ELSEVIER

Artificial Intelligence 120 (2000) 235–250

Artificial Intelligence

www.elsevier.com/locate/artint

Computer chess move-ordering schemes using move influence

Kieran Greer [1]

School of Information and Software Engineering, Faculty of Informatics, University of Ulster at Jordanstown, Newtownabbey, N. Ireland, BT37 0QB, UK

Received 12 June 1999; received in revised form 1 February 2000

Abstract

The chessmaps heuristic is a pattern-oriented approach to ordering moves for the game of chess. It uses a neural network to learn a relation between the control of the squares and the influence of a move. Depending on what squares a player controls, the chessmaps heuristic tries to determine where the important areas of the chessboard are. Moves that influence these important areas are then ordered first. The heuristic has been incorporated into a move-ordering algorithm that also takes account of immediate tactical threats. Human players also rely strongly on patterns when selecting moves, but would also consider immediate tactical threats, so this move-ordering algorithm is an attempt to mimic something of the human thought process when selecting a move. This paper presents a new definition for the influence of a move, which improves the performance of the heuristic. It also presents a new experience-based approach to determining what areas of the chessboard are important, which may actually be preferred to the chessmaps heuristic. The results from game-tree searches suggest that the move-ordering algorithm could compete with the current best alternative of using the history heuristic with capture moves in a brute-force search.

Keywords: Chessmaps heuristic; Computer chess; Search heuristic; Pattern-oriented; Neural network

1. Introduction

This paper will present a move-ordering algorithm that attempts to mimic something of the human thought process when selecting chess moves. At the heart of this is a heuristic called the *chessmaps heuristic*, that orders moves depending on which areas of the chessboard they *influence*. The definition for the influence of a move is given in Section 2.3.

[1] Email: krc.greer@ulst.ac.uk.

0004-3702/00/$ – see front matter
PII: S0004-3702(00)00026-6

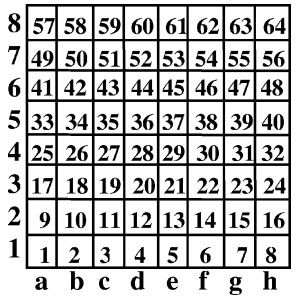

Fig. 1. Layout of a chessboard divided into 64 sectors. The sectors are numbered from 1 to 64 as shown.

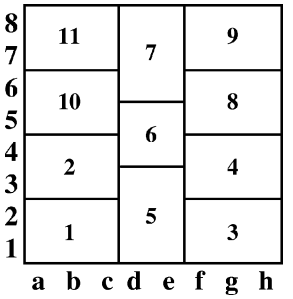

Fig. 2. An alternative layout of a chessboard divided into 11 sectors.

If a chessboard is divided into specific areas, which could be of any size or shape, then the term *sector* is used to define one of these areas and Figs. 1 and 2 illustrate a chessboard divided into different numbers of sectors. For Fig. 1 the sector size is only 1 square large and tests showed that this was the most effective sector size. The chessmaps heuristic uses the output of a neural network to order these sectors into relative importance. Moves that influence important sectors are then thought to be stronger than moves that influence less important sectors only and so would be looked at first. Details of preliminary tests using the move-ordering algorithm for game-tree searches are reported in [6]. This paper will present an improvement made to the chessmaps heuristic and provide further support to the validity of the move-ordering algorithm through the results of more game-tree searches.

Two methods will be used to try and determine what sectors are the most relevant to a position. The first method is the knowledge-based chessmaps heuristic, which uses a neural network to try and learn a relation between the control of the squares and the influence of a move. The definition for the control of a square is given in Section 2.1. For this method to be sensible, the control of the squares must contain important information for any chess position. The chess concept of *space* is represented by how much of the chessboard we control. It is known that a player with a space advantage often attacks, while a player with a space disadvantage often defends. So the control of the squares can be used to define a very basic strategy, which the neural network managed to learn to some degree (see Section 3). Also, as the control of the squares is calculated by determining what squares the pieces attack or defend, it is directly related to the movements of the pieces and the relationships between the pieces. There is thus important information contained in the control of the squares, but this needs to be represented to the computer program in an appropriate form and one attempt is presented in this paper. The second method is experience-based, where the sectors are ordered according to the results of the previous search of the game-tree. This is really an alternative to the chessmaps heuristic. A novel aspect of both of these sector-ordering methods is that they order moves depending on their influence; that is, the piece does not need to actually move to the sector in question itself. Other move ordering heuristics have been concerned with the squares that the pieces actually move to.

