Knowledge-Based Systems 34 (2012) 34–42

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Aggregating consistent endgame knowledge in Chinese Chess

Bo-Nian Chen [a], Pang-Feng Liu [b], Shun-Chin Hsu [c], Tsan-sheng Hsu [a,*]

[a] Institute of Information Science, Academia Sinica, Nankang, Taipei 115, Taiwan
[b] Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan
[c] Department of Information Management, Chang Jung Christian University, Tainan 711, Taiwan

ARTICLE INFO

Article history:
Available online 6 December 2011

Keywords:
Computer Chinese Chess
Endgame knowledge-based system
Conflict resolution
Lattice graph
Efficient algorithm

ABSTRACT

We often incorporate endgame heuristics as part of the evaluation function for Chinese Chess programs. In order to aggregate endgame knowledge effectively, we propose a Chinese Chess endgame knowledge-based system to construct a large set of consistent endgame heuristics, called *endgame knowledge base*, which is used in our program, Contemplation. The knowledge-based system consists of the acquisition module, the inference module, the inquiry module and the verification module. This system implements our graph model that has the functionality of maintaining consistency and improving its correctness. The experimental results on self-play test show that the playing strength of Contemplation has a distinct enhancement with this knowledge base.

1. Introduction

Shannon [5] proposed an architecture to combine knowledge and the type A search for a Chess program. An improvement of minimax game tree search algorithm, $\alpha - \beta$ pruning, is proved that for perfect ordering game trees, the time complexity of the search is $O(d^{1/2})$, where d is the depth of the game tree, by Knuth and Moore in 1975 [6]. After that, Reinefeld discovered the NegaScout searching method that perform better than $\alpha - \beta$ pruning algorithm in most cases, but sometimes takes more time to do the re-search [2].

The state-space complexity and the game-tree complexity of Chess is 10^{46} and 10^{123}, respectively. Comparing to the latest solved game, Checkers, whose complexity is 10^{21} and 10^{31}, it is still believed far from being solved in the near future [7]. Thus, the researchers divide a game into the opening game, the middle game, and the endgame. For the achievement of the endgame, van den Herik and Herschberg suggested the concept of the *retrograde analysis* to build endgame knowledge bases in 1985. Thompson [8] uses retrograde analysis to construct endgame knowledge bases and provide perfect play in endgame positions. Ten years later, his endgame knowledge bases can contain 6-piece endgames [9].

Each game has its significant domain knowledge. Although Chess and Chinese Chess are both the games of checkmates, the latter has several distinct features: (1) various tactics of cannon, (2) a palace that limits the mobility of kings, (3) pawns which cannot be promoted.

In the aspect of research, there are two critical problems in Chinese Chess. The first is the specialized rules for repetition of positions. The Asian rule is an international rule that is relatively consistent than other rules for repetition of positions. This special rule also increases the complexity of endgame knowledge bases due to the graph history interaction (GHI) problem [1].

The second problem in Chinese Chess is the *dynamic* values of *material combinations*, i.e., a set of remaining pieces, in the endgame. Since there is no precise definition of endgame, we define the endgame as the position set that both players have no more than five units of strong pieces, i.e., rooks, knights, and cannons. A rook is considered as two units of strong pieces. The problem makes it difficult to evaluate a correct score in the endgame and thus causes some uncertainty for transforming positions into endgames by piece exchanging. The problem will be described in Section 2.

Retrograde analysis is also an important problem solving method for computer Chinese Chess endgames [13]. Such knowledge bases provide perfect play in tournaments, but it is not easy to go into these pre-created endgames during a real tournament due to the fact that Chinese Chess endgames contain much more pieces than other similar games such as Chess. In order to increase the usage rate of knowledge bases in practice, we proposed a systematic method to construct a large consistent knowledge base that contains heuristics about material combinations in 2009 [4]. The retrograde analysis method is *the position-wise configuration* whose knowledge base consists of positions. The value of each position is the game-theoretical value: win, draw or loss. Our work is to collect material combinations and to assign their values to be used in an evaluation function. The value of each material combination reflects the heuristic a Chinese Chess master usually

* Corresponding author.
E-mail address: tshsu@iis.sinica.edu.tw (T.-s. Hsu).

0950-7051/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2011.11.020

understand whether the endgame is advantageous or not without considering possible extreme positions of the pieces.

Knowledge aggregation is an important problem in knowledge-based systems [11]. It is also critical in Chinese Chess. In this paper, we propose a knowledge-based system that can acquire, inference, inquire, verify on the knowledge of material combinations. Knowledge acquisition automatically generates material combinations and perform a score prediction method to obtain their score values. The inferencing technique resolves conflicts in the knowledge base. Inquiry is a mechanism for the system to ask questions when there is inconsistency to improve the quality of knowledge. Verification of knowledge facilitates Chinese Chess human experts to verify and improve the correctness of knowledge base.

The main contribution of the paper is that we design a graph model to represent our data and thus provide the functionality to ensure the consistency and to improve the correctness of the knowledge base. The consistency versus correctness problem will be discussed in Section 2. According to our experiments, the graph model is useful in building a reliable knowledge base for Chinese Chess.

This paper is organized as follows. Section 2 defines our problem and illustrates our concept in graph theory. Section 3 describes the architecture of the endgame knowledge-based system. Section 4 provides our methods for knowledge acquisition and implements the inference component of the system. Section 5 discusses the inquiry and verification process. Section 6 shows the overall experiment of the knowledge-based system by a self play test. In Section 7, we make concluding remarks.

2. Theoretical foundations

In this section, we define the notations of pieces, material combinations, and advantage scores of material combinations. We then discuss the problem of dynamic material combinations, followed by our graph model for maintaining consistency. The main meta-knowledge used includes the piece additive rule and the lattice model with its properties. We will also state some important properties about the graph model.

2.1. Notations

There are seven types of pieces in Chinese Chess: king (K), guard (G), minister (M), rook (R), knight (N), cannon (C) and pawn (P). The *static material value* of each type of pieces is roughly defined as follows: king (∞), guard (2), minister (2), rook (10), knight (5), cannon (5), pawn (1). Rooks, knights and cannons are called *strong pieces* and pawns are *weak pieces*. Strong and weak pieces are called *attacking pieces* because they are able to cross the river which divides the board into two territories. Guards and ministers are *defending pieces*. Their mission is to protect the king.

