Chinese Dark Chess

Bo-Nian Chen[1], *Bing-Jie Shen*[2], *and Tsan-sheng Hsu*[3]

Taipei, Taiwan

ABSTRACT

Chinese dark chess is a popular and easy-to-learn game in Asia. The characteristic of possible revealing of unknown pieces makes it different from Chinese chess or Western chess. Players with luck may win a game by chance. Thus, there is a probabilistic behavior that a player has to consider.

Computer Chinese dark chess problems can be divided into three phases: 1) the opening game, 2) the middle game, and 3) the endgame. Revealing pieces reasonably and effectively is the main issue in the opening game. In the middle game, the choice of revealing pieces or moving pieces becomes the critical issue. Designing a good evaluation function is also important both in the middle game and endgame. In an advantageous endgame, how to capture all opponent's pieces to win the game is also a hard problem.

Search-based methods, such as $\alpha\beta$ pruning, are only suitable in the positions where revealing actions do not influence the results. However, according to our experiments, programs that reveal pieces reasonably and effectively have better playing strength. In this paper, we introduce the game Chinese dark chess and give some research topics about this game. We also discuss some strategies for considering the revealing actions in this paper.

Keywords: Chinese dark chess, dark chess, half chess, Banqi

1. INTRODUCTION

Chinese dark chess is a popular version of Banqi, a variation of Chinese chess that only uses half of the board, also called *dark chess*, *half chess* or *blind chess*. [4] It is played in oriental countries where Chinese chess is played, but with much more players. Most people playing Chinese chess play Chinese dark chess. A lot more people can play Chinese dark chess while not able to play Chinese chess.

The pieces in Chinese dark chess are the same as those in Chinese chess: each player has sixteen pieces of one color [5], including one king (K/k) [6], two guards (G/g), two ministers (M/m), two rooks (R/r), two knights (N/n), two cannons (C/c), and five pawns (P/p). The board size consists of 4×8 squares, totally thirty two squares. Unlike Chinese chess, pieces are placed within squares in Chinese dark chess. In the beginning of a game, all pieces are randomly placed on the board with piece icons facing down so that piece types are unknown.

When playing a game, two players make actions alternately. There are two kinds of actions: 1) *the flipping action* that reveals an unknown piece that is facing down, which is abbreviated to an *unknown piece* in this paper and 2) *the moving action* that moves one revealed piece of his own color from a source to a destination or for the cannon,

[1]Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. email:f92025@csie.ntu.edu.tw.

[2]Department of Radiation Oncology, Far Eastern Memorial Hospital, Taipei, Taiwan. email:dale@iis.sinica.edu.tw.

[3]Corresponding author. Institute of Information Science, Academia Sinica, Taipei, Taiwan. email:tshsu@iis.sinica.edu.tw.

[4]The game, dark chess, is also a chess variant that players can only see their own pieces and the grids of their legal moves; half chess is also a chess variant that uses half chess board; blind chess sometimes refers to blindfold chess, which is a training method that players do not look at the board during playing.

[5]One side is Red. The other side is Black.

[6]We use upper-cased letters for the Red side and lower-cased letters for the Black side.

it can jump over a piece. The unknown piece being flipped is called a *revealed piece*. From the above discussion, we know the player who plays first must flip a piece.

The color of the first flipped piece owns that color for the rest of the game. The opponent holds the other color. All types of pieces except cannons can only move or capture a piece one square up, down, left and right within the 4×8 grid area. Cannons move in the same way as other pieces, but when capturing a piece, just like Chinese chess (Yen *et al.*, 2004), they need to *jump* over exactly one piece to capture a piece of any distance in the same row or column. The indicated piece for a cannon to jump over is called a *carriage*.

Each piece type has a rank, as shown in Table 1. A piece can capture the pieces with equal or lower ranks. However, there are exceptions as follows: 1) a cannon can only capture pieces of any rank by jumping over a piece, 2) the king cannot capture pawns, and 3) pawns can capture the king.

A player wins when either the opponent has no legal move, i.e., stalemate, or all pieces are captured. The game ends in a draw when both players do not capture or reveal a piece within 40 plies. Repetition of positions also results in a draw.

Table 1: The rank of piece types.

Piece types	Red	Icon	Black	Icon	Rank	Exceptions
King	K	帥	k	將	1	cannot capture pawns
Guard	G	仕	g	士	2	none
Minister	M	相	m	象	3	none
Rook	R	車	r	車	4	none
Knight	N	馬	n	馬	5	none
Cannon	C	炮	c	炮	6	can only capture all types of pieces by jumping over
Pawn	P	兵	p	卒	7	can capture the king

The notation for recording moves is the same as *the algebraic chess notation*. Since the board size is 4×8, we have the letters a to d to label the rows from the left to the right. The columns are numbered from 1 to 8 bottomed up. Moving a piece is recorded as $S - D$ where S stands for the source location and D stands for the destination location. Revealing a piece is recorded as $P(T)$, where P indicates the place of the piece to be revealed and T indicates the revealed piece type, upper-cased characters for red pieces and lower-cased characters for black pieces. A sample opening game is shown in Figure 1.

To record a complete game, only the sequence of moves and result of a flipping action are needed. For an incomplete game, i.e., a middle game or an endgame, we need the information of its initial position. A position contains pieces on the board, the numbers of all types of unknown pieces and the player, his color, to move next.

