made up from six piece types (Rook, Queen, King, Pawn, Bishop and Knight). As CLAMPAnalyser has no knowledge of the rules of chess many of the moves, even though they are given a score, are illegal. Each piece will be given a score for a move to *each* of the sixty-four squares on the chessboard.

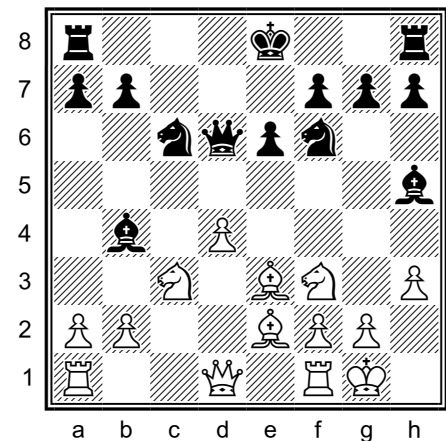


Figure 7.1:
Coats v Parkin 5th May 2001

¹⁸ The board taken from a tournament game can be represented by the Fen: r3k2r/pp3ppp/2nqpn2/7b/1b1P4/2N1BN1P/PP2BPP1/R2Q1RK1 w kq - 3 12

Using knowledge of how a chess piece can legally move on the board and considering the configuration above (Coats v Parkin), white can make one of forty-one possible moves. In order to compare the CLAMPAnalyser score with the score from a commercial chess program illegal moves have been removed from the output and the resulting moves are shown in the table 7.1 (below), with each move assigned the score output by CLAMPAnalyser after analysing the board for four-piece chunks. The score from CLAMPAnalyser is a count of the number of chunks that have been found in the library for the move to the positions shown. Against the CLAMPAnalyser score is shown the score produced by Fritz (a commercial chess program), based on an evaluation of the move and looking twelve-ply deep. The 'Fritz score' equates to the gain resulting from the move in terms of a pawn value.

The table below shows the CLAMPAnalyser score and Fritz score for each possible moves in the 'Coats v Parkin' chessboard.

MOVE FROM:	MOVE TO:	CLAMP SCORE	FRITZ SCORE		MOVE FROM:	MOVE TO:	CLAMP SCORE	FRITZ SCORE
Be2	Bd3	231657	-0.41		Nf3	Ng5	216931	-0.44
Be2	Bc4	212682	-0.37		Pa2	Pa3	417532	-0.06
Be2	Bb5	192263	-0.44		Pa2	Pa4	248241	-0.5
Be2	Ba6	85221	-3.34		Pb2	Pb3	222510	-0.47
Be3	Bg5	274531	-0.25		Pd4	Pd5	192147	-0.91
Be3	Bd2	205979	-0.31		Pg2	Pg3	152032	-0.47
Be3	Bf4	197600	-3.62		Pg2	Pg4	85885	-0.19
Be3	Bh6	129318	-3.03		Ph3	Ph4	38088	-0.62
Be3	Bc1	76306	-0.5		Qd1	Qb3	327807	-0.12
Kg1	Kh1	115648	-0.34		Qd1	Qc2	290939	-0.34
Nc3	Na4	329186	-0.28		Qd1	Qa4	288587	0
Nc3	Nb5	221909	-0.06		Qd1	Qd2	265918	-0.41
Nc3	Ne4	199718	-3.16		Qd1	Qc1	205312	-0.44
Nc3	Nd5	151850	-3.59		Qd1	Qe1	193843	-0.37
Nc3	Nb1	128618	-0.5		Qd1	Qb1	156168	-0.5
Nf3	Nh4	351078	-0.53		Qd1	Qd3	150672	-0.28
Nf3	Nd2	344666	-0.41		Ra1	Rc1	206116	-0.06
Nf3	Ne5	280526	-0.19		Ra1	Rb1	120512	-0.25
Nf3	Ne1	272470	-0.44		Rf1	Re1	191577	-0.25
Nf3	Nh2	244546	-0.53					

Table 7.1: – Move 'To' CLAMP and Fritz score comparison

A linear least squares fit between the score from CLAMPAnalyser and the score obtained by Fritz gave a correlation coefficient of 0.31. Although this is not a strong correlation¹⁹ the result shows *some* correlation between CLAMPAnalyser and Fritz move scores. The result is interesting because as previously stated, CLAMPAnalyser is not using any chess knowledge, for example CLAMPAnalyser has no knowledge if a move would result in the loss of a piece or be a disastrous tactical blunder, yet CLAMPAnalyser has produced a score for each move that correlates, albeit with a correlation coefficient of 0.31, with the score produced by a chess engine which uses conventional minimax methods to evaluate a move looking twelve-ply ahead.

The Coats v Parkin example is a typical chessboard configuration. However, to evaluate the effectiveness of CLAMP a simple test was performed on a number of chessboard configurations from a sample of mid-game positions. The boards were taken from tournament games between expert chess players, and from games that were *not* in the dataset used to build the library files. Inclusion of games that were used to build the libraries would distort the results as the board configuration and the move outcome will certainly match. The aim of the experiment is rather to test unfamiliar chessboard configurations to see if the ‘chunk composition’ of the board can be associated with the resulting move. The actual move made by the player resulting from the configuration was compared with the output from CLAMPAnalyser, with a ‘success’ attributed to the players move being in the top 50% of CLAMPAnalyser’s ordered list. The *null hypothesis* for the test is that CLAMPAnalyser’s list is *random* and CLAMPAnalyser will place the player’s move in equal proportions within the top and bottom 50% positions. If the move played is above the 50% point on CLAMPAnalyser’s list then the test will be assigned a

¹⁹ The correlation coefficient is an indication of the closeness of linear fit between two sets of variables. A coefficient of ‘one’ is a perfect fit whereas ‘zero’ is no fit at all.

'success', if the move is on the 50% point, or below, within CLAMPanalyser's list then the test is assigned 'failure'.

After analysing a number of configurations, the number of 'successes' can be divided by the number of boards tested to give a 'percentage success' figure. The percentage success figure can then be used to compare the effectiveness of different scenarios. A percentage success greater than 50% is an indication that the process is significant and not random; the higher the percentage the more significant the process, and the better the scenario is for 'predicting' which piece will be chosen by the player for the next move.

A percentage of 'success' figure for all of the test chessboards was calculated as follows:

$$\% \text{Success} = (\text{number of successes}) / (\text{number of boards tested})$$

In order to test CLAMPanalyser, a sample of chessboards that were used in the process of building the chunk libraries was analysed. This analysis is included because it is a standard step in machine learning systems, and was provided in order to validate the process.

A chunk size of four pieces was used for the analysis. The results are tabulated below:

Analysis using a chunk size of four pieces	Result
Number of pieces making a chunk	4
Number of chessboards analysed	250
Number of boards with one or more chunks on the board found within one or more chunk libraries:	241
Number of 'Successes' according to the hypothesis cf. page 86.	166
Number of 'failures'	84
Standard deviation of score results	64.5
Percentage success	66.4%

Results from the verification of CLAMPanalysis using a sample from the 'training' chessboards.

The results of the verification test show that in 66.4% of cases the chess player selected a move that was within the top 50% of the move list which was ordered in 'likelihood of being played' by the CLAMPAnalyser program. CLAMPAnalyser therefore produces a result that is better than a random selection

Having verified CLAMPAnalyser using a sample of chessboards from the training set further analysis was performed using chessboards that were not included within the training data. The results based on chunk sizes of two to five pieces using a sample of one thousand chessboards are listed as follows:

Chunk size	Number of Successes	Number of Failures	Number scored boards	%Success
2 piece	657	343	995	65.7%
3 piece	661	339	995	66.1%
4 piece	668	332	995	66.8%
5 piece	393	603	567	39.3%

Table 7.2: 'Percentage Success' comparison for the whole board area.

7.2. A few large chunks or many small chunks?

Suppose a chess expert moved a piece on a particular chessboard within a game and, at another time, exactly the same board configuration was seen in another game. Assuming the move made in the first instance was the best move we can also assume that exactly the same move would be chosen for the other game. If the chunk size were such to encompass all of the pieces on the chessboard then the configuration would be encapsulated into the one chunk. Very large chunks are therefore more influential in directing moves but at the same time, as the chunks enlarge in size they become increasingly rare. The likelihood of seeing exactly the same board configuration in the mid-game is unlikely as the number of possible games within the domain of chess is very large (Shannon 1950).

The approach taken with this research was to work with distinct chunk sizes (chunks consisting of the same number of pieces). The reason for this was to obtain insight into the effectiveness of chunks as the number of pieces that make up the chunk change. Chunks with a large size are *rare but specific* whereas small chunks are *frequent and general*. Therefore, if CLAMPanalyser is working with a library of small chunks we can expect to find a large number of chunks from the chessboard being associated with moves in the library files; the suggested moves that CLAMPanalyser generate being the consensus of scores from all chunks. On the other hand, when working with large chunk sizes, a smaller number of more influential chunks will be available.

There is therefore a 'trade-off' in performance as the chunk size increases. In some cases, when working with large chunk sizes CLAMPanalyser will be unable to find any of the chunks in the library, and will be unable to suggest moves. Table 7.2 (above) shows the number of boards scored with chunk sizes two to five. Chunks with two, three and four pieces had almost all of the boards successfully scored; however, when using a chunk size of five only 567 out of the one thousand boards (56.7%) were successfully scored.

Although a full analysis of chunk sizes above five pieces was not possible due to the computational complexities in processing, indications were that the number of boards scored continued to decline as the chunk size increased above five pieces.

7.3. Adjusting for 'move rareness'

Some moves in chess occur more frequently than others. A piece may, for example often move to the centre positions of the board and during the course of several games a piece may move to the same position numerous times. The actual

distribution of move frequencies are systematically presented in table 6.1 on page 59. On inspection of table 6.1, the arrival of a Knight on square 'D6' is relatively common such that to find one thousand instances of this move 499,105 chessboards were required, whereas the arrival of the Knight on H8 is rare, requiring 23,366,342 different chessboards to find one thousand instances of this occurrence. A move of a piece to a square can therefore be accompanied by a 'rareness' figure.

CLAMP ignores the 'rareness' of moves when building the chunk libraries. Each library file was compiled using a collection consisting of one thousand arrivals of each piece on each square. Each move, when building the chunk library, is presented to CLAMP in equal significance with no information as to how frequently the move occurs in the game of chess. To compensate for this, when CLAMPAnalysers is calculating the likelihood of a move from a chessboard, an adjustment for the 'rareness of a move' can be applied to the score from CLAMPAnalysers by dividing the chunk count by a factor of the 'move rareness' figure.

Unlikely moves have a high 'rareness' value. If CLAMPAnalysers suggests a move that is rare, dividing the 'move score' by the rareness figure decreases the significance of the move. Dividing the number of chunks associated with each move by the 'move rareness' figure improves the score by a small percentage. The adjusted scores for two, three, four and five piece chunks are shown in the table below:

Chunk size	%Success	%Success with 'move rareness' adjustment
2 piece	65.7%	69.7%
3 piece	66.1%	70.1%
4 piece	66.8%	69.7%
5 piece	39.3%	39.6%

Table 7.3: Percentage Success comparison for the whole board area.

Unless otherwise stated, the move rareness adjustment is included as part of the success score calculations in the results in this thesis from this point forward.

7.4. Changing the ‘success’ threshold

Comparing CLAMPanalyser’s output to test if the chosen move appears within the top 50% of CLAMPanalyser’s ordered list will test to see if CLAMP’s move ordering is significant compared to a random ordered list. The results displayed in table 7.2 (cf. page 88) show that for chunks sizes two, three and four CLAMP performs better than a random ordering. The following table shows the results from an analysis of one thousand chessboards with the ‘success’ threshold set between 40% and 90% and with move rareness adjustment applied to the results.

Chunk Size	2	3	4	5
40% threshold	77.7%	78.9%	79.5%	45.1%
50% threshold	69.7%	70.1%	69.7%	39.6%
60% threshold	62.3%	61.6%	59.9%	34.9%
70% threshold	52.3%	50.5%	49.6%	28.7%
80% threshold	28.9%	28.3%	28.8%	19.5%
90% threshold	15.1%	15.5%	15.2%	10.2%

Table 7.4: Percentage success figures with various thresholds

The percentage success against threshold data are plotted below:

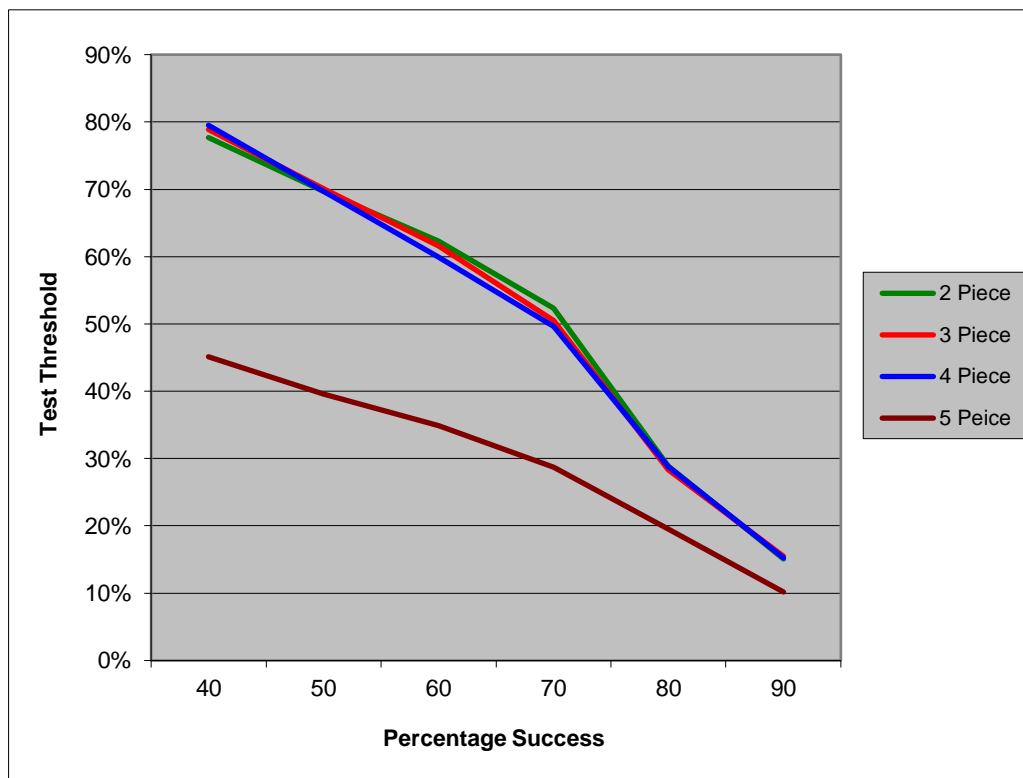


Figure 7.2: Effect of change in threshold on percentage success²⁰.

Figure 7.2 illustrates that similar results are obtained for each chunk size (with the exception of chunk size five, for the reasons given above) across the range of 'success thresholds'.

Unless stated otherwise, a success threshold of 50% is assumed for all results documented in this thesis.

7.5. The relationship between pieces within chunks

The analysis so far has focused on chunks that have been extracted from the whole area of the chessboard. To explore the properties of chunks, chunks were extracted

²⁰ Five piece chunks do not perform as well as other chunk sizes shown in figure 7.2 because five piece chunks score only 56.7% of the boards in the sample.

from the board using a variety of methods. For each extraction method CLAMP produced a chunk library. Using each library in turn, the comparison between methods was made using a large number of test chessboards, in the mid-game section of tournament games between experts or Master players. One thousand test chessboards were scored in each case. By using a large sample of games as opposed to considering a single chessboard, a more statistically reliable result was achieved.

Three different methods were used to refine the compilation of chunks. The first method combined all of the chess pieces from the entire board. This is a systematic analysis of the board and assumes no knowledge of the rules of the game. The second method combines pieces from a subset of the board so that pieces within the chunk are restricted to being in close proximity. With this method all pieces on the board are combined but, using a visual analogy, the chessboard was viewed through a sliding window that moved over the entire board. The analysis of a subset of the board assumes that the pieces that combine to make effective chunks exist in close proximity.

The third method was to only combine pieces that are in defending relationships with each other. This method requires knowledge of how pieces move and the significance of the colour of the pieces. Pieces on the board that are *not* 'protected' by a piece of the same colour are ignored. Pieces on the chessboard are combined to make chunks with the caveat that the pieces within the chunks are in defending relationships with each other - they have the property that they are defending another piece within the same chunk.

For each of the above methods the size of the chunk, or in other words, the number of pieces being combined to make the chunks was varied. For each scenario CLAMP produced a chunk library and the library was used to analyse a number of

test boards. A figure was obtained for the effectiveness of the chunk library at predicting the next piece to be played for each of the test boards. The aim of this analysis was to find the most effective method and chunk size, for selecting the piece to be moved.

7.5.1 The number of chunks in a chunk library

Table 7.5 (below) shows results for the analysis when using chunk sizes of two, three, four and five pieces. The results show a chunk size of three being slightly better than a chunk size of two and four. The number of chunks within the chunk library *increases* with the chunk size. The reason for this is, as the chunk size becomes larger the number of combinations of pieces making up the chunk also increase, and therefore, when building the chunk library a larger number of chunks will be stored in the library. However, when analysing a chessboard as the chunk size increases, the number of chunks matched within the library *decreases* because the chunks become more specialised. The chance of finding a large chunk is less likely as that chunk may not have been seen in the sample of chessboards that made the collection during the process to build the library.

7.5.2 The 'effectiveness' of a chunk

Table 7.5 (below) shows a low 'success' score for the chunk size 'five.' The low score is largely due to the absence of matching chunks from the test boards being present within the five-piece chunk library. The number of successes is limited to the number of boards that match at least one chunk from the chunk library, and in the five-piece case shown in the table only 567 boards out of the one thousand samples achieved this. If however CLAMP had constructed the libraries using a larger collection of

moves then the number of scored boards in this analysis would increase accordingly. Therefore, a more appropriate measure of the *effectiveness* of a chunk should be based on the score *divided* by the number of scored boards. In this thesis this measure is referred to as the ‘*effect*’.

In the results reported in this section the ‘effect’ value generally increases with increasing chunk size, which is consistent with section 7.2 (above). In practical situations however, when analysing a single chessboard, it may not be possible to match any chunks from the chessboard to chunks in the chunk library, particularly when using large chunk sizes. A practical example of how chunking with large chunk sizes and the ‘number of boards scored’ figure can be used with confidence when analysing just one chessboard is described in section 7.11 (cf. page 147).

7.5.3 Analysis of the whole board area.

Analysis of the whole board for piece configurations that frequently exist prior to a piece move (cf. page 88) was performed considering chunks sizes of 2,3,4 and 5 pieces with move rareness applied and a ‘50% success’ threshold. The results are summarised below:

Chunk size	Chunks In Library	Number of Successes	Number of Failures	Number scored boards	% Success	% Effect	%Standard Error
2 piece	1261864	697	303	995	69.7%	70.1%	6.27%
3 piece	7596060	701	299	995	70.1%	70.5%	7.11%
4 piece	27127049	697	303	995	69.7%	70.1%	7.88%
5 piece	45646773	396	604	567	39.6%	69.8%	10.72%

Table 7.5: Percentage success comparison for the whole board area.

Note: The values shown in table 7.5 are taken from results presented in tables 7.2 and 7.3. The columns in table 7.5 are as follows:

Chunk size.	The number of pieces that make up a chunk.
Chunks In Library	The number of chunks that make up the library.
Number of Successes	The number of instances where the actual move taken was one of the pieces in the highest 50% CLAMP score values.
Number of Failures	The number of instances where the actual move taken was not one of the pieces in the highest 50% CLAMP score values.
Number scored boards	The number of boards that were successfully scored. A 'scored board' is a board that has at least one chunk associated with a move.
% Success	The number of successes divided by the number of sample boards.
% Effect	The number of successes divided by the number of scored boards.
Standard Error	The Standard Error shown in the above table shows the adjustment, plus or minus, and applied to the 'percentage effect'. The Standard Error is reported with all similar results in this thesis and denoted by 'SE'.

Table 7.6: An explanation of the columns shown in table 7.5

The following table shows the percentage success with 95% confidence limits. The table, and the graph below, shows that the percentage success is comfortably in excess of two standard deviations of the 50% null hypothesis for chunk sizes 2,3,4 and five pieces.

Chunk Size	% effect (mean)	% Standard Error	low 95% confidence limit	high 95% confidence limit
2 piece	70.1	6.27	63.83	76.37
3 piece	70.5	7.11	63.39	77.61
4 piece	70.1	7.88	62.22	77.98
5 piece	69.8	10.72	59.08	80.52

Table 7.7: Chunk 'effect' with 95% confidence limits.

It is noted that a chunk size of five pieces show a small decrease in the mean compared to sizes of two, three and four pieces. The decrease is attributed to the

reduction in the number of large (five piece) chunks found on the chessboards that are frequently occurring within the training data (cf. page 88).

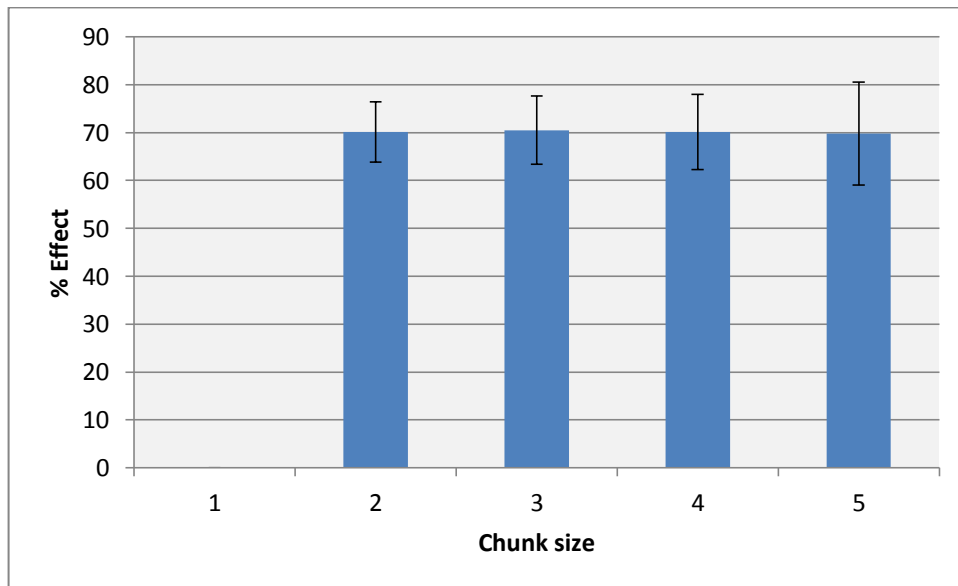


Figure 7.3: Percentage success comparison for the whole board area. (The error bars represent a 95% confidence interval).

7.5.4 Chunks and meaning

CLAMP's criterion for recognition of a chunk, when considering the whole board, is based simply on the frequency of occurrence of piece constellations. A human player is likely to be a lot more discriminating in his recognition of chunks and as a result may remember fewer configurations, but the configurations may have a greater significance. An example of being more discriminating could be by only selecting chunks that are related in other ways, such as chunks with pieces that are in close proximity, or chunks with pieces that are in attacking or defending relationships with each other. The thesis continues with an analysis of chunks that includes pieces in close proximity and pieces that are in defending relationships.

7.5.5 Analysis of chunks in small grouped areas on the chessboard

Papers by Gobet and Jansen (1994) and also Walczak (1992) report work done on analysis of boards with chess pieces that are in close proximity to each other (cf.

Page 26). Papers by Simon and Gilmartin (1973) and also Gobet and de Groot (1996) with respect to eye movement show that expert chess play conforms to this notion and in addition the player examines attacking and defending positions.

Limiting the view to a small area on the board considerably reduces the number of possible combinations of pieces. The original analysis for the whole board area was repeated but limiting the scope of a chunk to, for example, a 4x4 square area on the board. In this mode all combinations of pieces were obtained but only pieces that were located within an area of sixteen squares were significant. The chessboard was viewed with a 4x4 window, which scanned over the board and pieces within the 'window' at each juncture were combined to make chunks. Considerably fewer chunks were found with this limitation. Windows sizes of 3x3, 4x4, 5x5, 6x6 and 7x7 squares were processed and results compared.

The results are tabulated below. The points of interest are the positions of maximum percentage success. Areas start with the 3x3 size but where the percentage success shows a decline the analysis was stopped. Uncalculated scenarios are shown with x'.