A *material combination* is defined as the set of pieces in a position. We use a string of pieces to denote a material combination. The string starts from the red king, followed by the other red pieces, followed by the black king, and followed by other black pieces. For example, KCMKRP is a material combination that the red side has a king, a cannon and a minister; the black side has a king, a rook and a pawn. Without loss of generality, we consider the red side as *the attacking player* and the black side as *the defending player*.

2.2. Elements of material combinations

A material combination is attached with an endgame source identifier, an advantage score, an invariable flag and a modified flag. They are described as follows:

Endgame source identifier. The source where the piece of knowledge comes from, such as text books, human annotations or automatic generation by our system.

Advantage score. The advantage score of a material combination is in the range of [0–9]. The values 0 (sure win), 1 (mostly win), 2 (advantageous), 3 (slightly advantageous) represent the scores that the red side is in advantage; 4 represents that any one player has a chance to win; 5 means draw, which says no one can win; 6 is the opposite of 3, 7 is the opposite of 2, 8 is the opposite of 1, and 9 is the opposite of 0.

Invariable flag. It is used in the knowledge inferencing component to avoid the algorithm to modify the scores of the material combinations assigned manually by Chinese Chess human experts.

Modified flag. This attribute is also used by the knowledge inferencing component to record entries that are modified by the algorithm in an iteration.

Graph information. We will describe our graph model in Section 2.5. The information such as the neighbors of a node and the conflict neighbors of a node are maintained for the inference module.

Each material combination has exactly one *mirrored material combination* such that the red pieces and the black pieces are swapped. For *symmetric material combinations* that the red side and the black side contain same pieces, their mirrored material combinations reflect to themselves. The scores of such material combinations of equal power can only be either 4 or 5.

2.3. The problem of dynamic material combinations

The distinction of attacking pieces and defending pieces makes it hard to evaluate the correct score of a position. For example, KCPGMMKGGMM is an endgame that the red side can force a win, but in KNPGMMKGGMM endgame, the game generally ends in a draw. The two endgames, shown in Fig. 1, only differ by one strong piece of the same material value, but their conclusions are different. Not only attacking pieces are critical for endgames, defending pieces are also critical in many endgames. For example, in the endgames KPPKGG and KPPKMM, the red side has many chances to win, but KPPKGM is generally a draw endgame. Thus, we may conclude that KGM has better defensive power than KGG and KMM. However, KNPKGM and KNPKGG are generally red-win endgames but in the endgame KNPKMM, the black side has more chance to achieve a draw. A conclusion that KMM is better than KGG and KGM is inconsistent with that of the previous example. Therefore, it is hard to find a consistent rule for all of the peculiar results in endgames.

The problem of having dynamic values for material combinations makes it difficult to exchange pieces in Chinese Chess. For example, the red side can always find a winning strategy in KRKNGG endgame, but there are some drawing cases in KRKNMM endgame. According to this, when we need to exchange pieces and reduce the current position into certain endgames, choosing KRKNGG ensures the red side to win the game, but choosing

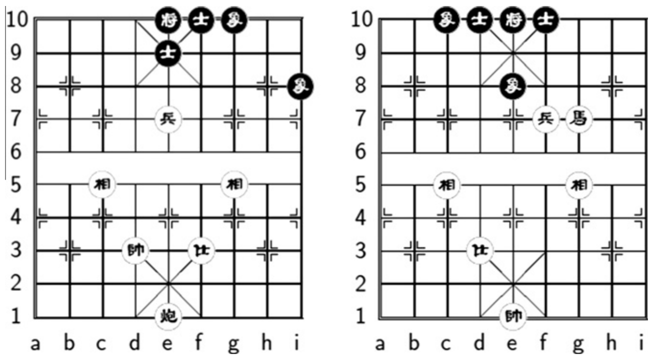

Fig. 1. The material combinations KCPGMMKGGMM v.s. KNPGMMKGGMM.

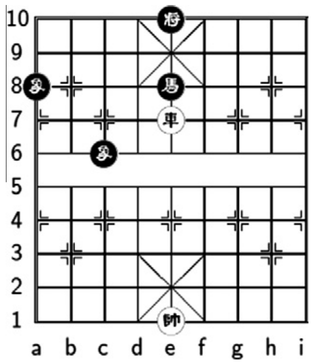

Fig. 2. The classical drawing configuration in KRKNMM.

KRKNMM has some risk of getting a draw, as shown in Fig. 2. Such knowledge about material combinations is important in the middle game and the endgame, but it is hard to acquire a consistent set of knowledge since there are more than one million of them. As a result, we propose a method based on graph theory to obtain values of commonly used endgame material combinations.

2.4. The main meta-knowledge

All material combinations in Chinese Chess follow the *piece additive rule* that for a material combination, adding pieces to a player cannot make it to be less advantageous than the original one if we only consider material combinations. Conversely, removing a piece from a player in a material combination cannot make this player to be be better than that was in the original one.

This property introduces the graph representation for material combinations, called a lattice. The lattice model described in the next section maintains consistency according to the piece additive meta-knowledge.

2.5. The lattice model

The idea we adopted is to apply meta-knowledge as a lattice model [10]. 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. By applying the piece additive rule, material combinations 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 the lattice structure as Γ. For two adjacent nodes x and y, $x \rightarrow y$ represents that the red side is at least as advantageous in x as in y. In the lattice, $x \rightarrow y$ implies that x and y are comparable and the meet of x is y. Each node in the lattice can be viewed as a single entry of knowledge. An edge between two nodes represents the relation of the corresponding two entries. Fig. 3 is an example of the lattice model.

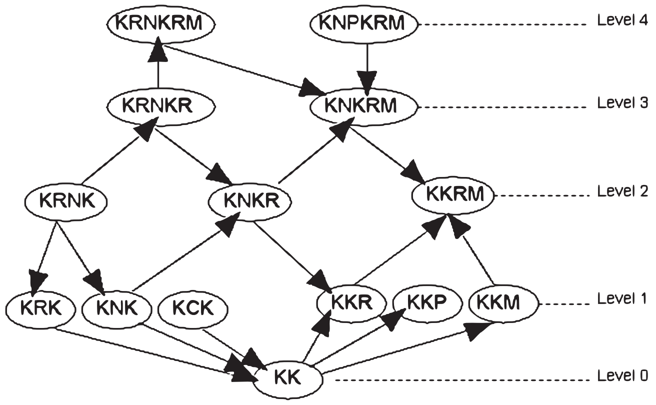

Fig. 3. An example of lattice model for endgame.