By considering the initial unknown pieces, Chinese dark chess is a two-player zero sum game with incomplete information (van den Herik, Uiterwijk, and van Rijswijck, 2002). Another example of incomplete information game is a chess variant, Kriegspiel (Ciancarini and Favini, 2007; Sakuta and Iida, 2000) or dark chess. Kriegspiel has many rule sets. Players can only see their own pieces and the grids of their legal moves, which is similar to dark chess as described above.

The state-space complexity of the game is $2^{32}(32!/((5!)^2(2!)^{10}))$, roughly 10^{37}. A game lasts for about 90 moves in average. Each position has an average of 32 valid moves. The game-tree complexity is roughly 10^{135}. We can see that the state-space complexity of Chinese dark chess is between Draughts and Western chess, whose state space complexities are 10^{30} and 10^{46}. The game-tree complexity of Chinese dark chess is between Western chess and Chinese chess. It seems that a game-tree search algorithm is sufficient to handle the game. However, the nature of incomplete information during the play makes the game more complex. We will discuss a detailed analysis about the game-tree search complexity in Section 2.1. The complexity of Chinese dark chess is similar to chess and Chinese chess. However, the distinguishing feature is its characteristic in incomplete information.

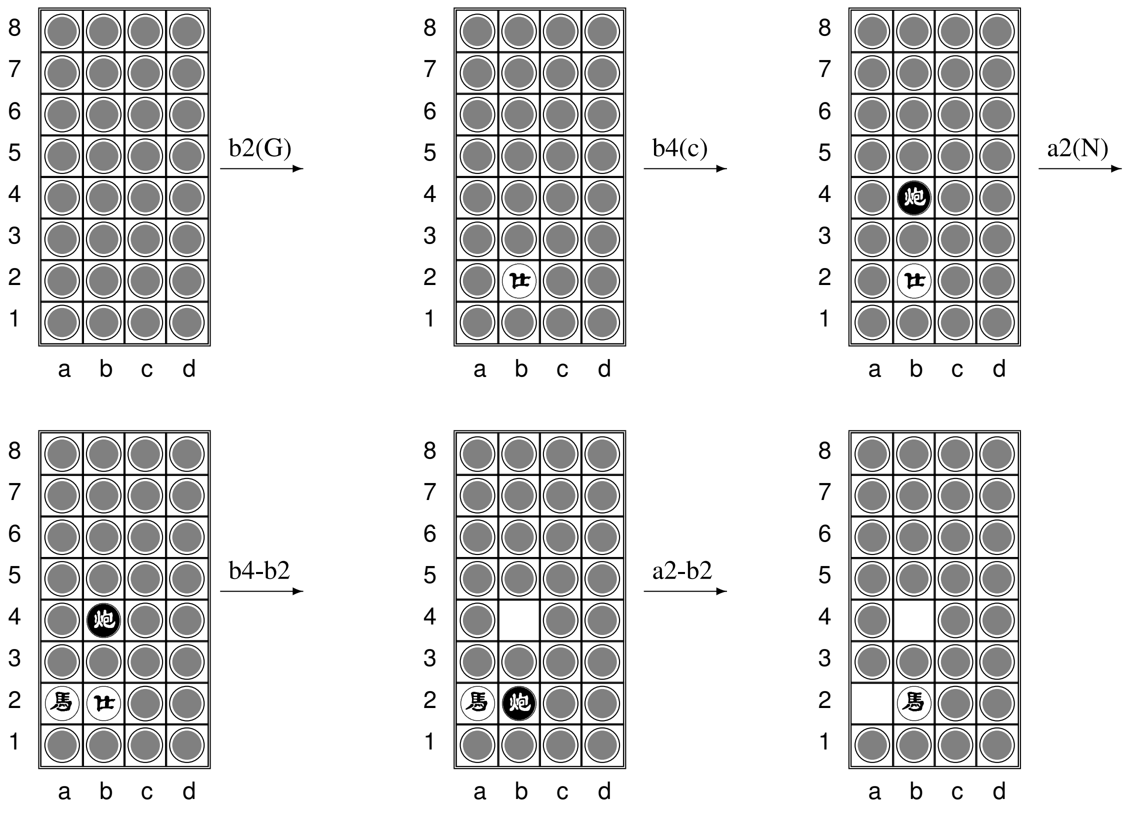

Figure 1: A sample opening game.

This paper is organized as follows. In Section 2, we discuss the problems in each phase of a game in Chinese dark chess. In Section 3, we introduce a baseline algorithm and an exhaustive search algorithm to discuss about the properties and research issues in Chinese dark chess. In Section 4, we propose refined algorithms that provide better flipping handling strategies and give some experimental results to compare these algorithms. In Section 5, conclusions and future work are presented.

2. RESEARCH PROBLEMS IN COMPUTER CHINESE DARK CHESS

As an incomplete information game, the Chinese dark chess problem is hard to be either weakly-solved or strongly-solved. Unknown pieces make the result of a position undecidable. An exhaustive search algorithm cannot precisely obtain the scores of positions with unknown pieces, but it can compute their approximated value.

In this section, we will discuss the complexity of an exhaustive search in Chinese dark chess, and possible research problems in the opening game, the middle game and the endgame.

2.1 Analysis of the Search Complexity

To start a game, we can only reveal a piece. In the opening game, there may be a few revealed pieces and many unknown pieces in a position; some pieces are captured. When more pieces are revealed and captured, the branching factor reduces. To analyze the branching factor, we use the following definitions.