Local areas	Chunks In Library	Number of Successes	Number of Failures	%Success	%Boards scored	Effect	SE
local 3x3 Size 2	114943	579	421	57.9%	85.1%	68.4%	7.16%
local 3x3 Size 3	80819	579	421	57.9%	82.0%	70.6%	9.21%
local 3x3 Size 4	23855	424	576	42.4%	54.2%	78.2%	11.38%
local 3x3 Size 5	3944	205	795	20.5%	22.3%	91.9%	20.01%
local 3x3 Size 6	x	x	x	x	x	x	x
local 3x3 Size 7	x	x	x	x	x	x	x
local 4x4 Size 2	253476	577	423	57.7%	87.2%	66.2%	7.81%
local 4x4 Size 3	343409	590	410	59.0%	84.0%	70.2%	8.87%
local 4x4 Size 4	210120	521	479	52.1%	68.2%	76.4%	11.39%
local 4x4 Size 5	85679	393	607	39.3%	49.9%	78.8%	12.19%
local 4x4 Size 6	26139	186	814	18.6%	21.0%	88.6%	17.17%
local 4x4 Size 7	x	x	x	x	x	x	x
local 5x5 Size 2	453492	639	361	63.9%	90.5%	70.6%	7.16%
local 5x5 Size 3	998254	590	410	59.0%	84.9%	69.5%	8.62%
local 5x5 Size 4	1057802	575	425	57.5%	78.6%	73.2%	9.33%
local 5x5 Size 5	726212	481	519	48.1%	65.6%	73.3%	9.57%
local 5x5 Size 6	388865	390	610	39.0%	51.8%	75.3%	12.07%
local 5x5 Size 7	x	x	x	x	x	x	x
local 6x6 Size 2	721550	585	415	58.5%	87.5%	58.5%	7.25%
local 6x6 Size 3	2593877	611	389	61.1%	88.5%	71.5%	8.34%
local 6x6 Size 4	4918042	587	413	58.7%	84.3%	69.6%	8.33%
local 6x6 Size 5	6179518	535	465	53.5%	76.3%	70.1%	8.95%
local 6x6 Size 6	4621214	390	610	39.0%	54.9%	71.0%	10.78%
local 6x6 Size 7	x	x	x	x	X	x	x

Table 7.8: Analysis of chunks in small local areas

The graph below shows a comparison of the percentage effect figures against chunk size, for each board area listed above. The graph shows each area achieving a *percentage effect* increasing as the chunk size increases.

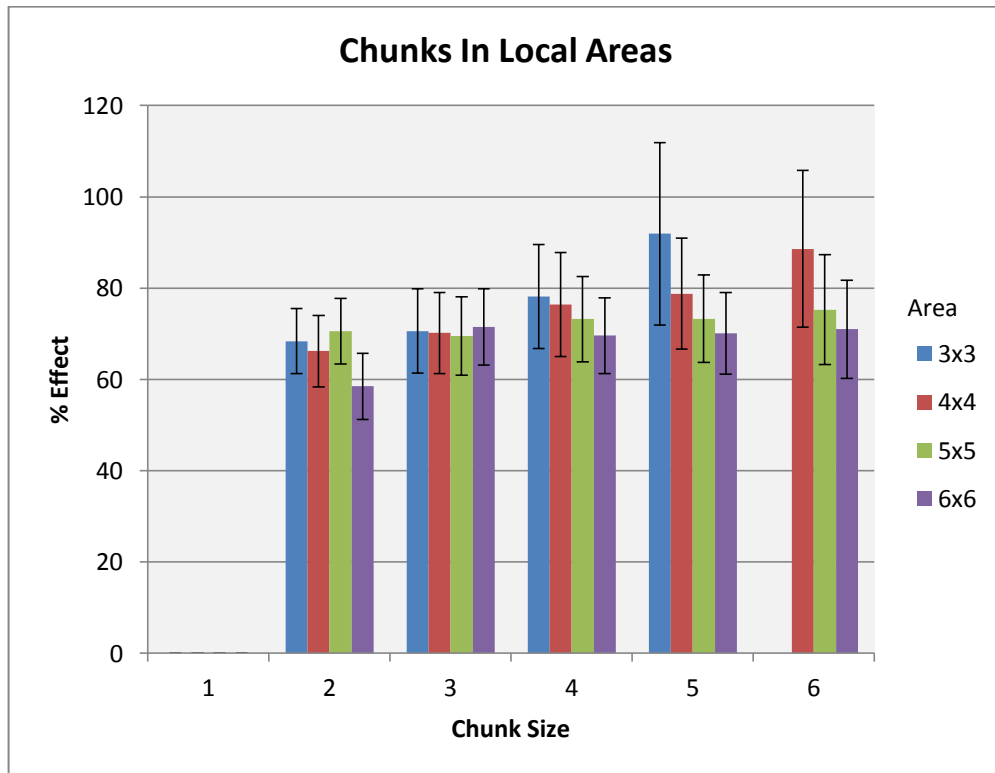


Figure 7.4: A comparison of small area analysis and chunk size (The error bars represent a 95% confidence interval).

The results shown in table 7.8 show that as the *percentage effect* increases with increasing chunk size the number of chunks found within the chunk library (the percentage success) decreases. This result is consistent with the argument given in paragraph 7.5.1. It can also be seen from table 7.8 that the number of chunks found (the number of successes) in small local areas on the chessboard increases with a larger board area. Walczak (1992) reported in an analysis of chessboards that “most chess patterns are contained within a sixteen squares area on the board. An analysis of eighty games from a former world chess champion eighty-six chunks were acquired within a 4x4 area. Increasing to 5x5 resulted in just one extra chunk, the

chunk being one of the existing eighty-six chunks plus two additional pieces”.

However, the analysis reported in this thesis, which is based on a much larger sample of games showed a significantly higher number of chunks present in a 5x5 as opposed to a 4x4 square area.

The *percentage effect* when restricting chunks to a small area on the board is comparable to results obtained from an analysis of the whole board area, however, the number of chunks in the respective chunk libraries can be dramatically different (cf. table 7.5 and 7.8). When analysing a chessboard, CLAMPanalyser compiles all chunks for the whole of the board under test, the difference in each method is the choice of chunk library that is searched. The ‘local area’ libraries contain a smaller number of chunks than those contained in the ‘whole board’ libraries, for example, for a chunk size of four pieces the whole board library has 27,127,049 chunks giving a percentage effect of 70.1%, whereas when limiting the pieces with chunks to a 5x5 area reduces the percentage effect to just 69.0% but the number of chunks in the corresponding library reduce to 1,057,802 (a reduction to approximately 4% of the original size).

As all of the 1,057,802 chunks within a local area library are contained in the ‘whole board’ library by virtue of the systematic way the libraries are constructed, it is possible that the chunks within the 5x5 local area chunk library make up a high proportion of the chunks that are found when searching the whole board libraries. The chunks contained in the ‘local area’ libraries can therefore be considered to be ‘more salient’ as comparable results are obtained to the ‘whole board’ libraries despite having considerably fewer chunks within the libraries.

It is therefore possible that a property of many of the chunks that are significant (in that they can be indicators of a move to make) within the game of chess is that they are composed of pieces that are in close proximity to each other. A

chunk library that has been built with the addition of this knowledge is therefore smaller as it contains fewer chunks, but the chunks that are contained are more salient. The theoretical implications of this result, on a computer system, are that smaller chunk libraries can be produced requiring less storage space and enabling a faster searching through the library to find specific chunks. The result also shows the implication of additional knowledge (that effective chunks normally consist of pieces in close proximity) can produce a result where a higher number of chunks are meaningful in terms of moves made.

It can be seen from figure 7.3 that a chunk size of four pieces within a 3x3 square archives the highest percentage success for the scenarios shown on the chart. The mean percentage success for this mode is 78.2% and with a standard error of 11.4%,

7.5.6 Analysis of chunks comprising of pieces in defensive relationships

An analysis of configurations that include only pieces that were in defensive relationships with each other (cf. page 26) was performed and the results compared with results from the 'whole board' and 'local area' analysis. 'Defensive relationships' with respect to the pieces within a chunk are defined such that each piece within the chunk is protecting another piece within the same chunk (cf. page 46). If the opponent were to take the 'protected' piece then the opponent could in turn be taken by the 'protecting' piece. In practice many of the pieces on the chessboard are arranged in clusters so that each piece protects another. The rationale behind analysing chunks in defensive relationships is based on papers by Wilkins (1980) "Human Masters, whose play is still much better than the best programs, appear to use a knowledge intensive approach to chess. They seem to have a huge number of

stored 'patterns' and analysing a position involves matching those patterns to suggested plans for attack or defence". Experiments with chess players by McGregor and Howes (2002) propose that the attack/defence relationship of pieces is more significant to skilled chess players than the size of the chunk on the board. McGregor and Howes (op. cit.) argue against the notion that proximity of pieces is a factor in chunk structure and by experiments on chess player's ability to remember configurations, gives evidence in favour of the relationship of pieces being the main factor in chunk selection. CLAMP was modified to build chunk libraries based on boards that consisted only of pieces in defending relationships to each other; the analysis ignored all 'passive' pieces on the board, that is, within a chunk, any pieces that did not defend another piece were ignored. Chunk patterns with two, three, four, five, six and seven pieces in defensive relationships were analysed.

It should be noted that the recognition of defensive chunks in a board being examined is only approximate as it fails to take account of the fact that some or all defence relationships in the chunks could be suppressed by intervening pieces.

In order for CLAMP to extract chunks that contained pieces in defensive relationships, some knowledge of how pieces move and their scope, was required, for example, a bishop requires a diagonal which is not blocked by a piece between itself and the piece it is defending, however, the knight can jump over an obstructing piece. This knowledge was required when building the chunk library so that only pieces in defensive relationships were processed and ultimately stored in the library, however, when *using* the libraries to analyse a board CLAMP analyser simply combines all pieces on the whole chessboard, without any knowledge of chess semantics, and searches for the chunks within the library. Knowledge of how a piece defends another is therefore not required when analysing a chessboard.

The results of the analysis of defending chunks are tabulated below:

Chunk size (pieces)	Chunks In Library	Number of Successes	Number of Failures	Number of boards scored	%Success	% Effect	SE
2	44266	696	304	995	69.6%	69.9%	6.38%
3	45096	644	356	954	64.4%	67.5%	7.04%
4	119857	666	334	965	66.6%	69.0%	7.57%
5	147510	593	407	825	59.3%	71.9%	8.54%
6	158275	538	462	714	53.8%	75.4%	9.32%
7	129313	338	662	427	33.8%	79.2%	12.07%

Table 7.9: A comparison of chunk library size and percentage success (defensive chunks)

The results are shown in the graph below:

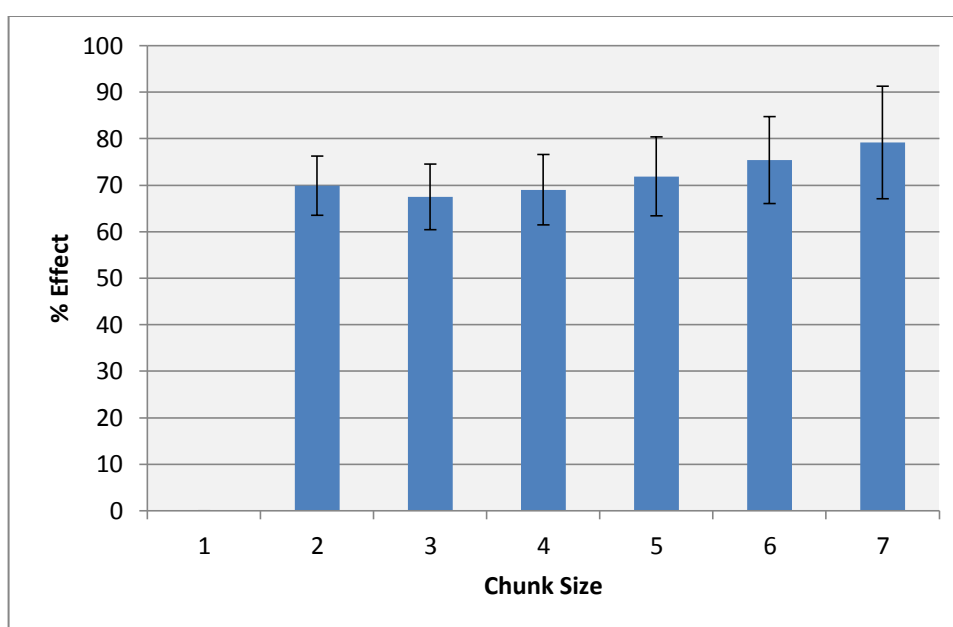


Figure 7.5: Percentage Success using 'defensive chunks' (The error bars represent a 95% confidence interval).

Using the same argument as used for chunks in local areas in paragraph 7.5.5, all of the chunks in the defensive libraries are contained within the 'all board' libraries, however, the percentage success figure is comparable between the two methods, moreover, the number of chunks when using pieces in defending relationships reduce to less than 0.5% of the number of chunks in the whole board library (the number of chunks in the library for whole board, four piece chunks is 27,127,049,

whereas the number of chunks in the four piece defending library is 119,857, the defending chunk library is 0.44% of the size of the library for the whole board).

It is therefore possible that a property of many of the chunks that are significant within the game of chess is that they are composed of pieces that are in defensive relationships with each other. It is also possible as the percentage success figures for the 'local area' and the 'defensive' methods give similar results so many of the defensive chunks have pieces that are in close proximity and are therefore also present within the 'local area' libraries.

A comparison of the 'percentage effect' for each chunk size was made for chunks extracted from the whole board and chunks extracted from pieces restricted to defending relationships. The results are shown in the graph below:

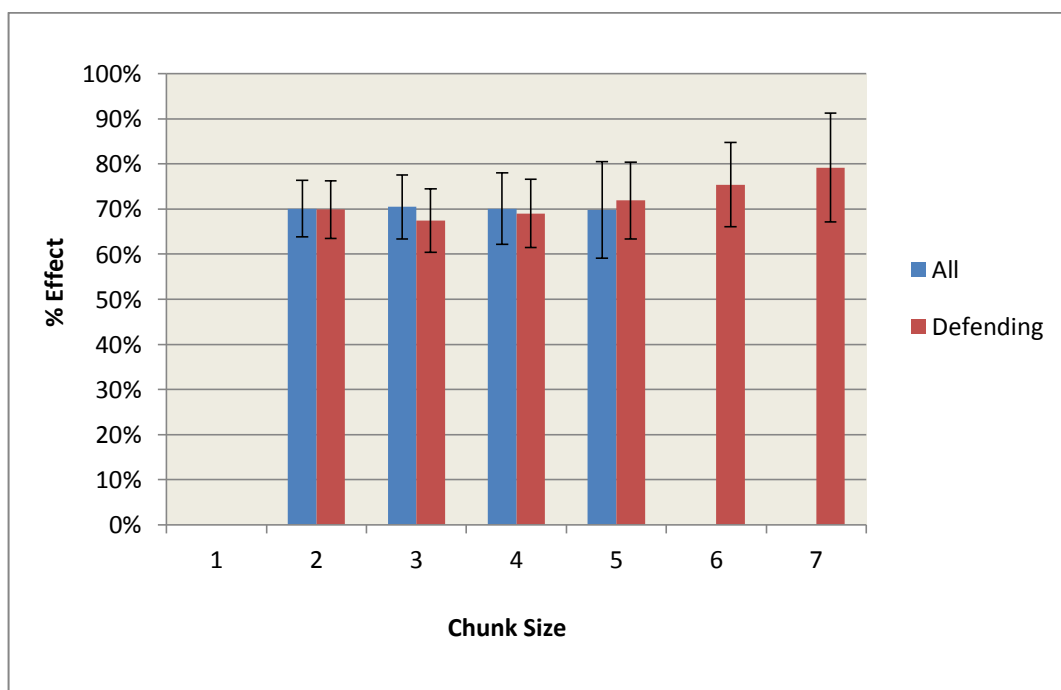


Figure 7.6: A comparison between 'defending' and 'all board' methods (The error bars represent a 95% confidence interval).

The following table compares the chunk libraries for three methods that give a similar percentage success figure:

Chunk Type	Chunks In Library	Number of Successes	Number of Failures	Percentage of boards scored	Standard Deviation	%Success Including no-score boards	Effect
Whole Board: Size 4	27127049	697	303	99.5%	26.4%	69.7%	70.1%
local 5x5: Size 4	1057802	666	334	96.5%	23.1%	66.6%	69.0%
Defending: Size 4	119857	666	334	96.5%	26.0%	66.6%	69.0%

Table 7.10: A comparison of three methods using four piece chunks

The results presented in this section compare the effectiveness of methods. When analysing the whole board area all pieces are combined. The ‘whole board’ analysis is therefore comprehensive, including all chunks on the chessboard. The number of chunks found is therefore larger than the ‘local group’ and ‘defensive’ methods, which restrict the pieces that can be included within chunks. The whole board ‘method’ may also contain a large number of chunks that are not relevant, or significant, to the move to be played. Local group and defensive analysis do not capture all of the significant chunks, but still achieve a high percentage success figure because the ‘local group’ and ‘defensive’ libraries have a high percentage of significant chunks. ‘Defensive’ and ‘local’ chunks can therefore archive good results when suggesting a move to be played even though libraries consisting of a small number of the total chunks that are present within the ‘whole board’ chunk library.

7.6. Changing the ‘success’ threshold with defensive chunks

Setting the ‘success’ threshold (or the ‘null hypothesis’ point (cf. page 86)) so that if the actual move taken appears within the top 50% of CLAMPanalyser’s ordered list

will test CLAMP against a random ordering, whereas, changing the threshold from 50% to other values can be used to compare the methods employed by CLAMP. The following table shows the results from an analysis of one thousand chessboards, using defensive chunks of various sizes, with the 'success' threshold set between 40% and 90%.

Chunk size:	3	SE	4	SE	5	SE	6	SE	7	SE
40% threshold	73.4%	6.38%	7.04%	7.57%	67.3%	8.54%	59.5%	9.32%	37.1%	12.07%
50% threshold	64.4%	6.38%	66.6%	7.57%	59.3%	8.54%	53.8%	9.32%	33.8%	12.07%
60% threshold	56.1%	6.38%	56.8%	7.57%	50.8%	8.54%	46.4%	9.32%	29.3%	12.07%
70% threshold	46.5%	6.38%	46.1%	7.57%	41.4%	8.54%	37.6%	9.32%	24.0%	12.07%
80% threshold	28.0%	6.38%	28.1%	7.57%	25.4%	8.54%	22.9%	9.32%	13.9 %	12.07%
90% threshold	12.3%	6.38%	12.9%	7.57%	12.4%	8.54%	11.9%	9.32%	7.6%	12.07%
% boards scored	95.4%	-	96.5%	-	82.5%	-	71.4%	-	42.7%	-

Table 7.11: Percentage success with varying 'success threshold' for defensive chunks

The results are displayed on the graph below:

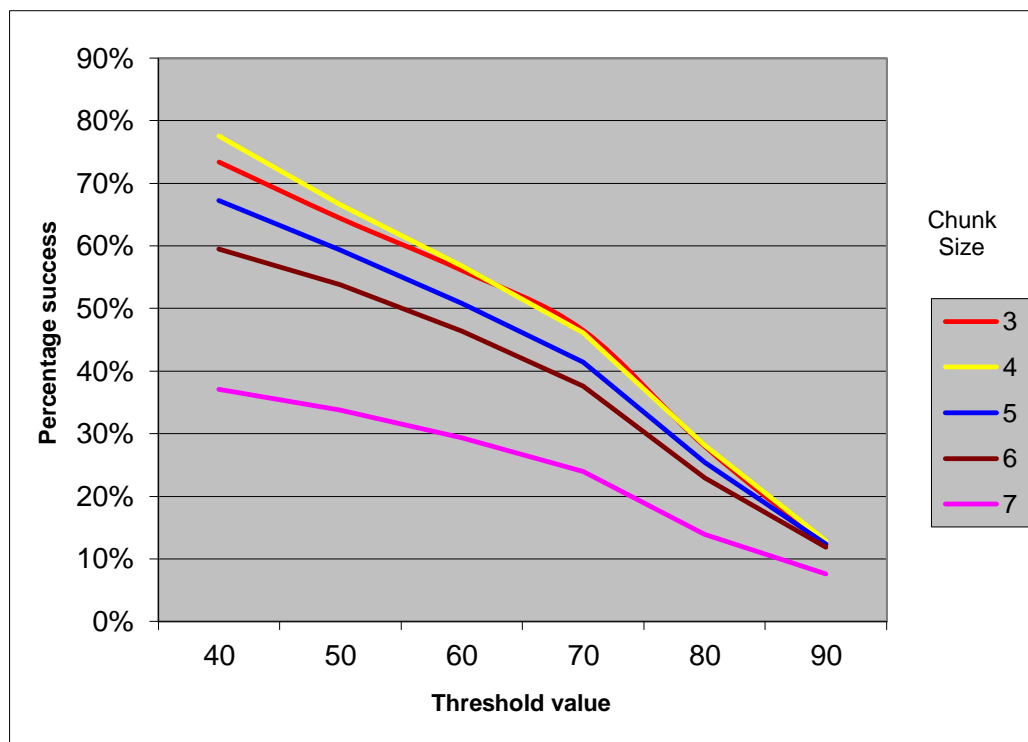


Figure 7.7: Graph of percentage success with varying threshold settings (defensive chunks).

The results displayed in figure 7.7 show, when the threshold value is low, a varying percentage success result with differences in chunk size. A chunk size of *four pieces* gave the highest percentage success when the threshold values are less than 60%. Higher threshold values yielded a lower percentage success, and less of a distinction between chunk sizes.

7.7. An analysis of de Groot's Position 'A'

Adriaan de Groot presented the chess configuration which has become known as 'de Groot's Position A' to five Grandmaster players, including Alekhine, Keres, Euwe and Flohr, and also to other expert players at the 1938 AVRO tournament²¹.

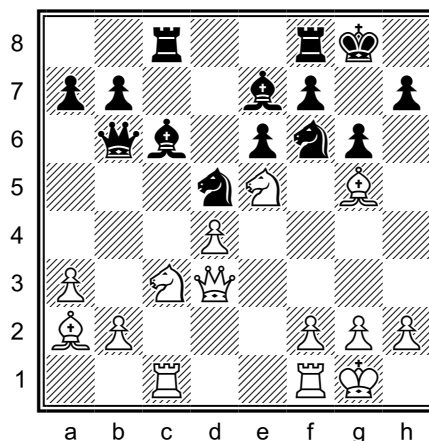


Figure 7.8: de Groot Position 'A'

All Grandmasters except Flohr chose the move: **Bxd5**. Move **Nxc6** was also considered and was thought to be equally strong by Alekhine and was preferred by Flohr. Other players chose weaker, but safe moves: **Rfe1** or **Bh6**.

An analysis of the chessboard by CLAMPanalyser (an analysis of the 'Position A' configuration) produced a list of moves in order of the number of chunks that

²¹ The 'Position A' FEN is: 2r2rk1/pp2bp1p/1qb1pnp1/3nN1B1/3P4/P1NQ4/BP3PPP/2R2RK1 w - - 0 1

support the move to be played, based on the four-piece chunks found on the whole board. The move **Bxd5** did not score well at position forty-ninth on the list of moves out of possible fifty-seven moves. Move **Nxc6** fared better at position twenty, and **Bh6** was at position twenty-four. The 'safe' move **Rfe1** however, was positioned second.

The safe moves, and in particular **Rfe1**, are therefore frequently occurring moves for similar piece configurations, whereas **Bxd5** is probably a rare tactical move which is not normally seen with pieces similar to this configuration. However, it should be noted that the Grandmasters were able to find the move **Bxd5** within a short space of time suggesting that chunking could be present within the Grandmasters thinking process for this move, but clearly, CLAMP's chunking process is very simplistic when compared with human cognition.

The table below shows the 'Score' from CLAMPAnalyser for each move. Column 'F' shows the order in which the Fritz chess program (searching to a depth of twelve ply) assigns moves and is shown for comparison with the CLAMPAnalyser ordering.²²

From	To	Score	F	From	To	Score	F	From	To	Score	F
Rf1	Rd1	693558	4	Ne5	Nc6	257060	3	Qd3	Qb5	157609	
Rf1	Re1	685485	2	Ba2	Bc4	256227		Ne5	Ng6	157215	
Nc3	Na4	522029		Ne5	Nc4	254717		Qd3	Qd1	155330	
Qd3	Qc2	407274		Bg5	Bf6	252325		Qd3	Qf3	148540	
Ph2	Ph3	395134	5	Bg5	Bh6	248520		Qd3	Qh3	109353	
Ne5	Nd7	362237		Pf2	Pf4	244847		Ne5	Ng4	109217	
Qd3	Qe2	349526	6	Bg5	Bd2	244450		Rc1	Ra1	105196	
Pa3	Pa4	337046		Qd3	Qb1	240452		Qd3	Qa6	104852	
Pd4	Pd5	330458		Pf2	Pf3	225081	9	Qd3	Qg3	99011	
Nc3	Ne6	327255		Nc3	Nd1	223866		Qd3	Qe4	93798	
Ba2	Bb1	321210		Nc3	Nb1	209507		Ba2	Bd5	89228	1
Nc3	Nb5	317700		Bg5	Bf4	208938		Pg2	Pg4	87402	
Pb2	Pb4	316133		Pg2	Pg3	208047		Qd3	Qe3	86611	
Qd3	Qd2	309224		Rc1	Rb1	204788	8	Ph2	Ph4	72047	7
Bg5	Be3	306666		Kg1	Kh1	186945		Ne5	Nf7	64672	
Pb2	Pb3	289354		Ne5	Nf3	175799		Qd3	Qf5	56125	
Nc3	Nd5	283839		Ba2	Bb3	175405		Qd3	Qg6	15466	
Bg5	Bh4	282060		Qd3	Qc4	167558					
Nc3	Ne2	258413		Rc1	Rc2	160796	10				

**Table 7.12: de Groot Position 'A' move scores
(compiled using the *four-piece whole board* chunk library)**

Comparing the list of moves ordered by the CLAMP score, and the order of moves output by Fritz, CLAMP successfully identifies a number of good moves within the top seven positions of the CLAMPAnalyser list. The top seven moves 'identified' by CLAMP include Rfe1, Rfd1, h3 and Qe2, all of which are not unreasonable. Table 7.13 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '57' being the worst move.