Pattern recognition plays an important role in a chess player's thought process. Chess masters store about 50000 chunks of chess information, represented by patterns. They then

retrieve this information and apply it to any new chess position. The information that they store would not be just the squares that the pieces sit on, but also the relationships between the different pieces. Work done on the psychology of chess players including their thought processes when selecting moves can be found in [4] and [12]. Much of what goes into selecting a move occurs during the first few seconds of looking at a position, when a player scans the board trying to recognise relevant chunks of information and gain an overall impression. The control of the squares is a crude attempt at giving a general first impression of the position and would contain much less information than the pattern chunks that a human player stores but is a step in this direction. There have been various attempts at pattern-based approaches to learn to play chess. One recent attempt is [9], but there are also other works (e.g., [3,8,13]). A recent study of machine learning attempts applied to computer chess can be found in [5]. Neural networks are particularly well suited to the problem of pattern recognition. Because the information being fed into the neural network for this method is very general, it is possible to represent all phases of a chess game, which is necessary to generate a general search heuristic. Other methods concerned with extracting rules that determine if specific moves can be played would not be practical here, because of the number of rules or cases that would need to be stored.

2. Definitions used for the chessmaps heuristic

The chess position and move must be pre-processed before they can be used and the following definitions are required for this pre-processing. The definition for the control of the squares is the same as has been used previously, but a new definition for the influence of a move will be presented.

2.1. Control of a square

Some general rules define the control of a square. A square on the chessboard can be given one of three values: 1 if White controls the square, -1 if Black controls the square and 0 if the square is *neutral*. A square can either be *occupied* or *empty* and we will firstly consider the case where the square is occupied. If a white piece occupies the square and it is not attacked or defended, then the control of the square is defined as neutral. If the piece is defended and not attacked then White controls the square. The complications arise when the piece is attacked by Black, where a capture sequence on the square needs to be made to determine who controls the square. If Black can capture on the square, but loses material if he does so then the square is under White's control. If however he gains material then the square is under Black's control. If Black neither gains nor loses material by capturing then the control of the square is neutral. Equivalent rules apply for a black piece occupying a square. If the square is empty, then White controls it if he can move a more valuable piece to the square without loss of material and Black controls the square if he can move a more valuable piece there. The loss of material is again determined by capture sequences on the square. The control of the square is said to be neutral if both sides can move a piece of equal value to the square without loss of material, or neither side can move a piece to the square.

To calculate the control of a square it is firstly necessary to record all of the pieces that directly or *indirectly* attack the square. A piece indirectly attacks a square if it is doubled with another piece on the square and so could move there only after the first piece has done so. Also taken into consideration is if the piece is pinned to the king, where a piece cannot enter into a capture sequence or move if this exposes its king to check. So only legal captures and moves are considered. When determining who controls the square, capture sequences are made on ascending material value of the capturing piece, so pawn captures are considered first and king captures last. The capture sequences can also be terminated early (not all captures made) if this leads to a favourable evaluation for the capturing side. For example, making all possible captures on a square for one player may lead to an overall material loss, but making just the first capture could mean a material gain. The following example in Fig. 3 may help to explain the process used to determine the control of a square.

In this position it is Black to move and we are wondering if he can move his knight to g5, so we are just considering the capture sequences on this square. Note that the white knight on e4 is pinned to the white king and the white bishop on d2 can capture on the square only after the white queen has done so. If we were to consider all capture moves on the square then the capture sequence would be:

1. BNf7−g5. WRg3 × g5.
2. BRg7 × g5. WQf4 × g5.
3. BRg8 × g5. WBd2 × g5.
4. BQe7 × g5. WNe4 × g5.

And White has ended up a rook ahead. However, if Black were to terminate the capture sequence after the third set of captures, then he would have won a queen for a rook and a knight and be ahead in material. As White cannot safely move a piece to the square, the control of the square belongs to Black. To ensure that the correct capture sequences are performed, not only stored is the material value of the pieces that attack a square, but also the conditions under which they can move to the square. Some pieces can move directly to a square and some can move to a square only after another piece has done so. For the white bishop in Fig. 3, the condition under which it could move to g5 would be *AfterWhiteQueen* and for the white knight it would be *AfterBlackQueen*, but Black's

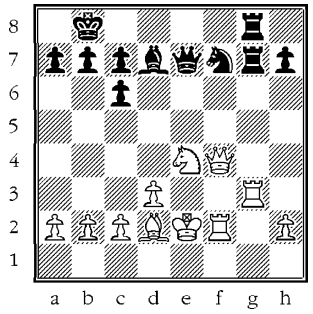

Fig. 3. Example chess position before Black's move Ng5.

knight could move directly to the square and would be given the condition *Direct*. It is then easy to determine when these conditions have been met, making the move legal. When determining the control of a square we consider each square separately and do not worry that capturing on one square could mean the loss of material on another square.