2.6. Basic properties of the lattice model

Structure and node number. In the lattice, the least element is KK. We always expand the lattice by adding a node which have one more red or black piece of an existing node. The number of possible piece combinations for one side is $3^5 \times 6 = 1458$. Totally, there are $1458^2 = 2{,}125{,}764$ possible material combinations in Chinese Chess. A lattice can be divided into levels. Each level contains material combinations of the same total number of pieces. In Chinese Chess, there are at most 32 pieces, and at least 2 pieces, that is the two kings, on a position. Hence, there are totally 31 levels in our lattice.

Piece additive rule. The material combination component in the node x is $x.m$. The material combination is composed of red pieces and black pieces, which are denoted as $x.m.r$ and $x.m.b$, respectively. Its advantage score is denoted as $x.v$.

Score values 4 and 5 are conceptually equal because no player has an advantage. Thus, we perform a transformation on a node to map the advantage score into the *advantage class*:

$$AC(x.v) = \begin{cases} x.v, & \text{if } x.v \leqslant 4; \\ x.v - 1, & \text{if } x.v > 4. \end{cases} \tag{1}$$

Now we formalize the piece additive rule. If a directed edge is from x to y implies:

$$x.m.r \supseteq y.m.r \quad \text{and } x.m.b \subseteq y.m.b. \tag{2}$$

Thus, we have:

$$AC(x.v) \leqslant AC(y.v) \quad \text{if } x \rightarrow y. \tag{3}$$

This rule holds in all cases in material combinations. This rule will be used to verify the consistency of our knowledge base.

Conflict. The score values of adjacent nodes violate the piece additive rule, that is,

$$AC(x.v) > AC(y.v) \quad \text{if } x \rightarrow y. \tag{4}$$

We say that rule x conflicts with rule y.

A *safe node* is a node x that for each of its undirected neighbor y, $AC(x.v) > AC(y.v)$ if $x \rightarrow y$ and for all $AC(x.v) < AC(y.v)$ if $y \rightarrow x$. Conversely, a *conflict node* is a node that is not safe. The corresponding edge is called *conflict* and the corresponding connected node is called the *conflict neighbor*.

Consistent lattice. A lattice model Γ with no conflict nodes is said to be a *consistent lattice*. If all nodes are safe, by transitivity, the lattice must be consistent.

Conflict node number. We use $CN(\Gamma)$ to denote the total number of conflict nodes.

2.7. Advanced properties of the lattice model

Monotonic property. For a consistent lattice, the score values of the nodes in any directed path between an arbitrary node pair (x,y) follow a monotonic non-decreasing property from x to y.

Conflict propagation. Assume there is an error node v in the lattice. The node has k neighbors, denoted as $N(v)$. We define the neighbors of v that conflicts with v as $C(v)$. The ones that still keep consistent with v are $\overline{C}(v)$. Thus, we have

$$N(v) = C(v) \cup \overline{C}(v), \tag{5}$$

and their corresponding numbers are

$$|N(v)| = |C(v)| + |\overline{C}(v)|. \tag{6}$$

The larger the value $|C(v)|$ of each error node v is, the more error we can capture by the *conflict resolution algorithm*, which is described in Section 4.

The average $|C(v)|$ for each error node v in the lattice is called the *conflict propagation rate*. It is a theoretical measurement of the effectiveness of the lattice model. Note that error nodes are different from conflict nodes. Error nodes refer to the nodes whose advantage score is different with the "correct" one, while conflict nodes may be erred or not.

Isolated subsets. An *isolated subset* is an undirected connected component with too few nodes. The correctness of a consistent lattice with a large amount of nodes is considered to be high. However, there may have several connected components in the lattice. Even if all the connected components are consistent, the isolated subsets with only a few number of nodes may still contain errors because there are not enough nodes to generate enough number of co-relations.

2.8. Consistency maintenance

The lattice model is mainly used to maintain consistency. All inconsistent information can be detected by a conflict discovery algorithm in Section 4.2.2. For an arbitrary lattice Γ, the *inconsistency function INC*(Γ) can measure its inconsistency. In addition, the *node inconsistency function INC*(x) measures the inconsistency of a node x. For example, we can use the *inconsistent percentage* of a node which the number of inconsistent neighbors divided by the number of its neighbors as its node inconsistency function, denoted by

$$INC(x) = IP(x) = \frac{|C(x)|}{|N(x)|}, \tag{7}$$

to know the probability of x being erred. Thus we can simply use the conflict node number as the inconsistency function.

$$INC(\Gamma) = CN(\Gamma) = \sum_{x \in \Gamma} \delta(INC(x)). \tag{8}$$

where

$$\delta(INC(x)) = \begin{cases} 1 & ,\text{if } INC(x) > 0; \\ 0 & ,\text{if } INC(x) = 0. \end{cases}$$

The concept of *critical nodes* is defined as the nodes that have many neighbors. *Severe nodes* are defined as those with a high $INC(x)$ value.

Our conflict resolution algorithm in Section 4 is a greedy method that always chooses a node with the maximum $INC(x)$ value to revise.

2.9. Consistency versus correctness

With the help of the lattice model, we can maintain a graph that captures the consistency information. However, it is hard to know the "correctness" of a knowledge base, which means the score values of nodes are correct.

Theoretically, it is very difficult to obtain a "correct" lattice Γ^*. Even human experts have different opinions in facing complicated endgames. Their conclusions may also change with time. However the knowledge of human experts is consistent all the time. So we define the difference of two nodes by only one level without changing the advantage is a *slight error*. In addition, the *tolerant correct number* is the number regarding the entries with slight errors as tolerably correct.

In the next section, we will start to introduce our endgame knowledge base system.