²² Results were obtained by the Chess program 'Fritz version 10' using the 'explain all moves' feature. Analysis was to a depth of 10 ply.

From	To	Fritz move preference
Rf1	Rd1	4
Rf1	Re1	2
Nc3	Na4	47
Qd3	Qc2	30
Ph2	Ph3	5

Table 7.13

The moves associated with the top five CLAMP scores three out of the moves compare favourably with the analysis of the same configuration by the Fritz chess engine. The exceptions are the move to Na4 which, although this move is supported by a high proportion of chunks the move is tactically poor as it could result in the loss of the knight, similarly the move Qc2 could result in the loss of the pawn on 'd4'.

The analysis of 'Position A' by CLAMP is included in this thesis for general interest although few conclusions can be drawn from the performance of CLAMP on one chessboard considered in isolation.

7.8. The Bratko/Kopec tests

The Bratko/Kopec test consists of twenty-four chessboard configurations designed by Dr. Ivan Bratko and Dr. Danny Kopec in 1982 to test a player's ability at the game of chess. The tests are used to rate a player's knowledge of chess and in addition have been used to test the power of computer chess programs. The tests consist of a chessboard configuration and the corresponding 'best' moves for each of the twenty-four tests. The moves are classed as either a 'tactical move' or a 'pawn lever' type move.

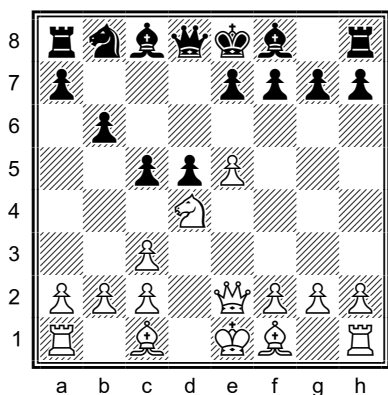
A 'pawn lever' move is a move by a pawn that ultimately damages the opponent's pawn structure by capturing an opposing pawn. The player's pawn is put in a position where it can be taken by the opponent and so the pawn is often sacrificed. In some instances the player's pawn structure may also be improved, as a result of a pawn lever move.

CLAMP does not perform very well with many of the Bratko/Kopec tests. The poor performance is believed to be partly due to the tactical nature of many of the 'best moves', as 'tactical moves' normally require knowledge of the game of chess - which CLAMP does not possess. Tactical moves are generally not related to chess pieces being in frequently occurring positions, as is the case with positional moves, but normally require knowledge of the semantics of the game. Tactical moves tend to be more rare and less repeatable than positional moves.

The results below show only the Bratko/Kopec tests where *white* is to play (a total of twelve tests) as the libraries compiled for this research work were for 'white to move' only configurations. CLAMPanalyser used the library with four-piece chunks 'whole board area' for this analysis. In the twelve tests only one scenario has the 'best move' within the top four of CLAMPanalyser's rank-ordered list of suggested moves. If however the test configurations are analysed by the Fritz chess engine

then it can be seen that many of the top ordered moves are highly rated by Fritz. In five out of the seven pawn-lever tests CLAMPAnalysers suggested within its top four choices moves that are within Fritz top four moves. In two out of five tactical tests CLAMP suggested within its top four choices moves that are within Fritz top four moves. The fact that CLAMP performed slightly better with ordering pawn lever configurations as opposed to tactical configurations is consistent with chunking theory. The results of each test are reported in the following pages.

7.8.1. Bratko/Kopec Test 4 (best move: pawn lever)



Result: The ‘best move’ (marked with ‘<’ alongside in the table below) for ‘test 4’ according to the Bratko/Kopec test is rated the 38th choice out of a possible thirty-eight scored positions, however the sixth choice in CLAMP’s rank order (Nf3) is actually the second highest scoring move (shown with ‘2’ alongside) when the chessboard is analysed by Fritz.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 4. Column ‘F’ shows the ordering of the move assigned by Fritz.

<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>
Bc1	Bg5	474395		Pg2	Pg3	264308		Qe2	Qg4	152290	
Bc1	Bd2	434346		Qe2	Qd2	254985		Pb2	Pb4	147631	
Pa2	Pa3	398470		Qe2	Qh5	247290	5	Ra1	Rb1	146494	
Bc1	Be3	369474		Ke1	Kd1	244264		Pf2	Pf4	145460	
Bc1	Bf4	367386		Bc1	Bh6	212982		Ph2	Ph4	121797	
Nd4	Nf3	364672	2	Nd4	Nc6	205980		Qe2	Qd1	117279	
Ph2	Ph3	309563		Pf2	Pf3	205363		Rh1	Rg1	104693	
Pb2	Pb3	304146		Nd4	Nf5	204345		Qe2	Qa6	101879	
Pa2	Pa4	298771		Qe2	Qd3	201949		Qe2	Qe3	96046	
Pc3	Pc4	297073		Qe2	Qb5	195325		Ke1	Kd2	78482	
Nd4	Nb3	291897	3	Qe2	Qc4	177933		Pg2	Pg4	76638	
Nd4	Nb5	284993	4	Nd4	Ne6	164475		Pe5	Pe6	70692	<
Qe2	Qf3	277261		Qe2	Qe4	164306					

Table 7.14: Bratko/Kopec Test 4 CLAMP Scores.

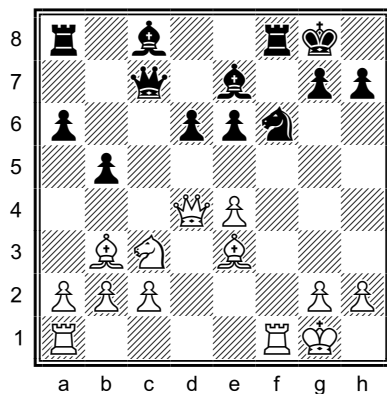
Table 7.15 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '38' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Bc1	Bg5	8
Bc1	Bd2	13
Pa2	Pa3	19
Bc1	Be3	11
Bc1	Bf4	12

Table 7.15

The moves associated with the top five CLAMP scores compare reasonably favourably with the analysis of the same configuration by the Fritz chess engine, with all moves being in the top 50% of Fritz's move preference list.

7.8.2. Bratko/Kopec Test 5 (best move: tactical)



Result: The 'best move' (marked with '<' alongside in the table below) according to the Bratko/Kopec test is rated the 14th choice out of a possible forty-seven moves scored by CLAMP (note that not all of the moves scored by CLAMP are legal moves)

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 5. The second move in CLAMP's ordered list is rated as the third best move by Fritz.

From	To	Score	F	From	To	Score	F	From	To	Score	F
Ra1	Re1	796082		Be3	Bd2	257942		Bb3	Be6	69954	
Ra1	Rd1	621528	3	Kg1	Kh1	257438		Qd4	Qb6	65762	
Ra1	Rc1	618780		Nc3	Nb1	206836		Qd4	Qd6	65616	
Pa2	Pa3	484378		Bb3	Bc4	191401		Ph2	Ph4	62558	
Qd4	Qd2	471313		Qd4	Qa4	187782		Qd4	Qd5	61251	
Nc3	Na4	457269		Qd4	Qd3	178713		Bb3	Bd5	52083	
Pa2	Pa4	400196	4	Rf1	Rf3	155701	2	Qd4	Qe5	46331	
Nc3	Nd1	367210		Be3	Bf2	152375		Qd4	Qc5	41655	
Ph2	Ph3	365476		Pg2	Pg3	139519		Rf1	Rf5	34233	
Be3	Bg5	354725		Rf1	Rf2	129534		Rf1	Rf6	31065	
Rf1	Rb1	352371		Bb3	Ba4	123158		Qd4	Qf6	28678	
Nc3	Ne2	334226		Pg2	Pg4	121429		Qd4	Qa7	24340	
Pe4	Pe5	328682		Qd4	Qc4	109192		Kg1	Kf2	7690	
Nc3	Nd5	314926	<	Be3	Bc1	103979					
Be3	Bh6	300975		Qd4	Qd1	99096					
Be3	Bf4	277532		Rf1	Rf4	87873	5				
Nc3	Nb5	258640		Qd4	Qb4	71385					

Table 7.16: Bratko/Kopec Test 5 CLAMP Score.

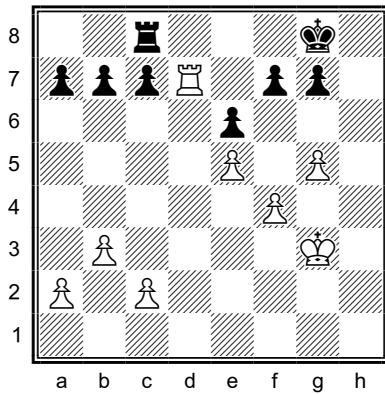
Table 7.17 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '47' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Ra1	Re1	7
Ra1	Rd1	3
Ra1	Rc1	8
Pa2	Pa3	6
Qd4	Qd2	18

Table 7.17

The moves associated with the top five CLAMP scores compare reasonably favourably with the analysis of the same configuration by the Fritz chess engine, with all moves being in the top 40% of Fritz's move preference list.

7.8.3. Bratko/Kopec Test 6 (best move: pawn lever)



Result: The ‘best move’ is Pg6. which is rated 14th out of twenty-seven moves which have been scored by CLAMP. The fourth move in CLAMP’s ordered list is fourth in the list of moves ordered by Fritz.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 6:

<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>
Pa2	Pa3	27157	5	Pe5	Pe6	4033		Rd7	Re7	805	
Pc2	Pc3	27067	6	Rd7	Rd3	3168		Kg3	Kf3	640	
Pc2	Pc4	19742		Rd7	Rd4	2966		Kg3	Kh3	621	
Pa2	Pa4	15659	4	Rd7	Rd8	2831		Kg3	Kh4	439	
Rd7	Rd1	11307		Pg5	Pg6	2479	<	Rd7	Rc7	426	
Pf4	Pf5	11054		Kg3	Kg2	2350		Rd7	Rf7	276	
Pb3	Pb4	6364	3	Kg3	Kf2	2327		Kg3	Kg4	141	
Rd7	Rd2	4658		Rd7	Rd5	1885		Rd7	Re7	805	
Kg3	Kh2	4374		Rd7	Rd6	1099		Kg3	Kf3	640	2

Table 7.18: Bratko/Kopec Test 6 CLAMP Scores

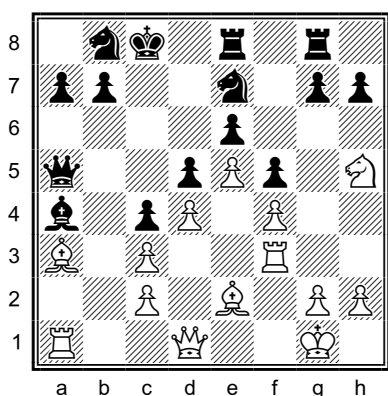
Table 7.19 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with ‘1’ being the best move and ‘27’ being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Pa2	Pa3	5
Pc2	Pc3	6
Pc2	Pc4	7
Pa2	Pa4	4
Rd7	Rd1	15

Table 7.19

The moves associated with the top five CLAMP scores compare reasonably favourably with the analysis of the same configuration by the Fritz chess engine, with all moves being in the top 50% of Fritz's move preference list.

7.8.4. Bratko/Kopec Test 7 (best move: tactical)



Result: The 'best move' (marked with '<') alongside in the table below) according to the Bratko/Kopec test is rated the 29th choice out of a possible thirty-eight moves on CLAMP's list.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 7:

From	To	Score	F	From	To	Score	F	From	To	Score	F
Be2	Bd3	135498		Ra1	Rb1	70620		Qd1	Qf1	27036	
Be2	Bc4	122895		Nh5	Ng3	67850		Nh5	Nf6	23082	<
Ph2	Ph3	117367		Ba3	Bd6	62818	2	Pg2	Pg4	22770	
Rf3	Rf1	111055		Qd1	Qb1	61194		Rf3	Rg3	20997	
Qd1	Qe1	107875		Ba3	Bb4	55295	5	Pe5	Pe6	16931	
Pg2	Pg3	106605		Ba3	Be7	53064		Ph2	Ph4	16406	
Ra1	Rc1	104592		Rf3	Rh3	49238		Rf3	Re3	13655	
Ba3	Bb2	90278		Rf3	Rf2	44530		Nh5	Ng7	13325	
Pd4	Pd5	85219		Pf4	Pf5	42641		Kg1	Kf1	8934	
Qd1	Qc1	83908	3	Ba3	Bc1	40721		Kg1	Kf2	6368	
Qd1	Qd3	77919		Ra1	Ra2	38956		Rf3	Rd3	6327	
Qd1	Qd2	76942	4	Be2	Bf1	30657					
Kg1	Kh1	71114		Ba3	Bc5	28512					

Table 7.20: Bratko/Kopec Test 7 CLAMP Scores

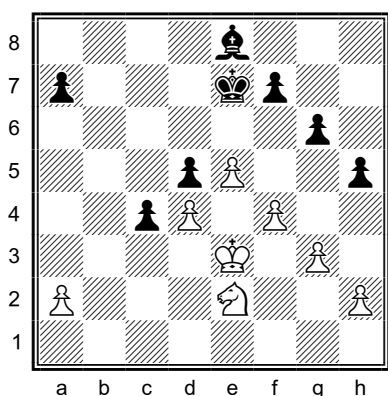
Table 7.21 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '38' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Be2	Bd3	32
Be2	Bc4	30
Ph2	Ph3	21
Rf3	Rf1	24
Qd1	Qe1	7

Table 7.21

The moves associated with the top five CLAMP scores compare poorly with the analysis of the same configuration by the Fritz chess engine. The moves associated with the top two CLAMP scores are tactically poor as they could result in a loss of the piece.

7.8.5. Bratko/Kopec Test 8 (best move: pawn lever)



Result: The 'best move' is Pf5. which is rated 7th out of sixteen moves which have been scored by CLAMP however the second choice in CLAMP's rank order (Ph3) is actually the fourth highest scoring move when the chessboard is analysed by Fritz.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 8:

<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>
Pa2	Pa3	12393	4	Ph2	Ph4	4386	
Ph2	Ph3	12248	3	Pe5	Pe6	2486	
Ne2	Nc1	8533		Ke3	Kf2	1335	
Ne2	Nc3	8291	2	Ke3	Kd2	577	
Pa2	Pa4	8201		Ke3	Ke4	232	
Pf4	Pf5	5902	<	Ke3	Kd3	228	
Ne2	Ng1	4831		Ke3	Kf3	220	
Pg3	Pg4	4538					

Table 7.22: Bratko/Kopec Test 8 CLAMP Scores

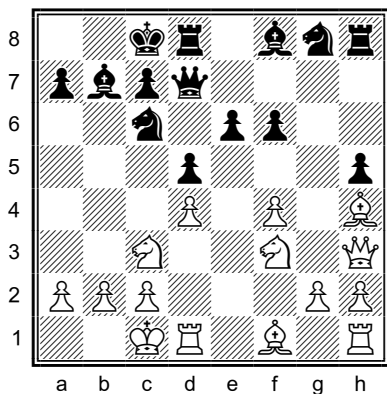
Table 7.23 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '16' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Pa2	Pa3	4
Ph2	Ph3	3
Ne2	Nc1	9
Ne2	Nc3	2
Pa2	Pa4	11

Table 7.23

The moves associated with the top five CLAMP scores compare reasonably favourably with the analysis of the same configuration by the Fritz chess engine, however the move 'a4' is tactically poor as it could result in the loss of the piece.

7.8.6. Bratko/Kopec Test 9 (best move: pawn lever)



Result: Result: The ‘best move’ is Pf5. which is rated 28th out of thirty-eight moves which have been scored by CLAMP however the first choice in CLAMP’s rank order (Kb1) is actually the fifth highest scoring move when the chessboard is analysed by Fritz.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 9:

From	To	Score	F	From	To	Score	F	From	To	Score	F
Kc1	Kb1	108828	5	Nc3	Nd5	21820		Nc3	Nb1	9960	
Bf1	Be2	46424		Nc3	Ne4	21089		Rd1	Re1	9585	<
Bf1	Bd3	44442	3	Bh4	Bg5	19113		Nf3	Ng1	8591	
Pa2	Pa3	41596		Pd4	Pd5	19044		Qh3	Qg4	8137	
Nc3	Na4	31204		Pb2	Pb3	17426		Rh1	Rg1	7879	
Bf1	Bb5	31076	2	Bf1	Bc4	16789		Pb2	Pb4	7449	
Nf3	Ne5	29965		Bf1	Ba6	14564		Qh3	Qf5	5310	
Bh4	Bg3	29565		Pa2	Pa4	14318		Bh4	Be1	4994	
Nc3	Ne2	27934		Qh3	Qg3	13747		Kc1	Kd2	4804	
Nc3	Nb5	27766	6	Nf3	Ne1	13736		Rd1	Rd2	4444	
Nf3	Nd2	27201		Bh4	Bf2	12715		Qh3	Qe6	3525	
Nf3	Ng5	26756		Bh4	Bf6	10138		Pg2	Pg4	3459	
Pg2	Pg3	21972	4	Pf4	Pf5	10010		Rd1	Rd3	2883	

Table 7.24: Bratko/Kopec Test 9 CLAMP Scores

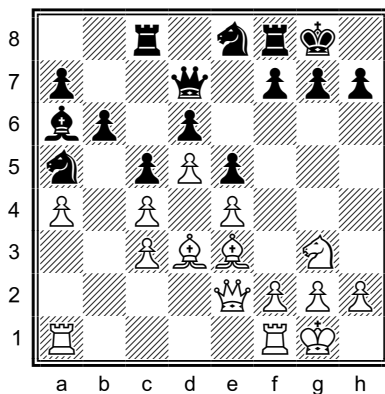
Table 7.25 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with ‘1’ being the best move and ‘38’ being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Kc1	Kb1	5
Bf1	Be2	8
Bf1	Bd3	3
Pa2	Pa3	10
Nc3	Na4	23

Table 7.25

The moves associated with the top five CLAMP scores compare reasonably favourably with the analysis of the same configuration by the Fritz chess engine.

7.8.7. Bratko/Kopec Test 11 (best move: pawn lever)



Result: The ‘best move’ (marked with ‘<’ alongside in the table below) according to the Bratko/Kopec test is rated the 9th choice out of a possible thirty-six, however the third choice in CLAMP’s rank order (Re1) is actually the fourth highest scoring move when the chessboard is analysed by Fritz. Fritz also scored Rb1 as its third choice, which is fourth in CLAMP’s ordered list.

The table below shows all possible moves, with the score assigned by CLAMP for configuration test 11:

<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>
Ra1	Rc1	823028		Pf2	Pf3	255594		Ra1	Ra3	161908	
Rf1	Rd1	744714	5	Be3	Bd2	253801		Qe2	Qd1	158106	
Rf1	Re1	721288	4	Qe2	Qb2	246547		Ng3	Nh5	154676	2
Rf1	Rb1	539637	3	Ng3	Nf5	227789		Qe2	Qh5	137465	
Qe2	Qc2	435685		Kg1	Kh1	219768		Be3	Bc5	135898	
Ph2	Ph3	422464		Qe2	Qe1	218067		Qe2	Qa2	117778	
Pa4	Pa5	388034		Be3	Bd4	204563		Ph2	Ph4	90527	
Qe2	Qd2	355413		Bd3	Bc2	193701		Pd5	Pd6	83222	
Pf2	Pf4	324268	<	Ng3	Nh1	189365					
Bd3	Bb1	307055		Be3	Bc1	178688					
Pc4	Pc5	305381		Qe2	Qg4	177824					
Be3	Bg5	304848		Qe2	Qf3	177389					
Be3	Bh6	271174		Ra1	Ra2	172697					
Pe4	Pe5	259295		Be3	Bf4	163758					

Table 7.26: Bratko/Kopec Test 11 CLAMP Scores

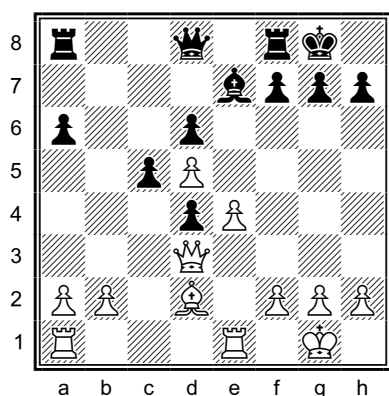
Table 7.27 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with ‘1’ being the best move and ‘36’ being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Ra1	Rc1	17
Rf1	Rd1	5
Rf1	Re1	4
Rf1	Rb1	3
Qe2	Qc2	22

Table 7.27

The most of the moves associated with the top five CLAMP scores compare reasonably favourably with the analysis of the same configuration by the Fritz chess engine.

7.8.8. Bratko/Kopec Test 13 (best move: pawn lever)



Result: The ‘best move’ (marked with ‘<’ alongside in the table below) according to the Bratko/Kopec test is rated the 19th choice out of a possible forty-six positions scored by CLAMP, however the first choice in CLAMP’s rank order (Rc1) is actually the second highest scoring move when the chessboard is analysed by Fritz.

Rd1 is scored second in CLAMP’s list and third by Fritz; Rb1 is scored fourth in both CLAMP’s list and Fritz. The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 13:

From	To	Score	F	From	To	Score	F	From	To	Score	F
Re1	Rc1	702832	2	Bd2	Bc3	287573		Qd3	Qh3	131406	
Re1	Rd1	650142	3	Pe4	Pe5	283595		Ph2	Ph4	123786	
Pa2	Pa3	505598	5	Pb2	Pb4	263487	<	Pg2	Pg4	118111	
Re1	Rb1	442731	4	Pf2	Pf3	249974		Bd2	Ba5	110389	
Pa2	Pa4	426944		Qd3	Qb1	191949		Pd5	Pd6	110010	
Bd2	Be3	418787		Kg1	Kh1	189124		Qd3	Qe3	106982	
Ph2	Ph3	417401		Pg2	Pg3	188708		Qd3	Qa6	97372	
Qd3	Qc2	404200		Qd3	Qf3	188322		Re1	Re2	78894	
Bd2	Bg5	394022		Qd3	Qc4	172607		Re1	Re3	73430	
Pb2	Pb3	388613		Qd3	Qd4	170358		Qd3	Qf1	70316	
Qd3	Qe2	374733		Qd3	Qb5	158894		Kg1	Kf1	38174	
Qd3	Qb3	328366		Qd3	Qa3	142002		Re1	Rd1	0	
Bd2	Bh6	321539		Bd2	Bc1	136577		Re1	Rc1	0	
Bd2	Bf4	317416		Qd3	Qc3	134305		Re1	Rb1	0	
Re1	Rf1	308866		Qd3	Qg3	133948					
Pf2	Pf4	300562		Bd2	Bb4	132266					

Table 7.28: Bratko/Kopec Test 13 CLAMP Scores

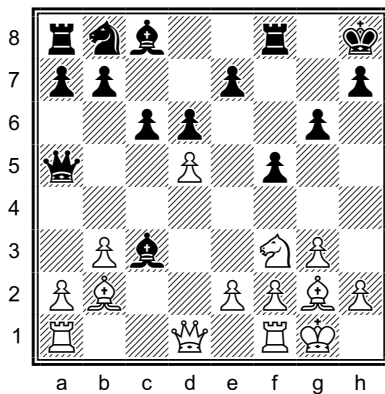
Table 7.29 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '46' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Re1	Rc1	2
Re1	Rd1	3
Pa2	Pa3	5
Re1	Rb1	4
Pa2	Pa4	17

Table 7.29

The moves associated with the top five CLAMP scores compare reasonably favourably with the analysis of the same configuration by the Fritz chess engine.

