Conflict Resolution of Chinese Chess Endgame Knowledge Base

Bo-Nian Chen[1,⋆], Pangfang Liu[2], Shun-Chin Hsu[3], and Tsan-sheng Hsu[4,⋆,⋆⋆]

[1] Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. r92025@csie.ntu.edu.tw
[2] Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. pangfeng@csie.ntu.edu.tw
[3] Department of Information Management, Chang Jung Christian University, Tainan, Taiwan. schsu@mail.cjcu.edu.tw
[4] Institute of Information Science, Academia Sinica, Taipei, Taiwan. tshsu@iis.sinica.edu.tw

Abstract. Endgame heuristics are often incorperated as part of the evaluation function used in Chinese chess programs. In our program, Contemplation, we have proposed an automatic strategy to construct a large set of endgame heuristics. In this paper, we propose a conflict resolution strategy to eliminate the conflicts among the constructed heuristic database, which is called *endgame knowledge base*. In our experiment, the correctness of the obtained constructed endgame knowledge base is high enough for practical usage.

Keywords: Computer Chinese chess, Endgame Knowledge Base, Conflict Reduction

1 Introduction

A game of Chinese chess, like chess, can be divided into three phases: 1) opening game, 2) middle game, and 3) endgame. Opening game is the first phase of a game that almost all pieces are on the board. After about 20 plies, both two players have moved their pieces to their important places and the game becomes middle game. After exchanging some pieces, the game goes into endgame phase.

The most popular technique used to solve the opening game problem is constructing opening databases that stores all possible choices in previous games [1]. In the middle game, people often use a nega-scout algorithm with a good evaluation function and a nice move ordering scheme to obtain a good solution [2]. There are also strategies in artificial intelligence that automatically generate middle game evaluation functions [3]. In the endgame phase, the performance of

⋆ Supported in part by National Science Council (Taiwan) Grants 97-2221-E-001-011-MY3.
⋆⋆ Corresponding author.

today's Chinese chess program is still not satisfiable compared to human experts. People often solve relatively small endgames by using retrograde algorithms [4].

Endgame databases constructed by retrograde analysis algorithms are perfect in the sense that the game-theoretical values of all positions matched in the database are available. However, there are two drawbacks in endgame databases: 1) it needs too many memory, and 2) practical endgames contain too many pieces for current retrograde algorithms to handle. Hence, the search algorithm still needs heuristic information about endgames. The problem then relies in how to generate effective heuristics in endgame. Although heuristics are not perfect, they can be applied to practical endgames and are useful in tournaments.

Our intuition is as follows. Each endgame is assigned with a heuristic, namely a level of advantage, disadvantage according to the material combination of the two sides. This assigned value reflects the heuristic a Chinese chess master usually understand whether the endgame is advantageous or not without considering positions of the pieces. The heuristics of two endgames may be obtained using different methods, such as from text books or human annotation. These heuristics, though have a high level of accuracy, may still contain errors. We observe that if the material combinations of two endgames differ by a small number of pieces, then the heuristic values of the two endgames are not independent. For example, knowing an endgame usually is advantage to the Red side, then after the endgame heuristic value of adding a RED piece cannot be worse than the original one. Assume an endgame A is related to many other endgames and further assume a large portion of the heuristics obtained so far is corrected, then A must be consistent with most of the related endgames. If this is not the case, then there is a high chance that the heuristic value of A is incorrect. Using this high level idea, we build an expert system to self-correct a large set of annotated heuristics.

In our previous work, we have designed a method to automatically generate a large number of heuristics of material combinations [5]. However, there are some conflicts in our endgame knowledge base. A small number of conflicts are enough to harm search algorithms. Hence, we propose an important conflict reduction algorithm that increases the consistency in endgame knowledge base.

In our paper, we will discuss about the following important issue relative to endgame problems: piece exchange when transform from middle game to endgame. In Chinese chess, using solely material values computed from summing all piece values to choose good exchanges may be incorrect in some cases. Our solution is to use a lot of heuristics of material combinations, called endgame knowledge base, to guide our program when piece exchange is needed.

2 Theoretical Foundations

In this section, we describe theoretical concepts about lattice used by our method and discuss the problem in our automatic generated endgame knowledge base.

