A MINORS HASH TABLE IN CHINESE-CHESS PROGRAMS

Jiao Wang[1], *Si-Zhong Li*[2], *Xin-He Xu*[3]

ShenYang, China

ABSTRACT

With the development of computer Chinese Chess, some effective algorithms are proposed. This paper presents a new algorithm named Minors Hash Table, an innovation of hash table application specific to computer Chinese Chess. For the sake of generality, this algorithm is applicable to most other Chinese Chess programs without much difficulty of code modification. The algorithm could increase the speed of almost every program by more than 10 percent, and this leads to significant performance improvement. This paper also describes the design, operations, complexity, performance and especially Minors hash collision prevention in parallel search related to Minors Hash Table. The results of our experiments show that the proposed algorithm is reliable and stable.

1. INTRODUCTION

Chinese Chess is the most popular board game in China. Along with the great success in computer Chess, more and more researchers focus on computer Chinese Chess, which is more complex than computer Chess but less complex than computer Go (Allis L.V., 1994). Nowadays, several Chinese Chess programs are able to defeat human grandmasters. As far as computer Chinese Chess is concerned, some unique algorithms can be discovered owing to the special characteristics and rules of Chinese Chess.

In this paper, a novel algorithm is proposed, named Minors Hash Table (abbreviated as MHT in this paper). The algorithm can be easily implemented and lead to more than 10 percent speedup in search time with respect to almost every Chinese Chess program.

Several experiments are designed to prove our theory in this paper. The experiments are mainly based on NEUCHESS, one of the strongest Chinese Chess programs, which has won three world-championships and has equal strength as human grandmasters. Our program implements DTS algorithm (M. Campbell, 1988; R. M. Hyatt, 1997) and can search in multi-thread mode. In order to prove the universality of MHT, experiments are also applied in QiXing, which is one of the top ten Chinese Chess programs and we can access its code freely. The hardware environment of the experiments can be seen in APPENDIX A.

Section 2 describes the special characteristics and piece rules of Chinese Chess, and then classifies the pieces into two categories according to their basic functionality. Section 3 analyzes the possibility of MHT and presents our design. Section 4 introduces the concrete operations in MHT. Section 5 designs some experiments in order to test the performance of MHT. Section 6 analyzes the performance of MHT and discusses several key points to achieve the best performance. Section 7 demonstrates an advanced Minors hash collision prevention of MHT in parallel search. Finally, our conclusion is presented in Section 8.

2. SPECIAL CHARACTERISTICS OF MINORS

[1] College of Information Science and Engineering Northeastern University, Email: wangjiao@ise.neu.edu.cn.
[2] College of Information Science and Engineering Northeastern University, Email: littolee82@gmail.com.
[3] College of Information Science and Engineering Northeastern University, Email: xuxinhe@ise.neu.edu.cn.
The material in this paper is based upon work supported by the NSFC-MSRA Joint Research Fund under Grant 60971057.

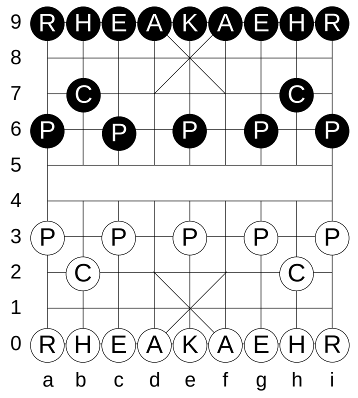

Figure 1: Board and pieces of Chinese Chess.

Chinese Chess is a two-player, zero-sum game with complete information, which includes a 90-legal-spot board and 32 pieces, as seen in Figure 1. The two sides are named RED and BLACK, whose pieces are signed with corresponding colors. The detailed characteristics of board and piece rules can be seen in the paper (Shi-Jim Yen, Jr-Chang Chen, Tai-Ning Yang, S.C. Hsu., 2004; Hans Bodlaender, 2000; Leventhal Dennis A, 1978). In the following subsections we describe and analyze the special characteristics of Chinese Chess, and then summarize some useful disciplines.

2.1 Mobility and Activity Scope of Pieces

The 32 pieces can be categorized into 7 kinds, each kind differing in their mobility and activity scope. Here, the mobility refers to the maximal number of legal moves of a specific piece, while the activity scope means the number of reachable spots of a kind of piece.

Piece	Abbreviation	Mobility	Activity Scope
rook	R	17	90
horse	H	8	90
cannon	C	17	90
pawn	P	3	55
king	K	4	9
elephant	E	4	7
advisor	A	4	5

Table 1: Mobility and activity scope of pieces.

As can be seen in Table 1, the details are discussed below.

- Rook and cannon have the largest mobility and they can reach every spot of the board.
- Pawn is never allowed to move backward and can only move forward before crossing the river. Pawn has three legal directions when it steps into the other side’s territory.
- Elephant is not allowed to cross the river. There are a total of seven legal spots for each side.
- King and advisor can move only within the palace, and both have at most four possible legal moves.

2.2 Pieces Classification

To clarify the functionality of pieces, Definition 1 is proposed.

Definition 1: (Pieces Classification)
All the seven kinds of pieces are split into two categories.
Majors= (Rook, Cannon, Horse),
Minors= (King, Pawn, Elephant, Advisor).

The classification is based on the following three perspectives.

(1) Activity scope: Majors can reach all the ninety joint points on the board, while Minors can only move within restricted scopes.
(2) Mobility: Minors have less mobility and are relatively unimportant for both attack and defense. Each Chinese Chess position has forty legal moves on average, and the contribution of Minors is relatively less before mid-game phase in comparison to Majors.
(3) Piece original functionality: Majors can either attack the opponent or protect their own king. Advisor and elephant, on the contrary, are to protect the king, whose shapes play a great role in king-safety. Therefore they can be regarded as protective pieces. Pawn is a special piece, whose limited mobility makes it scarcely attackable unless it occupies dangerous spots.