Definition 1. The variable n is the piece number remaining on the position. We have $n = n_r + n_f$ where n_r is the number of revealed pieces and n_f is the number of unknown pieces.

Definition 2. The variable n_t is the number of piece types that are still unknown. For example, we have $n_t = 14$ in the beginning of a game.

Definition 3. The estimated branching factor $b = M(p) + R(p)$ where p is the current position. $M(p)$ is the number of nodes for computing moving actions, and $R(p)$ is the number of nodes for computing flipping actions in position p. A position p contains information about n_r, n_f, n_t and other information such as locations of pieces.

The number of valid moves of a cannon is at most six. Other pieces can have up to four moves. Hence the upper bound of the number of moving actions is $4n_r + 8$. A position usually contains roughly equal numbers of red and black pieces, we estimate the number of valid moves of a player as $M(p) = (4n_r + 8)/2 = 2n_r + 4$. To enumerate all possible combinations, the exhaustive search algorithm generates n_t children for a flipping action in the search tree. The number of nodes for computing flipping actions is then defined as $R(p) = n_f \times n_t$. The estimated branching factor b is thus $b = 2n_r + n_f \times n_t + 4$.

Here we show our analysis of branching factors in different stages of a game in Table 2. We make three assumptions in this analysis: 1) the analyzed games are reasonable games, not randomly generated positions, 2) when there are $32 - n_f$ pieces revealed, half of the revealed pieces are captured 3) we assume the revealed pieces are uniformly distributed among all pieces.

Table 2: Branching factors in five stages of a game.

Stage	$p = (n_r, n_f, n_t)$	$M(p)$	$R(p)$	b
1	(0, 32, 14)	4	448	452
2	(4, 24, 14)	12	336	348
3	(8, 16, 12)	20	192	212
4	(12, 8, 4)	28	32	60
5	(16, 0, 0)	36	0	36

The second assumption is based on the observation that as the game goes on, the piece on the board decreases. The empty grid becomes more from the opening game to the middle game and from the middle game to the endgame.

To explain the uniform distribution assumption, when we reveal fourteen pieces, the number of all piece types are decreased by one. If the number of revealed pieces is less than fourteen, the types having higher occurence probability are computed first. There are three levels of probabilities. Pawns are classified in the highest level, kings are classified in the lowest level, and the rest of types of pieces are classified in the middle level. For example, if sixteen pieces are revealed, fourteen of them are mapped to each type of pieces with additional two pieces being pawns. Thus, both kings are revealed and the value of n_t is 12.

Stages 1 and 2 are in the opening game; stages 3 and 4 are in the middle game; stage 5 is in the endgame. The analysis shows that the branching factor in an exhaustive search algorithm is huge in the opening and middle game. Solutions other than exhaustive search for evaluating flipping actions, such as Monte Carlo-based methods (Kocsis and Szepesvári, 2006), may be needed to tackle this issue.

2.2 Issues in the Opening Game

In chess or Chinese chess, opening book construction is a common strategy to solve the problem in the opening game. We can build an incomplete game tree for the opening game, and compute the utility of each opening move by statistical strategy (Chen and Hsu, 2001). The opening game problem in Chinese dark chess is more difficult because there are many unknown pieces in the opening game. Unknown pieces in Chinese dark chess makes the result of a certain opening move indeterminable.

As a result, the first idea is using a search algorithm to handle the opening game. A possible research problem is to reduce nodes searched in positions with unknown pieces. There is a trade off between the correctness and the efficiency. One possible improvement is to apply Monte Carlo tree search on the positions with unknown pieces to compute an approximate solution. However, since the playouts need to be done until a game is finished in a traditional Monte Carlo method, one may try to limit the plies by using other criteria such as a limited depth. Another choice is to consider only some selective, not all possible piece types for an unknown piece. This method is likely to reduce the number of nodes searched to a certain order and still retain reliability.

2.3 Issues in Piece Flipping

Three critical issues about flipping actions are: 1) when should a player flip pieces, 2) which place should a player flip, and 3) how to analyze the positions after flipping a piece. Some possible heuristics for the three critical issues are listed follows.

When should a player flip pieces? In the opening game, a player frequently flips pieces. It is generally considered appropriate to flip pieces when we cannot capture any piece and our pieces are not being captured. However, the heuristic is not always correct. For example, when our piece is being threatened and unable to move, flipping a piece may cause an exchange of pieces, or discover a new piece that protects the piece being threatened.

Which piece should a player flip? When the opponent king is alive and our pawns are not yet revealed, we can try to flip the unknown pieces adjacent to the opponent king. When at least one of our cannons are not revealed, we can flip a piece that is behind the carriage in the same row or column of an opponent's piece to increase the chance of capturing pieces.

How to analyze the positions after flipping a piece? A flipping action results in n_t possible variations. The variable n_t represents the number of unknown piece types. When $n_t = 1$, the evaluation function of the position after a flipping action is a precise value. Otherwise, we can only obtain an expected value out of n_t possible choices. When we compare two values in the search algorithm, a score that is an expected value is less reliable. For example, we have two choices in a position with the scores of 10 and 20, respectively. The score of the first choice is a precise value that is computed according to the best moves. The score of the second choice is an expected value of three flipping actions. Assume the original scores of the flipping actions are -20, 0, 80, respectively. If we only consider the score itself, we will select the value 20. However, there is a probability of $2/3$ that the result will be worse than the choice of the value 10.