7.8.9. Bratko/Kopec Test 14 (best move: tactical)



Result: The 'best move' (marked with '<') alongside in the table below) according to the Bratko/Kopec test is rated the 3rd choice out of a possible thirty-two positions scored by CLAMP.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 14:

<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>
Nf3	Nd2	939370		Pb3	Pb4	573087		Bb2	Bc1	227328	
Bg2	Bh1	937629		Qd1	Qc1	532626	5	Pd5	Pd6	177277	
Qd1	Qd2	865691	<	Nf3	Ne5	499241		Pg3	Pg4	156525	
Pa2	Pa3	856011		Bb2	Ba3	492595		Ph2	Ph4	138197	
Qd1	Qc2	827568	4	Nf3	Ng5	486776					
Rf1	Re1	770952		Ra1	Rb1	405579					
Nf3	Ne1	752782		Bb2	Bc3	391611	3				
Pe2	Pe3	716420		Qd1	Qe1	367372	2				
Nf3	Nh4	710429		Pd5	Pc6	356273					
Nf3	Nd4	709700		Bg2	Bh3	327054					
Pe2	Pe4	693070		Kg1	Kh1	324048					
Pa2	Pa4	678859		Qd1	Qd3	267610					
Ph2	Ph3	665722		Qd1	Qb1	263978					
Ra1	Rc1	600505		Qd1	Qd4	250290					

Table 7.30: Bratko/Kopec Test 14 CLAMP Scores

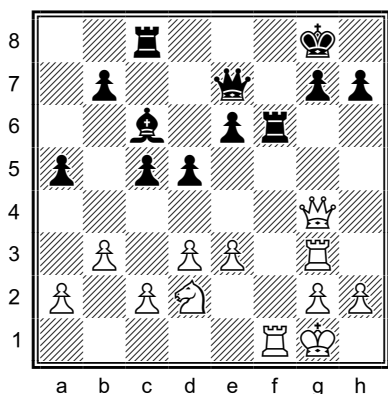
Table 7.31 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '32' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Nf3	Nd2	7
Bg2	Bh1	23
Qd1	Qd2	1
Pa2	Pa3	16
Qd1	Qc2	4

Table 7.31

The moves associated with the top five CLAMP scores compares reasonably with the analysis of the same configuration by the Fritz chess engine with the exception of 'Bh1' and 'a3', both of which are tactically poor as they can lose material

7.8.10. Bratko/Kopec Test 15 (best move: tactical)



Result: The 'best move' (marked with '<' alongside in the table below) according to the Bratko/Kopec test is rated the 35th choice out of a possible forty-four, however the fourth choice in CLAMP's rank order (Rf3) is actually the fourth highest scoring move when the chessboard is analysed by Fritz.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 15:

<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>
Pd3	Pd4	223143		Qg4	Qa4	113440		Rg3	Rh3	46603	
Pa2	Pa3	208251		Nd2	Ne4	112631		Rf1	Rf4	43089	4
Rf1	Rc1	196072		Rf1	Rd1	105747		Qg4	Qe4	31980	
Rf1	Rf3	192680	3	Kg1	Kh1	95824		Qg4	Qf5	31528	
Pe3	Pe4	185806		Rf1	Rb1	93952		Qg4	Qg7	27861	<
Pa2	Pa4	181244	5	Qg4	Qh5	69711		Qg4	Qf4	27514	
Pc2	Pc4	175453		Qg4	Qh4	61205		Qg4	Qg5	25921	
Nd2	Nb1	161104		Qg4	Qc4	61082		Ph2	Ph4	24606	
Ph2	Ph3	160948		Rf1	Ra1	60010		Qg4	Qe6	23843	
Pc2	Pc3	144831		Pb3	Pb4	57138		Rf1	Rf5	22876	
Rf1	Re1	144465		Qg4	Qd4	56975		Qg4	Qg6	15730	
Qg4	Qe2	133627		Qg4	Qf3	54747		Rf1	Rf6	15647	2
Nd2	Nf3	129613		Qg4	Qb4	54614		Kg1	Kf2	5583	
Nd2	Nc4	124816		Rf1	Rf2	50575		Rf1	Rf3	0	
Qg4	Qh3	116050		Qg4	Qd1	49575					

Table 7.32: Bratko/Kopec Test 15 CLAMP Scores

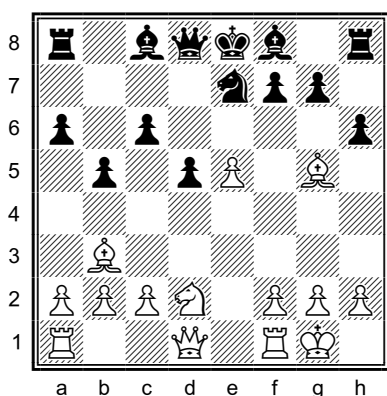
Table 7.33 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '44' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Pd3	Pd4	10
Pa2	Pa3	9
Rf1	Rc1	13
Rf1	Rf3	3
Pe3	Pe4	11

Table 7.33

The moves associated with the top five CLAMP scores compare reasonably with the analysis of the same configuration by the Fritz chess engine, with all five moves being within the top 25% of top moves scored by Fritz.

7.8.11. Bratko/Kopec Test 16 (best move: tactical)



Result: The ‘best move’ (marked with ‘<’ alongside in the table below) according to the Bratko/Kopec test is rated the 10th choice out of a possible thirty-five.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 16:

<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>	<i>From</i>	<i>To</i>	<i>Score</i>	<i>F</i>
Pa2	Pa3	450075		Nd2	Nc4	238246		Pf2	Pf3	146347	
Pc2	Pc3	425795		Bg5	Bf4	237652	3	Bb3	Bc4	142428	
Pa2	Pa4	362787		Nd2	Nf3	225476		Kg1	Kh1	140785	
Ph2	Ph3	307219		Bg5	Be7	218320	4	Qd1	Qb1	134311	
Qd1	Qe2	301220		Qd1	Qe1	208300		Nd2	Nb1	132277	
Pc2	Pc4	293176		Qd1	Qf3	207369		Qd1	Qg4	124560	
Rf1	Re1	291386		Qd1	Qh5	196826	2	Bb3	Ba4	116872	
Bg5	Bh4	277750		Pf2	Pf4	190494		Pg2	Pg4	61992	
Bg5	Be3	256595	5	Qd1	Qc1	188446		Ph2	Ph4	55419	
Nd2	Ne4	249521	<	Pg2	Pg3	167577		Bb3	Bd5	51565	
Ra1	Rc1	248032		Bg5	Bh6	150211		Pe5	Pe6	42097	
Bg5	Bf6	240372		Ra1	Rb1	149180					

Table 7.34: Bratko/Kopec Test 16 CLAMP Scores

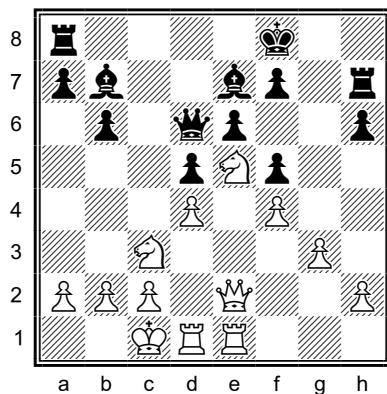
Table 7.35 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with ‘1’ being the best move and ‘35’ being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Pa2	Pa3	17
Pc2	Pc3	13
Pa2	Pa4	7
Ph2	Ph3	25
Qd1	Qe2	19

Table 7.35

The moves associated with the top five CLAMP scores compare with the analysis of the same configuration by the Fritz chess engine, however, most of the moves are tactically weak as they can result in a loss of material.

7.8.12. Bratko/Kopec Test 20 (best move: pawn lever)



Result: The 'best move' (marked with '<' alongside in the table below) according to the Bratko/Kopec test is rated the 25th choice out of a possible forty-two.

The table below shows all possible moves, with the score assigned by CLAMP, for configuration test 20:

From	To	Score	F	From	To	Score	F	From	To	Score	F
Pa2	Pa3	84625		Pb2	Pb4	30689		Qe2	Qg4	13398	
Nc3	Na4	68029		Ne5	Ng4	29452		Qe2	Qb5	13173	
Kc1	Kb1	66919	5	Qe2	Qf2	28335		Qe2	Qa6	12912	
Ne5	Nd7	66588		Ne5	Nc4	27783		Qe2	Qe4	12532	
Pb2	Pb3	64061		Nc3	Nb1	27695		Qe2	Qf1	11185	
Pa2	Pa4	51168		Re1	Rf1	25535		Ne5	Nf7	10504	
Ph2	Ph3	50834	<	Ne5	Nd3	25137		Qe2	Qc4	9856	
Pd4	Pd5	49339		Ne5	Ng6	23082		Rd1	Rd3	8689	
Qe2	Qd2	47589		Qe2	Qf3	20987		Qe2	Qg2	8209	
Nc3	Nd5	47109		Pg3	Pg4	20157	3	Re1	Rg1	7212	2
Nc3	Ne4	46971		Qe2	Qd3	19475		Re1	Rh1	2274	
Nc3	Nb5	37886		Ph2	Ph4	18430		Kc1	Kd2	2249	
Ne5	Nc6	37618		Qe2	Qh5	17029	4				
Ne5	Nf3	32087		Qe2	Qe3	15390					
Pf4	Pf5	30969		Rd1	Rd2	15104					

Table 7.36: Bratko/Kopec Test 20 CLAMP Scores

Table 7.37 (below) shows the top five CLAMP scores with the position of the move in order of preference from an analysis by the Fritz chess engine, with '1' being the best move and '42' being the worst move.

<i>From</i>	<i>To</i>	Fritz move preference
Pa2	Pa3	19
Nc3	Na4	16
Kc1	Kb1	5
Ne5	Nd7	34
Pb2	Pb3	26

Table 7.37

The moves associated with the top five CLAMP scores compare poorly with the analysis of the same configuration by the Fritz chess engine. The move 'Nd7' is tactically poor as it could result in a loss of the piece.

7.9. An analysis of the top moves

The analysis described so-far in this thesis has tested if the move played by the chess player was one of the suggested moves (suggested by CLAMP) appearing in the top half of the list of all possible moves which had been sorted in descending order of 'likelihood to be played' by CLAMP. This chapter has reported results from the comparison of methods using the whole board areas, sub-sections of the board and pieces in defending positions, for various chunks sizes. The analysis focused on the piece type and the position that a piece is moved to.

Another comparison method (which was used by Gobet and Jansen (1994) to test the 'CHUMP' program), is to quantify the percentage of cases that give a correct move within the top few 'predictions' (in this instance a 'prediction' is a move that is supported by the highest number of chunks, and the 'correct move' is the move that was actually played by the chess player). This analysis will therefore focus on the moves that CLAMP attributes the highest scores; low-scoring moves and chessboard configurations that give no scores are treated similarly. The chessboards were taken

from tournament games that were not used in the process to build the libraries and had an average 40.47 possible moves on the board with a standard deviation 5.86.

If move predictions were a result of a random selection then each move would have a uniform probability of approximately 1/40 of being selected. This estimate ignores the small variation in the number of possible moves in the sample of chessboards used for the evaluation. The probability of selecting a move and this being the best move therefore equates to 2.5% or 25 from a sample of one thousand chessboards. The chance of a random move being in the top two positions is similarly 5%, or 50 from a sample of one thousand chessboards (by adding the probabilities of the two positions), and 10%, or 100 from a sample of one thousand boards for the move to within the top four positions.

Table 7.37 (below) shows the probability that a *randomly selected* move will appear within the top 'n' best moves,

Where:

'p-1' is the best move.

'p-2' the move is within the top two positions.

'p-3' the move is within the top three positions.

'p-4' the move is within the top four positions.

p-1	p-2	p-3	p-4
25	50	75	100

Table 7.38: The probability of random moves from a sample of 1000 boards

If CLAMP suggests a 'best move' then, to be significant (that is, better than a random selection), the percentage of best moves that match the player's chosen move must occur more than 2.5% of the time. Similarly, for 'p-2' a better than random selection would be a percentage better than 5% and so on.

7.9.1. Using ‘Whole Board’ chunk libraries

The results from the analysis of one thousand chessboards, in the mid-game section of tournament games (games that were not used to build the chunk libraries), using chunk libraries based on the whole board area are shown in the table below. The table reports the number of actual moves played that matched the move associated with the highest score from CLAMP (p-1). In addition, the number of actual moves played that were within the top two scoring outputs (p-2), top three scoring outputs (p-3) and top four scoring outputs (p-4) from CLAMP.

		number of actual moves played	low	high
Chunk size 2	p-1	52	42	62
	p-2	86	72	100
	p-3	125	109	141
	p-4	170	151	189
Chunk size 3	p-1	52	42	62
	p-2	95	81	109
	p-3	132	116	148
	p-4	162	143	181
Chunk size 4	p-1	52	42	62
	p-2	97	83	111
	p-3	135	119	151
	p-4	173	154	192
Chunk size 5	p-1	42	32	52
	p-2	76	62	90
	p-3	107	91	123
	p-4	139	120	158

Table 7.39: Chunks from entire board area

The ‘low’ and ‘high’ figures show the 95% confidence limits for each result. The results shown in table 7.39 (above) are based on a sample (‘N’) of one thousand

games. As the number of samples are high, a binomial distribution is approximately equal to the standard deviation for the data. Assuming that each game will select one move from a possible ('p') of forty options, the standard deviation for column 'p-1' can be estimated:

$$\begin{aligned}
 (\text{SD}) &= \sqrt{Np(1-p)} \\
 &= \sqrt{1000 * 1/40 * 39/40} \\
 &= 4.937
 \end{aligned}$$

Subtracting two standard deviations from the results shown in the column 'p-1' shows that the results reported in table 7.38 are comfortably in excess of two standard deviations (the results exist within a 95% confidence interval) from the random probability of 2.5%. Similarly, the standard deviations for the other columns are as follows:

Chunk Size	Standard deviation
p-1	4.93
p-2	6.892
p-3	7.888
p-4	9.747

Table 7.40: Standard deviations for each column

The results for p-1, p-2, p-3 and p-4 reported in table 7.39 show figures that are above the random percentage figures, indicating that the results from CLAMP are significant, albeit by a small amount. All of the results are in excess of two standard deviations, from the random positions for each column. As the size of the chunk increases to five pieces, the percentage of boards that were successfully scored decreases (cf. page 88). The highest p-1, p-2, p-3 and p-4 percentages were found when a chunk size of four was used.

7.9.2. Using 'Defensive' chunk libraries

The following table shows the scores when using libraries that were built from chunks with pieces in defensive relationships:

		number of actual moves played	Low	high
Chunk size 2	p-1	35	25.14	44.86
	p-2	62	48.22	75.78
	p-3	99	83.22	114.78
	p-4	129	109.51	148.49
Chunk size 3	p-1	36	26.14	45.86
	p-2	74	60.22	87.78
	p-3	102	86.22	117.78
	p-4	142	122.51	161.49
Chunk size 4	p-1	38	28.14	47.86
	p-2	83	69.22	96.78
	p-3	117	101.22	132.78
	p-4	150	130.51	169.49
Chunk size 5	p-1	49	39.14	58.86
	p-2	86	72.22	99.78
	p-3	122	106.22	137.78
	p-4	151	131.51	170.49
Chunk size 6	p-1	50	40.14	59.86
	p-2	90	76.22	103.78
	p-3	117	101.22	132.78
	p-4	156	136.51	175.49
Chunk size 7	p-1	40	30.14	49.86
	p-2	74	60.22	87.78
	p-3	104	88.22	119.78
	p-4	124	104.51	143.49

Table 7.41: Analysis using defensive chunks

All of the results shown in table 7.41 (above) show that when using chunk sizes of 3,4,5,6 or 7 pieces the results are in excess of two standard deviations from the random positions for each column (cf. table 7.38). The results when using a chunk size of two shows a lower 95% confidence interval which is very close to the random distribution value (cf. page 136).

The number of moves played shown in table 7.41 are slightly lower than but comparable to, results obtained when using the whole board area. However, the defending libraries are considerably smaller in size (cf. 104).

7.9.3. Using small grouped area chunk libraries

The tables below show results obtained from using libraries generated from chunks in small local areas on the chessboard:

		number of actual moves played	low	high
Chunk size 2	p-1	39	29	49
	p-2	68	54	82
	p-3	94	78	110
	p-4	129	110	148
Chunk size 3	p-1	53	43	63
	p-2	101	87	115
	p-3	136	120	152
	p-4	160	141	179
Chunk size 4	p-1	58	48	68
	p-2	104	90	118
	p-3	142	126	158
	p-4	162	143	181
Chunk size 5	p-1	40	30	50
	p-2	76	62	90
	p-3	87	71	103
	p-4	102	83	121

Table 7.42: Chunks with pieces in 3x3 local areas

All of the results shown in table (above) show that when using chunk sizes of 2, 3,4 or 5 pieces the results are in excess of two standard deviations from the random distribution value (cf. page 136).

		number of actual moves played	low	high
Chunk size 2	p-1	41	31	51
	p-2	91	77	105
	p-3	123	107	139
	p-4	157	138	176
Chunk size 3	p-1	48	38	58
	p-2	94	80	108
	p-3	141	125	157
	p-4	177	158	196
Chunk size 4	p-1	53	43	63
	p-2	104	90	118
	p-3	141	125	157
	p-4	185	166	204
Chunk size 5	p-1	50	40	60
	p-2	88	74	102
	p-3	117	101	133
	p-4	152	133	171
Chunk size 6	p-1	33	23	43
	p-2	57	43	71
	p-3	74	58	90
	p-4	90	71	109

Table 7.43: Chunks with pieces in 4x4 local areas

All of the results shown in table (above) show that when using chunk sizes of 2, 3,4 or 5 pieces the results are in excess of two standard deviations from the random the random distribution (cf. page 136). When using a chunk size of six pieces, the lower 95% confidence limit is below the value for the random distribution value (cf. page 136).

		number of actual moves played	low	high
Chunk size 2	p-1	35	25	45
	p-2	70	56	84
	p-3	108	92	124
	p-4	145	126	164
Chunk size 3	p-1	50	40	60
	p-2	81	67	95
	p-3	107	91	123
	p-4	148	129	167
Chunk size 4	p-1	44	34	54
	p-2	83	69	97
	p-3	121	105	137
	p-4	153	134	172
Chunk size 5	p-1	47	37	57
	p-2	89	75	103
	p-3	123	107	139
	p-4	158	139	177
Chunk size 6	p-1	40	30	50
	p-2	96	82	110
	p-3	131	115	147
	p-4	169	150	188

Table 7.44: Chunks with pieces in 5x5 local areas

All of the results shown in table (above) show that when using chunk sizes of 2, 3,4,6 or 6 pieces the results are in excess of two standard deviations from the random distribution value (cf. page 136).

		number of actual moves played	low	high
Chunk size 2	p-1	33	23	43
	p-2	59	45	73
	p-3	98	82	114
	p-4	124	105	143
Chunk size 3	p-1	37	27	47
	p-2	65	51	79
	p-3	101	85	117
	p-4	131	112	150
Chunk size 4	p-1	35	25	45
	p-2	60	46	74
	p-3	98	82	114
	p-4	129	110	148
Chunk size 5	p-1	40	30	50
	p-2	78	64	92
	p-3	105	89	121
	p-4	142	123	161
Chunk size 6	p-1	46	36	56
	p-2	76	62	90
	p-3	102	86	118
	p-4	130	111	149

Table 7.45: Chunks with pieces in 6x6 local areas

All of the results shown in table 7.45 (above) are close to or in excess of two standard deviations with respect to the lower confidence limits, from the random distribution values (cf. page 136).

7.10. Top move analysis and the number of boards scored

When considering chunks within a small area on the board, a chunk size of four within a local area of 4x4 squares gives the best overall results with 18.5% of moves played, being within the top four of CLAMP's output. With this configuration, the highest scoring move from CLAMP is the actual move played 5.3% of the time. The '4x4 area – four piece chunks' chunk library is not only slightly better than the 'whole board – four piece chunk' library but it contains 210,120 chunks, which is a fraction of the 27,127,049 chunks contained in the 'whole board – four piece chunk' library, equating to a 99.8% reduction in chunks within the library. The 4x4, 4 piece chunk however only found chunks on 682 out of the sample one thousand boards.

As the chunk size *increases*, the number of boards that are successfully scored *decreases*. As reported earlier in this thesis, large chunks are *rare but specific* (cf. page 88) so if CLAMP can find large chunks in the configuration and match these to a move in the library then the move has a high likelihood of being the 'best' move. Using large chunks however, will fail to score many boards, for example in the above analysis the '3x3 – five piece chunk' analysis only scored 223 chessboards out of the one thousand in the test. However, if at least one chunk from the chessboard is found within the chunk library then the accuracy of CLAMP's prediction is higher when using large chunk sizes than for chunks made up from a small number of pieces.

The percentage probability that CLAMP will correctly score a move within the top four positions *provided the board has at least one chunk matching a move within the chunk library* can be calculated by dividing the p-4 score by the 'Number of boards scored' value, and then multiplying the result by 1000, in the above tables. The adjusted scores for each of the scenarios are shown below:

Chunk Size	p-4	Number of boards scored (#)	Adjusted p-4
2	17.0%	995	17.1%
3	16.2%	995	16.3%
4	17.3%	995	17.4%
5	13.9%	567	24.5%

Table 7.46: 'Adjusted p-4' percentage - Whole board

Chunk Size	p-4	Number of boards scored	Adjusted p-4
2	12.9%	995	13.0%
3	14.2%	954	14.9%
4	15.0%	965	15.5%
5	15.1%	825	18.3%
6	15.6%	714	21.8%
7	12.4%	427	29.0%

Table 7.47: 'Adjusted p-4' percentage - Defending chunk libraries

Local area (squares)	Chunk size	p-4	Number of boards scored	Adjusted p-4
3x3	2	12.9%	851	15.2%
3x3	3	16.0%	820	19.5%
3x3	4	16.2%	542	29.9%
3x3	5	10.2%	223	45.7%
4x4	2	15.7%	872	18.0%
4x4	3	17.7%	840	21.1%
4x4	4	18.5%	682	27.1%
4x4	5	15.2%	499	30.5%
4x4	6	9.0%	210	42.9%
5x5	2	14.5%	905	16.0%
5x5	3	14.8%	849	17.4%
5x5	4	15.3%	786	19.5%
5x5	5	15.8%	656	24.1%
5x5	6	16.9%	518	32.6%
6x6	2	12.4%	875	14.2%
6x6	3	13.1%	855	15.3%
6x6	4	12.9%	843	15.3%
6x6	5	14.2%	763	18.6%
6x6	6	13.0%	549	23.7%

Table 7.48: 'Adjusted p-4' percentage – Local area chunk libraries

7.11. Using chunks to suggest a move: a design strategy

This section is included to illustrate how libraries with large chunks can be used in a practical scenario, taking into consideration that large chunks are rarely repeated between chessboards and, as a result, many large chunks may be absent from the chunk libraries.

Suppose a program has to be written to suggest the best chess move with a 'confidence' percentage, how could such a program be written? A good strategy when applying chunking to practical applications would be to use the libraries with the smallest number of chunks (so that searching through the library is faster) to find associated moves, and starting with a library that gives the highest 'Adjusted p-4' percentage, and then iterate through numerous chunk libraries until a move is predicted.

If the hypothetical program can make four guesses when analysing a chessboard the probability of the correct move being within the four guesses can be calculated for each of the chunk libraries by using column labelled 'Adjusted p-4' in tables 7.46, 7.47 and 7.48. The probability that a score will be assigned by the chunk library is calculated from the column labelled 'number of boards scored' from the above tables.

In order to maximise the program speed, where possible, the smallest chunk libraries should be used to minimise the search.

With reference to the results reported in section 7.9, the process could, for example, use chunk libraries in the sequence shown in the table below:

Sequence	Board area (squares)	Chunk Size (pieces)	Number of chunks in library	Number of boards scored	Probability of getting a score	Probability of the 'correct move' being in top 4 scores
1	4x4	6	26139	210	21.0%	42.9%
2	5x5	6	388865	518	51.8%	32.6%
3	4x4	4	210120	682	68.2%	27.1%
4	3x3	3	80819	820	82.0%	19.5%
5	8x8	3	7596060	995	99.5%	16.3%

Table 7.49: Iteration sequence to predict top moves.

The move predictor program will perform the following steps in the order below:

1. Using the '4x4 six-piece chunk' library will score approximately 21% of boards; if a score is obtained then the first four moves will contain the best move with a probability of approximately 43%. The chunk library is small with 26139 chunks and so searching this library will be fast.
2. If no score is obtained from step 1 (above) then the '5x5 six-piece chunk' library should be tried. The '5x5 six-piece chunk' library will score approximately 52% of boards and will predict the best move being within the top four outputs with a probability of approximately 33%.
3. If no score is obtained from step 2 (above) then the '4x4 four-piece chunk' library should be tried. The '4x4 four-piece chunk' library will score approximately 68% of boards and will predict the best move being within the top four outputs with a probability of approximately 27%.
4. If no score is obtained from step 3 (above) then the '3x3 three-piece chunk' library should be tried. The '3x3 three-piece chunk' library will score approximately 82% of boards and will predict the best move being within the top four outputs with a probability of approximately 20%.
5. If no score is obtained from step 4 (above) then the 'whole board three-piece chunk' library should be tried. The 'whole board three-piece chunk' library will

score approximately 99% of boards and will predict the best move being within the top four outputs with a probability of approximately 16%.

The above sequence will search for moves associated with chunks, starting with large (six piece) chunks and decreasing to three-piece chunks. The sequence attempts to find *rare but specific* chunks first, with each step moving increasingly towards *frequent and general* chunks. The final stage (whole board, three piece chunk library) will score 99% of boards albeit with an accuracy probability of 16% that the best move is in the four suggested outputs.