Furthermore, in some specific situations, protective pieces can also help attacking its opponent. Three special situations exist as follows.

- In Chinese Chess, cannon cannot capture pieces unless a rack is given. So Minors can be used as racks to help attacking.
- Due to king face-to-face rule, Minors can choose the right time to expose their king, so as to improve attack intensity.
- Minors can limit the mobility of enemy's pieces, which is another kind of attack.

3. DESIGN OF MINORS HASH TABLE

Hash table is an important technique in computer games. Transposition Table (Nelson H.L., 1985; Breuker, D. M., Uiterwijk, J. W. H. M., Herik and H. J. van den., 1996; Breuker, D. M., Uiterwijk, J. W. H. M. and Herik, H. J. van den., 1997) and Pawn Table (R.M. Hyatt, R. and H.L. Nelson., 1985) are widely used in computer Chess, both of which are based on hash table. Apart from them, MHT is another extended application of hash table in computer Chinese Chess.

This section demonstrates the design and implementation of MHT. Section 3.1 analyzes the principle of MHT. Section 3.2 describes the function design of the algorithm. Section 3.3 explains the typical structure designed in NEUCHESS.

3.1 Minors Unchanged Rate

We have classified all the pieces into Majors and Minors in Section 2. Chinese Chess is a fierce game, and Majors moves occupy higher proportion than Minors moves in the game tree. Therefore Majors moves account for the vast majority of all branches, and Minors moves remain unchanged on those nodes. On the other hand, evaluations of game positions account for most computing time and are mainly invoked on leaf nodes. Thus we are more concerned about the rate at which Minors keep unchanged on leaf nodes. To validate this hypothesis, Definition 2 is proposed.

Definition 2: (Minors Unchanged Rate and Majors Unchanged Rate)
Several abbreviations are defined as follows.
L: The set of leaf nodes which invoke the evaluation function.
MI: The number of unique Minors status in L.
MA: The number of unique Majors status in L.
N: The number of elements in L.
MIUR: Minors Unchanged Rate.
MAUR: Majors Unchanged Rate.

MIUR and MAUR can be computed as below.

$$MIUR = 1 - MI/N$$
$$MAUR = 1 - MA/N$$

An experiment is designed on NEUCHESS to gain MIUR and MAUR. Five phases are categorized according to Majors material value (rook accounts for 2, horse and cannon accounts for 1, therefore the Majors number is the sum of all Majors corresponding value of both sides on the board). One thousand positions are collected in each of the five phases respectively, and the search configuration can be seen in Appendix B. The data on leaf nodes is collected.

Majors material value	MI	MA	N	MIUR	MAUR
11~16	533,855	23,236,341	36,592,662	98.5%	36.5%
9~10	632,869	6,112,267	11,467,666	94.5%	46.7%
7~8	1,720,133	3,793,245	9,852,583	82.5%	61.5%
4~6	1,553,888	1,392,894	5,902,084	73.7%	76.4%
0~3	214,442	92,086	719,429	70.2%	87.2%

Table 2: MIUR and MAUR on leaf nodes.

MIUR on leaf nodes is very high in all five phases, as can be seen in Table 2. With the Majors number decreasing, the MIUR decreases too. Comparing with MIUR, MAUR is relatively low when Majors material value is greater than 6. This indicates the Majors contribute more to the game tree than Minors before endgame, especially in opening-game.

Moreover, there can be some correlation between MIUR and forward pruning search algorithm (Smith, S. J. J. and Nau, D. S., 1994; Y. BjÄornsson, T.A. Marsland and J. Schaeffer., 1997). NEUCHESS uses many forward pruning algorithms, much more than QiXing. The same experiments applied on QiXing, and the MIUR is higher than that in NEUCHESS in all phases. Because forward pruning algorithms cut many nodes due to the predicted influence of moves, and Minors moves are generally considered unimportant. Therefore most of the Majors moves remain and are used to expand the game tree. Under such condition, it is evident that MIUR is relatively lower. The issue will be discussed detailedly in Section 5.2.

3.2 Function Design of Minors Hash Table

It is a very special characteristic of Chinese Chess that MIUR is very high on leaf nodes, suggesting that computations may be saved if repeated evaluations of Minors can be avoided. If we can store Minors information in some kind of table and retrieve it when Minors status is matched, the goal can be achieved. Fortunately, hash table is a nice tool to solve similar issues in the field of computer games. We propose an algorithm entitled MHT, which is based on hash table.

Minors evaluation should be separated from global evaluation before implementing MHT. Minors evaluation evaluates Minors irrespective of the Majors status. In addition, necessary operations of MHT are carried out which consist of storage and retrieval processing in evaluation function.MHT can provide not only evaluation score, but also some valuable Minors information. The status of Minors is significant for global evaluation especially in king-safety evaluation in Chinese Chess. MHT can provide that information when probe hits instead of computing them in the global evaluation.

It should be noted that MHT is only used in the evaluation function, not in the search process as Transposition Table. Transposition Table stores sub-game-tree search information of temporal nodes and the information can be used when identical position is found. Transposition Table is an algorithm with a bit risk because almost all programs use the information for forward pruning. As far as MHT is concerned, it is a harmless algorithm without any risks.

However, the MAUR is also very high in all phases. But due to the characteristics of Chinese Chess, it does not make sense to evaluate Majors separately.

3.3 Attributes of Minors Hash Table

An example that is applied in NEUCHESS can be seen in Table 3.

Type	Bits	Name
U64	64	checksum
int	32	score
char	8	phase
char	8	advisor_num_r
char	8	advisor_num_b
char	8	elephant_num_r
char	8	elephant_num_b
char	8	pawn_num_r
char	8	pawn_num_b
char	8	fatal_pawn_num_r
char	8	fatal_pawn_num_b
char	8	king_door_r
char	8	king_door_b

Table 3: Attributes of one entry in MHT.