3. Chinese Chess endgame knowledge-based system

The Chinese Chess endgame knowledge-based system consists of an acquisition module, an inference module, an inquiry module and a verification module. The acquisition module adds entries into the endgame knowledge base. The inference module resolves conflicts among the knowledge base. After the inferencing process, there might be some conflicts that cannot be resolved using existing rules. The inquiry module then arranges these "questions" and ranks them according to their severeness for Chinese Chess human experts to make the final judgement. When the knowledge base becomes consistent, we then apply the verification module to help us finding out potentially incorrect knowledge.

The constructed knowledge base is used by our program, Contemplation, as one feature in its evaluation function. In addition to using piece and location values in the evaluation function, information about the benefit of material combinations gradually significant near endgame.

3.1. The acquisition module

The acquisition module enables us to build a large knowledge base. Its working flow contains three steps: (1) extend an existing endgame knowledge base, called *the extending procedure*, (2) use the method of predicting unknown material combinations to automatically assign scores for extended nodes, called *the predicting procedure*, and (3) store the resulting value of the selected instance into the knowledge base. Fig. 4 shows the process of building such a knowledge base.

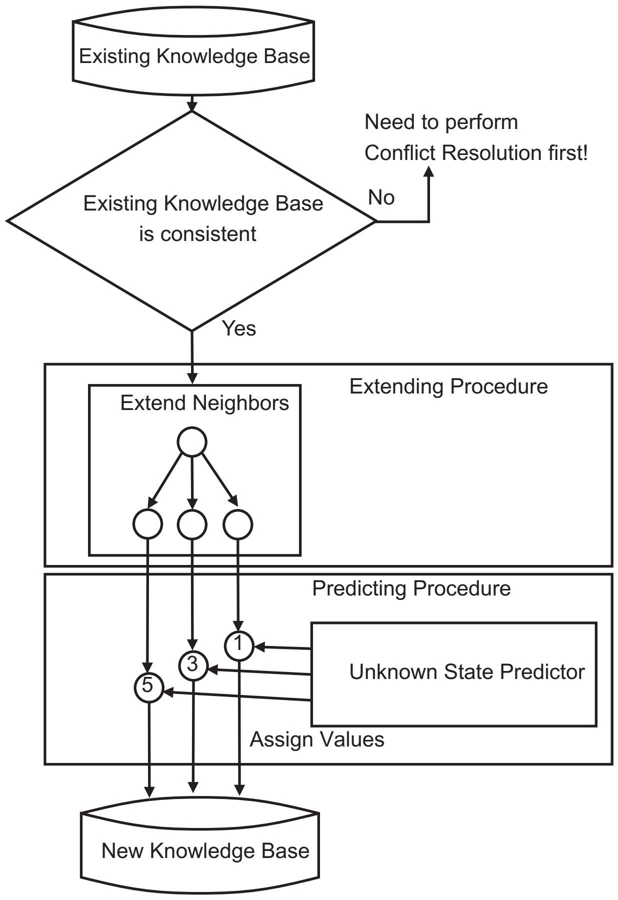

Fig. 4. The working flow of the acquisition module.

The first endgame knowledge base is called an *elementary endgame knowledge*. It is a small knowledge base that is constructed manually.

The extending procedure retrieves an instance *i* from the current knowledge base, and extends it by finding the neighbors of *i*. By definition in Section 2, the neighbors of a node are those nodes of either adding a piece or deleting a piece from the original node. There are 12 types of pieces in Chinese Chess, excepting the two kings. As a result, there are at most twelve additions, twelve deletions, totally up to 24 operations that can be applied to a newly added node.

For each newly added node, the predicting procedure calls the unknown state predictor to assign a score for it. The reliability of the constructed endgame knowledge base relies on its consistency. The target knowledge base contains the original nodes plus their neighbors.

3.2. The inference module

The inference module takes the advantage of the lattice model and is able to find all the conflicts by repeating the following four steps: (1) discovering conflicts, (2) selecting a candidate with conflicts, (3) selecting the best score value for the selected candidate, and (4) revising the score value of the selected instance.

The first step discovers all conflicts in the lattice. If the lattice is not consistent, we need to apply our greedy algorithm to resolve the conflicts. We first select the instance with the highest error value, and then try all score values to find out the most suitable score, i.e., the one that causes the least amount of conflicts, for the instance. Detail issues will be discussed in Section 4.

After performing the acquisition module and the inference module, we obtain a consistent end-game knowledge base. We can repeat the process by feeding the consistent knowledge base into the automatic extending algorithm to obtain a larger knowledge base.

3.3. The inquiry module

Since we use a greedy method in the inference module, it may end up with some unresolvable conflicts because of reaching a local minimal. The objective of the inquiry module is to identify instances that cause the most problem to our system and send them to Chinese Chess human experts to make the final judgement. They only need to make critical revisions that cannot be done by our system. The major conflict resolution work is performed by the inference module.

Note that the score of a material combination only indicates its "average" behavior. Chinese Chess human experts are good at identifying average behavior of material combinations. If Chinese Chess human experts give a score with a severe error, it will be quickly found by the inference module. Therefore, the inference module and the inquiry module working together makes the knowledge base more reliable.

3.4. Correctness verification

A consistent knowledge base may still have errors in the lattice due to: (1) isolated subsets that are all incorrect; and (2) there are errors that are consistent with the others that are also errors.

We will discuss two types of methods to improve the correctness of the obtained endgame knowledge base: (1) the random sampling verification and (2) further revision using advanced meta-knowledge.

4. Knowledge acquisition and inference

The process of the knowledge acquisition module includes the extending procedure and the predicting procedure. In the beginning, we need an elementary endgame knowledge base that is constructed manually. The extending procedure expands the neighbors of the nodes in the lattice model one by one. For each expanded node, we apply the predicting procedure to assign its score value.

For practical usage, search algorithms can be confused by having only a small amount of conflicts. To avoid the problem, we propose a conflict resolution algorithm that modifies conflict nodes one by one greedily. When the algorithm stops and the knowledge base still contains conflicts, we ask the help of Chinese Chess human experts to do a small amount of manual modification. Then we rerun our self-correcting algorithm. After iterations of modification, we can obtain a consistent endgame knowledge base.

4.1. The acquisition module

The elementary knowledge base contains the fundamental knowledge that will be used for the acquisition module. Chinese Chess human experts assigned all advantage scores for each instance in the elementary knowledge base. The elementary knowledge base contains the material combinations of up to one strong piece, a pawn and all possible combinations of guards and ministers.

