# An Analysis for Strength Improvement of an MCTS-Based Program Playing Chinese Dark Chess

Chu-Hsuan Hsueh[1], I-Chen Wu[1,*], Wen-Jie Tseng[1], Shi-Jim Yen[2], and Jr-Chang Chen[3]

[1]Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
{hsuehch,icwu,wenjie}@aigames.nctu.edu.tw

[2]Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan
sjyen@mail.ndhu.edu.tw

[3]Department of Applied Mathematics, Chung Yuan Christian University, Taoyuan, Taiwan
jcchen@cycu.edu.tw

**Abstract.** Monte-Carlo tree search (MCTS) has been successfully applied to many games recently. Since then, many techniques are used to improve the strength of MCTS-based programs. This paper investigates four recent techniques: early playout terminations, implicit minimax backups, quality-based rewards and progressive bias. The strength improvements are analyzed by incorporating the techniques into an MCTS-based program, named DARKKNIGHT, for Chinese Dark Chess. Experimental results showed that the win rates against the original DARKKNIGHT were 60.75%, 71.85%, 59.00%, and 82.10%, respectively for incorporating the four techniques. The results indicated that the improvement by progressive bias was most significant. By incorporating all together, a better win rate of 84.75% was obtained.

**Keywords:** Monte-Carlo tree search, Chinese dark chess, early playout terminations, implicit minimax backups, quality-based rewards, progressive bias.

## 1 Introduction

Monte-Carlo tree search (MCTS) [5] has been successfully applied to many games, such as Go [16][18][19][20], General Game Playing (GGP) [3], Backgammon [43] and Phantom-Go [4]. Since then, many techniques were proposed to improve the playing strength of MCTS-based programs. In this paper, four recent techniques are studied and analyzed.

1. *Early playout terminations* [2][17][29][31][32][33][44][45]: Terminate a playout much earlier when it is very likely to win, lose, or draw.
2. *Implicit minimax backups* [29]: Guide MCTS selections by using minimax scores from heuristic evaluations of positions together with the original simulated win rates.
3. *Quality-based rewards* [35]: Use simulation length and terminal state quality to adjust the rewards returning from simulations.

* Corresponding author

4. *Progressive bias* [8][9][10][23][24][34]: Use heuristic scores of moves to differentiate newly generated nodes in selections.

The techniques are analyzed through the game Chinese dark chess (CDC), a partially observable (PO) game widely played in Chinese community. This paper incorporates the above four techniques into an MCTS-based CDC program, named DARKKNIGHT which won two CDC tournaments [42][48] including the 17th Computer Olympiad, and analyzes how well these techniques can improve the game-playing strength.

The experimental results showed that progressive bias improved the most, which reached a win rate of 82.10% against the original DARKKNIGHT which serves as the baseline program. The improvements reached a win rate of 60.75% for early playout terminations, 71.85% for implicit minimax backups, and 57.50% and 59.00% respectively when using simulation length and terminal state quality for quality-based rewards. By incorporating all together, DARKKNIGHT was improved with a win rate of 84.75%.

This paper is extended from a preliminary version [22] by proposing some modification to implicit minimax backups, adding progressive bias, and investigating more on combinations of the above techniques. The structure of this paper is organized as follows. Section 2 reviews the MCTS algorithm, the game CDC and the previous work for CDC game-playing programs. Section 3 presents the above four techniques, including the reviews of these techniques and the incorporations into DARKKNIGHT. Section 4 shows the experimental results. Finally, Section 5 makes concluding remarks.

## 2 Background

In this section, MCTS algorithm is briefly reviewed in Subsection 2.1. Then the game CDC and the previous work for CDC are reviewed in Subsections 2.2 and 2.3.