2.2. Chessmap

A *chessmap* is an abstract representation of a chess position, where each square is given one of the three control values as defined in Section 2.1 and maps out the territorial control of each side. Appendix A illustrates the chessmap and the influence of a move generated for the position in Fig. A.1. If we consider Fig. A.1, then in this position the strategies for both sides are well defined. White has the greater space on the queenside and attacks here, while Black plays for a kingside attack. The chessmap for this position, given in Fig. A.2, seems to strongly agree with these strategies and the following simple algorithm could be used to generate a chessmap for a position:

(1) For each square on the chessboard record which pieces attack the square and the conditions under which they can move to the square.
(2) For each square on the chessboard then perform the following:
  2(a) If the square is occupied then, if the piece is attacked, calculate the capture sequence on the square to determine who controls it. If the piece is defended and not attacked then the control of the square belongs to the side that owns the piece. If the piece is not attacked or defended the square is neutral.
  2(b) If the square is not occupied then make each legal move to the square in turn and record the highest valued piece for each side that can move to the square without loss of material on the square. The side that can move the higher valued piece to the square then controls the square. The control of the square is neutral if the piece values are equal or neither side can move a piece to the square.

2.3. Influence of a move

A new definition for the influence of a move has been tried in this paper and would work best with the 64 sectors of Fig. 1. Previous tests indicated that this was the most accurate number of sectors. The influence of a move is meant to represent a player's intentions when he moved a piece to a certain square. Unfortunately, when automatically generating the move influence, a lot of sectors are included in the influence that are not really relevant to the move at all. So this process is a bit fuzzy, but as will be seen the definition used here seems to be slightly more accurate than the previously used definition, which can be found in [6]. A computer program has been written to automatically process the chess positions and moves. Every time a move is made, an array in the computer program is updated. This array is called *valueboard* and stores for each sector the sum values of all pieces that directly or indirectly attack that sector. The value of any piece that sits in the sector is also added to the sum value for that sector. It is possible to take the valueboard values before and after the move and use this as the influence of the move in some way.[2] Consider

[2] I would like to mention Dr. Piyush Ojha who supervised my D.Phil. research on this work. He suggested the possibility of using the valueboard values for the move influence in some way, but this method was not tested at the time.

the position in Fig. A.1, where the valueboard for this position is given in Fig. A.3. If we consider the move Rc1 for White, then after this move the valueboard is changed to the one in Fig. A.4. Fig. A.5 then shows the differences in the two valueboards. Table A.1 gives the values for each piece that are used in the valueboard array, or stored as the material value of the piece for the capture sequences and these values may look a bit strange. It is necessary to give the rooks and bishops of each side different values. This is because when we move one rook this may set a certain condition to true in a capture sequence and we need to be able to determine that one rook move is not confused with the other rook moving. The same applies for the bishops. Note that the king is given a value of ± 40 in the valueboard array when being used as an attacking piece but ± 200 when included in a capture sequence.

The influence of the move can then be defined as follows: If a sector (or square for 64 sectors) in the valueboard array is changed in favour of the side to move (positively for White or negatively for Black), then the influence of that sector is set to 1. If the value is unchanged, then the influence is set to 0 and if it is changed in favour of the opposing side, then the influence is set to -1. Fig. A.6 illustrates what the influence of the move Rc1 in position of Fig. A.1 would be and the following algorithm could be used to determine the move influence:

(1) Store the valueboard array values before the move.
(2) Make the move on the board and then re-calculate the valueboard array values.
(3) Subtract the old valueboard values from the new valueboard values.
(4) The resulting values for each sector will then determine the move influence as follows:
    4(a) If it is a White move, then if the difference in the valueboards for a sector is positive then the move influence for that sector is $+1$. If it is negative then the move influence is -1 and if the difference is 0 then the move influence is also 0.
    4(b) If it is a Black move, then if the difference in the valueboards for a sector is negative then the move influence for that sector is $+1$. If it is positive then the move influence is -1 and if the difference is 0 then the move influence is also 0.

This new definition only stores the *new* sectors influenced by the move and not all sectors influenced. As we generally move a piece to attack a new square this may help to remove some of the inaccuracy of the older definition that included all sectors influenced by the move. Including all sectors may have made the definition more fuzzy, as a greater number of sectors that are not really associated with the move may also have accidentally been included in the move influence. Note that with 64 sectors, the sector that the piece moves to would not be considered as being influenced with the new definition. This definition also gives a negative value to sectors that are weakened by the move, which helped when training a neural network, but does not affect any move ordering. As will be explained in Section 3, when ordering moves only sectors with a positive value are considered.

3. Training and testing a neural network

A feedforward neural network was trained with the backpropagation algorithm[3] [1]. The neural network was trained on a 10000 position training set and then tested on a 10000 position test set. These data sets were generated from complete and randomly chosen chess games taken from master and grandmaster play. For each position in the training or test set, the input for the neural network was a 70-element vector and the desired output was a 64-element vector. The input vector consisted of the values 1, 0 or −1. The first 64 elements represented the chessmap values of the position and the final 6 elements represented the relative positions of the kings. The king positions were included in the hope that they would help the neural network to determine when to suggest attacking or defensive strategies, or possibly in the endgame. Earlier tests showed that the inclusion of these pre-computed features improved performance only very slightly, but that they had an influence on how the sectors were ordered. Each king was defined as being either on the queenside, in the centre or on the kingside. This required 3 input nodes, one for each area, where a value of 1 meant that the king was in that area of the board and a value of −1 meant that it was not. Thus to represent the positions of both kings, a total of 6 extra input nodes were required. The desired output was a vector quantifying the influence of the move played in that position. Each output node represented a sector and each element of the desired output vector was either 1, 0 or −1 as defined by the move influence definition of Section 2.3. The input and desired output values for the position of Fig. A.1 in Appendix A can be found in Figs. A.7 and A.6 respectively and these are the sort of values that the neural network would be trained on.