2.4 Issues in the Middle Game

The critical issue here in the middle game is how to design a good evaluation function. To decide which pieces to remain on the board is an important issue in the middle game of Chinese dark chess. For example, the king, though has the highest rank, is usually captured by pawns or cannons in early phase of a game if revealed. Therefore, guards becomes as important as king. If both sides have neither king nor guards, ministers become the pieces of the highest rank.

To design a suitable evaluation function, we may consider to represent piece values either by static or dynamic strategy. *Static piece value strategy* uses a fixed table of the values of all pieces while *dynamic piece value strategy* uses variant table. In Chinese dark chess, pawns are powerful pieces to attack the opponent king. The material value of a pawn should be a high value when the opponent king is alive. After the king is captured, pawns are more or less useless and can be ignored. By static piece value strategy, a pawn with a low value may be easily captured. We can use dynamic strategy that gives pawns higher values when the opponent's king is alive, and gives them a low value otherwise.

Cannons are very powerful when carriages exist and the pieces that can threaten them are a distance away. A good arrangement with cannons can even threaten pieces of high ranks such as king or guard. If a cannon is limited in the surrounding of opponent pieces or unknown pieces, they can be easily captured. In a good evaluation function, the score of a cannon is decided according to the situations above. Thus, we need to solve the following two problems. The first is to consider the influence of other pieces on cannons. The second is to compute the safety of a cannon.

It is easy to draw a game in the middle game by performing repeated moves. If a program tends to use repeated moves to draw all the time, the games are very boring. A program that plays repeated moves in disadvantageous positions is reasonable. Playing repeated moves in an advantageous position misses the chance of winning a game. Hence, the judgement of advantage or disadvantage is a very important issue in the middle game.

2.5 Issues in the Endgame

Retrograde analysis is a popular algorithm to construct endgame databases in many games. Several games are solved by retrograde analysis, such as checkers (Schaeffer, 2005). In more complex games like chess, databases of 6 pieces or less are available (Thompson, 1996). In Chinese chess, current largest database contains two strong pieces, i.e., attacking pieces other than pawns, on both sides (Wu, Liu, and Hsu, 2006). One can also use retrograde analysis to build some endgame databases.

Some issues in endgame are discussed in the following. First, for pieces that are not pawns and cannons, only the relative strength of pieces matter. For example, the endgame GN v.s. MN is equivalent to MN v.s. RN.

To illustrate the concept of the Chinese dark chess endgame problem, we first discuss the characteristics of simple endgames. The simplest endgame contains a red piece and a black piece. The distance of two pieces is defined as their Manhattan distance. The player with the piece that can capture the other conditionally wins when the distance of the two pieces is an odd number in his turn. Otherwise, the game is a draw.

A petty piece is defined as the piece that does not have attacking power but are used to change the turn. In the left of Figure 2, the player who moves first loses the game. However, in the right of Figure 2, black wins no matter who plays first. If black plays first, he can move the petty piece, the pawn, and the position reduces to the situation that red plays first. In Figure 3, we show an example of how to draw a disadvantageous game by using a petty piece. In the left figure, if red moves first, he loses the game, otherwise the game is a draw. In the right of Figure 3, no matter who plays first, the game is guarenteed to be a draw. If red moves first, he can move the petty piece to avoid being captured.

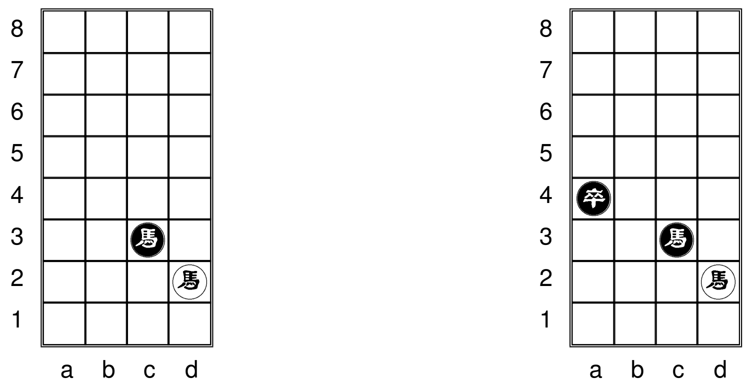

Figure 2: Winning by a petty piece in the endgame of two equal-ranked pieces.

In endgames containing petty pieces, a player needs to use strategical method to win or draw a game. How to use petty pieces reasonably is a problem to be studied.

3. IMPLEMENTING A CHINESE DARK CHESS PROGRAM

In this section, we will compare two basic search methods for Chinese dark chess. The first method is intuitively a baseline method that do not handle flipping actions. The second method is an exhaustive search algorithm that considers both moving actions and flipping actions. When searching a position with a flipping action, it expands all unknown pieces with all types that are possible to appear. In our experiments, there is a great gap in playing strength between the baseline program and the exhaustive search program. This emphasizes that reasonably handling flipping actions is important in the game of Chinese dark chess.

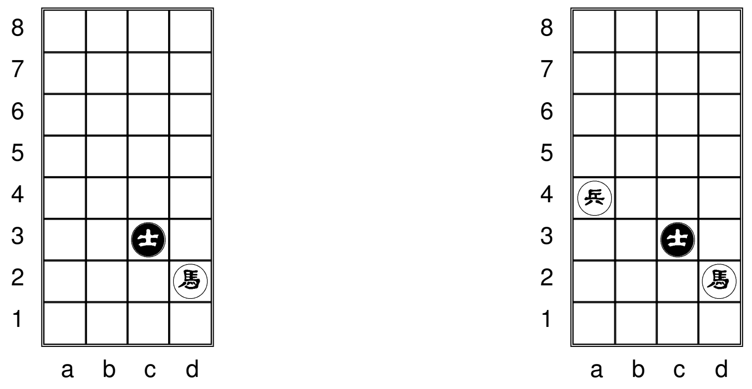

Figure 3: Drawing by a petty piece in a disadvantageous endgame.