### 2.1 Monte-Carlo Tree Search (MCTS)

Monte-Carlo Tree Search (MCTS) [5] is a best-first search algorithm on top of a search tree, named *UCT* [28], using Monte-Carlo simulations as state evaluations. There are four phases in MCTS:

1. Selection. A path is traversed from the root to one of the leaves following a selection policy. A popular selection policy is to select the children with the maximum values of the *upper confidence bounds* (*UCB*) function [1]:

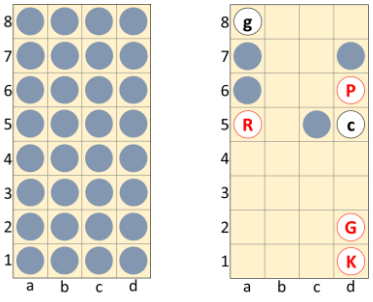
$$UCB_i = Q_i + C\sqrt{\frac{\ln N}{n_i}} \quad (1)$$

   where $Q_i$ is an estimator for the value of node $i$, $n_i$ the visit count of node $i$, $N$ the visit count of the parent of node $i$ and $C$ a constant representing the weight of exploration. Commonly, $Q_i$ is a win rate, $w_i/n_i$, or a mean value for rewards, where $w_i$ is the win count of node $i$.
2. Expansion. One or more children are expanded from the selected leaf.

3. Playout. Following a playout policy, the playout is played from the position corresponding to the selected leaf to some terminal state of the game.
4. Backpropagation. The result of the playout, e.g., rewards, is updated from terminal state back to all its ancestors in the selected path during selection.

### 2.2 Chinese Dark Chess

Chinese dark chess (CDC) [12][47], widely played in Chinese community, is a two-player zero-sum non-deterministic game played on a 4×8 square board as illustrated in Fig. 1 (a) and (b). It is also a partially observable (PO) game with symmetric hidden information. The two players, *Red* and *Black*, respectively own identical sets of sixteen pieces with different colors. The set of pieces is the same as those in Chinese chess. The piece types are shown in Fig. 1 (c). Each piece has two faces, one showing the piece type and the other showing the piece cover which is identical for all pieces. When the piece type faces up, the type is revealed and known by both players. When it faces down, it is unrevealed and unknown.




(a) (b)

| *Piece types* | | | # | *Rank* |
|---|---|---|---|---|
| Name | Red | Black | *pieces* | |
| King | K | k | 1 | 1 |
| Guard | G | g | 2 | 2 |
| Minister | M | m | 2 | 3 |
| Rook | R | r | 2 | 4 |
| Knight | N | n | 2 | 5 |
| Cannon | C | c | 2 | 6 |
| Pawn | P | p | 5 | 7 |

(c)

**Fig. 1.** (a) Initial board, (b) a board example and (c) piece information

Two kinds of actions are allowed in CDC: *flipping* and *moving*. Each flipping action flips a piece, namely making the piece type revealed. Each moving action is to move a revealed piece by the player owning the piece. All pieces can be moved to empty neighboring squares (with one square up, down, left or right). Pieces except cannons can be moved to neighboring squares with capturing the opponent pieces that have equal or lower ranks shown in Fig. 1 (c) with the following exceptions: Pawns can capture the opponent king, but not the other way around. A piece that can be captured by another piece $p$ as above is said to be *capturable* by $p$ in this paper. Cannons have a different capturing rule called *one-step-jump* [12][47] in which any opponent pieces in the same row/column can be captured by jumping over exactly one piece in that row/column. For example, in Fig. 1 (b), the black cannon (c) at d5 can capture the red rook (R) at a5 or the red king (K) at d1.

Initially, all 32 pieces are unrevealed and placed randomly, as in Fig. 1 (a). So, the probability distribution of unrevealed pieces are all equal. The first player flips one of the 32 pieces and then owns the set of pieces of the revealed color, while the second

player owns the other set. A game is played by the two players in turn. As flipping actions performed, the number of unrevealed pieces decreases. When there remains at most one unrevealed piece type, the game becomes deterministic.

Finally, a player wins by capturing all of the opponent pieces or making the opponent have no legal moves. The game also ends with a draw when both players play without any capturing and flipping within 40 plies, or when the same position appears three times.

### 2.3 Previous Work for CDC

In [12], the state space complexity of the game CDC was estimated to be $10^{37}$ between those of Draughts and chess, while the game tree complexity was estimated to be $10^{135}$ between those of chess and Chinese chess without considering *chance nodes* [37]. In [47], Yen *et al*. estimated the game tree complexity with chance nodes to be $10^{207}$ between those of Chinese chess and Shogi.

In the early stage, most CDC programs [12][41] were developed using *alpha-beta search*. Chen *et al*. published a paper [12] about alpha-beta search in their CDC program. Some more research work for CDC includes opening books [11], endgame databases [13][14][15][38], solving smaller CDC [6][7] and game variations [25].