The classification task to be learned by the neural network was then as follows: Given the chessmap representation of a chess position and the king positions, the neural network would learn to recognise what areas of the chessboard were important to the position and what areas were not. It did this by attempting to learn the move influence pattern for the position in question, so that when the same position occurred again it would suggest the same move influence. As already stated, this move influence pattern is a bit fuzzy. However, if a relation does in fact exist between the square control and the influence of a move, then similar positions should have move influence patterns that include a similar core number of important sectors. Each move influence pattern may then also contain a number of other sectors, but because these will not occur in as many patterns, they will not be recognised by the neural network as being as important. The weight values associated with the core sectors for a particular type of position will then be reinforced the most and these sectors will obtain the largest output values when a position of this type is encountered again. All positions were considered from the White side, so when it was Black to move the position was reversed. Testing suggested that a 3-layer architecture was to be preferred to a 2-layer architecture, with 16 hidden nodes being a good number. Thus the architecture of the neural network was 70-16-64. After training, the continuous valued actual output values of the neural network were used as the sector ordering, where sectors corresponding to nodes with larger output values were ordered first. In this way the neural network was being used

[3] The neural network simulator used was an older version of Don Tveter's backpropagation package called rbp386f. An up to date version of his software can be found at http://www.dontveter.com/nnsoft/nnsoft.html.

more like a statistical classifier, to produce something similar to a probability distribution for the sectors for particular types of position. Because the output of the neural network is being used to order the sectors, it was important to consider not just the error value, but also to obtain a good spread of values over all of the output nodes. Without this a good ordering could not be obtained. Thus weight values could be chosen before the neural network had reached its minimum error if this produced a good spread of the output values.

After the neural network was trained on the training set, its performance was measured as follows: For each position in the test set, the sectors were ranked by the neural network. The highest ranked sector that was influenced by the player's move in that position was then calculated. Percentage values were then calculated for the entire test set that indicated how accurate the sector ordering would be if only a certain number of sectors were to be used. For example, 20% of the time the move played may have influenced the highest ranked sector, but 30% of the time it may have influenced one of the two highest ranked sectors, and so on. Note that when calculating these percentage values, or when ordering the moves, only sectors positively influenced by the move are considered. The percentage values generated for the test set for just the top 15 sectors are given in Table 1 in the '% Accuracy for a neural network' row. The values in this row can be interpreted as follows: 25.3% of the time the move played would have influenced the highest ranked sector. If the move played did not influence the highest ranked sector, then 13% of the time it would have influenced the second highest ranked sector, and so on. So summing the percentage values for a particular number of sectors will give some idea of how accurate the sector ordering is likely to be for that number of sectors. These values are slightly down on those published previously, but the new definition for the influence of a move includes fewer sectors, making it more accurate. The second row of this table gives the average number of moves that are grouped together for each of the 15 highest ranked sectors. So when looking at the highest ranked sector there were an average of 4.6 moves that influenced it. There were then an average of 3.2 moves that did not influence this sector but influenced the second highest ranked sector, and so on.

These values can be compared with the likely values produced by a random move ordering. There were an average of about 32 moves per position for the entire test set, so if we consider a random move ordering, then the move played is equally likely to be ordered from position 1 to 32. This means that there is a 50% chance that it will be ordered in the top half of all moves, or in the top 16 moves. Looking at the values of Table 1 we can see that the top 4 sectors have an accuracy of 54.8% and the number of moves looked at for these sectors is 12.6. Or alternatively, the top 6 sectors would look at a total average of 16 moves and the percentage accuracy for this is 64.3%. Although these values are not

Table 1
Values indicating the accuracy of the sector ordering for the test set generated by a neural network and the average number of moves looked at for each sector

Top 15 sectors	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
% Accuracy for a neural network	25.3	13	9.2	7.3	5.4	4.1	3.6	3.1	2.8	2.4	2	1.6	1.6	1.5	1.5
Average number of moves	4.6	3.2	2.6	2.2	1.8	1.6	1.3	1.2	1	0.9	0.9	0.8	0.7	0.6	0.6

Table 2
Values indicating the accuracy of the sector ordering for the test set generated by a fixed ordering taken from the training set

Top 15 sectors	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
% Accuracy for a fixed ordering	20.4	16.8	7.6	5.2	4	2.9	5.6	2.9	1.4	1.4	2.5	1.3	1.2	2.8	2.9

overly impressive, they do show a clear improvement on a random move ordering. It is also possible to compare these values with a fixed sector ordering that has been generated from the training set results. For the entire training set, the number of times each sector was ranked first was calculated. A fixed sector ordering was then calculated by ordering the sectors on descending order of how many times they were ranked first. For example, the sector relating to the e4 square was ranked first 1372 times out of a total of 10000 and so would be ordered first by the fixed sector ordering. The results of using this fixed sector ordering to predict the move influence in the test set is given in Table 2. The values are actually slightly worse than using a neural network, where the difference for the top sector is most significant. In this case the total for the top 4 sectors is 50% exactly and for the top 6 sectors it is 56.9%, where these values are also worse than when using a neural network, but better than a random ordering.