When extending the knowledge base, the extending procedure gets the instances from the original knowledge base, generates expanding nodes, assigns score values for each expanded nodes, and finally stores these expanded nodes into the knowledge base.

An instance in the knowledge base maps to a node in the lattice model. The expanded nodes of an instance are the neighbors of the indicated node in the lattice by adding or deleting a piece. For each expanded node, we apply the unknown state predictor to compute its score value.

The strategy in our extending procedure is by *selective extension*, which only extends nodes most likely to maintain the consistency of the original structure. For a node x_{org} with the score value less or equal to 3, i.e., slight advantage for red, we only expand the node with adding a red piece or deleting a black piece to make $x_{new}.v \leqslant x_{org}.v$. Symmetrically, we add a black piece or delete a red piece for $x_{org}.v \geqslant 6$ to make $x_{new}.v \geqslant x_{org}.v$. If the score value of x_{org} is 5, i.e., a draw score, we only add pieces for it. The goal is to maintain the monotonic property in the lattice.

The idea of the *unknown state predictor* simulates what human players used to evaluate unknown endgames. By exchanging pieces, human experts can accurately infer the values of material combinations that were previously unheard of. For example, KNKGGMM is generally draw. When the result of KRNKRGGMM is in question, the black side can easily exchange his rook directly with the red rook, and the result will be a draw. This strategy is called *material reduction*. Whether the position in question is also draw depends on the expected values of all possible exchanges.

Another example is KRPKNGGMM shown in Fig. 5. If the red side exchanges a pawn with two black guards, the resulting

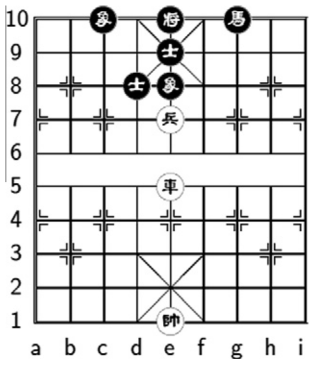

Fig. 5. A KRPKNGGMM position.

material KRKNMM can win easily, but there are still some drawing configurations. However, if the pawn is exchanged with two black ministers, the resulting material, KRKNGG would absolutely be a win. However, it may be easy for a pawn with a rook to exchange with two black guards than exchanging with two black ministers. Thus KRPKNGGMM is not likely to be a sure win.

Since exchanging piece correctly is important, our unknown state predictor method incorporates the concept of material reduction that uses a *material exchange table* and *piece-exchange method* to evaluate the score value of a material combination [3].

We design a probabilistic model that predicts the results of unknown material states by exchanging pieces. Both players can exchange pieces when necessary. A material exchange table is introduced to compute the probabilities of being able to carry out such an exchange.

The mobility of pieces are different. The easiness to exchange a certain piece for pieces of another type is also different. A *helper piece* can be any piece that is not being exchanged, but it can be used to facilitate such an exchange. Each player can select one piece as a helper piece. An *active player* is the one who wants to make a certain exchange happens. A *passive player* is the one who is forced to make an exchange. Generally, actively exchanging pieces with the assistance of a helper piece increases the chance of being able to carry out such an exchange. Contrarily, passively exchanging pieces with the aid of a helper piece may reduce the chance of pieces being exchanged. Hence, we manually construct a two-dimensional material exchange table to record the probability of exchanging for each type of piece with the assistance of helper pieces.

There are 6 types of pieces besides to the king. We use 36 tables to cover all possible combinations. Each table contains the exchange probabilities of the specified active piece with all possible helper pieces and the specified passive piece with all possible helper pieces.

Now suppose we are identifying the score value of a node x. We predict $x.v$ by performing a sequence of exchanges. An exchange is denoted as $e = \text{ex}(p_1,h_1,p_2,h_2)$ where p_1 is the red piece to be exchanged, h_1 is the helper piece of p_1, p_2 is the black piece to be exchanged, h_2 is the helper piece of p_2. $Pr(e)$ can be looked up in a material exchange table. The probability of a sequence of exchanges $E = < e_1,e_2,\dots,e_n>$ to occur is

$$Pr(E) = \prod_{i=1}^{n} Pr(e_i).$$

The score value of a sequence of exchanges E is $S(E)$, which is obtained by querying the material combination after the sequence of exchanges from the knowledge base. If the resulting material combination is not available in the knowledge base, $S(E)$ will be "unknown". The solution will only be found in the candidates E whose $S(E)$ are not "unknown".

If the red side is the active player, the score value of the identified node $x.v$ is computed as

$$x.v = \min_{Pr(E)>PLB} S(E).$$

Similarly, if the black side is the active player, the score value of the identified node $x.v$ is computed as

$$x.v = \max_{Pr(E)>PLB} S(E).$$

In our work, we suggest the probability lower bound $PLB = 10\%$ since it produces one of the best results.

4.2. The inference module

The predicting module is used to predict the score values of extended nodes. After the procedure, we will obtain a knowledge base with some conflicts. We will describe theoretical concepts about the lattice used by our method and discuss the problem in our automatic generated endgame knowledge base.

4.2.1. The concept of conflict resolution

We use Fig. 6 to illustrate our concept. In the figure, we can find that the central node, KCCKNCPGG, has a score value 3, and the four nodes on the top have score values higher than 3. All these nodes have a directed edge to the central node, but they are worse than the central node in the aspect of the red side. Hence, there must be conflicts in the top four nodes or the central node. The first material combination has four inconsistent edges. Thus, inconsistent percentage is 80%. The second material combination has only one inconsistent edge, whose conflict neighbor is the central node of the top part of Fig. 6. Thus the inconsistent percentage is 9.09%. In this example, the first material combination is more likely to be incorrect and should be modified first.

The conflict resolution algorithm has a premise that the major part of the knowledge base is correct. In the beginning, we need the elementary endgame knowledge base that is considered as "correct" for the conflict resolution algorithm. The algorithm discovers and modifies the score values of the nodes in the automatically-generated endgame knowledge base that are inconsistent with the helps of in the elementary endgame knowledge base.