2.1 Using Lattice to Represent the Material Structure

There are seven types of pieces in Chinese chess: king (K), guard (G), minister (M), rook (R), knight (N), cannon (C), and pawn (P). A *material combination* is defined as the set of pieces in a position, e.g., KCMKRP is a material combination that the red player has the king, a cannon and a minister; the black player has the king, a rook and a pawn. A material combination is attached with a score that describes its advantage without position information. Each material combination has exactly an *mirrored material combination* such that the red pieces and the black pieces are swapped. The *material structure* consists of a set of material combinations. In our discussion, the material structure represents all of the material combinations in our endgame knowledge base. *Invariable nodes* are those modified or verified by our human expert and cannot be changed by our conflict reduction algorithm.

The material combination structure can be viewed as a lattice. A lattice is a partially ordered set(poset) that all non-empty subsets have a join and a meet in mathematical order theory. A join is the least upper bound of an element or a subset; a meet is the greatest lower bound of an element or a subset. All material combinations in Chinese chess follow the *piece additive rule* that for a material combination, adding pieces to a player cannot make him be disadvantageous if we only consider material combination, not specific positions. The piece additive rule also claims that removing a piece from a material combination cannot be better than the original material combination. By applying piece additive rule, the material structure can be transformed into a lattice which is a directed graph. The node in the lattice represents a material combination. The edge connects two material combinations that differ only one piece. We define $x \rightarrow y$ as two adjacent nodes and the directed edge represents that the red player is at least as advantageous in x as in y. In lattice, $x \rightarrow y$ means that x and y are comparable and the meet of x is y.

2.2 Construction Strategy of Material Structure

In the lattice, the least element is KK. We always expand the material structure by adding either a red piece or a black piece. The number of possible piece types on one side is $3^5 \times 6 = 1,458$. Totally, there are $1,458^2 = 2,125,764$ possible material combinations in Chinese chess. An example of material structure is shown in Fig. 1. A lattice can be divided into several levels. Each level contains material combinations of the same piece number. In Chinese chess, there are at most 32 pieces and at least 2 pieces, two kings, on a position. Hence, there are totally 31 possible levels in our lattice.

It is not always correct to give a score to the material combinations on each side. For example, KPPKGG and KPPKMM are generally red-win endgames when pawns are not yet move to the palace of the opposite side in the starting position of these endgame, but KPPKGM is generally a draw endgame. Thus, we may conclude that KGM is better than KGG and KMM for defense. However, KHPKGM and KHPKGG are generally red-win endgames but KHPKMM is a

draw endgame when the pawn stands in the last line of the palace of the opposite side. The conclusion that KMM is better than KGG and KGM is inconsistent with the last case.

The score value of a node is in the range of $[0-9]$. The values 0 (win), 1 (most win), 2 (advantage), 3 (slight advantage) represents the score that the red side is in advantage, 4 represents that any one player has a chance to win, 5 means almost draw, 6 is oppsite to 3, 7 is opposite to 2, 8 is opposite to 1, and 9 is opposite to 0.

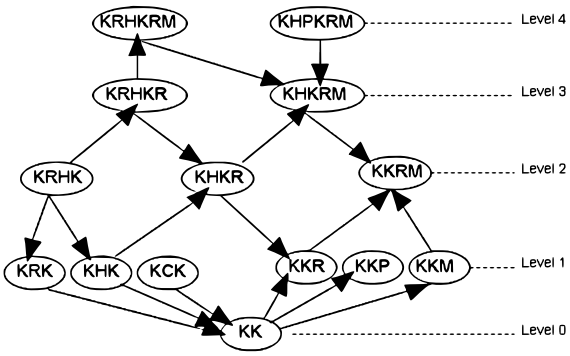

Fig. 1. An example of lattice structure for endgame.

In our automatically endgame knowledge base construction, we first created a basic endgame knowledge base manually. A probablistic method is used to evaluate the score value of generated material combinations. By using a systematic generating algorithm, we have constructed a large endgame knowledge base. The endgame knowledge base is used by middle game search algorithm to find paths to enter advantageous endgames by exchanging pieces.

Inconsistency is defined as the case that score values of adjacent nodes violate the piece additive rule. The two nodes are called *inconsistent node*, the corresponding edge is called *inconsistent edge* and the corresponding connected node is called the *Inconsistent neighbor*. *Inconsistent percentage* of a node indicates the number of inconsistent neighbors divided by the number of its neighbors.