Some other tests were also performed to try and gain further insight into the kind of information that the neural network had learned. Centralisation can be tested by looking at which sectors the neural network most often ranks first in any position. For the test set, the sectors most often ranked first in decreasing order were: e4, d3, e2, d5, d4, e5, c2, d1, c4 and c5. The number of times these squares were ranked first was 1423 times out of a total of 10000 for the e4 square, down to 318 times for the c5 square. So the neural network shows a strong tendency towards centralisation. In another test all chessmap values were made the same, either all 1, 0, or -1 and the neural network produced a sector ordering for this. This was done to see how the neural network would interpret complete domination by either side or a completely neutral chessboard, from White's point of view. The chessboards of Figs. 4–6 show the top 10 ranked sectors from these 3 sets of input values. When White controls the whole chessboard the sectors ranked first are very far advanced, consistent with an attacking strategy. When the chessboard is completely neutral the sectors are more centralised to defensive and slightly more defensive again when Black controls the whole

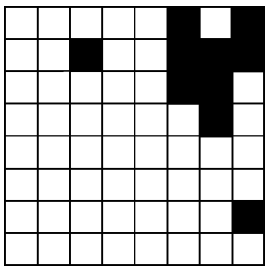

Fig. 4. The top 10 sectors when all control values are 1.

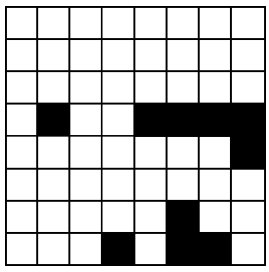

Fig. 5. The top 10 sectors when all control values are 0.

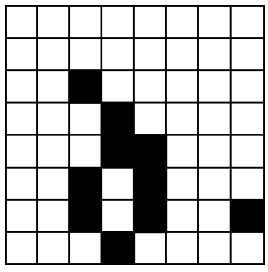

Fig. 6. The top 10 sectors when all control values are -1.

chessboard. This suggests that the neural network might have learned something about when to attack or defend, which would be a very basic strategy.

4. A move-ordering algorithm that uses the chessmaps heuristic

The chessmaps heuristic only provides a general guide for the move ordering and is probably not accurate enough by itself. It can however be used as an effective secondary move ordering method behind other more precise heuristics. Because it is positional in nature, one option would be to extract tactical threats to complement the positional evaluation. With this in mind, the moves are divided into three types: *forced*, *capture* and *other moves*. All of the information required to determine these moves is saved in the chess position or when creating a chessmap, so only a small amount of extra processing is required to produce this move-ordering scheme.

A forced move is defined here as one where a piece is forced to move in order to avoid it being captured with a loss of material. Note that this is different to the idea of a forced move in common chess terms, where a forced move is the only good or acceptable move in the position. Forced moves are ordered on descending value of the piece forced to move, so king moves are considered first and pawn moves last. The forced moves for a single piece are then ordered depending on which sectors they influence. Capture moves are ordered on the material difference between the capturing and the captured piece, with the greatest material difference in favour of the capturing side being ordered first.

All other moves (not forced or capture) are then ordered by the chessmaps heuristic, where we look at each sector in turn and record which remaining moves influence it. The group of moves for each sector can be further ordered by a heuristic called the *closeness heuristic*, which measures how close a move moves a piece to a sector. The piece must be aligned with the sector (a bishop or queen along a diagonal, and a rook or queen along a row or column) and then the number of squares between the piece and the sector is calculated as the piece's closeness value. Moves for each sector are then ordered on ascending closeness value. Note that the influence of a move includes the case where one piece can move out of the way of another piece, so that the second piece now attacks a sector that it previously did not. This change would be recorded in the valueboard arrays and also means that the piece that has just moved may not influence the sector in question itself. As the piece just moved does not directly influence the sector, it is given a default closeness value of 8. Pawns, knights and kings can only have a closeness value of 0 (in the sector), 1 (one move away), or the default value. The closeness heuristic is not particularly accurate, but at least it allows for an ordering of the moves grouped for a sector and there is some logic in moving the piece closer to the relevant area. If the sector chosen is not the most important then maybe another sector in that general area is, so this could help to concentrate forces in the correct area.

The moves can be further sub-divided into *safe* and *unsafe*, where a safe move does not result in loss of material on the square that the piece is moved to and an unsafe move does. These six categories of moves can then be looked at in whatever order is considered best and the ordering used for this paper is:

(1) Safe capture moves.
(2) Safe forced moves.