7.12. Chapter conclusion

This chapter has reported results from the analysis of chunks produced by the program CLAMP. The results show that chunk patterns can be used to suggest chess moves with a probability that is higher than a random selection. Results reported compared the effectiveness of different sizes of chunk and various filters used in the procurement of chunks, including restricting chunks to pieces in close proximity or restricting chunks to pieces in defending relationships. The corresponding reduction in chunk library size without significant loss of accuracy intimating that many of the effective chunks exist as pieces in defending positions or with close proximity.

This chapter tested the effectiveness of the selection of a move to play by comparing the output of CLAMPanalyser with one thousand sample games. The games comprised of tournament transcripts between master chess players (games that were not used by CLAMP to build the chunk libraries) with CLAMPanalyser matching the human player's moves with a 'better than random' result comfortably with a 95% confidence limit.

Specific chessboard configurations were also tested, such as de Groot's Position 'A' and the Bratko/Kopec configurations, with results compared with the output from a commercial chess program. The Bratko/Kopec test showed that CLAMPAnalyser performed slightly better with positional as opposed to tactical moves.

The final section illustrated how a hypothetical program could use several chunk libraries to select a move, based on the probability that a library will 'score' a chessboard and the likelihood that the best move is in the top four suggestions.

The results of the tests on CLAMPAnalyser show that knowledge of chunks within a computer system can be used to suggest good chess moves from an analysis of the piece constellations on the board. The result is consistent with human chunking theory that suggests that an expert human chess player has knowledge of chunks, and that this knowledge, or "perceptual advantage", can direct attention to relevant moves (de Groot 1978, pp. 307).

8. AN EVALUATION OF PIECES MOVED *FROM* A POSITION

The analysis reported so far in this thesis was based on libraries built from collections that were compiled from pieces *arriving* on squares. This chapter examines the results when CLAMP was configured to produce libraries using collections of *departures* from squares. In this case the collections (cf. page 53) are compiled from board configurations that exist immediately prior to the move of a piece. The result of this process is that CLAMPanalyser will produce a list of pieces in order of their likelihood of being moved from their current positions. Pieces *from* a position are less useful when suggesting a chess move as the destination of where the piece is moved to is unknown. The number of possible moves is reduced to the number of pieces on the board of the colour who is to move. Only 'white' moves are considered in this chapter, as this is sufficient for proof of concept.

When building a chunk library, the name of the collection includes the piece/position of the chess piece prior to the move. This method generates a score for each (white) piece on the board, for example, the 'Coats v Parkin' configuration (cf. page 84) results in fourteen scores, one for each white piece. No knowledge of the rules of chess, or how pieces move, is used and in some instances a score is allocated to a piece even if the piece cannot legally move (for example, in the 'Coats v Parkin' configuration (cf. page 84), the pawn on f2 is blocked by the white knight on square f3).

The analysis using *departures* gave a score for a move from a position for each piece on the chessboard. In the same way as the process described in chapter 6 gave indicators for moves to a square, analysis of moves from squares gave an indication of which *pieces* were likely candidates to be moved.

With the 'analysis of departures from a square' the CLAMPanalyser score is the total of the number of chunks that exist on the chessboard that can be found in

the library file that is associated with a move of the piece from the square. The following table shows CLAMPAnalysers scores based on four piece chunks, for piece departures and the corresponding Fritz score. The Fritz score is based on the best (the highest scoring move) that the piece can make.

MOVE FROM:	CLAMP SCORE	FRITZ SCORE		MOVE FROM:	CLAMP SCORE	FRITZ SCORE
Be2	237562	-0.37		Pd4	189049	-0.91
Be3	131419	-0.25		Pg2	107479	-0.19
Kg1	036547	-0.34		Ph3	035862	-0.62
Nc3	207605	-0.06		Qd1	303675	-0.12
Nf3	254543	-0.19		Ra1	148715	-0.06
Pa2	233678	-0.06		Rf1	127730	-0.25
Pb2	165369	-0.47		Pf2	094086	N/A

Table 8.1: – Move ‘From’ CLAMP and Fritz score comparison

The scatter graph below shows the CLAMP score compared with the Fritz score. The line drawn on the graph shows the linear least squares analysis with a correlation coefficient of 0.307

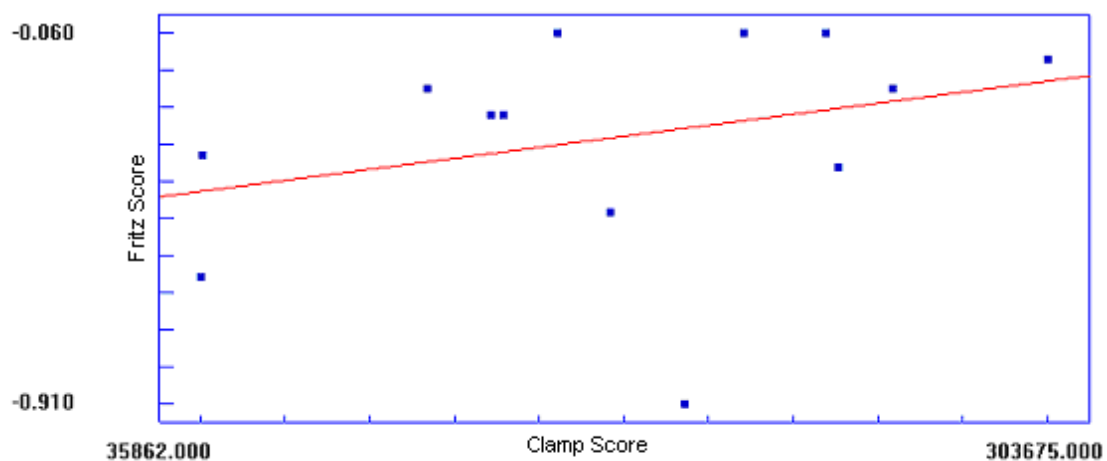


Figure 8.1: Move ‘From’ CLAMP and Fritz score comparison

8.1. 'Move From' scores by using 'Move To' analysis

When considering moves from a square in the above procedure, no consideration is given as to the number of positions that a piece can move to, for example, if there are no viable moves for a piece on the chessboard under examination then the piece should not be considered for a move. Similarly, if there are just a few viable positions a piece can be moved to, then it may be advantageous to take this into account when calculating the score for the piece. A piece departing from a square may move to several alternative positions with varying merit. An enhancement to the 'Move To' method is to attribute the 'best' score, from the possible destination squares. By calculating the chunk move score for pieces moving to a position a more meaningful result can be achieved.

The 'Coats v Parkin' configuration (cf. page 84) has thirteen possible white piece moves. Table 7.1 shows the scores for all available squares a piece can move to. By taking the move with the *highest* CLAMPAnalyser score and assigning this score to the *piece*, CLAMPAnalyser will assign scores to pieces based on the most frequently played move according to the chunks that exist on the chessboard. This method assumes that if a piece is moved then it will be moved to its highest scoring position. The score assigned to a piece is therefore the highest *potential* score for that piece.

The table below shows the scores from table 7.1 by assigning the *highest* scores to each piece (the highest CLAMP score and the highest Fritz score for a piece moving *from* a position). Similarly the table shows the 'Fritz Score' as the highest scores that a piece can acquire by selecting the best move for each piece calculated by Fritz.

MOVE FROM:	CLAMP SCORE	FRITZ SCORE	MOVE FROM:	CLAMP SCORE	FRITZ SCORE
Be2	231657	-0.37	Pd4	192147	-0.91
Be3	274531	-0.25	Pg2	152032	-0.19
Kg1	115648	-0.34	Ph3	38088	-0.62
Nc3	329186	-0.06	Qd1	327807	0
Nf3	351078	-0.19	Ra1	206116	-0.06
Pa2	417532	-0.06	Rf1	191577	-0.25
Pb2	222510	-0.47			

Table 8.2: – Move ‘From’ using highest ‘Move To’ CLAMP and Fritz score comparison

The scatter graph below shows the CLAMP score compared with the Fritz score. The line drawn on the graph shows the linear least squares analysis with a correlation coefficient of 0.616

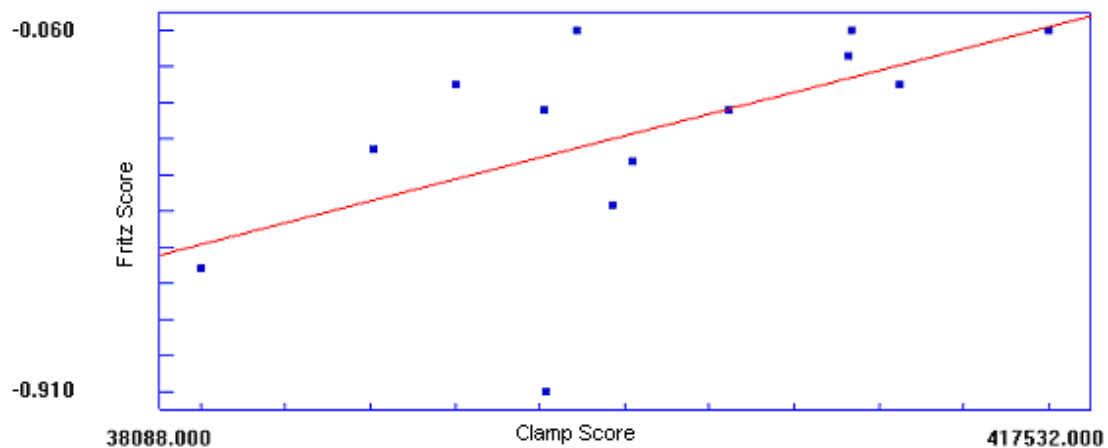


Figure 8.2 Move ‘From’ using highest ‘Move To’ CLAMP and Fritz score comparison

When considering a piece *move from* a position, the correlation between CLAMP and Fritz is higher using *move from scores by using ‘Move To’ analysis* than with ‘*Move From*’ method reported in this section (although the comparison of methods reported in this section was based on just one chess configuration - the ‘Coats v Parkin’ configuration (cf. page 84)).

8.2. Combining the likelihood of a *Move To* with the *Move From* score

The 'Move From' and 'Move To' methods each produce a likelihood score for a piece departure from a square and a piece arrival on a square. The 'Move From' score gives no indication of where the piece is moved to, and similarly the 'Move To' score no indication of the source. In some circumstances, for example where there is more than one piece of the same type on the board, the 'Move To' analysis cannot differentiate the actual move from source to destination.

This section shows how the two methods can be combined to give a score for the move of a piece from its old position to the new position.

Table 8.3 (below) lists the move scores based on 'Move From' and 'Move To' analysis. The 'Move From' and 'Move To' likelihood percentages are combined (as the product from the two methods) to give an overall likelihood score. The 'Fritz score' is included for comparison with the combined score.

MOVE FROM	MOVE FROM SCORE	MOVE FROM LIKELIHOOD %	MOVE TO	MOVE TO SCORE	MOVE TO LIKELIHOOD %	COMBINED LIKELIHOOD %	FRITZ SCORE
Be2	237562	78.2%	Bd3	231657	55.5%	43.4%	-0.41
Be2	237562	78.2%	Bc4	212682	50.9%	39.9%	-0.37
Be2	237562	78.2%	Bb5	192263	46.1%	36.0%	-0.44
Be2	237562	78.2%	Ba6	85221	20.4%	16.0%	-3.34
Be3	131419	43.3%	Bg5	274531	65.8%	28.5%	-0.25
Be3	131419	43.3%	Bd2	205979	49.3%	21.4%	-0.31
Be3	131419	43.3%	Bf4	197600	47.3%	20.5%	-3.62
Be3	131419	43.3%	Bh6	129318	31.0%	13.4%	-3.03
Be3	131419	43.3%	Bc1	76306	18.3%	7.9%	-0.5
Kg1	36547	12.0%	Kh1	115648	27.7%	3.3%	-0.34
Nc3	207605	68.4%	Na4	329186	78.8%	53.9%	-0.28
Nc3	207605	68.4%	Nb5	221909	53.2%	36.3%	-0.06
Nc3	207605	68.4%	Ne4	199718	47.8%	32.7%	-3.16
Nc3	207605	68.4%	Nd5	151850	36.4%	24.9%	-3.59
Nc3	207605	68.4%	Nb1	128618	30.8%	21.1%	-0.5
Nf3	254543	83.8%	Nh4	351078	84.1%	70.5%	-0.53
Nf3	254543	83.8%	Nd2	344666	82.6%	69.2%	-0.41
Nf3	254543	83.8%	Ne5	280526	67.2%	56.3%	-0.19
Nf3	254543	83.8%	Ne1	272470	65.3%	54.7%	-0.44
Nf3	254543	83.8%	Nh2	244546	58.6%	49.1%	-0.53
Nf3	254543	83.8%	Ng5	216931	52.0%	43.6%	-0.44
Pa2	233678	77.0%	Pa3	417532	100.0%	77.0%	-0.06
Pa2	233678	77.0%	Pa4	248241	59.5%	45.8%	-0.5
Pb2	233678	77.0%	Pb3	222510	53.3%	41.0%	-0.47
Pd4	189049	62.3%	Pd5	192147	46.0%	28.7%	-0.91
Pg2	107479	35.4%	Pg3	152032	36.4%	12.9%	-0.47
Pg2	107479	35.4%	Pg4	85885	20.6%	7.3%	-0.19
Ph3	35862	11.8%	Ph4	38088	9.1%	1.1%	-0.62
Qd1	303675	100.0%	Qb3	327807	78.5%	78.5%	-0.12
Qd1	303675	100.0%	Qc2	290939	69.7%	69.7%	-0.34
Qd1	303675	100.0%	Qa4	288587	69.1%	69.1%	0
Qd1	303675	100.0%	Qd2	265918	63.7%	63.7%	-0.41
Qd1	303675	100.0%	Qc1	205312	49.2%	49.2%	-0.44
Qd1	303675	100.0%	Qe1	193843	46.4%	46.4%	-0.37
Qd1	303675	100.0%	Qb1	156168	37.4%	37.4%	-0.5
Qd1	303675	100.0%	Qd3	150672	36.1%	36.1%	-0.28
Ra1	148715	49.0%	Rc1	206116	49.4%	24.2%	-0.06
Ra1	148715	49.0%	Rb1	120512	28.9%	14.1%	-0.25
Rf1	127730	42.1%	Re1	191577	45.9%	19.3%	-0.25

Table 8.3: 'Move From' and 'Move To' analysis results

The moves shown as a scatter graph on the graphs below:

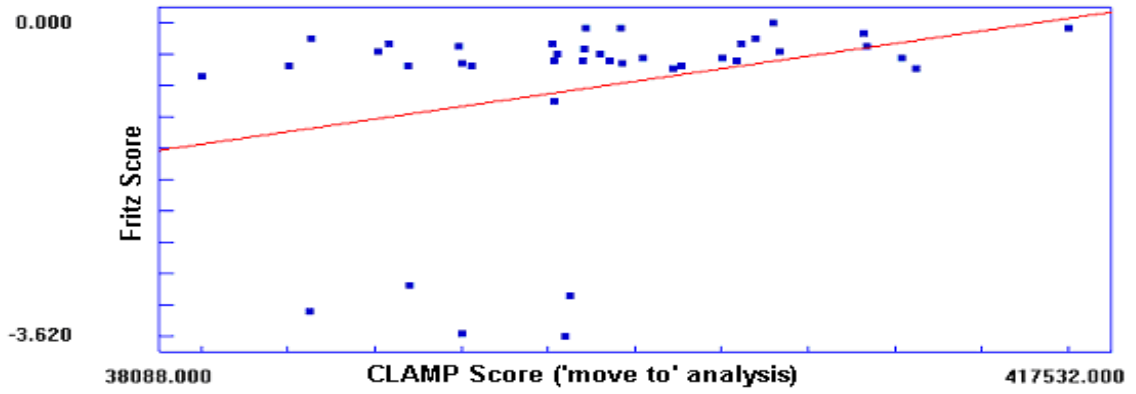


Figure 8.3: CLAMP / Fritz comparison ('Move To' analysis)

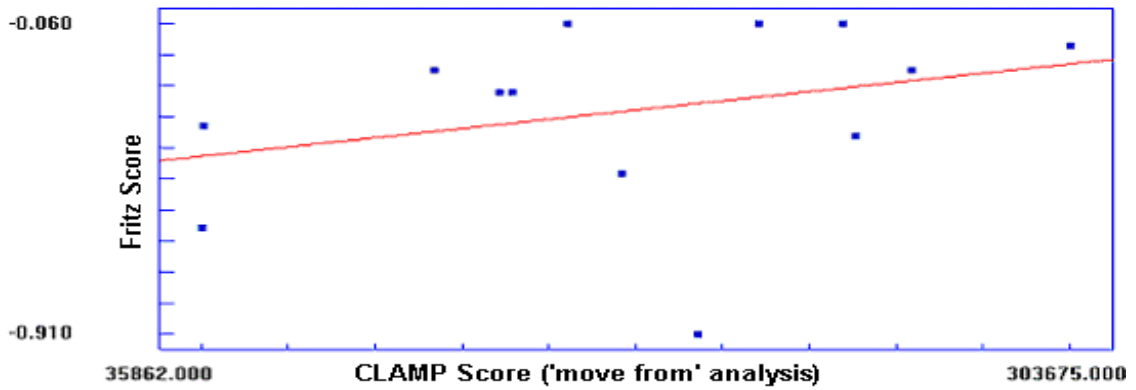


Figure 8.4: CLAMP / Fritz comparison (move from analysis)

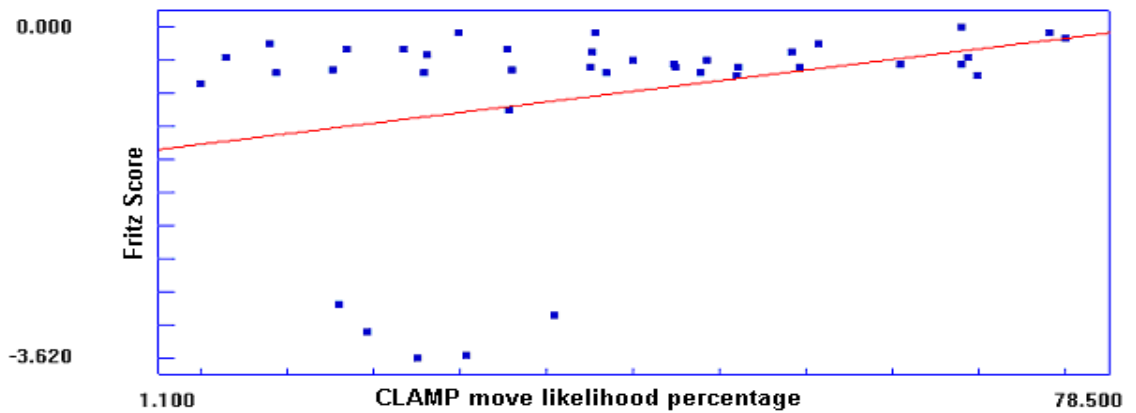


Figure 8.5: CLAMP and Fritz move likelihood comparison²³

²³ The correlation coefficient for the Fritz score and CLAMP move likelihood percentage is 0.314

Combing the ‘Move To’ and ‘Move From’ likelihood has little effect on the correlation coefficient between Fritz and CLAMPanalyser scores, however, this process reduces the ambiguity of a move where a type of piece could move to a position from more than one origin. The top ten move-likelihood’s for Fritz, ‘Move To’ and combined ‘Move To/Move From’ scores are different in each case as shown in table 8.4 below:

Move ordering	Fritz	Move To	Move To and Move From
1	Qa4	Pa3	Qb3
2	Nb5	Nh4	Pa3
3	Pa3	Nd2	Nh4
4	Rc1	Na4	Qc2
5	Qb3	Qb3	Nd2
6	Ne5	Qc2	Qa4
7	Pg4	Qa4	Qd2
8	Bg5	Ne5	Ne5
9	Rb1	Bg5	Ne1
10	Re1	Ne1	Na4

Table 8.4: Move ordering comparison

8.3. Chapter conclusion

This chapter has introduced the method for suggesting the move of a piece *from* a square on the chessboard by analysing the component chunks on the board and using knowledge in a chunk library based on the frequently played moves from a piece on the chessboard. The suggested move from a piece can then be combined with the suggested move to a square, by using the chunk libraries compiled as described in earlier chapters of this thesis. The resulting move ordering has less ambiguity regarding which piece is moved in situations where two chess pieces of the same type can access the same destination.

9. AN APPLICATION OF CHUNKING TO A CHESS PLAYING PROGRAM

“Put one pound of Alpha Beta Prunes in a jar or dish that has a cover. Pour one quart of boiling water over the prunes. The longer the prunes soak, the plumper they get. –

Alpha Beta Acme Markets, Inc, La Habra, California” – Knuth and Moore, 1975

9.1. A brief look at the MINIMAX routine

The vast majority of chess programs use a variation on the MINIMAX search to select the move to make. The MINIMAX algorithm, which was proposed by Claude Shannon (Shannon 1950), is essentially a simple process that ‘moves’ each piece on the board, trying every possible legal position for the piece. The MINIMAX algorithm can be used to explore domains where an action results in a number of consequential responses, each response then evokes a further action and so on, resulting in an ever expanding tree of possible solutions with increasing depth of search. MINIMAX (or more normally the alpha-beta routine which is an enhancement to the MINIMAX routine) is commonly used in games to test possible moves and responses between two players, in particular the alpha-beta search is used extensively within the game of chess where all possible moves and counter-moves are evaluated many positions in advance. The routine starts with a move with its own colour (for example, Black), moving one piece one position. The program then moves each white piece, trying every legal position in turn. Each move requires the opponent to move all of his pieces through all possible moves, resulting in a huge branching structure illustrated below:

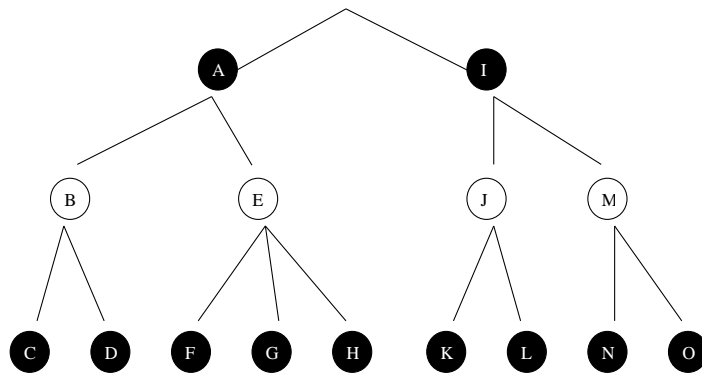


Figure 9.1: An example of the MINIMAX function to explore all moves and counter moves to a depth of three ply.

The diagram shows the black pieces having two possible moves, 'A' and 'I'. The program will make move 'A' (a black piece) and then corresponding move 'B' (a white piece) . Following this, a black piece is moved and so on. The game is played out by alternating black and white moves in turn to the depth required (typically between eight and sixteen) by the program.

Having reached the maximum depth (in the above diagram this is shown as a depth of three), the program backtracks to the previous move ('B') and tries an alternative response ('D'). Every legal response is tried, but in the above example 'B' is shown to have two possible moves.

Having exhausted all responses to move 'B' the program backtracks to move 'A' and tries the next white move 'E'.

The program plays all possible moves of both black and white pieces for several turns. This gives all of the possible scenarios for the game looking ahead by the number of turns. Each of the final scenarios are scored, with the highest scoring result being passed back to the root level so that the program can choose a move (either 'A' or 'I' in the above diagram) to achieve the best result.

A simple scoring method could be to count the number of black pieces on the board minus the number of white pieces. From blacks point of view, the higher the

number then the stronger black is. Black will favour moves that increase the score and white will favour moves to decrease it.

Having evaluated all scenarios for the deepest levels, the program can assign values to each node (each of the moves taken) according to the rule:

- If the move was taken by white then select the lowest score
- If the move was taken by black then select the highest score.

Consider the following example, which shows the score at each node:

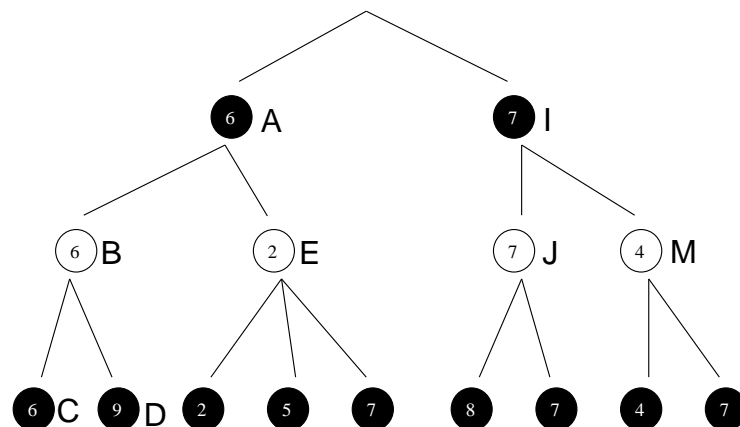


Figure 9.2: A simple MINIMAX example with the score at each node.