Although most of the score values in our endgame knowledge base are accurate, only a little conflict is sufficient to let the search algorithm to select wrong moves. To resolve the problem, we propose a conflict reduction algorithm that reduces the number of inconsistent nodes. When our algorithm cannot progress further, we ask the help of a Chinese chess expert to do a small amount of modification. Then we rerun our self-correcting algorithm. After some iterations of modification, we can obtain a zero-conflict endgame knowledge base. To further

increase correctness, we ask our Chinese chess expert to verify a randomly selected sample to evaluate the percentage of errors and do further modifications.

3 Basic Conflict Reduction Algorithms

In our automatically-generated endgame knowledge base [5], there are some inconsistent material combinations that causes the search algorithm to exchange piece incorrectly. For example, consider there are two adjacent material combinations, as shown in Fig. 2. To illustrate the concept of conflicts, we can find that the central node, KCCKHCPGG, has a score value 3, and the leftmost node, KCCKHPGG, has a score value 6. There is a conflict because the black side is more advantageous when a black cannon is taken. The first material combination has four inconsistent edges, inconsistent percentage is 80% and the second material combination has only one inconsistent edge, inconsistent percentage is 9.09%. In this example, the first material combination is more likely to be incorrect and should be modified first.

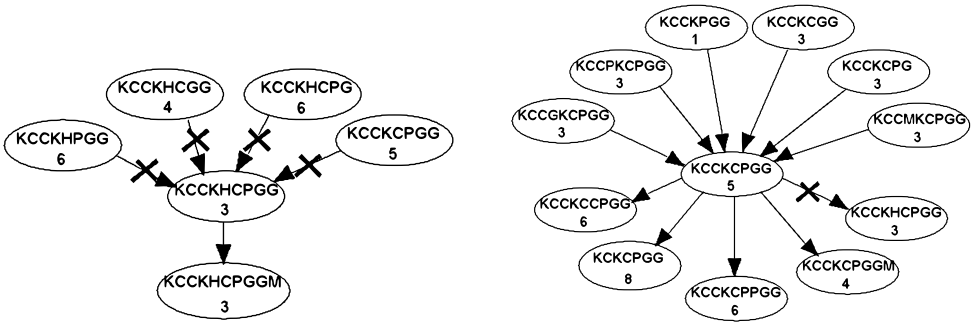

Fig. 2. Examples of two adjacent inconsistent nodes. The number in each node is its score value. The edge with a cross means an inconsistent edge.

We need a standard endgame knowledge base that is considered as "correct" for the conflict reduction algorithm. The algorithm discovers and modifies the nodes in automatically-generated endgame knowledge base that are inconsistent with some nodes in the standard endgame knowledge base.

3.1 Conflict Computation

If there is a conflict in a lattice, there must be some nodes having inconsistent neighbors. Conflict computation procedure computes the number of inconsistent neighbors of each nodes. When finishing the computation, information that we

actually want to know is the inconsitent number and information of the inconsistency level. The level of inconsistency is relative to the inconsistent percentage of the node. We define 10 levels of inconsistency, from level 0, level 1, ... , to level 9. Level 0 represents the inconsistent percentage in $(0\% - 10\%]$, level 1 represents $(10\% - 20\%]$, etc. Information of the number of nodes is in each inconsistency level. It simplifies the process of reducing conflicts.

Our idea is a greedy method that always modifies a node of the highest inconsistency level. We use *inconsistent edge checking* to find conflicts between nodes. Inconsistent edge checking algorithm has two targets: 1) two neighbor nodes, and 2) two mirrored material combinations. We implement the piece additive rule which is defined in Section 2.1 for the first target. The second target, a mirrored material combination pair in our lattice is virtually considered as the same material combination. Inconsistent edge checking algorithm can ensure the consistency of mirrored material combinations.

The algorithm of computing conflicts simply uses inconsistent edge checking to summary the inconsistency neighbors for each node. Computing conflicts of the whole lattice can be done in $O(MN)$ time, where M is the maximum number of the neighbors of a node, and N is the number of nodes.

3.2 Conflict Reduction Algorithm

The conflict reduction algorithm, shown in Algorithm 3.2 finds inconsistent nodes in our lattice and modifies the score values of some nodes to reduce the number of inconsistent nodes. It takes four steps to finish the work: 1) conflict computation, 2) candidate selection, 3) score value selection, and 4) modification. Our algorithm repeats the four steps until no more candidate can be selected. The first step, conflict computation, is described in Section 3.1. The next three steps are described as follows.

```
procedure ConflictReduction()
    do loop
        err_num = ConflictComputation();
        ResetUpdated(); // set the updated flag as not updated
        if(err_num = 0 or no any modification)
             break;
        do loop
            node = CandidateSelection();
            if(no valid node)
                break;
            score = ScoreValueSelection(node);
            Modify(node, score); // change the score value of a node
        end loop
    end loop
end procedure
```

Algorithm 3.2. Conflict reduction algorithm.