The checksum and score are two fundamental attributes of MHT.

- **Checksum:** The 64-bit checksum is the hash key for verification. It is a very common practice in hash table. In Section 4 this shall be discussed in more detail.
- **Score:** Evaluation score for Minors. It should be noted that the score is set in the respect of the red side.
- **Phase:** Phase is a very special attribute in MHT. In Chinese Chess, the Minors importance constantly changes with the progress of the game. Normally, almost every advanced program has several very different evaluation functions for Minors, corresponding to different phases. The phase attribute is to represent the evaluation functions by which the Minors score is computed. Furthermore, the score is valid only if the phase attribute stored in the table is identical to the phase of current node, where the evaluation function is called. Phase indicates the Majors information corresponding to the Minors score. In NEUCHESS, there are two values for phase, one is *BeforeEndgame* and the other is *InEndgame*. *BeforeEndgame* means the current processing of the game is in opening-game or mid-game, while *InEndgame* is in endgame. However, it does not matter if readers separate the whole game into more phases.

Other attributes represent important information of Minors.

- **pawn_num_r, pawn_num_b:** Pawns' number for each side. The information is essential for evaluation especially in endgame.
- **elephant_num_r, elephant_num_b, advisor_num_r, advisor_num_b:** Elephants' and advisors' number for each side. Those purely protective pieces' numbers are the key points to distinguish whether the king is in danger or not.
- **king_door_r, king_door_b:** Some special shapes of Minors can be stored in the table, e.g. King-door. King-door is the only escape spot of king when two advisors form a kind of shape.

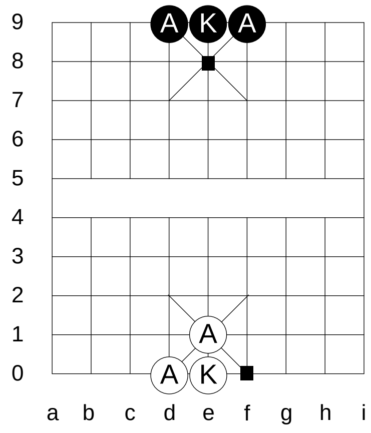

Figure 2: King-door.

Figure 2 shows an example of king-door. The spot marked bold dot is the king-door. King-door is the only escape spot of king when two advisors form a special kind of shape. It is very dangerous when king-door is controlled by opponent and many enemy Majors are attacking.

- **fatal_pawn_num_r, fatal_pawn_num_b:** Some pawns are extremely dangerous with the help of Majors when they occupy important spots.

The attributes listed above are only the typical implementation details of NEUCHESS, and should not be unchangeable rules. Actually, we have omitted some attributes which are hard to understand for readers. Attributes can be customized to meet the specific demands of readers.

4. OPERATIONS OF MINORS HASH TABLE

This section describes the operations of MHT, including initialization (Section 4.1), addressing and verification (Section 4.2), and storage/retrieval processing (Section 4.3).

4.1 Initialization

4.1.1 Preparing Zobrist Array for MHT

Normally, a Chinese Chess position is comprised of pieces specific information and the side to move, but the latter can be neglected in MHT. Minors' information can be encoded by 64-bit hash key as signature values by Zobrist hashing (Zobrist A.L., 1988).

Preparing Zobrist hashing array goes first in initialization, which is a two-dimensional arrays and each attribute contains a 64-bit random number.

$$U64\ Zobrist[9][55] \tag{1}$$

The first dimension denotes piece type, and the second coordinates. There are eight kinds of pieces (king, advisor, elephant and pawn for both sides) for Minors, and totally 9 kinds in the first dimension with zero meaning no piece. Pawn has the largest activity scope among Minors and it is obvious that the maximum size of the second dimension is 55. MHT and Transposition Table are two very different applications without conflict, therefore we can simply reuse the random number array built by Transposition Table, avoiding building another coordinate system of Minors.

4.1.2 Memory Allocation

Before using MHT, a block of memory should be allocated to avoid instant allocation. Fortunately, each entry occupies a fixed memory size and the maximal entry number (signed as S_{bucket}) can be set in initialization. Moreover, all the entries should be reset before being used. Minors signature (signed as MS) of any position is computed by Formula 2. Only the Minors status is concerned, therefore the right side of the formula only includes Minors information of both sides.

$$MS = \underset{m \in Minors}{XOR}\ Zobrist[m][Pos(m)] \tag{2}$$

Where $Minors$ is the set of Minors pieces and $Pos(\bullet)$ means the coordinate of a specific piece. It is a time-consuming way to generate Minors signature every time entering evaluation function. However, we could record and maintain the Minors signature throughout the game tree, which is a better mechanism to achieve the same result (Zobrist A.L., 1988).

4.2 Addressing, Verifying and Replacing

Using 0 to 2^{64}-1 to create index of MHT is reasonable, so that all the address and entries are one-to-one mapped. However it is astonishing when we calculate the memory requirement. In our example, each entry occupies 192-bit memory, and totally 402,653,184T RAM is in demand. For contemporary computers, it is impossible to afford such a large memory, therefore a compromise scheme should be found. Fortunately, there is a successful experience of solving similar problems in computer games. In that approach, 64-bit Minors signature is limited within 0 to $S_{bucket} - 1$. Formula 3 shows how to calculate the address.

$$Address = MS \% S_{bucket} \tag{3}$$

In accordance with that method, it is inevitable to cause collision that different positions can be mapped to the same address. In retrieval processing, to solve this kind of collision, the 64-bit Minors signature is also stored in the table, which is called checksum. Thus additional verification operation should be included in retrieval processing. While in storage processing, newer data should replace older ones.