For example, node 'B' has two possible moves ('C' or 'D'). Move 'C' results in a score of '6' and move 'D' results in a score of '9'. As this is white's node the lowest score ('6') will be assigned to the node 'B'

Node 'A' has two possible moves, 'B' and 'E'. As node 'A' is a black move the highest score will be taken from nodes 'B' and 'E', and in this case, black will chose to make the move 'I'.

The algorithm assumes both 'Black' and 'White' always play their best moves.

9.2. Optimising the MINIMAX search with alpha-beta pruning.

A mid-game chessboard may allow typically thirty-five legal moves. As each one of the possible moves will have roughly the same number of counter moves on each level of the search, the search tree will rapidly expand as the depth of search increases. A commercial chess program would search typically between eight and sixteen ply deep giving a potentially huge number of nodes. Alpha-beta pruning is an optimisation to MINIMAX that can dramatically reduce the number of nodes, thereby reducing the processing requirement for a chess computer.

Considering the search tree in shown in figure 9.2 above, if the minimum score for all child nodes is returned to a 'white' node and the maximum score of all child nodes is returned to a 'black' node, a white node will select '6' from the two resulting child nodes '6' and '9' as '6' is the lowest number.

If black nodes on the same level return values less than '6' then a value of less than '6' will be returned to the white parent node. In the above example a value of '2' is returned by the deepest level. The black level above the white node will select the highest value from its child nodes, and in the example the value '6' will be selected. Knowing the value in the white node, when more scores are calculated on the deepest level (referring to the diagram above) they can be compared against this value. If a value less than the value in the first white node is found then this new value, or a value less than this, will be returned to the white node. It is not necessary to continue processing any more nodes because they will not be considered and additional branches are 'pruned'. The alpha-beta search tree therefore reduces to the following:

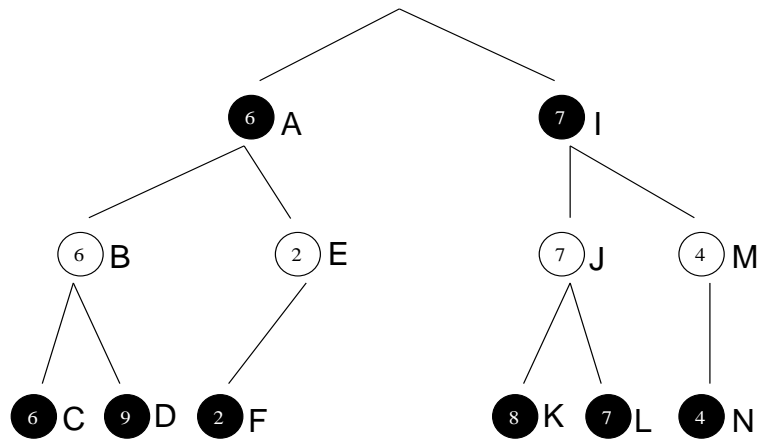


Figure 9.3: The Alpha Beta search tree after 'pruning'.

In the above example, if a value of '2' is returned to node 'E' It is not necessary to find other scores because only a lower score than '2' will be passed back to node 'E'. But as node 'A' will choose the highest score (as it is a black node) from nodes 'B' and 'E', node 'A' will select '6'.

Nodes 'G' and 'H' can be skipped as whatever their score is it will have no influence on the selection. By optimising in this way considerable savings can be made, depending on the order in which the nodes are examined.

The advantage of the alpha-beta algorithm over the MINIMAX routine is that instead of evaluating every possible solution alpha-beta will 'prune' some branches "to speed up the search processes without loss of information" (Knuth & More 1979), resulting in a saving in processing time. However, the order in which branches are processed is crucial to the efficiency of the alpha-beta process. If the branches are processed 'worst move first' then no branches will be pruned, exploring all branches at each depth. Assuming a search depth of 'd' and a *branching factor*²⁴ of 'b', the number of tip nodes will be:

²⁴ In relation to chess, the 'branching factor' is the number of moves that can be made on the chessboard.

$$\begin{aligned} \text{nodes} &= b \times b \times b \times b \times b \dots \\ &= b^d \end{aligned}$$

If however the branches are processed 'best move first' then the first level consists of just one branch (the best move). Depth 'two' explores all possible positions resulting from the first move. Depth 'three' selects the perfect move for each configuration; depth 'four' explores all possible moves resulting from the previous move and so on. Assuming a search depth of 'd' and a branching factor of 'b' the number of tip nodes will be:

$$\begin{aligned} \text{nodes} &= 1 \times b \times 1 \times b \times 1 \dots \\ &= b^{d/2} \text{ (assuming an } \textit{even} \text{ search depth)} \\ &= \sqrt{b^d} \end{aligned}$$

By sorting the order in which branches are processed to 'best move first' the number of tip nodes reduce to the square root of the number of nodes evaluated in the 'worst move first' case. The number of nodes in practice lies somewhere between the two extremes as the order of processing is normally between the 'best' and 'worst' cases. A randomly ordered search would, on average, achieve a mid-position between best and worse. It can be shown that "a random ordered search reduces the average branching factor by approximately $\sqrt[4]{b^3}$ " (Nilsson 1998). Most chess programs apply heuristics that are specific to the rules of chess to order the moves applied to the alpha-beta search to improve the efficiency of the process. The research reported in this thesis uses chunking to sort the moves into an order of likelihood to be played. The chunking method has no intrinsic knowledge of the rules of chess but from

matching chunks (or groups) of chess pieces with similar chunks found on previously analysed chessboards the likelihood of a move can be estimated. This thesis uses the game of chess as an example of how chunking, without any knowledge of the domain, can improve the efficiency of an alpha-beta search.

9.3. Using CLAMP to optimise the alpha-beta search

“While current computers search for millions of positions a second, people hardly ever generate more than a hundred. Nonetheless, the best human chess players are still as good as the best computer programs” (Saariluoma 1995, pp 116).

The quotation from Saariluoma’s 1995 paper (op. cit.) suggests that expert human chess players employ a narrow search when selecting a move. This chapter attempts to emulate this (the narrowing of the search) by using chunking to optimise the alpha-beta search process within a conventional chess program. The chess program will model human behaviour by recognising chunk patterns, focusing on just a few pieces from the entire board. This approach, when describing human expert chess players, was documented by Adriaan de Groot (1978):

“Four distinct stages in the task of choosing the next move were noted. The first stage was the phase of orientation, in which the subject assessed the situation and determined a very general idea of what to do next. The second stage, the phase of exploration was manifested by looking at some branches of the game tree. The third stage, or phase of investigation, resulted in the subject choosing a probable best move. Finally, in the fourth stage, the phase of proof, saw the subject convince himself or herself that the results of the investigation were correct.”

The aim is to test whether chunk knowledge within a chess program improves the efficiency of a chess engine by reducing the width of a search tree. Chunk knowledge can be used to optimise the chess program by restricting move look-ahead to a selected subset of pieces and thereby narrowing the search tree of an alpha-beta routine. The chess program would model the expert player by restricting look-ahead to just some branches of the game tree. To find the best move, a chess program will test each possible move with the corresponding counter moves several ply ahead. The order in which moves are tried will have a very significant influence on the effectiveness of alpha-beta cut-offs.

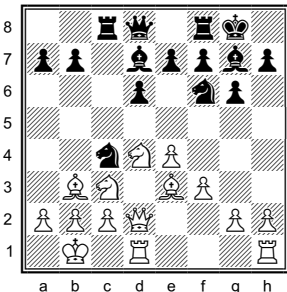
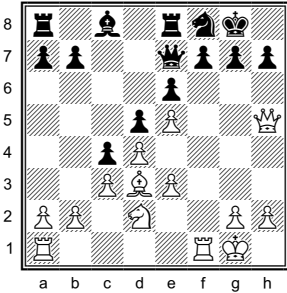
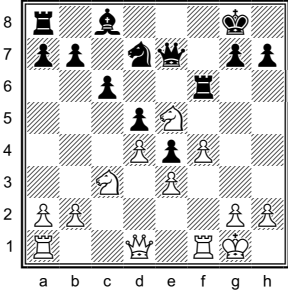
The chess program 'Beowulf' (a version of which was used in the ChessBrain project (Frayn 2006)) was modified to include move ordering from an analysis of chunks. The hybrid of Beowulf and CLAMP was then used to calculate chess moves. When making a move the chess engine compiled a list of three-piece chunks and then scored each legal move based on the starting configuration of the chessboard. The scores were used to order the sequence in which moves were applied to the alpha-beta search routine.

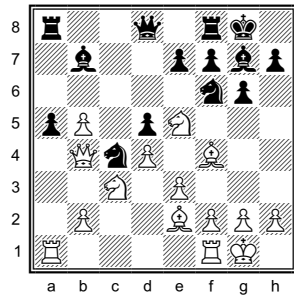
The chess engine was programmed to look ahead by four-ply and count the number of branches produced. The alpha-beta search in normal operation will move a piece to model moves and counter-moves, changing the original chessboard configuration, and therefore the chunks contained therein, with each depth of search. As the depth of search was limited to four-ply the impact of the changing configuration was small as the majority of pieces stayed in their original positions. The same move ordering was therefore applied at every node in the search.

A sample of one thousand chessboards from the mid-game section from tournament games were processed by the chess engine and the number of branches recorded. Care was taken to ensure that the sample chessboards were taken from a

dataset that was not used to build the chunk libraries. The same chessboards were processed again but this time with a *random order* applied to the move ordering. The random ordering should, on average, give a middle position between a ‘best move first’ and ‘worst move first’ move ordering. A comparison between the numbers of nodes searched when using ‘chunking ordered’ as opposed to a ‘randomly ordered’ sequence in which branches were processed was performed.

The examples in table 9.1 show typical example chessboard configurations and the number of nodes searched with random and sorted moves.

Chessboard configuration:	Random ordered	Chunk-sorted ordering	Saving in branches processed
	601996	270000	55.15%
	83376	50413	39.54%
	175607	64594	63.22%



352044 300572 12.15%

Table 9.1: Sample results from comparison analysis

The percentage decrease in the number of branches searched was calculated for each of the one thousand sample chessboards used in the analysis reported in chapter 7 and the results collated in grouped intervals. The results are reported in table 9.2 below.

Reduction	Frequency	Reduction	Frequency	Reduction	Frequency
-150 %	1	-65 %	3	20 %	18
-145 %	1	-60 %	3	25 %	22
-140 %	0	-55 %	0	30 %	25
-135 %	0	-50 %	3	35 %	32
-130 %	2	-45 %	2	40 %	30
-125 %	0	-40 %	2	45 %	42
-120 %	0	-35 %	8	50 %	48
-115 %	2	-30 %	8	55 %	41
-110 %	0	-25 %	7	60 %	48
-105 %	1	-20 %	6	65 %	74
-100 %	2	-15 %	12	70 %	80
-95 %	1	-10 %	9	75 %	84
-90 %	3	-5 %	13	80 %	84
-85 %	2	0 %	11	85 %	95
-80 %	4	5 %	15	90 %	67
-75 %	4	10 %	19	95 %	31
-70 %	1	15 %	25	100 %	0

Table 9.2: Results of analysis of chessboards showing the number of chessboards (Frequency) against each percentage reduction category.

Table 9.2 shows the results from the analysis of one thousand chessboards showing the percentage reduction in intervals and the number of instances (Frequency) within

each interval. Considering the results overall, the 'chunk ordered' process performed better than the 'random ordered' process. The graph in figure 9.4 shows the number of chessboards collated in each percentage interval.

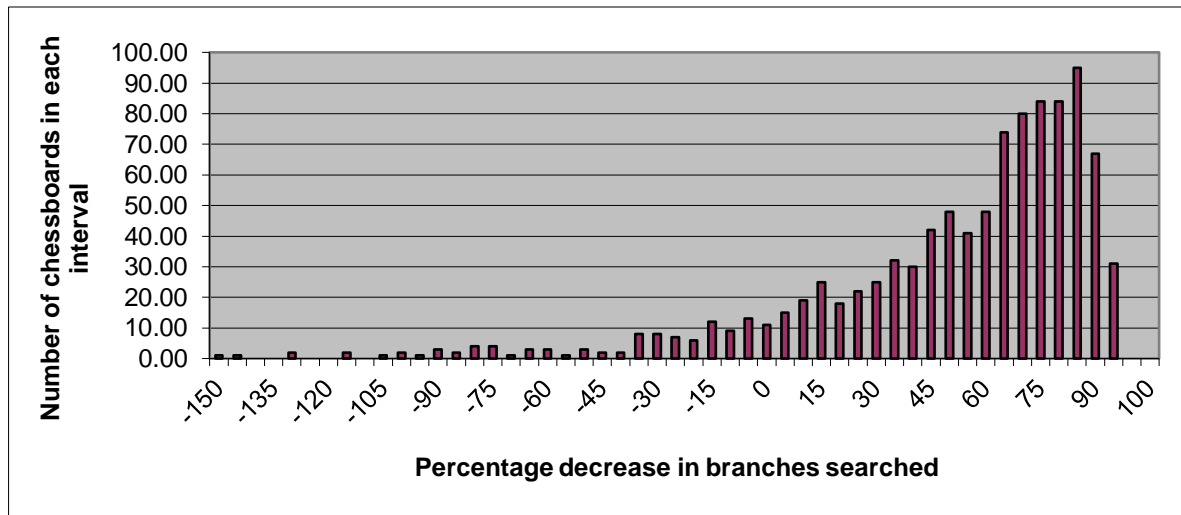


Figure 9.4: Graph showing the number of chessboards in each percentage interval

Using chunking based on three-piece chunks (from an analysis of the whole board area and without move rareness adjustment) to order the processing of branches in an alpha-beta search, a significant reduction in the number of branches searched was achieved when compared with a search based on random ordered moves. The majority (over 80%) of cases gave a decrease in the number of branches searched when using chunk ordered moves compared with a random ordering. The results show the highest number of cases (ninety-five chessboards, or 9.5% of the sample) giving a percentage decrease within the interval range of 85% to 90%. The mean percentage decrease for all one thousand cases was found to be 50.8%

The results shown above are based on an analysis using *three-piece chunks*. The chessboards were analysed using four and five piece chunks and the results tabulated as follows:

	3 Piece chunk	4 Piece chunk	5 Piece chunk
Mean % decrease:	50.75%	48.77%	49.58%

Table 9.3: Comparison of chunk size on search improvement

The same analysis was performed with three piece chunks but applying the output from CLAMP in *reverse* order, resulting in an increase of branches searched by a factor of 880 times.

9.4. Chapter conclusion

This chapter reported an application of chunking to optimise an alpha-beta search routine. It should be noted that the chunks were built *without any knowledge of the rules of chess or properties of the pieces*, yet despite this a significant reduction in the number of nodes searched was achieved when compared with an alpha-beta search using random ordered pieces. It is possible that similar techniques could be applied to searching algorithms in other domains where the parameters that define the domain are unknown.

10. CLAMP AND CHUMP: A COMPARISON

This chapter will make a comparison between CLAMP (cf. page 51) and CHUMP (CHUnking of Moves and Patterns - Gobet and Jansen 1994). As both programs analyse chess games to extract chunks and associate the chunks with moves, and use chunk knowledge to select chess moves, it was considered appropriate to include a chapter within the thesis to highlight the differences and similarities between the two programs.

A number of key points are discussed in the following paragraphs.

10.1. The aims of the programs

CLAMP was designed to investigate the properties of chunks that can be associated with chess moves. The size of chunks was varied to investigate the effectiveness of chunk size on the effectiveness of the chunks with respect to selecting a likely move. In addition, the relationship of the pieces within a chunk were investigated to compare the effectiveness of chunks compiled under various scenarios. CHUMP on the other hand, uses a combination of chunk sizes and filter methods to select the pieces which constitute a chunk. CHUMP is described as “a chess program”, which “models human ‘computational’ mechanisms” (Gobet and Jansen op. cit.). The discussion of CHUMP in the aforementioned paper seeks to aid understanding of the chunking mechanism in humans, and propose these methods as possible additions to competitive game playing programs.

10.2. Chunk acquisition

Both CHUMP and CLAMP acquire knowledge by the examination of experts' chess games, analysing the chunks (chess piece constellations) on the board and

associating the chunks to the move that was played. There are, however, a number of differences in methods between CHUMP and CLAMP.

CLAMP has three basic methods to find the chunks that are present on the board as follows:

10.2.1 Analysis of the whole board.

All of the pieces on each of the sixty-four squares on the chessboard are combined. Frequently occurring combinations are saved as chunks. This method ensures that all possible chunk patterns are found but it has the disadvantage that a large number of chunks are processed, many of which may not be relevant to the chess move to be played (cf. page 95). In this mode, CLAMP has no prior knowledge of the rules of chess

10.2.2 Analysis of local areas on the board

As many effective chunks consist of pieces that are in close proximity to each other CLAMP can analyse the chessboard to build chunks of pieces that exist within an area of 3x3, 4x4, 5x5, 6x6 or 7x7 squares (cf. page 98).

10.2.3 Analysis of pieces in 'defensive' relationships to each other.

In this mode, CLAMP will build chunks consisting only of pieces that are in defensive relationships with each other (cf. page 102).

Only one of the above options can be selected at a time. CLAMP can therefore compare the analysis of the same chessboards to measure the effectiveness of each method.

CHUMP uses a combination of methods to select the pieces on the chessboard which are processed as chunks as follows:

10.2.4 The 'eye-movement-simulator'.

CHUMP uses an 'eye-movement-simulator' to scan the chessboard directing 'attention' to twenty positions on the chessboard where pieces are expected to exist, based on the saccades of expert chess players. The eye-movement-simulator aims to give more meaning to chunks by applying expert's knowledge of expected chess piece positions on the board.

10.2.5 Attack, defence and proximity relationships

In addition to eye movement knowledge, CHUMP uses knowledge of attack, defence and proximity relationships to select relevant pieces to build meaningful chunks.

10.3 Chunk repository

In both CHUMP and CLAMP, chunks are associated with moves to be played. With CLAMP the association of a chunk and a move is the presence of a chunk within a 'library file'. CLAMP builds a library file for every possible move for each piece on the chessboard; if a chunk is associated with the move then the chunk will exist within the file, or indeed, within several files. CHUMP uses two discrimination nets based on the EPAM model of memory and perception (Feigenbaum and Simon, 1984). One discrimination net, containing the chunk information, with the chunks referencing a second net which contains the proposed moves.

10.4 Chunk Size

CHUMP recognised chunks with up to twenty pieces, although it has been found earlier in this thesis that frequently occurring observations of chunks with a size greater than ten pieces are rare within chess games (cf. page 62), indeed, estimates for the maximum number of items retained in human short-term memory are thought to be considerably lower than twenty (but excluding chunks recalled as a result of template learning).

CHUMP uses a combination of all chunk sizes at the same time. CLAMP, on the other hand, only works with one chunk size and, for technical reasons, the largest chunk size investigated as part of this research is seven pieces. Chunks of discrete sizes between three and seven were processed. A comparison of the effectiveness of chunk sizes has enabled inferences to be made about the number of pieces that make effective chunks.

The different approach regarding use of multiple chunk sizes between CHUMP and CLAMP is consistent with the purpose of the respective programs, that being as a model of human memory (CHUMP) or as a tool to investigate chunking parameters (CLAMP).

10.5 Move proposals

Normally when analysing a chessboard several relevant chunks will be found. Each chunk will be associated to one or more moves. CLAMP will propose the move to play that is supported simply by the highest number of chunks. CHUMP is more complex in the move proposal method, selecting on the move that is also supported by the highest number of chunks, but also by the number of times the chunk has been 'activated' within the training data (a chunk is activated each time the chunk is 'seen' within the training data).

It is possible that the performance of CLAMP would be improved if CLAMP similarly used the chunk activation as a factor in the move proposal logic. CLAMP uses a simple Boolean method with regard to chunk activation in that for a chunk to be significant its number have been activated on at least one per cent of the training board data. Infrequent chunks are not considered in the move proposal process if the chunk does not appear on at least one per cent of the training boards.

The implementation of CLAMP for this thesis is able to suggest moves where white is to play only as only 'white' moves were analysed (cf. page 56). CHUMP is able to suggest moves for both White and Black to play.

10.6 The move start and end squares

CHUMP associates chunks with a move of a piece from a starting square. CLAMP associates chunks with a move of a piece to a square (the starting position of the piece is considered to be implicit for most moves).

10.7 The size of the learning set.

The learning set used for CHUMP consisted of 300 games from just one Grandmaster (the former world champion, Mikhail Tal). Using CHREST (cf. page 18) to build a discrimination net of piece/positions arranged in a net (graph) object to associate chunks with moves. The training data for CLAMP was considerably larger, using in the order of 1.5 million games. The dataset comprised of tournament games between numerous Grandmasters.

10.8 Test results from the Bratko-Kopec positions

CHUMP was tested on the Bratko-Kopec positions, comparing the moves suggested by CHUMP with the 'Bratko-Kopec' best moves. CHUMP was able to score about 50% of the test boards, with the best move being in the top four suggested moves

after training with just 300 games. CHUMP achieved a 4.2% success at predicting the best move as the first choice. CLAMP was able to suggest moves for each of the boards tested but was unsuccessful in suggesting the best move within any of the top four moves.

As CLAMP is only able to suggest moves for 'white to play' only twelve of the Btатko-Kopec moves could be tested with CLAMP.

Both CLAMP and CHUMP achieved better results when scoring boards that resulting in a pawn-lever type move.

10.9 Test results from de Groot's Position 'A'

CLAMP and CHUMP were both tested using de Groot's Position 'A' configuration.

Neither program selected the best move (Bxd5), however, CLAMP's first suggestion was Rd1, and second suggestion Re1. Both of these moves are rated highly by the Fritz program.

CHUMP suggested g2-g3 and h2-h3 as its top two suggestion, either of which are described as 'not so bad' (Gobet and Jansen 1994).

10.10 Conclusion

CHUMP and CLAMP are two programs that analyse chunk patterns from expert player's games, in order to suggest a move to be played, based on the board having a similar chunk constituents. The design and function differences between CLAMP and CHUMP because each program was designed to perform a different task.

CHUMP is a model of human memory, selecting moves using a combination of abilities to emulate human cognition. CLAMP however is a program designed to investigate the properties of chunking in chess. CLAMP allows a comparison of the effectiveness of chunking methods as the chunk parameters are varied.

The size of the training set used for CLAMP was substantially larger than the training data used for CHUMP. From the results obtained from CLAMP it is likely that a considerable improvement in the performance of CHUMP could be gained by increasing the size of the training data appropriately. Despite the relatively small chunk knowledge that CHUMP acquires, CHUMP performs well, suggesting moves based on an analysis of the chunks found on the board.

CLAMP compares the effectiveness of the proposed moves with various chunk parameters, such as chunk size or the relationship between pieces. CLAMP is not designed to be a chess playing program, but instead, CLAMP is a tool to investigate the properties of chunks.

11. CONCLUSION

11.1 About chunking in chess

The research reported in this thesis describes original work, which seeks to gain understanding as to the nature of ‘chunks’ of chess pieces and their use by skilled players within the game of chess. The results reported show that chunks are present in large numbers within the piece configurations of typical chessboards. There are many chunks that are frequently occurring within chess games, a number of which have been isolated using CHREST (and supplied by Gobet) or by CLAMP as described in chapter 5 of this thesis. A simple analysis of the number of chunks found on the chessboard show that chunks can be associated with the stage of the game in terms of the number of moves prior to checkmate. The results reported in chapter 5 investigated the connection between the skill of the player and the number of chunks used in their chess play, however, no significant correlation between player skill and the number of chunks could be found.

The program *CLAMPanalyser* described in chapter 6 analysed chunks in more detail by associating chunks to the next move of a piece to a position on the chessboard. Testing the moves played by chess experts with the suggested moves from *CLAMPanalyser* consistently gave results that were better than a random choice of moves. Applying simple heuristics when extracting chunks, such as building chunks from pieces that are in close proximity, or restricting chunks to pieces that are in defending relationships, dramatically reduced the number of significant chunks that were required by *CLAMPanalyser* to suggest a move. Furthermore, selecting the size of the chunks (that is, the number of chess pieces that make up the chunk) can influence the probability of an accurate prediction of the best moves.

The results obtained from CLAMP and *CLAMPanalyser* showed that as the chunk size increased the accuracy of the likelihood prediction improved, however the

chance of finding large chunks on a chessboard is small. Chunk sizes of above five pieces in the mid-game section were rarely repeated from one game to another. Smaller chunks with, for example, three pieces were prevalent between games.

Adding additional knowledge to the chunk, such as the knowledge that many chunks are made from pieces that are in close proximity (cf. page 98), or that many of the pieces within a chunk are in defending relationships (cf. page 102), will, as stated above, considerably reduce the number of chunks in the chunk libraries without loss of accuracy of the move likelihood prediction. A 'four by four' square area on the board will reduce the number of chunks in the chunk library to less than 1% of the size when compared to a similar analysis for the whole board area. Furthermore, considering chunks that consist only of pieces in defending relationships, the number of chunks contained within the library reduces to 0.5% compared with the number of chunks within the 'whole board' library. We can therefore conclude that chunks of three or four pieces that are in close proximity, or are in defending relationships with each other, are useful for calculating the likelihood of a piece to be moved.