The first step, candidate selection, chooses the node with the highest conflict rank to be modified. Score value selection then tries all possible values and

computes the inconsistent percentage after modification. The value with the minimum inconsistent percentage is selected. If two scores of a node have the same minimum inconsistent percentage, we ask the help of our human expert. The detail discussion is in Section 4.3. During modification, we need to simultaneously update the mirrored material combinations when modifying a node to ensure their consistency.

In our algorithm, updated flag is used to avoid repeat selection of the same candidate in one iteration. Invariable flag identifies whether a node is modified by our 4-Dan expert and should be skipped by candidate selection procedure.

4 Refinements

In this section, we introduce the diffusing algorithm and other enhancing techniques. We also discuss the verification issue of a consistent lattice.

4.1 Diffusing Algorithm

Diffusing algorithm, as shown in Algorithm 4.1, is a recursive procedure that searches the neighbor nodes for the nodes with only one *consistent possible value*, which is the value not inconsistent with any invariable nodes. These nodes are trivial and should be modified first.

```
procedure Diffusing(n)
      // variable n represents the node to be diffused
      // variable nn represents a neighbor node of variable n
      if(n is updated before)
          return;
      for each nn of n do
          // compute possible values of nn that do not violate n
          // variable count records the number of possible values
          // variable score records which scores are possible values
          (count, score) = ComputeRelativePossibleValue(nn, n);
          // update if only one possible value
          if(count = 1)
              Modify(nn, score);
              Diffusing(nn);
          end if
      end for
end procedure
```

Algorithm 4.1. Diffusing algorithm.

By performing diffusing algorithm, all nodes with only one possible value are modified first and our algorithm reduces more conflicts.

4.2 Ranking and Scoring Strategies

We rank nodes in the lattice to indicate its degree of errors, called *conflict rank*. Our algorithm picks one with the highest rank to update its value. In the basic conflict reduction algorithm, we use inconsistent percentage as its conflict rank. Here we define a better conflict rank:

$$V = N_i \times N_n$$

In the above formula, V represents conflict rank, N_i represents the number of inconsistent neighbors, and N_n represents the number of neighbors. Instead of dividing N_n, the new conflict rank multiplying N_n to emphasize the importance of the number of neighbors.

In the step of score value selection, we use *corrected score* to measure the whole level of inconsistency. Our conflict reduction algorithm always selects the score value that minimizes the corrected score. In basic method, we use the number of inconsistent nodes as the corrected score. Here we have developed a new corrected score that favors small inconsistency levels as follows:

$$V_c = \sum_{i=0}^{9} 2^i \times I_i$$

In the formula, the value V_c means corrected score, I_i represents the information of inconsistency level i (see Section 3.1).

By using the new corrected scores, nodes with large inconsistent percentages are usually reduced to smaller percentages. The new corrected score improves the ability of identify better score values and thus decreases the probability of fall into local minimum.

4.3 Final Verification

Now consider that we have obtained a consistent endgame knowledge base. Unfortunately, there may be two types of errors in the lattice: 1) an isolated subset of the lattice are all incorrect; and 2) there are errors in some nodes that do not influence the consistency.

Checking by random sampling verification is a way to obtain the approximation of the correctness of the lattice. We selects a small number of nodes with a percentage p in the lattice randomly such that the distance of any two selected nodes is at least k. The selected material combinations are verified by the human expert. If n error nodes is reported, the approximated value of whole error nodes is n/p. The modified data can also be used to reduce the errors of the whole knowledge base.

5 Experimental Results

In this section, we use the practical endgame knowledge base in Contemplation as our test data. Then we show the reduction ability of the conflict reduction

algorithms, the correctness analysis by random sampling verification, and the comparison of the consistent endgame knowledge base with its original version.