3.1 Our Baseline Program

The baseline program shown in Algorithm 3.1 incorporates an nega-max search algorithm, a simple evaluation function designed by static piece value strategy, and *a flipping policy* to decide whether we want to make a moving action or a flipping action at the root. If a flipping action is selected, we randomly select an unknown piece to reveal. When we visit a node with no moving action, we use the *no-move cut strategy* that calls the evaluation function and returns the computed value.

```
function negamax($position$, $depth\_limit$)
$s = -\infty$
$next\_position$ = generate_moves($position$)
**if** $\#(next\_position) = 0$ or $depth\_limit = 0$ **then**
   **return** evaluate($position$)
**for all** $p$ in $next\_position$ **do**
   $v = -$negamax($p$, $depth\_limit - 1$)
   **if** $v > s$ **then**
      $s = v$
**if** $depth\_limit = initial\_depth$ **then**
   $c$ = evaluate($position$)
   **if** there are unknown pieces and $s \leq c$ **then**
      $v = -$negamax($position$, $depth\_limit - 1$) {∗perform null-move search∗}
      **if** $v > s$ **then**
         {* choose the best flipping action instead of the best moving action *}
**return** $s$
```

Algorithm 3.1: The baseline algorithm

3.1.1 Flipping Policy

The baseline program considers only moving actions. After the search algorithm returns a move with the best value given a fixed depth or time limitation, we have two choices: 1) perform the best move that the search algorithm suggested, and 2) perform a random flipping action. The flipping policy is a decision function to identify when we should make flipping actions. We list three possible flipping policies below.

The first policy is *move-first policy*. When there is a moving action, we perform the search algorithm and play the best move. Since a flipping action takes one ply, the move-first policy never wastes time flipping and just wait for the opponent to flip all unknown pieces unless no legal move is found. However, when there is no *effective move*, i.e., a move with a material gain, the program would rather make meaningless moves than try to flip an unknown piece. In addition, making meaningless moves sometimes leads to danger in the future. Furthermore, if both players use this policy, their games seem to be drawn all the time because they tend to make repeated moves.

To improve the move-first policy, we have the second policy, called *comparing policy*, which makes flipping actions when no effective move exists. The strategy that determines when to flip pieces is a *comparison algorithm*. The first idea is to compare the score of the best moving action and the average score of performing a flipping action, which is performed by a null-move search. If the score of performing a flipping action is not worse than the best moving action, we make a flipping action.

The comparison algorithm solves the problem of making meaningless moves. However, when a piece of the opponent is to be captured for sure, the program usually does not capture it because the score of the moves without capture the piece is also the same as the move to capture it. If the opponent flips surrounding pieces, the to-be-captured piece may have chances to escape. Furthermore, the revealed piece can even threaten the potential attacking pieces.

Our third policy, *take-advantage-or-flip policy* uses the *enhanced comparison algorithm* that compares the immediate evaluation score of the current position and the score of performing the best moving action. The immediate evaluation score is the score of the position before making any move. If the score after performing a best moving action is higher than the score of the original position, we choose the moving action. Otherwise, a null-move search is used to determine whether to use the searched moving action or perform a flipping action. If the score of the null-move search is higher than that of the previous best move, it means that the player gets higher value when he does not move any piece. Thus, a flipping action is suggested. In this method, the program immediately captures pieces that cannot escape. Thus, the behavior of a computer program is more like human players.

3.2 The Exhaustive Search Program

The exhaustive search program shown in Algorithm 3.2 incorporates $\alpha - \beta$ pruning algorithm on both moving actions and flipping actions. The variable s is the score returned by the alpha-beta pruning algorithm. It is used to compare with c, which is the score by performing an immediate evaluation function in $position$. The measurement of a flipping action is an important issue when searching. We measure the value of a flipping action by computing the expected value of the search results of all possible piece types in this program.

Canonical exhaustive search algorithm has a problem when comparing the expected values with precise scores of moving actions. The algorithm selects the flipping action with the highest expected value. Thus, the program makes unreasonable moves such as flipping a piece instead of rescuing a piece in danger when the score of the selected flipping action is higher than moving actions.

```
function exhaustive($position$,$depth\_limit$,$\alpha$,$\beta$)
$c$ = evaluate($position$) {* call evaluation function to compute the score of the current position *}
**if** (no legal move and no unknown pieces) or ($depth\_limit = 0$) **then**
  **return** $c$
s = alpha_beta($position$,$depth\_limit$,$\alpha$,$\beta$) {* normal alpha-beta pruning algorithm by considering only moving actions *}
**if** ($s \leq c$) and (there are unknown pieces) **then**
  **for all** $r$ in $possible\_flips\_places$ **do**
    $v = 0, n = 0$
    **for all** $t$ in possible_flip_types **do**
      $p$ = flip($position$, $r$, $t$) {* assign location r to be type t *}
      $v = v -$exhaustive($p$, $depth\_limit - 1$, $-\beta$, $-max\{s, \alpha\}$) $\times$ piece_num_of_type($t$)
      $n = n +$piece_num_of_type($t$)
    **if** $v/n > s$ **then**
      $s = v/n$ {* v/n is the expected score by flipping r *}
    **if** $s \geq \beta$ **then**
      **return** $s$
**return** $s$
```

Algorithm 3.2: The exhaustive search algorithm

Quiescence search (Yen *et al.*, 2004) technique is very popular in Western chess and Chinese chess programs. However, in Chinese dark chess, there is no major improvement of using this technique because the only forcing move, capturing, does not last for many plies in most cases due to the low mobility of pieces.