Chapter 8.3 reports a practical application of chunking. It has been possible to use chunking to order the pieces prior to applying to an alpha-beta search with a resulting decrease in the number of nodes searched. The reduction in search width was confirmed with one thousand chessboards, giving credence to the notion that chunking is an effective method to focus attention on the significant moves. The application of chunking in this example operates without knowledge of the rules of chess and used a chunk size of three pieces to a depth of four ply. An average reduction of 50% in the number of leaf nodes was achieved.

11.2 The question this thesis aims to answer

Chunking appears, even in the simplistic context described in this thesis, to yield reasonable predictive dividends with small chunk library inventories. The results obtained are consistent with psychological experiments that propose that human chess players are aware (either consciously or subconsciously) of the existence of chunks and their presence can direct an expert player to relevant moves. The research reported adds additional material to the argument for chunking being an aspect of an expert's cognitive processing. The question this thesis is trying to answer is however more complex. It is not possible to answer the question "*does the utilization of chunks within a chess-playing program provide a plausible model for the use of chunks by human players*" (cf. page 9) with an affirmative, as it is not possible to draw conclusions about the cognitive processes within a human player from the experiments in this thesis. However the thesis reports original findings, and therefore makes an original contribution to knowledge, regarding the nature of chunks within the chess domain and the plausibility of chunking as a mechanism for directing attention to relevant chess moves. The findings are indeed indicative, taking into account human limitations, that chunking is a technically viable process for the human expert.

The analysis performed and programs written were simplistic in nature, detecting chunks based on frequency of occurrence and crude heuristics (such as restricting the formation of chunks to a small area). These simple filters drastically reduced the number of chunks found, without a loss in accuracy. It is possible that the advanced 'filtering' that human expert could perform may select even fewer, but even more significant chunks. The results, even with the simple filtering, show that associating chunks with moves is a viable process, albeit with a limited probability,

therefore showing that knowledge of chunks can direct attention to moves that are likely to be played.

The program 'CLAMPAnalyser' employs a software implementation of a simple *recognition-association* technique. The software does not use complex search heuristics or statistical analysis but by the simple *recognition* of chunks (within the chunk libraries) and their *associations* with moves CLAMPAnalyser selects possible 'good' moves. The *recognition-association theory* is a descriptive term used by Holding to describe chunking as follows:

"In more detail, the recognition-association theory makes the assumption that chess mastery stems from knowing thousands of chess patterns. Recognition of one of these patterns during play is said to trigger the memory of an associated plausible move, which may then be selected or investigated by the player" (Holding 1992).

The results obtained by CLAMP show that this technique is a feasible method for associating chunks to moves. The only caveat to Holdings definition of recognition-association is that CLAMP's evaluation of moves is based on the recognition of *many* chunk patterns. Chunking is proposed to be a framework that "underlies many aspects of human learning" and if CLAMP models cognitive *perceptual chunking* then the research reported in this thesis shows that chunking is achievable with interlinked and associated memories (Gobet et al. 2001).

The advantage of using a computer program as a tool to investigate chunking is that the parameters and methods are exactly defined, as opposed to psychological experiments where the actual cognitive process is unknown. A psychological experiment may suggest that chunking is evident but the exact cognitive process is unknown and can only be proposed by inference. CLAMP however shows that frequently occurring configurations of three to six pieces *can* be used as an indicator for a move. CLAMP therefore shows that a human chess player *could* use a similar

method to focus the attention of a chess player onto a few moves. Chunk sizes of less than seven pieces have been shown to give measurable results which agree with Miller's (1956) hypothesis for chunking in human subjects, indeed chunk sizes of four or less yield a positive correlation between chunk configurations and the move made which is consistent with Cowan's (2001) paper. The positive results obtained by CLAMP using chunks of four pieces or less show that Cowan's measurements for human capacity limits are viable within the domain of chess. In addition, the size of the inventory is within a plausible range when considering the capacity of a human expert (Gobet and Lane 2010), for example a library consisting of 147,510 five piece 'defensive' chunk constellations offers a 15% likelihood of predicting one of the top four best moves. As mentioned above, the crude heuristics applied to the chunk library building process, such as only processing pieces that are in close proximity or pieces that are in defensive relationships, considerably reduce the number of chunks contained within the chunk libraries. It is plausible that an expert chess player will acquire chunk knowledge using considerably more efficient heuristics which would reduce the inventory of chunks even further.

12. RECOMMENDATIONS FOR FURTHER WORK

The results reported in this thesis show that chunks are present within chessboard configurations and, from a very simple analysis of chessboards taken from tournament games it can be shown that chunks can be used as a differentiator of skill groups (cf. page 44). Furthermore, by associating chessboards and their constituent chunks to the move that follows, chunks can be associated with moves. The program 'CLAMP' (cf. page 51) was developed to build libraries of chunks that are associated with moves and the program 'CLAMPanalyser' (cf. page 82) will list all possible moves in order of the number of chunks that support the move. In addition to combining pieces from the entire chessboard, simple heuristics have been applied to limit the scope of pieces that are grouped into chunks as well as the number of pieces that make up a chunk (cf. page 92). A filter on which pieces are combined to make chunks can reduce the number of chunks by over 99% with only a small decrease in the effectiveness of the process. The additional filtering of chunks is, in effect, adding knowledge to the chunk, which reduces the number of chunks by removing many unnecessary chunks from the library without significantly reducing the accuracy of CLAMPanalysis in selecting moves. Further work with CLAMP could focus on improving the filtering to restrict the pieces included in building chunks. If a human chess player performs a similar filtering process then, speculatively, the filter parameters could for example include knowledge of the rules of the game, and tactical experience. In this research filters were applied to chunks based on the proximity of pieces and the defending relationship of pieces. Further work could research the development of filters that include more aspects of chess knowledge.

The chunk libraries produced by CLAMP include the move associations and consequently encapsulate some domain knowledge (cf. page 76). It may be possible to apply production rule, or expert system, processing techniques to the chunk data.

The chunk libraries in this context would be termed the ‘training set’ (Liu and White 1991), however, as the training set consists of a very large number of items (the chessboard contains a large number of chunks) the process may be prohibitively complex.

Although CLAMP was developed for chess research it would not be difficult to modify the program for analysis in other domains, such as image processing and context aware object recognition. It is understood from literature on “visual cognition and cognitive neuroscience that the human and animal visual systems use these relationships [context awareness] to improve their ability of categorization” (Perko and Leonardis 2010). Using current techniques, image recognition is computationally complex and is generally considered to be an unsolved problem. Recognition of an object in the context of its surroundings can in some ways be considered a similar problem to playing the right move in chess, as for example parallels have been made between the work of a radiologist and the thinking of a chess master when diagnosing an x-ray film (Wood 2009). Development of systems employing recognition / association techniques for image analysis could have practical applications within fields such as medical diagnosis, robotics and security.

13. APPENDICES

13.1. Appendix 1: Summary of results

Library Type	From board area	Chunk Size	Library size (chunks)	50% null hypothesis %Success	Chunk success probability	Top 4 %Success	Top 4 success 'adjusted'	% Effect	SE
Area	3x3	2	114943	57.9%	85.1%	12.9%	15.2%	68.0%	7.16%
Area	3x3	3	80819	57.9%	82.0%	16.0%	19.5%	70.6%	9.21%
Area	3x3	4	23855	42.4%	54.2%	16.2%	29.9%	78.2%	11.38%
Area	3x3	5	3944	20.5%	22.3%	10.2%	45.7%	91.9%	20.01%
Area	3x3	6	-	-	-	-			
Area	3x3	7	-	-	-	-			
Area	4x4	2	253476	57.7%	87.2%	15.7%	18.0%	66.2%	7.81%
Area	4x4	3	343409	59.0%	84.0%	17.7%	21.1%	70.2%	8.87%
Area	4x4	4	210120	52.1%	68.2%	18.5%	27.1%	76.4%	11.39%
Area	4x4	5	85679	39.3 %	49.9%	15.2%	30.5%	78.8%	12.19%
Area	4x4	6	26139	18.6%	21.0%	9.0%	42.9%	88.6%	17.17%
Area	4x4	7	-	-	-	-			
Area	5x5	2	453492	63.9%	90.5%	14.5%	16.0%	70.6%	7.16%
Area	5x5	3	998254	59.0%	84.9%	14.8%	17.4%	69.5%	8.62%
Area	5x5	4	1057802	57.5%	78.6%	15.3%	19.5%	73.2%	9.33%
Area	5x5	5	726212	48.1%	65.6%	15.8%	24.1%	73.3%	9.57%
Area	5x5	6	388865	39.0%	51.8%	16.9%	32.6%	75.3%	12.07%
Area	5x5	7	-	-	-	-			
Area	6x6	2	721550	58.5%	87.5%	12.4%	14.2%	66.9%	7.25%
Area	6x6	3	2593877	61.1%	85.5%	13.1%	15.3%	71.5%	8.34%
Area	6x6	4	4918042	58.7%	84.3%	12.9%	15.3%	69.6%	8.33%
Area	6x6	5	6179518	53.5%	76.3%	14.2%	18.6%	70.1%	8.95%
Area	6x6	6	4621214	39.0%	54.9%	13.0%	23.7%	71.0%	10.78%
Area	6x6	7	-	-	-	-			
Area	7x7	2	-	-	-	-			
Area	7x7	3	-	-	-	-			
Area	7x7	4	14525807	61.2%	84.5%	13.7%	16.2%	72.4%	8.12%
Area	7x7	5	43904506	-	-	-			
Area	7x7	6	-	-	-	-			
Area	7x7	7	-	-	-	-			
All Board	8x8	2	1261864	69.7%	99.5%	17.0%	17.1%	70.1%	6.27%
All Board	8x8	3	7596060	70.1%	99.5%	16.2%	16.3%	70.5%	7.11%
All Board	8x8	4	27127049	69.7%	99.5%	17.3%	17.4%	70.1%	7.88%
All Board	8x8	5	45646773	39.6%	56.7%	13.9%	24.5%	69.8%	10.72%
All Board	8x8	6	-	-	-	-			
All Board	8x8	7	-	-	-	-			
Defensive	8x8	2	44266	69.6%	99.5%	12.9%	13.0%	69.9%	6.38%
Defensive	8x8	3	45096	64.4%	95.4%	14.2%	14.9%	67.5%	7.04%
Defensive	8x8	4	119857	66.6%	96.5%	15.0%	15.5%	69.0%	7.57%
Defensive	8x8	5	147510	59.3%	82.5%	15.1%	18.3%	71.9%	8.54%
Defensive	8x8	6	158275	53.8%	71.4%	15.6%	21.8%	75.4%	9.32%
Defensive	8x8	7	129313	33.8%	42.7%	12.4%	29.0%	79.2%	12.07%

Table 12.1: A summary of results from various chunk library scenarios

Library Type	From board area	Chunk Size	(p-1) percentage	(p-2) percentage	(p-3) percentage	(p-4) percentage	(p-4) Adjusted
Area	3x3	2	3.9	6.8	9.4	12.9	15.2%
Area	3x3	3	5.3	10.1	13.6	16.0	19.5%
Area	3x3	4	5.8	10.4	14.2	16.2	29.9%
Area	3x3	5	4.0	7.6	8.7	10.2	45.7%
Area	3x3	6	-	-	-	-	
Area	3x3	7	-	-	-	-	
Area	4x4	2	4.1	9.1	12.3	15.7	18.0%
Area	4x4	3	4.8	9.4	14.1	17.7	21.1%
Area	4x4	4	5.3	10.4	14.1	18.5	27.1%
Area	4x4	5	5.0	8.8	11.7	15.2	30.5%
Area	4x4	6	3.3	5.7	7.4	9.0	42.9%
Area	4x4	7	-	-	-	-	
Area	5x5	2	3.5	7.0	10.8	14.5	16.0%
Area	5x5	3	5.0	8.1	10.7	14.8	17.4%
Area	5x5	4	3.8	8.3	11.7	15.0	15.5%
Area	5x5	5	4.7	8.9	12.3	15.8	24.1%
Area	5x5	6	4.0	9.6	13.1	16.9	32.6%
Area	5x5	7	-	-	-	-	
Area	6x6	2	3.3	5.9	9.8	12.4	14.2%
Area	6x6	3	3.7	6.5	10.1	13.1	15.3%
Area	6x6	4	3.5	6.0	9.8	12.9	15.3%
Area	6x6	5	4.0	7.8	10.5	14.2	18.6%
Area	6x6	6	4.6	7.6	10.2	13.0	23.7%
Area	6x6	7	-	-	-	-	
Area	7x7	2	-	-	-	-	
Area	7x7	3	-	-	-	-	
Area	7x7	4	3.7	7.1	10.1	13.7	16.2%
Area	7x7	5	-	-	-	-	
Area	7x7	6	-	-	-	-	
Area	7x7	7	-	-	-	-	
All Board	8x8	2	5.2	8.6	12.5	17.0	17.1%
All Board	8x8	3	5.2	9.5	13.2	16.2	16.3%
All Board	8x8	4	5.2	9.7	13.5	17.3	17.4%
All Board	8x8	5	4.2	7.6	10.7	13.9	24.5%
All Board	8x8	6	-	-	-	-	
All Board	8x8	7	-	-	-	-	
Defensive	8x8	2	3.5	6.2	9.9	12.9	13.0%
Defensive	8x8	3	3.6	7.4	10.2	14.2	14.9%
Defensive	8x8	4	3.8	8.3	11.7	15.0	15.5%
Defensive	8x8	5	4.9	8.6	12.7	15.1	18.3%
Defensive	8x8	6	5.0	9.0	11.7	15.6	21.8%
Defensive	8x8	7	4.0	7.4	10.4	12.4	29.0%

Table 12.2: A summary of 'Top4' moves by analysis type

13.2. Appendix 2: The key for the axis: 'Move to piece/position'.

1	Pg8	41	Pg3	81	Rg4	121	Nd5	161	Bd8	201	Bd3	241	Kd6	281	Kd1	321	Qd4
2	Ph8	42	Ph3	82	Rh4	122	Ne5	162	Be8	202	Be3	242	Ke6	282	Ke1	322	Qe4
3	Pa7	43	Ra8	83	Ra3	123	Nf5	163	Bf8	203	Bf3	243	Kf6	283	Kf1	323	Qf4
4	Pb7	44	Rb8	84	Rb3	124	Ng5	164	Bg8	204	Bg3	244	Kg6	284	Kg1	324	Qg4
5	Pc7	45	Rc8	85	Rc3	125	Nh5	165	Bh8	205	Bh3	245	Kh6	285	Kh1	325	Qh4
6	Pd7	46	Rd8	86	Rd3	126	Na4	166	Ba7	205	Ba2	246	Ka5	286	Qa8	326	Qa3
7	Pe7	47	Re8	87	Re3	127	Nb4	167	Bb7	207	Bb2	247	Kb5	287	Qb8	327	Qb3
8	Pf7	48	Rf8	88	Rf3	128	Nc4	168	Bc7	208	Bc2	248	Kc5	288	Qc8	328	Qc3
9	Pg7	49	Rg8	89	Rg3	129	Nd4	169	Bd7	209	Bd2	249	Kd5	289	Qd8	329	Qd3
10	Ph7	50	Rh8	90	Rh3	130	Ne4	170	Be7	210	Be2	250	Ke5	290	Qe8	330	Qe3
11	Pa6	51	Ra7	91	Ra2	131	Nf4	171	Bf7	211	Bf2	251	Kf5	291	Qf8	331	Qf3
12	Pb6	52	Rb7	92	Rb2	132	Ng4	172	Bg7	212	Bg2	252	Kg5	292	Qg8	332	Qg3
13	Pc6	53	Rc7	93	Rc2	133	Nh4	173	Bh7	213	Bh2	253	Kh5	293	Qh8	333	Qh3
14	Pd6	54	Rd7	94	Rd2	134	Na3	174	Ba6	214	Ba1	254	Ka4	294	Qa7	334	Qa2
15	Pe6	55	Re7	95	Re2	135	Nb3	175	Bb6	215	Bb1	255	Kb4	295	Qb7	335	Qb2
16	Pf6	56	Rf7	96	Rf2	136	Nc3	176	Bc6	216	Bc1	256	Kc4	296	Qc7	336	Qc2
17	Pg6	57	Rg7	97	Rg2	137	Nd3	177	Bd6	217	Bd1	257	Kd4	297	Qd7	337	Qd2
18	Ph6	58	Rh7	98	Rh2	138	Ne3	178	Be6	218	Be1	258	Ke4	298	Qe7	338	Qe2
19	Pa5	59	Ra6	99	Ra1	139	Nf3	179	Bf6	219	Bf1	259	Kf4	299	Qf7	339	Qf2
20	Pb5	60	Rb6	100	Rb1	140	Ng3	180	Bg6	220	Bg1	260	Kg4	300	Qg7	340	Qg2
21	Pc5	61	Rc6	101	Rc1	141	Nh3	181	Bh6	221	Bh1	261	Kh4	301	Qh7	341	Qh2
22	Pd5	62	Rd6	102	Rd1	142	Na2	182	Ba5	222	Ka8	262	Ka3	302	Qa6	342	Qa1
23	Pe5	63	Re6	103	Re1	143	Nb2	183	Bb5	223	Kb8	263	Kb3	303	Qb6	343	Qb1
24	Pf5	64	Rf6	104	Rf1	144	Nc2	184	Bc5	224	Kc8	264	Kc3	304	Qc6	344	Qc1
25	Pg5	65	Rg6	105	Rg1	145	Nd2	185	Bd5	225	Kd8	265	Kd3	305	Qd6	345	Qd1
26	Ph5	66	Rh6	106	Rh1	146	Ne2	186	Be5	226	Ke8	266	Ke3	306	Qe6	346	Qe1
27	Pa4	67	Ra5	107	Nf7	147	Nf2	187	Bf5	227	Kf8	267	Kf3	307	Qf6	347	Qf1
28	Pb4	68	Rb5	108	Ng7	148	Ng2	188	Bg5	228	Kg8	268	Kg3	308	Qg6	348	Qg1
29	Pc4	69	Rc5	109	Nh7	149	Nh2	189	Bh5	229	Kh8	269	Kh3	309	Qh6	349	Qh1
30	Pd4	70	Rd5	110	Na6	150	Na1	190	Ba4	230	Ka7	270	Ka2	310	Qa5		
31	Pe4	71	Re5	111	Nb6	151	Nb1	191	Bb4	231	Kb7	271	Kb2	311	Qb5		
32	Pf4	72	Rf5	112	Nc6	152	Nc1	192	Bc4	232	Kc7	272	Kc2	312	Qc5		
33	Pg4	73	Rg5	113	Nd6	153	Nd1	193	Bd4	233	Kd7	273	Kd2	313	Qd5		
34	Ph4	74	Rh5	114	Ne6	154	Ne1	194	Be4	234	Ke7	274	Ke2	314	Qe5		
35	Pa3	75	Ra4	115	Nf6	155	Nf1	195	Bf4	235	Kf7	275	Kf2	315	Qf5		
36	Pb3	76	Rb4	116	Ng6	156	Ng1	196	Bg4	236	Kg7	276	Kg2	316	Qg5		
37	Pc3	77	Rc4	117	Nh6	157	Nh1	197	Bh4	237	Kh7	277	Kh2	317	Qh5		
38	Pd3	78	Rd4	118	Na5	158	Ba8	198	Ba3	238	Ka6	278	Ka1	318	Qa4		
39	Pe3	79	Re4	119	Nb5	159	Bb8	199	Bb3	239	Kb6	279	Kb1	319	Qb4		
40	Pf3	80	Rf4	120	Nc5	160	Bc8	200	Bc3	240	Kc6	280	Kc1	320	Qc4		

Table 12.3: Key for the axis 'Move to piece/position' for figures 6.9, 6.10 and 6.11.

13.3. Appendix 3: Supplementary media

A disc is attached to this thesis²⁵ containing the following components:

- Source files
- Data (including chunk libraries)
- Executable programs

The folders and their contents are described below.

13.4. CLAMPanalyser

The CLAMPanalyser folder contains an application 'CLAMPanalyser.exe' which runs on a PC under the Windows operating system. The program allows the user to enter a chessboard configuration in FEN format and select a chunk library.

CLAMPanalyser will assemble the chunks on the chessboard and score each move.

The program lists all possible moves²⁶, in order of moves with the highest number of supporting chunks, and scaled by the 'move rareness' figure. Moves are listed in descending order in the area named '*Moves scored here*'.

To use the program do the following:

1. Enter a chessboard in FEN format in the text box under the heading:

Enter the chessboard FEN here:

2. Use the scroll bar on the right of the list box titled:

Double click on the library file here:

To select a library file double click on the file to begin the analysis.

²⁵ Instructions on how to obtain the data are available via an Internet search for the string: 'CHUNKING-DATA-ANDREWCOOK'

²⁶ For simplicity *en passant* pawn moves, promotions and *castling* are not supported in CLAMPs repertoire of possible moves.

- Wait for the progress bar to complete. The results of the analysis showing each move is listed within the list box entitled:

Moves scored here:

A screen-shot of CLAMPanalyser program is show below:

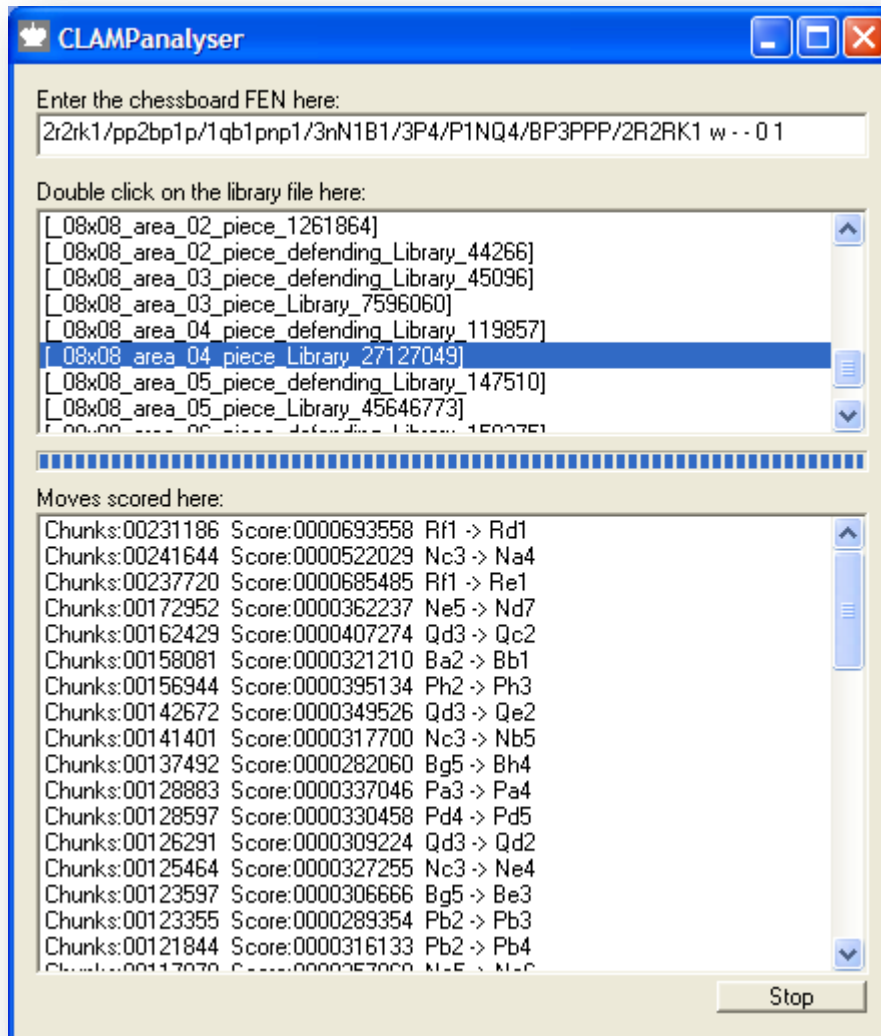


Figure 12.1: CLAMPanalyser program screen-shot

The results displayed in the 'moves scores here' window show each move starting and ending position and the number of chunks that support this move. The score is based on the number of chunks supporting the move divided by the move rareness-scaling factor.

13.4.1. The library naming convention for the CLAMPanalyser program

A library is a set of files, each file corresponding with the arrival of a piece to a square. The set of files are contained within a folder and the folder name describes the type of library.

A typical folder name is as follows:

`_08x08_area_02_piece_defending_Library_44266`

The folder name is constructed as follows:

1. **`_08x08_area`** denotes the board area used to restrict the pieces that make up a chunk. An **`'08x08_area'`** denotes the whole board area.
2. **`_02_piece`** denotes the number of pieces that make up the chunks. **`_02_piece`** indicates that the library is composed of two-piece chunks.
3. **`_Defending`**, (if present) denotes that the chunks within the library are made from pieces in defending relationships with each other.
4. **`_Library`** denotes that this folder is a library.
5. **`_44266`** (if present) shows the total number of chunks that make up the library.

13.4.2. Counting the number of chunks in a chunk library

A copy of executable program 'CountCks.exe' is present in each of the library folders.