5.1 Experiment Design

Our test data is the endgame knowledge base used by our program, Contemplation. There are three manually constructed endgame knowledge bases: END65, END60, and END50. The number of nodes in END65, END60, and END50 are 17,038, 422, and 1,499, respectively. The data to be tested is our automatic generated endgame knowledge base: END64, END59, END49 using methods in [5]. A extended endgame knowledge base is named as the number of the original knowledge base decreased by 1, i.e., END64, which is generated by extending END65, contains the neighbors of the nodes in END65. The number of nodes in extended knowledge bases END64, END59, and END49 are 47,621, 2,722, and 3,938, respectively. Our practical endgame knowledge base, ENDALL, combines six endgame knowledge bases, containing 69,595 nodes.

5.2 Results and Discussions

In the first experiment, we try our methods on endgame knowledge bases END64, END59, END49, and ENDALL. There are two methods to be compared: 1) basic conflict reduction algorithm, called BA, and 2) the algorithm with all refinements, called RA. The results are shown in Table 1. The number of iterations means the iterations needed for convergence. An iteration indicates handling all inconsistent nodes once in the lattice. To test the convergency, we need an extra iteration to ensure that no modifications is done in an iteration.

Table 1. Comparison of the reduction ability of the basic algorithm and the refined algorithm. The error after BA and the error after RA columns show the number of inconsistent nodes after performing the basic algorithm and the refined algorithm, respectively.

	DB size	org error	error after BA	iterations	error after RA	iterations
END64	47621	14616	9786	6	970	3
END59	2722	1786	1330	3	166	2
END49	3938	1362	438	6	45	3
ENDALL	69595	16488	11108	6	585	4

By using all refinement techniques, we obtain the knowledge bases with much less conflicts in less number of iterations. This reduces the work of the human expert to verify and modify the endgame knowledge base.

In the second experiment, we show the correctness of our endgame knowledge base after we performed random sampling verification. We have done three random sampling experiments. In each experiment, we use an algorithm described in Section 4.3 to generate different 695 nodes with parameters $k = 4, p = 1\%$. After the sampled nodes have been verified and modified, we performed our conflict reduction algorithm with all refinement techniques on the ENDALL knowledge base. We define the *error distance* as the difference between the score value of the consistent endgame knowledge base and the score value verified by our human expert. Because the score values 4 and 5 represent very similar class of advantage, the error distance between them is set to zero. In addition, the error distance between any score value and 4 is consider equal to the error distance between that score value and 5. We recorded the error distances of the modified data and checked whether it is inverted. An inverted result is a result that is wrong in the side who has advantage. For example, a node that the red side is in advantage and is marked as black win is an inverted result. The result of this experiment is shown in Table 2.

Table 2. The statistical analysis of the correctness in the zero-conflict endgame knowledge base ENDALL. IR means inverted results. D represents error distance.

	ErrNum	$D \geq 4$	$D = 3$	$D = 2$	$D = 1$	$D = 0$	IR
Sample1	92	0	2	15	75	0	1
Sample2	127	0	2	9	116	0	1
Sample3	99	0	2	16	80	1	0
Average	106.0	0.0	2.0	13.33	90.33	0.33	0.66
n/P	10600	0	200	1333	9033	33	66
%	15.23	0	0.28	1.91	12.97	0.00	0.00
Confidence	1533	0	200	1333	0	0	66
%	2.20	0	0.28	1.91	0.00	0.00	0.00

The probability of having error distance more than one is 2.20%. Note that a node with a score value 4 or 5 cannot be inverted. Hence, inverted results only happen when score values are less than 4 or more than 5. In other words, inverted results happen when the distance is more than or equal to one. They are also counted in the column of the distance is more than or equal to one. Although the absolute correctness is 85.77%, which is acceptable, the correctness with confidence is 97.70% ignoring one level difference. Evaluation of a material combination as "win" or "win in most cases" is a subjective choice. Even human master or grandmaster may have subjective judgement in many practical positions. Hence, we assume a difference of 1 level is tolerable.

In Table 3, we show the statistical comparison of the consistent endgame knowledge bases after verification and their original versions. Since different

original endgame knowledge bases may contain identical material combinations, we have filtered them when merging endgame knowledge bases. The score values of identical material combinations is set as the material combinations that first appear in the merging process. Hence, some material combinations may be counted more than once. They may even contain different original score values. For example, in END65, there are two errors with zero error distance, which are KHHMKHHG and KHHGKHHM. There are also two errors with zero error distance in END59, which are KCCGKCC and KCCKCCG. During merging operation, they are set as correct score values by chance, and the number of errors with zero error distance in ENDALL becomes two.