Recently, MCTS has been incorporated into CDC game-playing programs [26][27][47][48]. In [47], they developed an MCTS-based CDC program, named DIABLO, which used *non-deterministic nodes*, so-called in their article, to contain all *inner nodes* for all the possible outcomes of the flipping actions. Since non-deterministic nodes are conceptually similar to chance nodes, we simply use chance nodes in the rest of this paper. Thus, inner nodes are simply children of chance nodes. The win counts and visit counts of chance nodes were the sums from all their children. In the selection phase, roulette wheel selection was applied to chance nodes to select one of children according to the probability distribution of unrevealed pieces.

**Table 1.** The piece weights

| Piece type | K/k | G/g | M/m | R/r | N/n | C/c | P/p |
|---|---|---|---|---|---|---|---|
| Weights in [47] | 5500 | 5000 | 2500 | 1000 | 800 | 3000 | 800 |
| Weights in this paper | 55 | 50 | 25 | 10 | 8 | 30 | 8 |

In the playout phase, the program DIABLO [47] used piece weights, as listed in Table 1, when capturing pieces. Another MCTS-based CDC program, named DARKKNIGHT [48], was developed independently by some of the authors of this paper, but it used almost the same weights as in [47], except for reducing them by a factor of 100. The program DIABLO won some tournaments [30][39][46][49] in 2011 and 2012, while DARKKNIGHT won two CDC tournaments [42][48] including the 17th Computer Olympiad in 2013. DARKKNIGHT serves as the baseline program in this paper.

## 3 Incorporated Techniques

In this section, we discuss the four incorporated techniques, including early playout terminations in Subsection 3.1, implicit minimax backups in Subsection 3.2, quality-based rewards in Subsection 3.3, and progressive bias in Subsection 3.4.

### 3.1 Early Playout Terminations

For CDC and many other games, it is important to terminate playouts earlier for two reasons, speedup and/or accuracy of simulated results. The first reason is obvious. The second is that it is more accurate to terminate playouts at the time when they are very likely to win, lose, or draw. For example, it is normally a draw in a situation, `Gmm`, where Red has only one guard and Black has two ministers. In the case of keeping playing in the playout, Black may lose accidentally, while it is supposed to be a draw since `G` cannot capture either `m`. Thus, an early termination is actually more accurate than a continuous playout. Another example is `KGmm`. No matter where the pieces are placed and whether the pieces are revealed or not, Red always wins the game, since both `K` and `G` can capture `m`, but not *vice versa*. However, it may result in draws in playouts since Red may not be able to go to capture black pieces tactically during playouts.

In the rest of this subsection, we first review the previous work about early playout terminations and then describes our work to incorporate it into the baseline program.

#### 3.1.1 Previous Work

In [17], *Early Cutoffs* was proposed to collect information online, and terminate playouts when the game obviously favors some player from the collected information. They used it to save the time for more simulations for GGP.

Some other researches were to return results after making a fixed number of moves in playouts. In [31][32][33], for Amazons, Breakthrough and Havannah, they used evaluation functions to determine the winners who the evaluated scores favor. In [2], they did a shallow minimax search with fixed depths and then returned the evaluated scores directly. Their method, called *MCTS with Informed Cutoffs*, was tested for Othello, Breakthrough and Catch the Lion.

In [44][45], their methods for Lines of Action (LOA) were to check the scores by evaluation function every three moves, and then return the results in the following cases. If the scores exceeded some threshold, the results were wins. On the contrary, if the scores were below some threshold, the results were losses. Checking evaluation functions every three moves was for the sake of performance.

#### 3.1.2 Our Work

Playouts are terminated earlier when detecting a *likely outcome* which is *win*, *loss*, or *draw*, from Red's perspective. The detection rules are based on a material combination

[14][38], namely a set of remaining pieces on the board in CDC. For CDC, the total number of legal material combinations is 8,503,055 (= $(2^1 3^5 6^1)^2 - 1$). In our work, we analyzed all the combinations according to some heuristic rules. From the analysis, 750,174 are set to win, 750,174 loss, 108,136 draw, and the rest are unknown. For example, the likely outcomes for `KGmm`, `KGk` and `KGggm` are set to win, and those for `Gmm`, `CCccp` and `KPggm` are draw. Note that we call it a *likely* outcome since the outcome may not be always true in a few extreme cases as illustrated by the following example. For `KGggm`, Black can try to exchange one `g` with `G`, though it is hard to do so. If Black successfully exchanges `g` with `G` without losing `m`, then the game becomes a draw.