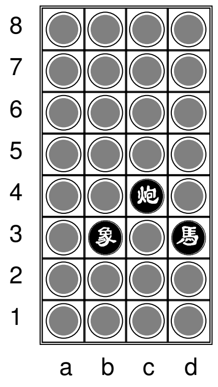

Figure 4: An example of capturing pieces after a flipping action.

Even if we perform exhaustive search, we still need a probabilistic model to evaluate the value of a position after a flipping action. An acceptable evaluation scheme is to compute the average value among the branches of all different outcomes after flipping. This method estimates possible scores of the positions after a flipping action. However, the method is not suitable for risk avoidance. For example, it is possible for the algorithm to reveal a king adjacent to the opponent pawn. It is also hard for the algorithm to capture pieces after taking a flipping action. The following are examples of intelligent behaviors that often taken by human experts. In Figure 4, The red side can reveal the unknown pieces adjacent to the three black pieces to capture them. If we get a red guard at $c3$, which occurs with very low probability, all of the three black pieces are in danger. Our algorithm reveals $c5$ because the probability of the piece at $c5$ can capture the cannon at $c4$ is high.

4. REFINEMENTS IN DEALING WITH FLIPPING ACTIONS

In this section, we introduce refined methods in making flipping actions to reduce the branching factor.

4.1 Refined Methods

Though the exhaustive search algorithm can handle flipping actions reasonably, its weakness is that it takes too much time computing flipping actions. As a result, the searching depth is then reduced. To avoid the exponentially exploration in branching factor, refinements are needed.

The first refinement modifies Algorithm 3.2 to expand the nodes of flipping actions only at the root node of a search tree, as shown in Figure 5. When the depth is less than initial depth, the algorithm only expands the nodes of moving actions The method is called *initial-depth flipping*. The program using this strategy is called VAR1.

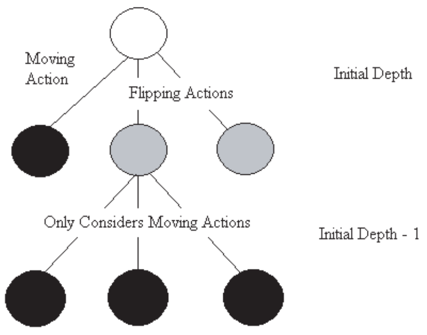

Figure 5: An example of initial-depth flipping.

Recall that when we visit a node with no moving action in BASIC program, we perform a no-move cut. This strategy may cause a horizon effect. For example, in Figure 6, black is ready to move. The correct move of black is to capture the red knight at $b7$ first, and then capture the red king at $d4$. If black captures the red knight at $b7$ first, it makes red no legal move. A no-move cut is applied and the resulting score is only an advantage of a knight.

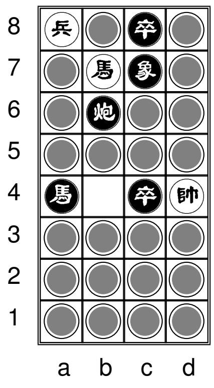

Figure 6: An example of horizon effect due to the no-move cut strategy.

The second refinement strategy reduces the branching factor when handling a flipping action. In an exhaustive search algorithm, we evaluate a flipping action of an unknown piece by expanding all possible piece types. Here we try to reduce the number of candidate piece types. This method is called *k-type expansion*.

We now give two possible variations of this kind of method: 1) *fixed piece order strategy* and 2) *random piece order strategy*. The first strategy uses a predefined candidate order of piece types to expand nodes. It also reduces the branching factor of exhaustive searching algorithm.

In fixed piece order strategy, the order of pieces is critical in the strength of the algorithm. However, due to the limit of fixed order, it is hard to derive an optimal order that suitable for general cases. As a result, a random piece order strategy is used as an improvement.

Although the value k has 14 choices, it still needs to select carefully. If k is too close to 14, we cannot decrease the order of branching factor. If k is too small, that means the program assume only a very small set of pieces can be revealed. It may let the program to make dangerous decision. As a result, a reasonable value of k is 6. We also used a small experiment to verify the value k. In our experiment that the version of $k = 6$ against other versions, the version of $k = 6$ has better performance.

5. EXPERIMENTAL RESULTS

5.1 Test Data

Here we show our experiments to verify the playing strength of the five programs discussed above. The baseline program is called BASIC. The exhaustive searching program is called ENHANCE. The first refined method with initial-depth flipping is called VAR1. The second refined method, k-type expansion with fixed piece order strategy, is called VAR2. The third refined method, k-type expansion with random piece order strategy, is called VAR3.

For fairness concern, the evaluation function of all programs are equal. Each player has a time limit of 30 seconds for each move. The differences of these programs are their search depth and their strategies in handling flipping actions. Since *initiative*, which is known as the advantage of playing first, exists in many games, our experiments also take initiative into account. A program needs to play against other four programs. There are totally $C(5, 2) = 10$ pairs. In each pair, a program needs to play both 100 games as the first player and 100 games as the second player.