4.3 Processing of Minors Hash Table

In the storage processing, old data is always replaced by the new one. In the retrieve processing, verification operation confirms that the data is exactly corresponding with current Minors status. Algorithm 1 demonstrates the processing of MHT in evaluation function, written in C++.

```
//the procedure of MHT based on original evaluation
0   int Evaluation (GameTree *tree, int rtm){
         int score; unsigned __int64 address;
        MINOR_HASH_ENTRY * entry, entryData;
        //address computing
1       address = (tree->MinorSignature)%BUCKETSIZE;
        //probe corresponding entry
2       entry = MinorsHashTable[address];
        //verification using checksum
3       if (entry->Checksum != tree->MinorsSignature
          || entry->phase != tree->CurrentPhase) {
            //not hit or not same phase, evaluate Minors normally
4           entryData = EvaluateMinors();
           //store the information into MHT
5          *entry = entryData;
        }
        //evaluate other things using Minors information
6       score = GlobalEvaluate(entry);
```

```
        //add Minors evaluation score
7       score += entry->score;
        //return the score, rtm(red to move) is a variable to indicate the moving side.
8       return (rtm ? score : -score);
    }
```

Algorithm 1: Processing of MHT in evaluation.

The bold lines are the additional operations beyond original evaluation function. If the Minors signature of the current position is equal to the checksum of the found entry, and meanwhile the phase value in the entry is identical to that of the current node, the Minors information can be directly retrieved and used. Otherwise all the attributes are filled by Minors evaluation function and stored into MHT as one entry.

4.4 Using MHT Information in Global Evaluation

It is easy to understand that Minors evaluation can be skipped when MHT hits, but how to use MHT information in global evaluation need to be concerned. So a simple example is given as follows.

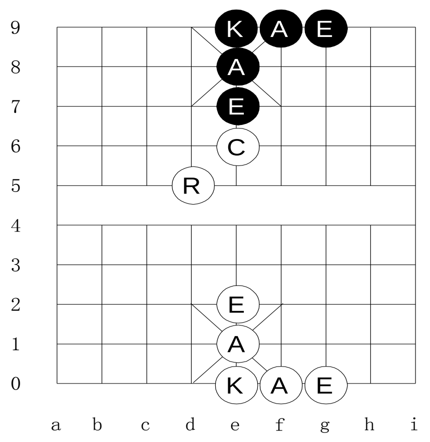

Figure 3: An example about using MHT information.

Can be seen in the Figure 3, the red side has a right overhead cannon and a rook controls black king_door, and it is extremely dangerous for the black side. To distinguish this special posture, several conditional judgments are executed in global evaluation function. Before this processing, MINOR_HASH_ENTRY which records MHT information is filled (line 1 to line 5 in Algorithm 1). So after the red right overhead cannon and "d" line rook being confirmed, the coordinate of black king_door can be directly achieved from MINOR_HASH_ENTRY. It is obviously that MHT can save time by avoiding king_door computing.

The given example above is a typical application of using MHT information in global evaluation, and actually much more similar cases can be found in NEUCHESS, especially in king-safety evaluation function.

5. COMPLEXITY AND PERFORMANCE IMPROVMENT

This section will discuss the complexity and performance improvement of MHT. Section 5.1 presents theoretical analysis. Section 5.2 discusses how to choose the appropriate MHT size. Section 5.3 designs experiments to show the performance improvement.

5.1 Time Complexity Analysis

Expectation of random variable X can be denoted as $E(X)$. The expectation of evaluation time without MHT can be computed by Formula 4.

$$E(T_{ENM}) = E(T_{EG}) + E(T_{EM}) + E(T_{GMS}) \quad (4)$$

Where T_{ENM} is the time for evaluation without MHT, T_{EG} for evaluating global information of the position except Minors, T_{EM} for evaluating Minors, and T_{GMS} for generating Minors status.

The expectation of evaluation time with MHT can be denoted as Formula 5. It should be noted that the cost in updating MS in game-tree is negligible and it is not made into account.

$$E(T_{EWM}) = E(T_{EG}) + E(T_{MR}) + (1-p) \times (E(T_{EM}) + E(T_{GMS}) + E(T_{MS})) \quad (5)$$

Where T_{EWM} is the time for evaluation with MHT, T_{MR} for MHT retrieval, and T_{MS} for MHT storage. p is the probability of MHT probe hit. Although building 64-bit index is impossible, our experiments show that p is close to MIUR if BUCKETSIZE is big enough (at least 2^{19}).

The difference between T_{ENM} and T_{EWM} can be computed by Formula 6.

$$Diff = E(T_{ENM}) - E(T_{EWM}) = p \times E(T_{EM}) + p \times E(T_{GMS}) - (1-p) \times E(T_{MS}) - E(T_{MR}) \quad (6)$$

Normally, T_{EM} is very complex and p is very high, however both the T_{MS} and T_{MR} can be negligible compared to T_{EM}. Therefore Diff is a positive number and MHT can bring significant time-saving in almost all cases.

5.2 Best Choice of MHT Size

In the implementation of NEUCHESS, all the attributes are encoded into three 64-bit variables, in order to eliminate hash collision in parallel search (more details can be seen in Section 7.2), so each entry occupies 192-bit. Readers may wonder how to choose the appropriate MHT size, in other words, how many entries are set in MHT is suitable. In this sub-section, the issue shall be discussed in two aspects.

5.2.1 Probe Hit Rate and MHT Size

As can be seen in Formula 6, only if p is very high and approximates MIUR, the performance improvement is prominent. Whereas, since complete indexing is impossible and collisions inevitably happen, p could not reach MIUR.

In normal sense, p has close connection with MHT size, and bigger memory space means fewer collision occurrences. An experiment is designed to discover the relationship between them. One thousand opening-game positions are collected, and the search configuration can be seen in Appendix B. The statistics of probe hit rate is collected for different MHT sizes.

Probe hit rate	Entry number (by power of 2)	MHT size (MB)
98.34%	20	24
98.70%	21	48
98.96%	22	96
99.08%	23	192
99.14%	24	384
99.18%	25	768

Table 4: Probe hit rate and MHT size in opening-game.