(3) Safe other moves.
(4) Unsafe capture moves.
(5) Unsafe forced moves.
(6) Unsafe other moves.

Appendix B gives a sector ordering and move ordering for an example chess position. As the majority of moves are ordered by the chessmaps heuristic, it would be possible to add other more precise heuristics without making the chessmaps heuristic redundant. In the next section, another heuristic that is tried is experience-based and records the last sector that caused a cutoff at each depth.

5. Testing the move-ordering algorithm in game-tree searches

Testing was done on a 266 MHz Pentium PC and two sets of tests will be presented in this section. The first set of tests compares the chessmaps heuristic with a random move ordering and also with the history heuristic [11]. As the complete move ordering method is not used here, the moves are only divided into safe and unsafe and then ordered by the sector ordering. All strategies performed a brute-force negamax search [10] to depth 5 with a quiescence search [2] at the leaf nodes, where the quiescence search consisted of making just the safe capture moves for each side. These were ordered on material difference, in the same way as the capture moves of the complete move ordering method. Test search runs were performed on the 24 Bratko–Kopec positions [7] and the average number of nodes searched in each position is given in Table 3. For all strategies a large percentage of nodes searched came from the quiescence search. The chessmaps heuristic and the history heuristic reduced the search size by nearly 80% compared to the random move ordering. The history heuristic searched only slightly fewer nodes than the chessmaps heuristic, but because of the time required for the chessmaps heuristic to generate the move ordering, the history heuristic can perform the search in much less time. Just over 25% less time was required. However, the new definition for the influence of a move has definitely helped the chessmaps heuristic, and reduced the number of nodes searched by approximately 9.5% compared to the old definition.

The second set of tests involved testing the complete move-ordering algorithm against the currently popular approach of firstly extracting capture moves and then ordering the remaining moves using the history heuristic. These tests were done on 54 positions, consisting of the Bratko–Kopec positions and 30 other middlegame positions chosen at

Table 3
The average number of nodes searched for the Bratko–Kopec positions for a brute-force search to depth 5, with a quiescence search at the leaf nodes

Search strategy	Average number of nodes
Random move ordering	2173388
Chessmaps heuristic	472375
History heuristic	441240

Table 4
The average number of nodes searched for 4 search strategies for 54 test positions. The values indicate the average number of nodes for an iterative brute-force search plus quiescence search to the depth indicated for the brute-force search.
The strategies are as follows:
(1) The move ordering method using the chessmaps heuristic.
(2) The move ordering method using the chessmaps heuristic, but also record the top sector at each depth.
(3) The move ordering method where the sectors are ordered in an experience-based way.
(4) Use the history heuristic but firstly extract the capture moves and order these first

	Average number of nodes for varying brute-force search depths			
Search strategy	3	4	5	6
Strategy 1	3549	22383	142320	748494
Strategy 2	3233	19049	112953	593250
Strategy 3	3395	19683	120615	640000
Strategy 4	4278	25779	155554	948161

random. The average number of nodes searched for the Bratko–Kopec positions was slightly less than the average number overall. Four different depths of brute-force search were tried, from depths 3 to 6, with a quiescence search at the leaf nodes. The results for four different search strategies searching to these four depths are presented in Table 4, where this table again gives the average number of nodes searched in each position. The first strategy was to use the move ordering method by itself. The second strategy used the move ordering method, but also added another simple heuristic. This heuristic stored the last sector that caused a cutoff at each depth. This sector was then retrieved and ordered first in any sector ordering, overriding the output of the neural network. The third strategy used the move ordering method, but ordered the sectors in an experience-based way. When each node in the search tree returned a move, the sector that the move was ordered by was recorded and its value in an array was incremented by 1. Thus sectors that are more important will be incremented more often and achieve larger values and this will build up a picture of where the more important areas of the chessboard are. Two arrays stored the sectors, one for White and one for Black; so separate strategies were recorded for each side. Note that forced or capture moves did not increment any sectors if they were found to be best. The final strategy firstly extracted capture moves, which were ordered first and then ordered the remaining moves using the history heuristic. This is another experience-based approach that is known to produce good results and could be used as a benchmark to compare other methods against. Iterative deepening [11] was also included in this set of test runs, so while the experience-based heuristics would benefit from the previous iterations, the knowledge-based approach of using a neural network would not. So a search to depth 3 would consist of a brute-force search to depth 1 followed by a quiescence search, then

a brute-force search to depth 2 followed by a quiescence search and finally a brute-force search to depth 3 followed by a quiescence search.