Totally, we have modified 24,486 material combinations in the final consistent endgame knowledge base. In the original endgame knowledge base, there are 1,652 errors of distance more than or equal to four, 2,392 errors at distance three, 2,286 errors at distance two, 18,154 errors at distance one, two errors between score value 4 and 5, and 1,064 inverted results.

About 9.10% of the original ENDALL knowledge base contain errors of $D \geq 2$; about 26.09% of them contain errors of $D = 1$; there are no errors of $D = 0$; about 1.53% of them contain inverted results.

Table 3. The statistical comparison of the original endgame knowledge bases and the final version of consistent endgame knowledge bases. IR means inverted results. D represents error distance.

	DB size	ErrNum	$D \geq 4$	$D = 3$	$D = 2$	$D = 1$	$D = 0$	IR
END65	17038	1100	6	24	174	894	2	80
END60	422	48	2	6	8	32	0	8
END50	1499	222	4	16	34	168	0	10
END64	47621	20908	1056	2042	1952	15858	0	708
END59	2722	1734	594	390	142	606	2	290
END49	3938	1982	60	32	50	1840	0	50
ENDALL	69595	24486	1652	2392	2286	18154	2	1064

6 Conclusions and Future Work

A complete mastering Chinese chess endgame problem is a hard problem even today. We construct endgame knowledge base for search algorithm to identify which kinds of endgames are beneficial. In this paper, we propose a conflict reduction algorithm to resolve the conflicts in our automatically-generated endgame knowledge base. The strategy is effective when handling large knowledge base with a relatively small percentage of conflicts. The resulting endgame knowledge base we have obtained is checked by random sampling verification and received

high accuracy. We use this modified knowledge base in our program, Contemplation, and find it to steadily improve it strength against its previous version. Its correctness is high enough for practical usage.

In the future, we will enhance our conflict reduction algorithm to be more sensible of advantage. For example, KCCKCPGG and KCCKCGG differs only by one pawn and they have score values 5 and 8, respectively. KCCKCRGG and KCCKCGG differ by a rook and they also have score values 5 and 8. The degrees of conflict-free expectation of the two above cases are different. In practical usage, if the two cases are both inconsistent, in the latter case has a more severe degree of conflict than the formal case. Another example is KRPKGGMM and KRPKGGM who are assigned 0 and 9 respectivly in compared with the same material combinations who are assigned 0 and 1, respectively. Although two cases include a conflict, the first score values is more severe because the difference of the score values is very obvious.

Combining the two representative examples, we can define a lattice with weighted edges. The weight is defined as follows:

$$w = D_m(m_1, m_2) \times D_s(Score(m_1), Score(m_2))$$

The variable w represents the weight which indicates the degree of conflicting. Function D_m computes the difference between the two material combinations m_1, and m_2; function D_s computes the difference between the two score values of m_1, and m_2. The weight value follows the order: $rook > cannon = knight > pawn > guard = minister$. The weight of guards and ministers should be adjusted dynamically: when the player has cannons, the weight values of guards and ministers should be bigger than the ones without. Function score retrieves the score value of a material combination. When a conflict occurs, the node with a larger weight value needs to be taken care of first.

References

1. J. C. Chen, and S. C. Hsu. Construction of online query system of opening database in computer Chinese chess. The 11th Conference on Artificial Intelligence and Applications. (2001)
2. S. J. Yen, J. C. Chen, T. N. Yang and S. C. Hsu. Computer Chinese Chess, *ICGA Journal*, Vol. 27, No.1, March 2004, pp. 3-18, ISSN 1389-6911. (2004)
3. B. N. Chen, P. F. Liu, S. C. Hsu, and T. S. Hsu. Abstracting knowledge from annotated Chinese chess game records. Computers and Games 2006, LNCS 4630, pp. 100-111. (2006)
4. P. S. Wu, P. Y. Liu, and T. S. Hsu. An external-memory retrograde analysis algorithm. In H. Jaap van den Herik, Y. Bjornsson, and N. S. Netanyahu, editors, *Lecture Notes in Computer Science 3846: Proceedings of the 4th International Conference on Computers and Games* , pages 145-160. (2006)
5. B. N. Chen, P. F. Liu, S. C. Hsu, and T. S. Hsu. Knowledge Inferencing on Chinese Chess Endgames. Computers and Games 2008, LNCS 5131, pp. 180-191. (2008)