As shown in Table 4, with the increment of MHT size, the probe hit rate increases too. Probe hit rate exceeds 99% and is close to MIUR when MHT size is bigger than 96MB. The experiment is based on opening-game positions. Furthermore, the experiment is applied on mid-game and endgame, and the results show they have similar property on probe hit rate as opening-game.

5.2.2 Performance Improvement and MHT Size

As discussed in Section 5.2.1, bigger MHT size means higher probe hit rate and lower collision probability, so more useful information could be stored in the table and more Minors evaluation time can be saved. However, it may not be concluded that the performance is directly in proportion to MHT size, as operations on larger space in memory cost more time. As shown in Formula 6, bigger MHT size means higher p, but T_{MR} and T_{MT} increase at the same time. Therefore, it is hard to say whether $Diff$ is positive.

An experiment is designed to clarify the correlation between the performance improvement and MHT size. One thousand positions from opening-game are collected, and the search configuration can be seen in Appendix B. Then the MHT size is adjusted and the corresponding consumed time is recorded. It should be noted that the average computing time without using MHT is 1398 seconds.

Average computing time (Second)	Performance improvement	Entry number (by power of 2)	MHT size (MB)
1194	14.59%	17	3
1184	15.31%	18	6
1173	16.09%	19	12
1170	16.30%	20	24
1168	16.45%	21	48
1167	16.52%	22	96
1173	16.09%	23	192
1176	15.88%	24	384
1185	15.24%	25	768

Table 5: Performance improvement and MHT size.

As can be seen in Table 5, the computing time corresponding different MHT size does not vary a lot. If the MHT size is lower than 3MB or higher than 768MB, the computing time is relatively longer. As far as contemporary computers are concerned, the appropriate setting of MHT size is at least 24MB and not more than 384MB, which can be regarded as empirical solution for the selection of MHT size.

5.3 Experiments on Performance Improvement

An experiment is designed to show performance enhancement, and the search configuration can be seen in Appendix B. One program is equipped with 96MB MHT and the other is not. Under such configuration, all the search details of two programs are identical except for computing time. One thousand games are played on MOVESKY (The most famous Chinese Chess web platform) using the program with MHT. After that the program without MHT runs the games again. The collected data is shown in Figure 4.

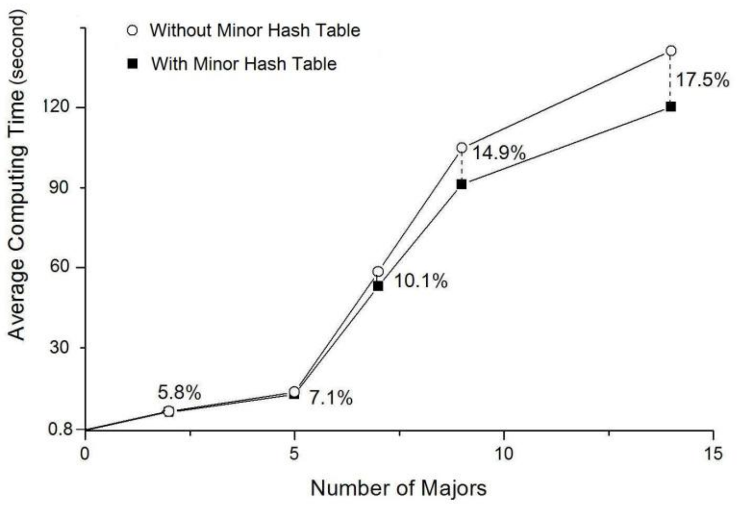

Figure 4: Performance improvement trend.

It can be seen that the performance improvement ratio decreases with the proceeding of games. The trend is accordant with the MIUR in Table 2, and the performance improvement ratio is very amazing especially before mid-game.

Furthermore, same experiments are applied on QiXing and 16.5% performance improvement ratio is gained on average, as QiXing has a more complex Minors evaluation and higher probe hit rate.

6. PERFORMANCE ANALYSIS

This section discusses the performance of MHT. Section 6.1 discusses the correlation between search strategy and performance improvement. Section 6.2 analyzes the performance discrepancy when we reset the MHT table after each move or not. Section 6.3 introduces the concept of occupancy rate and analyzes it in two different modes.

6.1 Search Strategy and Performance Improvement

As mentioned in Section 3.1, search strategy is closely correlated with MIUR, and it influences the performance of MHT to a large extent. Unlike Deep Blue, which is based on brute force search and relied on custom-built hardware(Hsu, Feng-hsiung, 2002; Murray Campbell, A. Joseph Hoane Jr. and Feng-hsiung Hsu., 2002), more selective search algorithms are developed in present-day Chess and Chinese Chess programs(Ye.C. and Marsland T.A., 1992; Marsland, T. A.,1986). The advanced programs normally utilize many forward pruning search algorithms. The efficiency and accuracy of search is the joint action of all search algorithms, and they may influence each other. These typical forward pruning algorithms include Nullmove (Donninger C., 1993; Heinz E.A., 1999; Omid David and Netanyahu N., 2002), Razor (Birmingham J.A. and Kent P., 1977; Heinz E.A., 2000), Futility (Heinz, E.A., 1998) and LMR (also known as History Pruning or History Reductions. Winands, M.H.M., Werf, E.C.D. van der and Herik, H.J. van den, 2006; Tord Romstad, 2003).

An experiment is designed to prove the correlation. In preparing for the experiment, the 1000 opening-game and mid-game positions are collected, since all the forward pruning search algorithms are forbidden in endgame in practice. To simplify the experiments, we disable all the forward pruning algorithms at first, and then we gradually enable each algorithm in NEUCHESS. Subsequently we make the program search the position, and the configuration can be seen in Appendix B. This experimental approach can eliminate the mutual effect of the algorithms. The collected experimental data is shown in Table 6.