When looking at the search sizes we can see that all strategies are in the same order for all search depths. The history heuristic with capture moves (strategy 4) searched the most nodes. The move ordering method using just the chessmaps heuristic (strategy 1) searched the second largest number of nodes. Ordering the sectors in an experience-based way (strategy 3) searched the third largest number of nodes and the strategy of ordering the sectors using the chessmaps heuristic, but also recording the top sector at each depth (strategy 2) searched the least number of nodes. With regard to the search times, the move ordering method using just the chessmaps heuristic (strategy 1) was the slowest. The other 3 strategies were very close to each other for search depths of 3–5 ply. At a search depth of 6 ply all strategies would take too long to be practical in a game-playing program, but the history heuristic with capture moves used appreciably less time. This was because of the increases in the sizes of the brute-force search tree. Recording the top sector at each depth would seem to be very effective and it helps to compensate for the times when the neural network gets the ordering badly wrong. Without this the experience-based approach would prove to be more reliable, which may be the case anyway.

6. Discussion and future work

This paper suggests an alternative move ordering method compared to the popular approach of using the history heuristic with capture moves. This method is shown to search fewer nodes but does not reduce the search times. It is also a complicated algorithm to implement and so the history heuristic may be preferred for a chess game-playing program if it can produce a similar or better performance. However, this new move-ordering algorithm may be of interest from a research point of view. It could have the pattern-oriented chessmaps heuristic as its basis, which is in line with the pattern-oriented approach of human chess players. The chessmaps heuristic attempts to represent the attack-defence relationships between the pieces, but only in a very general way. Another thing that human players do is to look for immediate tactical threats in the position and this is partly covered by firstly extracting capture and forced moves. Concerning the argument for an experience-based or knowledge-based approach, it would seem that the experience-based approach is still to be preferred. This can also be demonstrated in the move-ordering algorithm by replacing the chessmaps heuristic with an experience-based method for ordering the sectors. To perform as well as the other methods, the chessmaps heuristic needs the help of another simple experience-based heuristic that stores the top sector at each depth. The move-ordering algorithm has not been implemented in a chess game-playing program and so it is not possible to give any definite comment about its performance, but these results suggest that it might be a competitive way of producing the move ordering.

There are still various things that can be tested with this move-ordering algorithm. One thing would be to try and improve the performance of the neural network. A new definition for the influence of a move has been presented in the paper, so maybe other parameters could also be changed. No matter how accurate the neural network is however, the sector ordering can only attain a certain level of accuracy, as it only groups moves together (which

are then ordered by the closeness heuristic). So another promising area to look at would be to add other simple heuristics to the capture moves and forced moves heuristics that could suggest specific moves.

Appendix A. Figures illustrating the control of the squares and the influence of a move

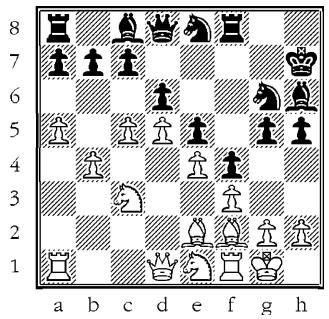

Fig. A.1. Position arising from the King's Indian Defence. Korchnoi–Kasparov, Dresden 1992.

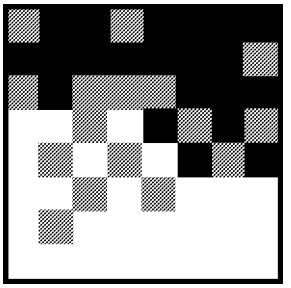

Fig. A.2. Chessmap generated from the position given in Fig. A.1. White controls the white squares. Black controls the black squares and the grey squares are neutral.

Table A.1
The relative values of each piece used in the valueboard array or as the material value of each piece used in a capture sequence. Bishop1 refers to the black squared bishops and bishop2 refers to the white squared bishops. Rook1 and rook2 refer to the rooks starting on a1 and a8 or h1 and h8 respectively

	pawn	knight	bishop1	bishop2	rook1	rook2	queen	king
White pieces	10	29	30	31	49	50	90	200 or 40
Black pieces	−10	−29	−30	−31	−49	−50	−90	−200 or −40

8	-50	-50	-171	-90	-168	-108	-89	-118
7	-60	-41	-129	-121	-119	-49	-99	-40
6	-10	-60	0	-129	-21	-168	-69	-70
5	70	60	40	139	-49	-70	-130	-10
4	169	10	31	110	49	-138	90	-99
3	50	90	29	150	20	170	30	-21
2	79	0	119	90	150	119	79	50
1	140	169	140	200	248	120	119	40
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**

Fig. A.3. Values in the valueboard array position in Fig. A.1 before the move Rc1.

8	-50	-50	-171	-90	-168	-108	-89	-118
7	-60	-41	-129	-121	-119	-49	-99	-40
6	-10	-60	0	-129	-21	-168	-69	-70
5	20	60	40	139	-49	-70	-130	-10
4	119	10	31	110	49	-138	90	-99
3	0	90	79	150	20	170	30	-21
2	29	0	169	90	150	119	79	50
1	140	169	140	200	248	120	119	40
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**

Fig. A.4. Values in the valueboard array for the position in Fig. A.1 after the move Rc1.