In order to retrieve an average value of branching factor, we collected $1,561$ positions of computer versus computer. The branching factor without flipping actions is 7.76. If we also consider flipping actions, the average branching factor becomes 32.30.

5.2 Measurements

We incorporate the idea of Global strength to measure the performance of each program with the following definitions:

- $GS_f(p)$: the global strength of p among all games when p plays first.
- $GS_s(p)$: the global strength of p among all games when the opponent of p plays first.
- $win_{p,q}(p)$: the number of games won by p when p plays first with q.
- $win_{q,p}(p)$: the number of games won by p when q plays first with p.
- $draw_{p,q}$: the number of games drawn when p plays first with q.

$$GS_f(p) = \sum_{q \ni P, q \neq p} \frac{(2 \times win_{p,q}(p) + draw_{p,q})}{2 \times n_{p,q}}$$

$$GS_s(p) = \sum_{q \ni P, q \neq p} \frac{(2 \times win_{q,p}(p) + draw_{q,p})}{2 \times n_{q,p}}$$

The variable $n_{p,q}$ represents the total number of games that p plays first with q. P is the set of players.

Define tournament score of p among all games:

$$TS(p,q) = (GS_{p,q}(p) + GS_{q,p}(p))/2$$

5.3 Results of Each Versions of Programs

The results of our experiments is shown in Table 3. Any two of the programs have played 200 games. In each pair, we use (w,l,d) to show their result, where w means that the first player wins, l means that the first player loses, and d means the game is a draw. The programs in the leftmost column play first. For example, in the third column of the second row, the result of BASIC plays first with ENHANCE is $(9,34,57)$, which means BASIC wins 9 games, loses 34 games, and 57 games are draws. The empty grids are programs play against themselves. They are not considered in our experiment.

Table 3: The pairwise comparison results of all methods. The programs in the leftmost column play first.

	BASIC	ENHANCE	VAR1	VAR2	VAR3	$GS_f(p)$	$GS_s(p)$	$TS(p)$
BASIC		(9, 34, 57)	(2, 74, 24)	(0, 52, 48)	(1, 53, 46)	0.2488	0.2900	0.2694
ENHANCE	(47,7,46)		(12, 41, 47)	(5, 17, 78)	(15, 25, 60)	0.4863	0.4988	0.4925
VAR1	(45, 0, 55)	(32, 29, 39)		(35, 14, 51)	(41, 13, 46)	0.6213	0.6750	0.6481
VAR2	(48, 1, 51)	(18, 18, 64)	(22, 30, 48)		(4, 5, 91)	0.5475	0.5525	0.5500
VAR3	(36, 0, 64)	(26, 3, 71)	(7, 38, 55)	(3, 2, 95)		0.5363	0.5438	0.5400

The result shows that ENHANCE is stronger than BASIC and all refined algorithms are stronger than ENHANCE. Among the three refined algorithm, VAR1 gets the highest tournament score. This means the initial-depth flipping is the most effective strategy comparing with other methods. In the following, we discuss about some issues about the results and the characteristics of these algorithms.

The winning rate of the exhaustive search program is much higher than the baseline program. This shows that the search depth and the strategy of handling flipping actions are both important.

In Chinese dark chess, a player having a great advantage sometimes needs to win the game with many moves because he has to capture all pieces of the opponent. However, in a middle game position, it is not necessary to search very deep to obtain a good move because the mobility of pieces except cannons in Chinese dark chess is much less than pieces in Chinese chess or Western chess.

In our experiment, we conclude that $RS_f(p) < RS_s(p)$ for all programs. This means initiative is a disadvantage in Chinese dark chess. The intuition is that the first player can only flip a piece. After the first player's move, the second player has a target that he or she can attack and has more possibility to get some benefit.

A majority of the games in our experiment end in a draw. There are two possibles. One is that the endgame position is theoretical draw. The other is that the player in advantage does not discover the path to win the game. When a program is in a position that is considered to be a winning position, it may needs deeper search or better knowledge to obtain a correct move. For example, Figure 7 shows a winning position that black has a great advantage. The only thing black needs to do is to capture the red pawn. It takes at least fourteen plies for the program to find such a winning strategy. Because the pieces except cannons can only move one grid each time, we can realize that endgames in Chinese dark chess is similar to endgames with many pawns in Western chess or Chinese chess.

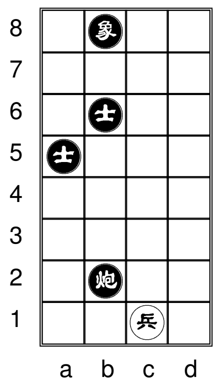

Figure 7: An example of a winning position.

According to the results, the flipping action is a strange action that is important but do not need to be consider all the time during search. To flip an optimal place is hard, but it is acceptable to flip a safe place. Besides flipping actions, the depth of searching and knowledge are also critical for a strong program.

5.4 Results of Humans versus Computers

In our experiment, we selected five persons to play against each of our programs. The rank of the five persons is shown in Table 4. Each person played 10 games with each of the three refined program versions: VAR1, VAR2, VAR3. In the 10 games with one program, the human plays first in 5 of them, the computer plays first in the other 5 games. The results is shown in Table 5. The (w, l, d) notation in the table is the same as in Table 3.

Table 4: The rank of the five persons who against our programs.

Rank	ID of person
1	4
2	1, 2, 5
3	3

This experiment also follows our observation of disadvantage in initiative. VAR1 is still the best version among all refined versions, whose tournament score is 0.2500.