Forward pruning algorithms enabled	Improvement rate of MHT
None	+15.52%
LMR	+13.99%

Razor and Futility	+13.60%
Nullmove	+12.91%
Enable all the above	+12.87%

Table 6: MHT performance and search algorithms.

Seen in Table 6, the search algorithms listed in the left column are sorted in ascending order according to their cutting aggressiveness, their corresponding improvement rate of MHT decreases. The improvement rates are all prominent, especially disabling all of them, in which case the program is some kind of a brute-force one.

We can more deeply explore the relationship between search strategy and performance improvement. The total search time can be computed as Formula 7. (total time is signed as *TT*, search processing is signed as *SP*, move generation is signed as *MG* and evaluation is signed as *EV*)

$$E(T_{TT}) = E(T_{SP}) + E(T_{MG}) + E(T_{EV}) \tag{7}$$

Where T_{SP} is the requisite time to proceed the search, including expansion and resumption of game tree, search information storage and retrieval, move ordering, operations on search tables and timing etc. T_{MG} is the time to generate needed legal moves to meet specific requirements, and T_{EV} is the time to evaluate static positions.

It is known T_{EV} can be reduced by MHT. Providing that the T_{EV} occupies higher proportion in T_{TT}, the time saved by MHT is more remarkable. It should be noted that more aggressive the search strategy, and more time consumed by T_{SP} and T_{MG}, lead to lower proportion of T_{EV} and less performance improvement of MHT.

The experimental statistics prove our hypothesis: the more aggressive the search strategy is, the lower the performance improvement rate of MHT.

6.2 Reset or Not

Unlike Transposition Table, the Minors shape information and their static evaluation score is irrelevant to game tree, therefore the search boundary and other search information could not be poisoned by MHT. All the Minors information retrieved from MHT is trustable and need not any additional verification, no matter whether the data is produced by current or former search processing.

Since the Minors significance has much difference in endgame phase and former phases (as can be seen in Section 3.4), the date stored in former phases is useless for endgame phases. For that reason, all the entries of MHT should be reset as game goes into endgame from later- mid-game.

The “reset” topic can be thought more deeply. How would the performance change if MHT is cleared after each move? It is obviously that the question link tightly with MHT size. In the experiment, two programs are built such that the first one resets MHT after each move and the other one does not. As we all know, providing that the two programs start at same position and they are limited to the same fixed search depth and single thread mode, the same game procedure would be got and only the computing time is different. Therefore one thousand opening-game positions are collected as start position. The two programs are set as shown in Appendix B, and they hold self-play games and the computing time in the games are accumulated.

The performance is inversely proportional with accumulated computing time, and the comparison of accumulate search time can be regarded as performance difference, which can be computed as Formula 8.

$$PerformanceDiff = (E(T_{NoReset}) - E(T_{Reset})) / E(T_{Reset}) \tag{8}$$

Where $T_{NoReset}$ is the accumulated computing time of the program does not reset MHT, while T_{Reset} is for the program that resets MHT after each move. The MHT size is set to different values and the experimental results are shown in Table 7.

Performance difference	MHT size (MB)
+0.18%	24
+0.03%	48
+0.12%	96
+0.11%	192
+0.30%	384
+0.42%	768

Table 7: Performance with reset versus without reset.

As can be seen in the table, if we clear the MHT after each move, not more than 0.5% performance is gained at all MHT size settings. It can be inferred that the performance of MHT has close connection with the pertinence of game tree. The Minors information generated by previous searches has little use to later search processing. Therefore whether resetting MHT after each move or not has little influence on MHT performance.

6.3 Occupancy Rate

The performance improvement of MHT can be seen generally in Section 5.2, but how many entries have useful data and the ratio to total entry number are still unknown. The ratio is named as occupancy rate.

In the implementation of NEUCHESS, each entry occupies 192-bit (more details can be seen in Section 7.2). So the entry number can be computed according to the MHT memory size. For example, 96MB MHT size means there are 2^{22} entries in the table.

To demonstrate the relationship between occupancy rate and MHT size, two experiments are designed. In both experiments, and the search configuration can be seen in Appendix B, after that the MHT size is changed and NEUCHESS plays 1000 games with QiXing from different start positions, which are selected by random in opening book. It is obvious that the same game procedure would be got, and the different MHT sizes lead to different computing times and average occupancy rates.

The only difference between the two experiments is that the first experiment resets all the entries after each move and the second one does not. The result of the first experiment is shown in Table 8.

Average occupancy rate	Used memory (KB)	MHT size (MB)
65.5%	16,098	24
46.9%	23,053	48
29.7%	29,197	96
16.9%	33,227	192
9.2%	36,176	384
4.8%	37,749	768

Table 8: Occupancy rate with reset operations.

As mentioned in Section 6.1, it is not necessary to reset MHT after each move in real circumstance, but the operation can avoid confusion. The trend of the occupancy rate can be clearly seen in all different sizes and different phases. As can be seen in the Table 8, the average occupancy rate corresponding with 24MB size is 65.5%, which means there are 686,818 from 1,048,576 entries having useful data while others being empty. The average occupancy rate decreases and the used memory increases with the increment of the MHT size. When the MHT size reaches 768MB, only 4.8% of all entries have useful data.

The second experiment shows the variation of occupancy rate if we do not reset MHT after each move. It means that the information in MHT which generated by the former search processing can be used for the later ones.

Peak occupancy rate	Used memory (KB)	Move number to reach peak occupancy rate	MHT size (MB)
99.9%	24,552	18	24
99.8%	49,054	52	48
99.6%	97,911	69	96
94.4%	185,598	76	192
75.9%	298,451	76	384
51.1%	401,867	76	768

Table 9: Occupancy rate without reset operations.