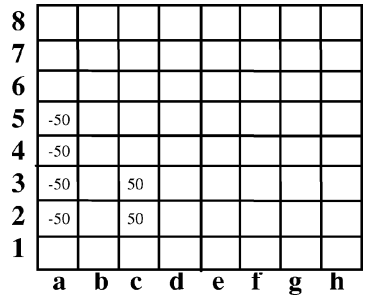

Fig. A.5. The difference in the valueboards of Figs. A.3 and A.4.

```
0 0 0 0 0 0 0 0 -1 0 1 0 0 0 0 0 -1 0 1 0 0 0 0 0 -1 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fig. A.6. A vector representing the influence of the move Rc1 on the position in Fig. A.1. There are 64 sectors numbered from a1 to h8 as in Fig. 1. This would be the output that the neural network would try to learn.

```
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0 1 -1 0 -1 1 1 0 1 -1 0 -1 0
0 -1 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 -1 -1 0 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 1
```

Fig. A.7. A vector representing the control of the squares for the chess position of Fig. A.1. Note that the positions of the kings are also included in the vector as the last 6 values. This would be the input values that are fed into the neural network.

Appendix B. An example position with the sector and move ordering for white

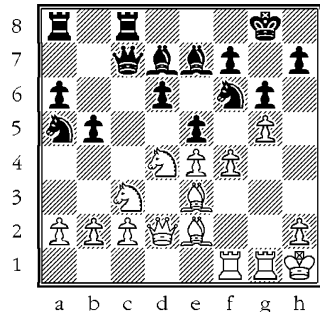

Fig. B.1. Example position for which a sector and move ordering is given below.

The sectors ordered by a neural network from the most to least important are: g4 h5 b4 c6 h3 f6 f4 c4 g6 d6 e7 e6 g7 b6 f7 c7 b7 d7 h7 d8 f8 f3 a7 d5 f5 c8 g2 b8 h8 h4 a8 a6 g8 d4 a4 a5 h6 d3 f2 b5 c5 e8 c2 e4 a3 b2 e3 h1 e5 g3 g5 h2 b3 c3 a1 d1 b1 d2 c1 f1 a2 e1 e2 g1.

The moves ordered and placed in their separate categories are:

(1) *Safe capture moves*:

WPg5 × f6 WPf4 × e5.

(2) *Safe forced moves*:

WNd4−b3 WNd4−f3.

(3) *Safe other moves*:

WQd2−d1 WPf4−f5 WPb2−b4 WPa2−a3 WNc3−d5 WNc3−b1
WNc3−d1 Wrg1−g3 WKh1−g2 WRf1−f3 WPb2−b3 WQd2−d3
WBe2−f3 WBe2−d3 WRf1−f2 WBe2−d1 WPh2−h4 Wbe3−f2
WQd2−e1 WRf1−d1 Wrg1−g2 WRf1−c1 WQd2−c1 WRf1−b1
WRf1−a1 WRf1−e1.

(4) *Unsafe capture moves*:

WNc3 × b5 WNd4 × b5 WBe2 × b5.

(5) *Unsafe forced moves*:

WNd4−f5 WNd4−e6 WNd4−c6.

(6) *Unsafe other moves*:

WPh2−h3 WNc3−a4 WBe2−g4 Wrg1−g4 WBe2−h5 WBe2−c4 WPa2−a4.

References

[1] J.A. Anderson, An Introduction to Neural Networks, MIT Press, Cambridge, MA, 1995.
[2] D.F. Beal, A generalised quiescence search algorithm, Artificial Intelligence 43 (1990) 85–98.
[3] H. Berliner, C. Ebeling, Pattern knowledge and search: The SUPREM architecture, Artificial Intelligence 38 (1989) 161–198.
[4] A.D. de Groot, Thought and Choice in Chess, Mouton, The Hague, 1965.
[5] J. Fürnkranz, Machine learning in computer chess: The next generation, ICCA J. 19 (3) (1996) 147–161.
[6] K.R.C. Greer, P.C. Ojha, D.A. Bell, A pattern-oriented approach to move ordering: The chessmaps heuristic, ICCA J. 22 (1) (1999) 13–21.
[7] D. Kopec, I. Bratko, The Bratko–Kopec experiment: A comparison of human and computer performance in chess, in: M.R.B. Clarke (Ed.), Advances in Computer Chess 3, Pergamon Press, Oxford, 1982, pp. 57–72.
[8] R.A. Levinson, A self-learning, pattern-oriented chess program, ICCA J. 12 (4) (1989) 207–215.
[9] E.M. Morales, Learning playing strategies in chess, Computational Intelligence 12 (1) (1996) 65–87.
[10] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, MA, 1984.
[11] J. Schaeffer, The history heuristic and alpha-beta search enhancements in practice, IEEE Trans. Pattern Anal. Machine Intelligence 11 (11) (1989) 1203–1212.
[12] H.A. Simon, M. Barenfeld, Information-processing analysis of perceptual processes in problem solving, Psychological Review 76 (5) (1969) 473–483.
[13] D. Wilkins, Using patterns and plans in chess, Artificial Intelligence 14 (2) (1980) 165–203.