Table 5: The pairwise comparison results of all methods. The programs in the leftmost column play first.

	Computer First	Human First	$GS_f(p)$	$GS_s(p)$	$TS(p)$
VAR1	(4, 17, 4)	(17, 5, 3)	0.2400	0.2600	0.2500
VAR2	(3, 21, 1)	(19, 2, 4)	0.1400	0.1600	0.1500
VAR3	(2, 19, 4)	(18, 3, 4)	0.1600	0.2000	0.2300

6. CONCLUSIONS

Chinese dark chess is a popular and easy-to-learn game in Asia. The characteristic of its probabilistic behavior in flipping makes it different from Chinese chess and Western chess. The opening game problem lies in how to flip pieces reasonably and effectively. In the middle game, we need a better evaluation function. The endgame is an novel problem that some techniques used in Chinese chess also work, but new techniques are needed for some special endgames, such as endgames with unknown pieces.

Game-tree searching techniques are also discussed in this paper. We have implemented $\alpha\beta$ pruning algorithm to handle moving actions. In flipping handling, we have proposed an exhaustive search algorithm and three refinements. The result shows that the Chinese dark chess problem is a trade off between the search depth and capability of making good flipping actions.

This paper is the first attempt to describe this interesting and popular game in English. We hope it can promote future researches for Chinese dark chess.

7. ACKNOWLEDGEMENT

This research is supported in part by National Science Council (Taiwan) Grants 97-2221-E-001-011-MY3. Tsan-sheng Hsu thanks his students in Theory of Computer Games course at National Taiwan University for fruitful discussions.

8. REFERENCES

Chen, J. C. and Hsu, S. C. (2001). Construction of online query system of opening database in computer CHINESE CHESS. *The 11th Conference on Artificial Intelligence and Applications.*

Ciancarini, P. and Favini, G. (2007). A program to play KRIEGSPIEL. *ICGA Journal*, Vol. 30, No. 1, pp. 3–24.

Herik, H. J. van den, Uiterwijk, J. W. H. M., and Rijswijck, J. van (2002). Games solved: Now and in the future. *Artificial Intelligence*, Vol. 134, pp. 277–311.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo Planning. *In: ECML-06. Number 4212 in LNCS*, pp. 282–293, Springer.

Sakuta, M. and Iida, H. (2000). Solving Kriegspiel-like problems: examining efficient search methods. *Computers and Games*, Vol. LNCS 2063, pp. 55–73.

Schaeffer, J. (2005). Solving CHECKERS. *ICGA Journal*, Vol. 28, No. 1, p. 32. ISSN 1389–6911.

Thompson, K. (1996). 6-piece endgames. *ICCA Journal*, Vol. 19(4), pp. 215–226.

Wu, P. s., Liu, P. y., and Hsu, T. s. (2006). An external-memory retrograde analysis algorithm. *Computers and Games*, Vol. LNCS 3846, pp. 145–160.

Yen, S. j., Chen, J. c., Yang, T. n., and Hsu, S. c. (2004). Computer CHINESE CHESS. *ICGA Journal*, Vol. 27, No. 1, pp. 3–18. ISSN 1389–6911.

9. APPENDICES

A Selected Game Records

ENHANCE v.s. VAR1

```
*  3. d8(P) d8-c8     4. a6(p) d7(R)
*  5. a5(p) b8(P)     6. a4(r) c8-d8
*  7. b5(c) b8-c8     8. b7-b8 a8-b8
*  9. b5-b8 c7(G)    10. b8-a8 c8-b8
* 11. a5-b5 b8-b7    12. a6-a5 c7-c8
* 13. b4(P) c8-b8    14. c5(r) b4-b5
* 15. d3(m) b8-a8    16. c5-b5 c6(k)
* 17. d5(N) c4(G)    18. b2(R) b7-c7
* 19. c6-c5 c4-b4    20. d6(g) a8-b8
* 21. d6-d7 c7-c6    22. c5-d5 b4-b5
* 23. a5-a6 b5-b4    24. a4-a5 b8-c8
* 25. b6(p) c6-c5    26. d5-d6 c8-b8
* 27. d7-d8 b8-b7    28. a3(c) b7-b6
* 29. a1(M) a2(p)    30. a3-a1 b2-a2
* 31. a5-b5 b6-b5    32. a7(C) a2-a1
* 33. d8-c8 b5-b6    34. a6-a5 c5-c6
* 35. d6-d7 c6-c7    36. d7-d6 a7-a6
* 37. d6-d5 c7-c6    38. d4(P) d4-d5
* 39. d3-d4 b6-b5    40. c2(P) c6-b6
* 41. c3(n) b4-c4    42. d1(K) c4-d4
* 43. d2-c2 b5-a5    44. b3(g) d1-d2
* 45. c3-c4 d4-c4    46. c2-c3 d2-d3
* 47. b3-b2 d3-c3    48. b1(N) b6-b7
* 49. c1(C) c4-b4    50. c8-d8 c1-d1
* 51. d8-c8 b1-c1    52. c8-c7 a6-a7
* 53. c7-c8 c3-c2    54. c8-b8 c2-b2
* 55. b8-a8 a5-a6    56. a8-b8 a7-a8
* 57. b8-a8 b7-a7    58. a8-b8 a6-b6
* 59. b8-a8 b6-b7    60. a8-a7 b7-a7
* VAR1 wins
```