In the experiment, it can be found that the occupancy rate reaches certain value and hardly changes with the carrying out of the game, and the value is named as peak occupancy rate. The peak occupancy rate and the average move number to stabilize it corresponding to different MHT sizes are shown in Table 9. The peak occupancy rate reaches 99.9% when the MHT size is set to 24MB or less, and decreases with the increment of MHT size. Only 51.1% of all entries are useful when the MHT size is 768MB. But the used memory increases with the increment of the MHT size, the same as that in reset mode.

7. LOCKLESS ALGORITHM IN PARALLEL SEARCH

Many Chinese Chess programs use parallel search algorithm to make full use of computing resource of multi-core computer and gain better performance (M. Campbell., 1988; M.G. Brockington., 1996; Valavan Manohararajah., 2001). This section discusses how to prevent hash collision in MHT, which is a very common issue in parallel search (Warnock, T. and Wendroff, B., 1988). Section 7.1 discusses the motivation and derivation of lockless algorithm. Section 7.2 presents the modification for lockless algorithm in primary MHT scheme. Section 7.3 designs an experiment to show the performance of lockless algorithm.

7.1 Motivation and Derivation of Lockless Algorithms

In parallel search, hash collision inevitable happens if no prevention is adopted (R.M. Hyatt and Cozzie A., 2005). To prove the existence of hash collisions of MHT, an experiment is designed. The search configuration of NEUCHESS can be seen in Appendix B and the verification is added. When probe hits, the normal Minors evaluation function is called to distinguish whether the retrieved information from MHT is correct, and the disparity implies the collision happens.

One hundred games are played on CHESSSKY using this program. As is expected, Minors hash collision occurs occasionally. The emergence of collisions is random, in each position search, *26* collisions found under the worst condition but no collision happens sometimes. On average, approximately *8* collisions are found in each search. Although the collision rate is low compared with the amount of searched nodes (about 15 million per search on average), it may lead to wrong evaluation score and poison the search tree. Even in some specific cases, the search results could be influenced by Minors hash collision.

According to the experience in preventing generating corrupt data in parallel processing, lock/unlock technique and mutex signal control are two popular methods. The common collision preventions in parallel search incur a significant penalty, because the frequency of storage/retrieval operation is rather high in computer games. The nice solution is lockless algorithm, whose original idea comes from Dr. Robert Hyatt (R.M. Hyatt and T. Mann., 2002), and we need to modify MHT to meet the requirements.

7.2 Implementing Lockless Hashing in MHT

To implement lockless algorithm, there are two steps to complete the modification in our primary scheme.

(1) New Attributes Design and Wrapping Information

New entry has three 64-bit attributes, and the requisite information should be wrapped into them. Because bitwise is used to record the information, an 8-bit char like Table 3 is not necessary. Detailed information is shown in Table 10.

Bits	Name	Category(64 bit)
64	checksum	VerificationKey
32	score	ScoreData
2	phase	InforData
2	advisor_num_r	
2	advisor_num_b	
2	elephant_num_r	
2	elephant_num_b	
3	pawn_num_r	
3	pawn_num_b	
3	fatal_pawn_num_r	
3	fatal_pawn_num_b	
7	king_door_r	
7	king_door_b	

Table 10: Detail information.

(2) Encoding and Decoding

In storage processing, the *ScoreData* and *InformationData* are built by wrapping information, but *VerificationKey* should be computed by Formula 9.

$$VerificationKey = ScoreData \;\; XOR \;\; InforData \;\; XOR \;\; MinorSignature \qquad (9)$$

In retrieval processing, we use Formula 10 to validate data's correctness. The codes replace 3th line in Algorithm1.

$$\begin{aligned} &if\,(entry->ScoreData\; XOR\, entry->InforData\; XOR\, entry->VerificationKey \,!= MinorsSignature \\ &\quad ||\;(entry->InforData \& 0X\,2 \,!= CurrentPhase) \end{aligned} \qquad (10)$$

7.3 Accuracy and Performance of Lockless Algorithm in MHT

(1) Accuracy Analysis

Lockless algorithm allows different threads to write and read the same memory block concurrently. Bad data can be retrieved as well, but further verification is carried out on retrieved data. Position's information is involved in the data, therefore it can be quickly found whether the fetched information is corresponding to the current position.

In this new scheme, thousands of test games are carried out and no Minors hash collision is found anymore.

(2) Performance Analysis

The encoding and decoding in lockless algorithm take some computing time. Encoding operation costs more than decoding operation does, but encoding operation happens in higher proportion due to high hit rate. Each encoding operation plus decoding operation cost 7×10^{-5} second under our experiment circumstance. On average, those operations take nearly 0.03% computing time.

The lockless algorithm slows down the program to a certain extent, but the cost is acceptable. However, it is wise and reasonable to spend a little computing time in exchange of more accurate computation. An additional gift is that more information can be wrapped into the each entry of MHT than the former scheme. In our example, after wrapping all information in Table 10, there are still 60 unused bits available for more purposes.

8. CONCLUSION

The paper introduces a new algorithm named Minors Hash Table in computer Chinese Chess. All the implementation issues of MHT are presented in detail, which includes structure design, storage and retrieval processing etc. Furthermore, the complexity of MHT and several key factors which influence MHT performance are analyzed. Finally, the Minors hash collision prevention when implementing MHT in parallel search is discussed. The experimental results prove that the MHT is stable and remarkably improves performance for almost every program.

On the other hand, because there is no ELO rating system (Arpad E.Elo., 1978) in computer Chinese Chess, which has been proved efficient in computer Chess, we could not quantify the totally performance improvement. The experiments in the paper mainly base on self-play or specific opponent games, and the experimental results show the performance indirectly.

9. REFERENCES

Allis L.V. (1994). Searching for Solutions in Games and Artificial Intelligence. *Ph.D. Thesis*, University of Limburg, Maastricht, The Netherlands.

M. Campbell. (1988). Algorithms for the Parallel Search of Game Trees. M. Sc. *Thesis*, Technical Report TR 81-8, Computer Science Department, University of Alberta, Edmonton.