The program will count the number of chunks in the library folders. To use the program enter the chunk size used by the library and click on the 'count' button. The program will count the number of chunks in each library file and display the sum.

Note that within the library file chunks are not duplicated however some of the same chunks may appear in several of the library files within a library folder.

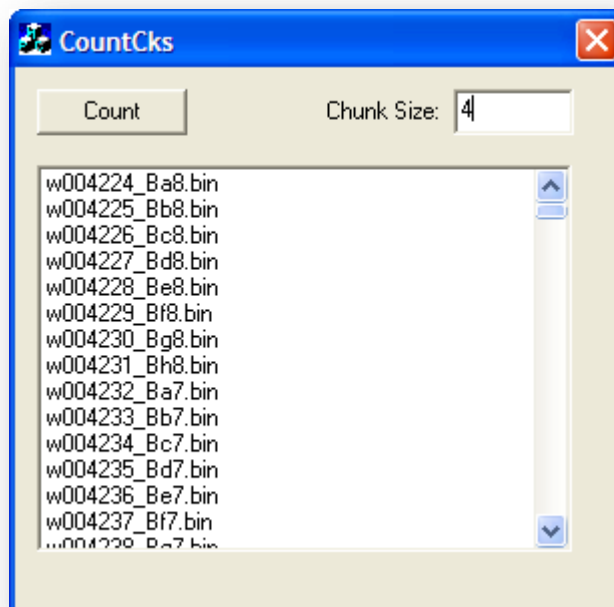


Figure 12.2: Screen-shot of the program CountCks.exe

13.4.3 The LibraryComparator

The LibraryComparator folder contains two executable programs that can be used to compare the output from the CLAMPAnalyser program. The executable programs are designed to run on a PC under the Windows operating system. The LibraryComparator program compares the move scores from one thousand sample typical chessboard configurations from the mid-game of tournament games between Master players, with the actual move that was played. Move scores for each of the test chessboards and the actual move played were previously compiled by CLAMPAnalyser using each of the chunk libraries, with a naming convention for the analysis files similar to the convention adopted for the libraries.

In addition to comparing the libraries the 'move rareness' (cf. page 89) scalar can be enabled. To analyse a file simply 'double-click' on the required file. The process will run and results reported in the text box at the bottom of the form.

A screen shot of the LibraryComparator program is shown below:

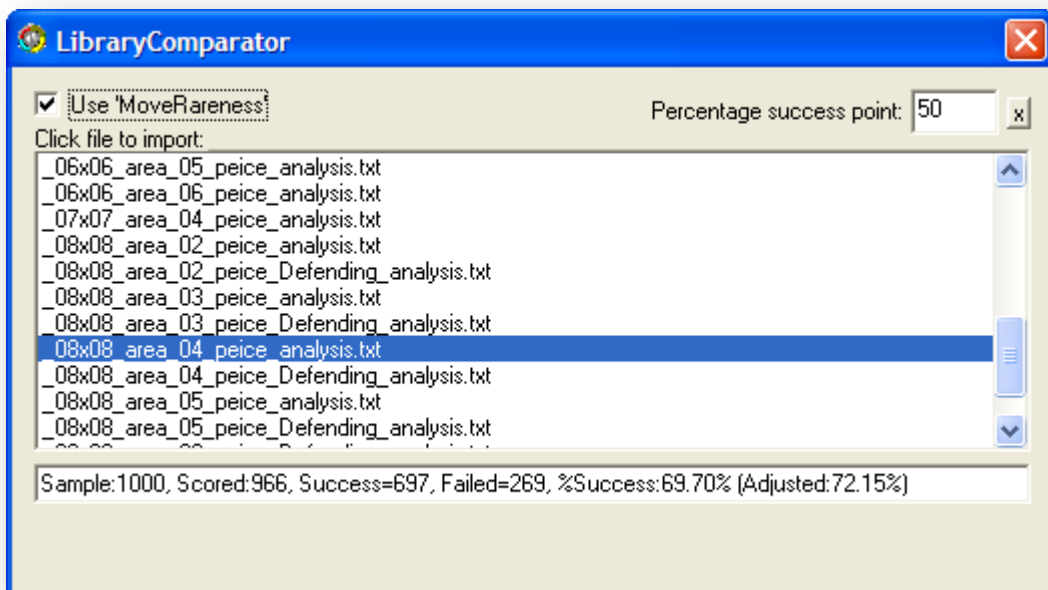


Figure 12.3: The LibraryComparator screen shot

The percentage success point (otherwise known as the ‘null hypothesis’ point (cf. page 86)) can be adjusted by entering a percentage into the box at the top right of the form.

13.4.4 The output from the LibraryComparator program

The results shown in the text box at the bottom of the form show the following fields:

Sample	The number of test chessboard configurations tested
Scored:	The number of test chessboards that had one or more chunks associated with a move.
Success	The number of instances that the ordered list from CLAMP placed the actual move played above the ‘percentage success point.
Failed	The number of instances that the ordered list from CLAMP placed the actual move played on or below the ‘percentage success point.
%Success	The number of successes / the number of the sample boards
Adjusted	The number of successes / the number of boards scored

Table 12.4: The output fields from the LibraryComparator program.

13.4.5 The 'TopMovesComparator' program

The TopMovesComparator program will show the results of the analysis of one thousand sample typical chessboard configurations from the mid-game of tournament games between Master players, displaying the number of boards were that the actual move played was one of the moves in the top positions of CLAMPAnalyser's ordered list.

TopMovesComparator allows analysis with, or without, the 'move rareness' scaling factor.

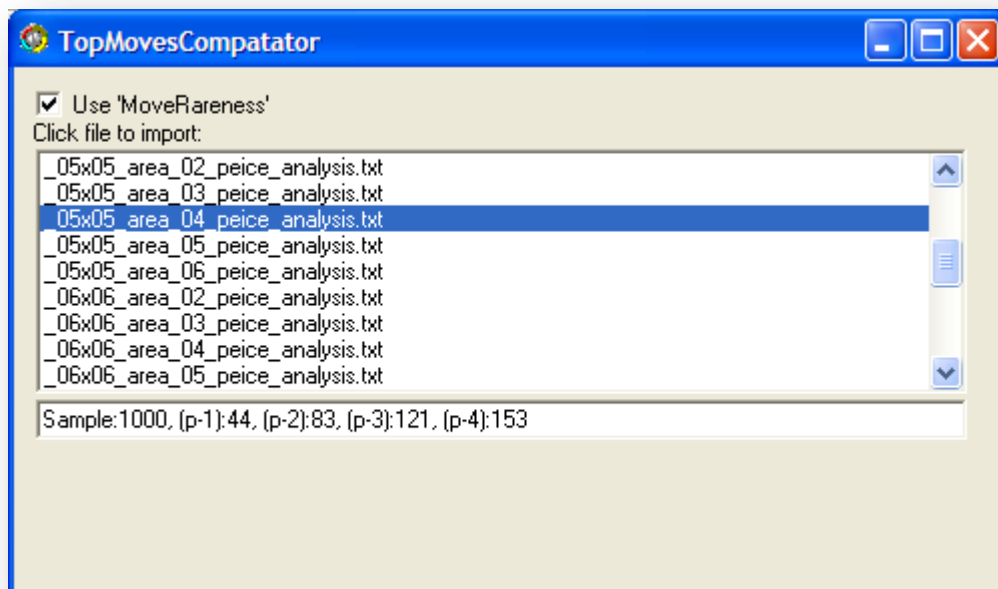


Figure 12.4 The 'TopMovesComparator' program screen-shot

13.4.6 The output from the TopMovesComparator program

The results shown in the text box at the bottom of the form show the following fields:

Sample	The number of test chessboard configurations tested.
p-1	The number of instances where the move played by the chess player is within the top position in CLAMPanalyser's ordered list output.
p-2	The number of instances where the move played by the chess player is within the top two positions in CLAMPanalyser's ordered list output.
p-3	The number of instances where the move played by the chess player is within the top three positions in CLAMPanalyser's ordered list output.
p-4	The number of instances where the move played by the chess player is within the top four positions in CLAMPanalyser's ordered list output.

Table 12.5: The output fields from the LibraryComparator program

13.4.7 SourceCode

The SourceCode folder contains the C++ source files for the CLAMP program. The project compiles under the mpiCC compiler to run on a cluster computer under the UNIX operating system.

Two sub-folders exist within this folder:

_CLAMP

The source files to build CLAMP are as follows:

clamp.cpp

CalcMoves.cpp

GenLibs.cpp

clamp.h

prototypes.h

GenLibs.h

CalcMoves.h

A makefile is included, the contents of which is as follows:

```
#####  
##### Makefile  
SRC = clamp.cpp GenLibs.cpp CalcMoves.cpp  
TRG = clamp  
#####  
$(TRG): $(SRC)  
    mpiCC -Wall -O2 $(SRC) -o $(TRG)  
#####  
all:  
    $(MAKE) $(TRG)  
#####  
clean:  
    rm -f *~ core *.o *.s $(TRG)  
#####
```

_CLAMPanalysis

The source files to build CLAMPanalysis are as follows:

- analyse.cpp
- analyse.h
- stats.cpp
- stats.h

A makefile is included, the contents of which is as follows:

```
#####  
##### Makefile  
SRC = analyse.cpp stats.cpp  
TRG = analyse  
  
#####  
$(TRG): $(SRC)  
    mpiCC -Wall -O2 $(SRC) -o $(TRG)  
  
#####  
all:  
    $(MAKE) $(TRG)  
#####  
clean:  
    rm -f *~ core *.o *.s $(TRG)  
#####
```

13.4.8 Collections_MoveTo

This folder contains the collection of one thousand chessboards for all moves to a position. The file names of the contents reflect the position that the piece was moved to after the piece was played, for example, a move of the Bishop to square 09 (numbering the chessboard from the top left corner as shown in table 12.6 below) would be:

B09_SOURCE.MAP

The format of the file is shown below:

```
Q.....B...bk....q.np...p.nr.....p...PP..P...P.rP.R.N.....R..K
.....k.pBpbrn.p.p.p...p.....P.....P..P.P.P...KP..B..R..
...rrk.pB....pp.p.....Pp..pQ.P.....q.n.P...P...P...RRK.
.....B..rp.....k..p.....p.....P...P.KP.....P.....
.....B.....p.....k..p...p.rP...K..R.....P.....
.....B.....p.....pK.....kP.....
.r.q.rk.pB.....p.n..npp..pp.p.....NP..PP..P.PP.QNP..R....RK.
..
```

Each line represents a chessboard. The period character represents a blank square.

Pieces are represented as follows:

P=Pawn, R=Rook, N=knight, B=Bishop, K=King, Q=Queen

Upper-case characters are white pieces and lowercase characters are black pieces.

The squares are sequenced from the top left corner reading to the right and then down as shown in table 12.6:

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Table 12.6: Chess square numbering

13.4.9 Collections_MoveFrom

This folder contains the collection of one thousand chessboards for all moves *from* a position. The file names of the contents reflect the position that the piece was moved from. The file naming convention and format of the contents is identical to the Collections_MoveTo collection.

14. BIBLIOGRAPHY

- Berliner H. and Campbell M., (1984) *"Using Chunking to solve chess pawn endgames"*. Artificial Intelligence, Volume 23, Issue 1, May 1984, pp. 97-120
- Beverly P. Wood M. D., MS (2009) *Visual Expertise*, Radiology, September 2009, 252 (3)
- Bilalic, M., McLeod, P., & Gobet, F. (2009). *Specialization effect and its influence on memory and problem solving in expert chess players*. Cognitive Science, 33, 1117-1143.
- Binet A. (1896/1996) *Psychologie des grands calculateurs et joueurs d'échecs*. Paris: Hachette.
- Bird, S., Klein E., Loper E. *Natural language processing with Python*, O'Reilly Media, Inc., 2009, ISBN: 0596516495, 9780596516499. Page 264
- Campbell M, Hoane A. J. Jr. and Hsiung-Hsu F., (2002) *"Deep Blue"*, Artificial Intelligence, Volume 134, Issues 1-2, January 2002, pp. 57-83
- Campitelli G, Gobet F, and Amanda Parker A. (2005), *Structure and Stimulus Familiarity: A Study of Memory in Chess-Players with Functional Magnetic Resonance Imaging*, The Spanish Journal of Psychology 2005, Vol. 8, No. 2, 238-245, ISSN 1138-7416
- Campitelli G, Gobet F, Head K, Buckley M, and Parker A., (2007), *Brain Localization Of Memory Chunks In Chess players*, International Journal Of Neuroscience, 2007, Vol. 117, No. 12, Pages 1641-1659
- Charness N, Reingold, Pomplun, Stampe, (2001) *The Perceptual Aspect Of Skilled Performance In Chess: Evidence From Eye Movements*, Memory & Cognition, 2001, 29 (8), 1146-1152
- Charness N. Reingold E. M., Pomplun M. and Stampe D. M., (2001) *"Visual Span in Expert Chess Players: Evidence From Eye Movements"*, Psychological Science, Volume 12 Issue 1 pp. 48 (January 2001)
- Chase S., (2000) *Pigeons and the Magical Number Seven*, Papiere zur Linguistik 62/63, 2000, pp 3-14. The magical number seven in language and cognition: empirical evidence and prospects of future research, Gertraud Fenk-Oczlon and August Fenk, University of Klagenfurt
- Chase W. G. and Simon H. A., (1973a) *"Perception in chess"*. Cognitive Psychology, Vol 4, Issue 1, January 1973, pp. 55-81
- Chase W. G. and Simon H. A., (1973b) *The mind's eye in chess*. W. Chase (ed.), Visual information processing. New York: Academic Press, 215-281.
- Cohen J. S., Westlake K. and Pepin M., (2001) *"High order chunking in serial pattern learning by rats in the T-maze"*, Learning and Motivation, 32, pp. 409-433,
- Cooke N. J. Atlas R. S. Lane D. M., Berger R, C, (1993) *"Role of High-Level Knowledge in Memory for Chess Positions"* The American Journal of Psychology, Vol. 106, No. 3 (Autumn, 1993), pp. 321-351, University of Illinois Press
- Cowan N. (2001) *The magical number 4 in short-term memory: A reconsideration of mental storage capacity*. Behavioural Brain Sciences 24:87-114.
- de Groot A. and Gobet F., (1996) *Perception and Memory in Chess*, Studies in the Heuristics of the Professional Eye, Van Gorcum 1996, ISBN 90 232 2949 5, pp 153
- de Groot, A. D., (1965, 1978) *Thought and choice in chess*, (2nd Edition) Mouton and Co, The Hague, Paris. (Original edition 1965)
- de Groot A. D. Gobet F., (1996) *Perception and Memory in Chess*, Van Gorcum & Comp. B.V., P.O. Box 43, NL-9400 AA Assen, The Netherlands. ISBN 90 2332 2949 5 (pp. 220)

Ericsson K. A., & Harris M. S., (1990, November). *Expert chess memory without chess knowledge: A training study*. Paper presented at the 31st Annual Meeting of the Psychonomic Society, New Orleans, Louisiana.

Ericsson K. A. and Kintsch W., (1995) *Long-term working memory*. Psychological review, 102, 211-245.

Feigenbaum E. A. and Simon H. A., (1984) "*EPAM-like models of recognition and learning*", Cognitive Science, Volume 8, Issue 4, October-December 1984, pp. 305-336

Feigenson L. and Halberda J., (2004) "*Infants chunk object arrays into sets of individuals*". Cognition 91 pp. 173-190

Fenk-Oczlon and Fenk, (2000) pp 3-14. *The magical number seven in language and cognition: empirical evidence and prospects of future research*, Gertraud Fenk-Oczlon and August Fenk, University of Klagenfurt

Finkelstein L. and Markovitch S., (1998) *Learning to play chess selectively by acquiring move patterns*. Computer Science Department, Technion, Haifa 32000, Israel.

Fountain S. and Benson D., (2006) *Chunking, rule learning, and multiple item memory in rat interleaved serial pattern learning*, Learning and Motivation, Volume 37, Issue 2, May 2006, Pages 95-112

Frayn C., Justiniano C., Lew K., (2006) *ChessBrain II – A Hierarchical Infrastructure for Distributed Inhomogeneous Speed-Critical Computation*, 2006 IEEE Symposium on Computational Intelligence and Games (CIG06)

Gobet, F. and Charness N., (2006) *Chess and games*. Cambridge handbook on expertise and expert performance (pp. 523-538). Cambridge, MA: Cambridge University Press. (<http://www.cambridge.org/>)

Gobet F. and Clarkson G. (2004) *Chunks in expert memory: evidence for the magical number four ... or is it two?* Memory. 2004 Nov;12(6):732-47.

Gobet F., (1998) *Pattern Recognition Makes Search Possible: Comments on Holding (1992)*, ESRC Centre for Research in Development, Instruction and Training, University of Nottingham

Gobet F. and Jansen P., (1994) *Towards a chess program based on a model of human memory*. In H. J. van den Herik, I. S. Herschberg, & J. W. Uiterwijk (Eds.), *Advances in Computer Chess 7*. Maastricht: University of Limburg Press.

Gobet F. and Simon H. A., (1996) *Recall of random and distorted positions: Implications for the theory of expertise*. Memory & Cognition, 24, 493-503.

Gobet F. and Simon H. A., (1996) *The roles of recognition processes and look-ahead search in time-constrained expert problem solving: Evidence from grandmaster level chess*. Psychological Science, 7, 52-55.

Gobet, F., & Simon, H. A. (1996). *Templates in chess memory: A mechanism for recalling several boards*. Cognitive Psychology, 31, 1-40.

Gobet F. and Simon H. A. (1998) *Expert chess memory: Revisiting the chunking hypothesis*. Memory, 6, 225-255.

Gobet, F. and Simon, H. A. (2000) *Five seconds or sixty? Presentation time in expert memory*. Cognitive Science, 24, 651-682.

Gobet F., Peter C. R., Lane Steve Croker, Peter C-H. Cheng, Gary Jones, Iain Oliver and Julian M. Pine, (2001) *Chunking mechanisms in human learning*, *TRENDS in Cognitive Sciences*, Vol 5, No 6, June 2001, ESRC Centre for Research in Development, Instruction and Training, School of Psychology, University Park, Nottingham, UK NG7 2RD

- Gobet, F. (2001). *Is experts' knowledge modular?* In Proceedings of the 23rd Meeting of the Cognitive Science Society (pp. 336-431). Mahwah, NJ: Erlbaum.
- Gobet F. and Jackson S., (2002) *"In search of templates"*, Cognitive Systems Research, Volume 3, pp. 35-44, (2002)
- Gobet F., (1998) *Pattern Recognition Makes Search Possible: Comments on Holding (1992)*, ESRC Centre for Research in Development, Instruction and Training, University of Nottingham.
- Gobet F. and Jansen P., (1994) *"Towards a chess program based on a model of human memory"*. In H. J. van den Herik, I. S. Herschberg, and J. W. Uiterwijk (Eds.), *Advances in Computer Chess 7*. Maastricht: University of Limburg Press.
- Gobet F., Lane P., (2010) *The CHREST Architecture of Cognition: The Role of Perception in General Intelligence*. *Advances in Intelligent System Research*, March 2010. ISBN: 978-90-78677-36-9
- Grover C. and Tobin R., *Rule-Based Chunking and Reusability*, School of Informatics University of Edinburgh
- Holding D. H., (1985) *The Psychology of Chess Skill* Lawrence Erlbaum Assoc. 1985
- Holding D. H., (1989) *The American Journal of Psychology*, Vol 102, No 1 (Spring 1989), pp 103 – 108, University of Illinois Press
- Holding D. H., (1992) *Theories of chess skill, Psychological Research*, March Vol 54(1) pp. 10 - 16
- Jongman R. W. (1968) *Het oog van de meester [The eye of the master]*. Assen, The Netherlands: Van Gorcum.
- Kennedy, W.G., & Trafton, J.G. (2006) *Long-term Learning in Soar and ACT-R*. In Proceedings of the Seventh International Conference on Cognitive Modeling (pp. 166-171). Trieste, Italy.
- Knuth and Moore. (1975), *An Analysis of the Alpha-Beta Pruning*, *Artificial Intelligence*, vol 6 pp 293-326
- Laird J. E., Rosenbloom, P.S., Newell, *Towards Chunking as a General Learning Mechanism*. AAAI-84 Proceedings. 1984
- Laird J. E., Rosenbloom, P.S., Newell A., *Chunking in Soar: The Anatomy of a General Learning Mechanism*, *Machine Learning 1*: 11-46, Kluwer Academic Publishers, Boston. Manufactured in The Netherlands. 1986,
- Lane. P. C. R., Gobet. F., *Perception in chess and beyond: Commentary on Linhares and Freitas (2010)*, *New Ideas in Psychology* (20 September 2010)
- Linhares A. (2008) *Decision-making and strategic thinking through analogies* (unpublished)
- Linhares and Freitas, 2010 A. Linhares and A.E.T.A. Freitas, *Questioning Chase and Simon's (1973) "perception in chess": the "experience recognition" hypothesis*, *New Ideas in Psychology* 28 (2010), pp. 64–78.
- Liu W.Z. and White A.P., (1991) *A Review of Inductive Learning*, in I.M.Graham and R.W. Milne, *Research and Development in Expert Systems VIII*, Proc. of Expert Systems 91, Cambridge Univ. Press, London, pp.112-126,1991.
- Logie and Gilhooly K. (eds.), *Working memory and thinking*. Hove: Psychology Press. 1998, pp 115 - 138
- Markovitch. (1998), *ICCA Journal*, Volume 21: Number 2 (June 1998) *Learning to Play Chess Selectively by Acquiring Move Patterns*. Lev Finkelstein and Shaul Markovitch. pp 100-119.

- McCarthy J., (1997). *AI as sport*. Science, vol 276, June 1997, pp.1518-1519
- McGregor S. J. and Howes, (2002) *The role of attack and defence semantics in skilled players' memory for chess positions*. Memory and Cognition. Vol 30 No 5, pp. 707-717
- Miller G. A., (1956) *The magic number seven, plus or minus two: some limits in our capacity for processing information*. Psychological Review 1956, 61, pp. 81-97
- Nilsson N. J., (1998), *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers Inc, San Francisco. ISBN 1-55860-535-5, pp 207
- Perko R., Leonardis A., (2010), *A framework for visual-context-aware object detection in still images*, Computer Vision and Image Understanding 114 (2010) 700–711
- Reynolds R., (1992), *Recognition of Expertise in Chess Players*, The American Journal of Psychology, Vol. 105, No. 3 (Autumn, 1992), pp. 409-415 University of Illinois Press.
- Ross, Philip E., (2006), *The Expert Mind*, Scientific American Magazine (August).
- Saariluoma P., (1991), *Aspects of skilled imagery in blindfold chess*. Acta Psychologica, 77(1), 65-89.
- Saariluoma P., (1994), *Location coding in chess*. Quarterly Journal of Experimental
- Saariluoma P. (1995), *Chess Players' Thinking: A Cognitive Psychological Approach*, Routledge, 1995, ISBN 0415120799, 9780415120791
- Saariluoma, P. (1998), *Adversary problem solving and working memory*. In R. Logie & K. Gilhooly (eds.), *Working memory and thinking*. Hove: Psychology Press. pp 115 – 138
- Saariluoma P. (2001), *Psicológica* (2001), 22, 143-164. *Chess and content-oriented psychology of thinking*, Pertti Saariluoma, University of Helsinki, Finland.
- Servan-Schreiber E. and Anderson J. R., *Learning Artificial Grammars with Competitive Chunking*, Journal of Experimental Psychology: Copyright 1990 by the American Psychological Association, Inc. Learning, Memory, and Cognition 0278-7393/90 1990, Vol. 16, No. 4, 592-608
- Shannon C. E. (1950) *Programming a Computer for Playing Chess*. Philosophical Magazine, Ser.7, Vol 41, No. 314 - March 1950. XXII.
- Simon and Barenfield., (1969) *Information-processing analysis of perceptual processes in problem solving*. Psychological Review, 76, 473-483.
- Simon H. A. and Gilmartin K., *A Simulation for chess positions*, *Cognitive Psychology*, 5 (1973) pp. 29-46
- Simon H. A. (1981). *The sciences of the artificial*, MIT Press Cambridge MA.
- Terrace HS, (1987), *Chunking by a pigeon in a serial learning task*, Nature. January 1987, pp. 8-14
- Terrace, H. (2001). *Chunking and Serially Organized Behaviour in Pigeons, Monkeys and Humans*. In R. G. Cook (Ed.), *Avian visual cognition*. [On-line]. Medford, MA: Comparative Cognition Press; available: www.pigeon.psy.tufts.edu/avc/
- Tichomirov G. K., and Poznyanskaya E. D., (1966) *An investigation of visual search as a means of analyzing heuristics*. *Voprosy Psikhologii*, 1966, 12, 39-53.
- Walczak S. (1992). *Pattern Based Tactical Planning*, International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI) Volume 6 No 5, Year 1992, pp 955-988
- Wilkins D. (1980). *Using patterns and plans in chess*, Artificial Intelligence, Vol 14. pp. 165-203