4.2.2. Conflict discovery

If there is a conflict in the lattice, there must be some nodes having inconsistent neighbors. Conflict discovery procedure computes the number of inconsistent neighbors of each nodes. We define 10 *inconsistency levels*, from level 0 to level 9. Level 0 represents the inconsistent percentage in $(0\% - 10\%]$, level 1 represents $(10\% - 20\%], \dots,$ level 9 represents $(90\% - 100\%]$. We use a level vector *lv* for the number of nodes in each inconsistency level.

Our idea is a greedy method that always modifies a node of the highest inconsistency level. We use the *inconsistent edge checking*

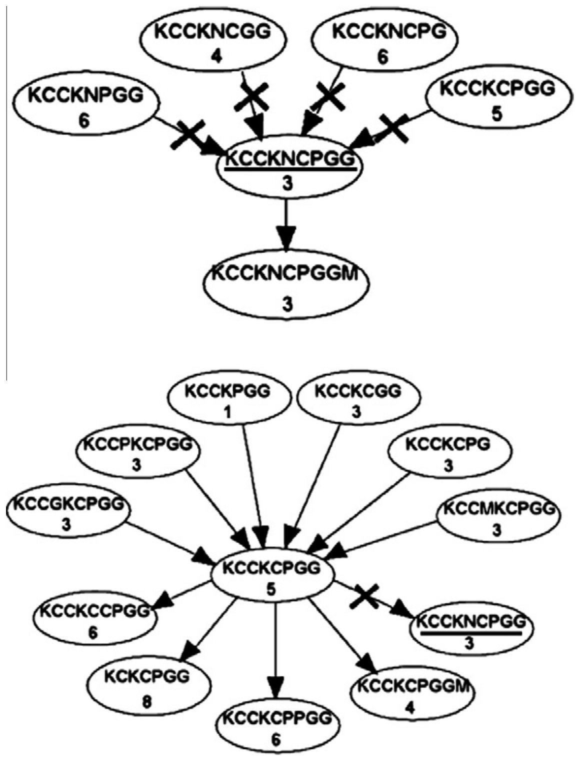

Fig. 6. Examples of two adjacent inconsistent nodes. The number in each node is its score value. The edge with a cross means a conflict.

to find conflicts between nodes. The inconsistent edge checking algorithm has two targets: (1) neighbor nodes, and (2) the mirrored material combination. We implement the piece additive rule which is defined in Section 2 for the first target. The second target, a mirrored material combination pair in our lattice is virtually considered as the same material combination. The inconsistent edge checking algorithm can ensure the consistency of mirrored material combinations.

The algorithm of discovering conflicts simply checks inconsistent edges to summary the conflict neighbors for each node.

4.2.3. Candidate selection

As described in Section 2, we use $INC(x)$ to measure the severeness of the inconsistency. The candidate selection is to choose the instance with the highest value of $INC(x)$.

There are two inconsistency functions: (1) inconsistent percentage and (2) inconsistent weight. The inconsistent percentage is the number of inconsistent neighbors divided by the number of its neighbors,

$$IP(x) = \frac{|C(x)|}{|N(x)|}.$$

Consider two nodes x and y where $IP(x) = IP(y)$ but $|N(x)| > |N(y)|$. Node x is likely to be more important because it relates to more nodes. If node x is updated to a value consistent with all its neighbors, the overall inconsistency will decrease more than the effect of node y. Hence, we define the *inconsistent weight* as

$$IW(x) = |C(x)||N(x)|. \tag{9}$$

This formula improves the $IP(x)$ measurement to make the algorithm to favor the nodes with more neighbors. These kind of nodes are considered as critical. If we modify critical nodes first, the algorithm takes less iterations to converge.

4.2.4. Score value selection

We use the inconsistency function $INC(\Gamma)$ to evaluate the performance of our algorithm. The inconsistency function after reassigning i to $x.v$ is denoted by $INC_{x,i}(\Gamma)$.

The score value selection is a sequential search algorithm on all score values i. Its optimal score value $x.v^*$ is calculated as

$$x.v^* = \operatorname*{argmin}_{i=0}^{9} \; INC_{x,i}(\Gamma). \tag{10}$$

Recall that the inconsistency function is

$$INC(\Gamma) = CN(\Gamma).$$

A node with many conflict neighbors is considered to be a severe node. In order to reduce the number of severe nodes, we include a vector lv that indicates its inconsistency level as described in Section 4.2.2. If there are two choices that result in the same number of conflict nodes, the choice that results in conflicts of a lower level is selected. So we introduce an exponential factor in the formula:

$$INC(\Gamma) = \sum_{i=0}^{9} 2^i lv_i. \tag{11}$$

Most nodes with a high inconsistent percentage are first reduced to the nodes with a low inconsistent percentage. The scores of these nodes are then modified to ensure the consistency. The new INC formula increases the probability for the algorithm to overcome local minimals.

4.2.5. Conflict resolution algorithm

The conflict resolution algorithm, shown in Algorithm 1 finds inconsistent nodes in our lattice and reassigns the score values of some nodes to reduce the number of inconsistent nodes by the following four steps: (1) conflict discovery, (2) candidate selection, (3) score value selection, and (4) modification. Our algorithm repeats the four steps until no more candidate can be found.

Algorithm 1: Conflict resolution algorithm.

```
procedure ConflictResolution ($\Gamma$)
err_num = ConflictDiscovery ($\Gamma$)
while err_num > 0 do
  ClearModifiedFlag ($\Gamma$)
  {*set the modified flag as not modified*}
  x = CandidateSelection ($\Gamma$)
  while $x \cdot modified = x \cdot invariable = false$
  do
  v = ScoreValueSelection (x)
  Modify (x, v) {*change score value*}
  $x \cdot modified = true$
  x = CandidateSelection ($\Gamma$)
  err_num = ConflictDiscovery ($\Gamma$)
end procedure
```

The conflict resolution algorithm scans all nodes in Γ in an iteration. The algorithm is surely to converge as it modify at least a node in an iteration. It is also true that for the modified knowledge base Γ', $INC(\Gamma') \leqslant INC(\Gamma)$. When the algorithm stops, either the knowledge base becomes consistent or it falls into a local minimal. For the formal case, we need the verification module to verify the correctness. For the latter case, we ask the help of the inquiry module.

As described in Section 2, we use the element, *modified_flag*, to filter the tried nodes in the same iteration. In addition, if the *invariable_flag* of a node is set, the conflict resolution algorithm skips it. This ensures the instances modified by Chinese Chess human experts cannot be further changed by our algorithm.