R. M. Hyatt. (1997). The Dynamic Tree-splitting Parallel Search Algorithm. *ICCA Journal*, Vol.20, No.1, pp.3-19.

Shi-Jim Yen, Jr-Chang Chen,Tai-Ning Yang and S.C. Hsu. (2004). Computer Chinese Chess. *ICGA Journal*, Vol.27, No.1, pp.3-18.

Hans Bodlaender. (2000). Chinese Chess. http://www.chessvariants.com/xiangqi.html.

Leventhal, Dennis A. (1978). The Chess of China. *Mei Ya Publications*.

Nelson H.L. (1985). Hash Tables in Cray Blitz. *ICCA Journal*, Vol.8, No.1, pp.3-13.

Breuker, D. M., Uiterwijk, J. W. H. M. and Herik, H. J. van den. (1996). Replacement schemes and two-level tables. *ICCA Journal*, Vol.19, No.3, pp.175-180.

Breuker, D. M., Uiterwijk, J. W. H. M. and Herik, H. J. van den. (1997). Information in Transposition Tables. *Advances in Computer Chess 8* (eds. H.J. van den Herik and J.W.H.M. Uiterwijk), pp. 199-211. Universiteit Maastricht, Maastricht, The Netherlands. ISBN 90-6216-2347.

R.M. Hyatt, R. and H.L. Nelson. (1985). Cray Blitz. *Advances in Computer Chess 4*, D. Beal(ed.). Pergamon Press, Oxford, pages 8-18.

Smith, S. J. J. and Nau, D. S. (1994). An Analysis of Forward Pruning. *In Proceedings of 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 1386-1391.

Y. BjÄornsson, T.A. Marsland and J. Schaeffer (1997). Searching with Uncertainty Cut-offs. *ICCA Journal*, Vol.20, No.1, pp.29-37.

Zobrist, A.L. (1988). A New Hashing Method with Application for Game Playing. *Technical Report. 88*, Computer Science Department, the University of Wisconsin, Madison.

Hsu, Feng-hsiung. (2002). Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. *Princeton University Press*, ISBN 0-691-09065-3.

Murray Campbell, A. Joseph Hoane Jr. and Feng-hsiung Hsu. (2002). Deep Blue. *Artifical Intelligence*. Vol.134, No.1, pp.57-83.

Ye.C. and Marsland, T.A. (1992). Experiments in Forward Pruning with Limited Extensions. *ICCA Journal*,Vol. 15, No. 2, pp. 55-66.

Marsland, T. A. (1986). A Review of Game-tree Pruning. *International Computer Chess Association Journal*, Vol.9, No.1, pp.3-19.

Donninger C. (1993). Null Move and Deep Search: Selective Search Heuristics for Obtuse Chess Programs. *ICCA Journal*, Vol. 16, No. 3, pp137-143.

Heinz E. A. (1999). Adaptive null-move pruning. *ICCA Journal*, Vol. 22, No. 3, pp. 123-132.

Omid David and Netanyahu N. (2002). Verified null-move pruning. *ICCA Journal*, Vol. 25 No. 3, pp. 153-161.

Birmingham, J.A. and Kent, P. (1977). Tree-searching and tree-pruning techniques. *Advances in Computer Chess 1*, M.R.B. Clarke (ed.), pp. 89-107, Edinburgh University Press.

Heinz E. A. (2000). AEL Pruning. *ICCA Journal*, Vol. 23, No. 1, pp. 21-32.

Heinz E. A. (1998). Extended Futility Pruning. *ICCA Journal*, Vol. 21, No. 3, pp.75-83.

Winands, M.H.M., Werf, E.C.D. van der and Herik, H.J. van den. (2006). The Relative History Heuristic. *Computers and Games, Lecture Notes in Computer Science* (LNCS 3846) (eds. H.J. van den Herik, Y. Björnsson, and N.S. Netanyahu), pp. 262-272.

Tord Romstad. (2003). An Introduction to Late Move Reduction. http://www.glaurungchess.com/lmr.html.

M. Campbell. (1988). Algorithms for the Parallel Search of Game Trees. M. Sc. Thesis, *Technical Report TR 81-8*, Computer Science Department, University of Alberta, Edmonton.

M.G. Brockington. (1996). A Taxonomy of Parallel Game-tree Searching Algorithms. *ICCA Journal*, Vol.19, No.3, pp.162–174.

Valavan Manohararajah. (2001). Parallel Alpha-beta Search on Shared Memory Multiprocessors. *Master's thesis*, Graduate Department of Electrical and Computer Engineering, University of Toronto, Canada.

Warnock, T. and Wendroff, B. (1988). Search Tables in Computer Chess. *ICCA Journal*, Vol. 11, No. 1, pp.10-13.

R.M. Hyatt and Cozzie A. (2005). The Effect of Hash Signature Collisions in a Chess Program. *ICCA Journal*, Vol.28, No.3, pp.131-139.

R.M. Hyatt and T. Mann. (2002). A Lock-less Transposition Table Implementation for Parallel Search Chess Engines. *ICCA Journal*, Vol.25, No.2, pp.63-72.

Arpad E.Elo. (1978). The Rating of Chessplayers, Past and Present. *Ishi Press*.

10. APPENDICES

APPENDIX A: EXPERIMENTAL SETUP

- NEUCHESS 6.0 by NEUCHESS team built in January 2009.
- QiXing1.01 by Jian-Feng Ye in August 2008.
- Workstation set up in 2007 (4-core 2.21GHz AMD875 CPU, 4G RAM, 512KB L1 cache, 2048 KB L2 cache, 90 nm).
- The workstation is installed Microsoft Windows Server 2003 Standard X64 Edition Service Pack2.

APPENDIX B: NEUCHESS SEARCH CONFIGURATION

- 16-depth limitation.
- Single-thread mode.
- No time-limitation.