5. Inquiry and verification

5.1. Inquiry

After performing the conflict resolution algorithm, it produces a list of conflict nodes sorted according to their inconsistency ratios. Human experts only need to give the score values for severe nodes. Then, we rerun the conflict resolution algorithm again to reduce more conflicts. After several iterations, we obtain a consistent knowledge base of size more than 120 thousands, which is impossible for any human expert to manually enter.

5.2. Correctness verification

In this section, we discuss two types of methods to improve the correctness of the obtained endgame knowledge base, namely the random sampling verification and the usage of advanced meta-knowledge.

5.2.1. k-random sampling verification

Checking by k-random sampling verification is a way to ensure the correctness of the lattice. We select 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 entries are verified by Chinese Chess human experts. If n error nodes are reported, the approximated value of the whole error nodes is n/p. For example, our END6 knowledge base contains 69,595 instances. END6C is the consistent version of END6. We perform k-random sampling verification on it. By using $k = 4$, $p = 1\%$, there are only 1,533 entries that are not slight errors. The tolerant correct number is 68,062 (97.80%) [3]. The modified data can also be used to reduce the errors of the entire knowledge base.

5.2.2. The usage of advanced meta-knowledge

The random sampling verification method can evaluate the correctness of a consistent endgame knowledge base, but it can only discover a small set of error nodes.

To further improve the correctness of the consistent endgame knowledge base, we first need to figure out the whereabout of potentially incorrect nodes.

Meta-knowledge can be viewed as rules describing advanced relations among the data in the knowledge base. If we can find some rules that are true for most cases, they will be good criteria to discover the existence of potential errors in the knowledge base.

In Section 2, we described the piece additive meta-knowledge that is the kernel knowledge for our graph model. We will illustrate two more examples of using meta-knowledge to find potential errors in the knowledge base: (1) the level gap analysis, (2) piece exchange analysis.

Consider KRNPGGMMKNCGGMM in Fig. 7 that the red side has three attacking pieces, a rook, a knight and a pawn, and the black side has a knight and a cannon. Its score is 2 (advantageous). However, its neighbor that a black minister is removed has a score of 0 (sure win). In this example, the difference of the material combination of the two nodes is one black minister, but the gap of scores is 2 which is too much most of the time for two endgames differ by only a defending piece.

Each adjacent node pair differ by only one piece. We want to analyze the gap variations between each adjacent node pair. In order to limit gap values within an upper bound, we select the pairs with the minimum distance to be analyzed. From knowledge of human experts, we observe that in most cases, the upper bound of the level gap between material combinations that differ by a guard or a minister is at most 2. We define the statement as *the level gap property*:

$$Pr(\gamma(x.m, y.m) \leqslant \mu) \geqslant \delta. \tag{12}$$

In the formula, γ is a function to obtain the level gap of two material combinations $x.m$ and $y.m$; μ is the upper bound of the level gap, which is defined as $\mu = 2$ in this paper; δ is the confidence index. The δ value is picked to be high enough such that we can use the level gap property to discover the material combinations that are potentially incorrect. Incorrect nodes with level gap more than or equal to 3 are also severe nodes that may cause the program to make big mistakes.

Another example is KNCPPGGMMKNNPPGGMM, whose score is 3 (i.e., slightly advantageous). After exchange a pawn, KNCPGGMMKNNPGGMM also has a score of 3. Thus we want to know whether it is possible for a pawn-exchanging to happen and to result in swapping of advantageous. Finally, we discover that if both sides still have pawns after the exchange, the advantage

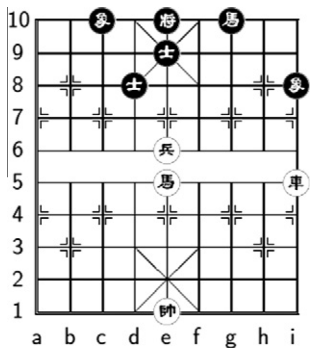

Fig. 7. A KRNPGGMMKNCGGMM position.

does not change. Since this is a general meta-knowledge, it can be used to further check the consistency of the knowledge base.

We expect advanced meta-knowledge to not only useful for the verification purpose, but also can be used later for other Computer Chinese Chess research.

6. Experimental results and discussions

In this section, we evaluate our endgame knowledge base used in a Chinese Chess playing program, Contemplation [12]. In the modification number comparison, we evaluate the number of modifications between several versions of knowledge bases. In the self-play experiment, we perform self-play by using different versions of endgame knowledge base to demonstrate the usefulness of our constructed knowledge base.

6.1. Modification number comparison

Our elementary knowledge base contains 18,959 instances. By extending it, we obtain a larger knowledge base of 69,595 instances, called END6. After resolving all the conflicts in it, we obtain a consistent knowledge base, called END6C. This version is then extended again to obtain a knowledge base of 123,985 instances, called END12. We also obtain END12C by resolving all of its conflicts. Furthermore, we continue to improve the correctness of END12C to obtain a much better refined version, called END12CR. Here we show the numbers of modifications in Table 1.

Recall the formula of advantage class:

$$AC(x.v) = \begin{cases} x.v, & \text{if } x.v \leqslant 4; \\ x.v - 1, & \text{if } x.v > 4. \end{cases}$$

The results in Table 1 compares the five versions. It also summarizes the number of instances whose scores differ by n, $n = 0, 1, \ldots, 8$, before and after modification. There are totally 9 levels for scores, defined as *modification level difference*. In Table 1, all difference values are computed from the advantage class. For example, $x.v = 3$ before modification and $x'.v = 1$ after modification. The level difference is $AC(x'.v) - AC(x.v) = 2$. If another instance $y.v = 2$ before modification and $y'.v = 8$ after modification, the modification level difference is $AC(y'.v) - AC(y.v) = 5$. The modification level difference 0 means the difference between advantage scores 4 and 5.

Table 1
Modification number comparison of endgame knowledge bases.

(*A*) END6 vs. END6C				
Number: 69,595		Total diff: 36,206		
0	1	2	3	4
35	9056	1765	906	2444
5	6	7	8	9
1941	14,223	5836	0	0
(*B*) END12 vs. END12C				
Number: 123,985		Total diff: 27,644		
0	1	2	3	4
10	13,138	6656	5201	1411
5	6	7	8	9
856	350	22	0	0
(*C*) END12C v.s. END12CR				
Number: 123,985		Total diff: 27,759		
0	1	2	3	4
157	13,750	9534	3582	666
5	6	7	8	9
44	18	8	0	0

Table 2
The (loss:win:draw) results of self-play test according to the versions in the first row.

	END6C	END12C	END12CR
Trivial	15:32:53	30:23:47	20:31:49
END6C		36:30:34	23:34:43
END12C			30:44:26

Table 3
The confidence intervals of the version in a specified column against the version in a specified row in the self-play test.

	END6C	END12C	END12CR
Trivial	(0.47, 0.70)	(0.37, 0.56)	(0.45, 0.64)
END6C		(0.38, 0.56)	(0.45, 0.66)
END12C			(0.46, 0.68)

Table 4
The Elo increases of the version in a specified column against the version in a specified row in the self-play test.

	END6C	END12C	END12CR
Trivial	(−20.9, 147.2)	(−92.5, 41.9)	(−33.3, 102.8)
END6C		(−85.0, 41.9)	(−34.9, 115.2)
END12C			(−27.9, 130.9)

By comparing Tables 1(A) and (B), we can observe that the amount of differences reduces when the size of the knowledge base doubled. An important discovery is that the effort to resolve conflicts is also reduced when a better prior knowledge base is used. The number of large modification level difference in Table 1(B) is much less than Table 1(A). In Table 1(C), the number of large modification level difference is much less than Table 1(B). It shows that the extending procedure is likely to produce less errors using a better prior knowledge base.

6.2. Self-play test

In order to estimate the usefulness of the end-game knowledge base for our Contemplation program, we conduct four versions in the self-play test: (1) trivial endgame knowledge, (2) END6C knowledge base, (3) END12C knowledge base and (4) END12CR knowledge base.

The first version is a baseline that contains manually added endgame knowledge that originally used by Contemplation.

In each game, the average time for each move is 5 s. Each pair of the versions play 100 games. The results is shown in Table 2; the confidence intervals is shown in Table 3; the Elo increases is shown in Table 4.

The experiment is performed on a server with a two dual-core AMD CPU's (2.8 GHz) and 20 GB of RAM. The rank 1 version is END12CR. It won more games than other three versions. Rank 2 version is END6C. Rank 3 version is Trivial version. The worst version is END12C. The result supports our conjecture that even only a few errors can confuse the search algorithm. END12CR won more games than other versions. However, it also lost more games comparing with END6C. END6C is the most stable version among all.

Our another conclusion is that we cannot win a game only with a perfect endgame knowledge base since to force a win in endgame needs a lot of moves to transform a winnable position into a win state. A good endgame knowledge base can make sure the process of transforming into an endgame is not disadvantageous.

7. Conclusions

Our goal is to aggregate knowledge to identify which kinds of endgames are useful to a search engine.

In this paper, we proposed a lattice model to ensure consistency and to improve the correctness of the knowledge base. Our Chinese Chess knowledge-based system can effectively acquire a large amount of consistent endgame knowledge base by coordinating with the inference module and the inquiry module. The verification module provides a mechanism that efficiently improves the correctness of the knowledge base.

We discover that if we have more consistent knowledge, we need less effort to build a larger consistent knowledge base, and even to further improve the correctness of the knowledge base.

The knowledge base can be integrated with our Chinese Chess program, Contemplation. The self-play experiment shows that END12CR is the best version, and END6C is the most stable version. In summary, quantity is as important as quality for knowledge-based systems. In the future, more experiments can be carried out in read tournaments in addition to the self-play ones.

Acknowledgment

1. The authors Hsu, T. -s. and Chen, B. N. are supported in part by National Science Council (Taiwan) Grants 97-2221-E-001-011-MY3.
2. Hsu, T. -s. is the corresponding author.
3. Some earlier ideas in this paper appeared in [4].
4. We thank anonymous reviewers for very helpful comments.

References

[1] A. Kishimoto, M.Müller, A general solution to the graph history interaction problem, in: Nineteenth National Conference on Artificial Intelligence (AAAI 2004), San Jose, CA, 2004.
[2] A. Reinefeld, An improvement of the scout tree search algorithm, in: ICCA Journal, volume 6, 1983, pp. 4–14.
[3] B.N. Chen, P.F. Liu, S.C. Hsu, T.-s. Hsu, Knowledge inferencing on Chinese Chess endgames, in: Computers and Games, vol. LNCS 5131, 2008, pp. 180–191.
[4] B.N. Chen, P.F. Liu, S.C. Hsu, T.-s. Hsu, Conflict resolution of Chinese Chess endgame knowledge base, in: Advances in Computer Games, vol. LNCS 6048, 2009, pp. 146–157.
[5] C.E. Shannon, Programming a computer for playing chess, in: Philosophical Magazine, vol. 41, 1950, pp. 256–275.
[6] D.E. Knuth, R.W. Moore, An analysis of alpha-beta pruning, in: Artificial Intelligence, vol. 6, 1975, pp. 293–326.
[7] H.J. van den Herik, J.W.H.M. Uiterwijk, J. van Rijswijck, Games solved: now and in the future, in: Artificial Intelligence, vol. 134, 2002, pp. 277–311.
[8] K. Thompson, Retrograde analysis of certain endgames, in: ICCA Journal, vol. 9, 1986, pp. 131–139.
[9] K. Thompson, 6-piece endgames, in: ICCA Journal, vol. 19, 1996, pp. 215–226.
[10] Kang, X., Li, D., Wang, S., A multi-instance ensemble learning model based on concept lattice, in: Knowledge-Based Systems, vol. 24, pp. 1203–1213.
[11] Kavurucu, Y., Senkul, P., Toroslu, I.H., Concept discovery on relational databases: new techniques for search space pruning and rule quality improvement, in: Knowledge-Based Systems, vol. 23, pp. 743–756.
[12] K.C. Wu, T.-s. Hsu, S.C. Hsu, Contemplation wins Chinese Chess tournament, ICGA Journal 27 (2004) 186–188.
[13] P.S. Wu and P.Y. Liu, T.-s. Hsu, An external-memory retrograde analysis algorithm, in: Computers and Games, vol. LNCS 3846, pp. 145–160.