In playouts, if the likely outcome for a material combination is one of win, loss and draw, the playouts end immediately and return the outcomes. The overhead for the detection is little, since we can simply lookup a table to check this. So, the detection is done for every move in playouts.

### 3.2 Implicit Minimax Backups

This subsection reviews the previous work for implicit minimax backups and then describes our work to incorporate it into the baseline program.

#### 3.2.1 Previous Work

In [29], the researchers proposed implicit minimax backups which incorporated into MCTS the minimax scores based on *heuristic evaluations* of positions. The minimax scores were used together with the estimator $Q_i$ to guide the selections of MCTS. Namely, $Q_i$ in Formula (1) was replaced by the following:

$$(1 - \alpha)Q_i + \alpha m_i \qquad (2)$$

where $m_i$ is the minimax score of node $i$ and $\alpha$ the weight of the minimax score.

In each leaf of UCT, a heuristic evaluation function was used to calculate the score of the position corresponding to the leaf, scaled to [-1, 1] through a sigmoid function. Then, the scaled score was also backed up as classical minimax search. For consistency, $Q_i$ was ranged in [-1, 1], too.

They improved the playing strength of the three games, Kalah, Breakthrough and LOA, and obtained win rates of around 60% to 70% in different settings of the self-play games. Their work also showed that even simple evaluation functions can lead to better performance. However, they found that this technique did not work on every game. They ran experiments on Chinese checkers and the card game Heart but obtained no significant improvement.

#### 3.2.2 Our Work

For CDC, we use as the heuristic evaluation function the weighted material sum, namely the difference of the total piece weights between the player and the opponent. The piece weights are listed in Table 1. The heuristic evaluations are then scaled to [-

1, 1] by a sigmoid function.

One issue to discuss is the minimax scores for chance nodes, which were not mentioned in [29]. An intuitive way is to use the probability distribution for unrevealed pieces, when calculating the expected values for chance nodes. For example, suppose to have four unrevealed pieces, two Ps, one k and one m (as in Fig. 1 (b)). The probability for P is 1/2, while those for the other two are 1/4. However, a problem occurs when unrevealed pieces are not in the UCT yet. For example, if k has not been flipped yet, the corresponding heuristic evaluation is unknown, making it hard to get the minimax scores for chance nodes.

In order to solve this problem, we use the ratios of visit counts as the probabilities for all children of chance nodes. For the above example, assume to flip P 3 times and m once, but none for k. Then, we use 3/4 for P and 1/4 for m.

Another issue is the quality of the heuristic evaluations including the resulting minimax scores. Although simple evaluation functions were shown leading to better performance, the scores may not reflect the values of positions well, compared to simulated win rates especially when the number of simulations is sufficiently large. To deal with this problem, we propose the following suggestion to dynamically decrease $\alpha$ in Formula (2) as numbers of visits increase:

$$\alpha = (1 - c_c)\sqrt{\frac{c_s}{c_s + c_N \times N}} + c_c \tag{3}$$

where $N$ is the visit count of the parent as in Formula (1), and $c_c$, $c_s$ and $c_N$ are all coefficients which decide where and how fast $\alpha$ converges. These coefficients make the adjustment of $\alpha$ more flexible.

### 3.3 Quality-Based Rewards

This subsection reviews the previous work for quality-based rewards and then describes our work to incorporate it into the baseline program.

#### 3.3.1 Previous Work

In [35], the researchers proposed new measurements to adjust the rewards obtained from simulations. Simulation length and terminal state quality were used as *quality assessments* of simulations to adjust the rewards from wins and losses.

In [35], simulation length, a domain independent quality assessment, was defined as the length of simulated game from the root to the terminal state in the simulation. Intuitively, in an advantageous position, the shorter the simulation length is, the better.

This technique maintained online sample mean and sample standard deviation of simulation length, denoted by $\mu_L^p$ and $\sigma_L^p$, for player $p$ when $p$ wins. With these statistical values, the reward $R_L$ for simulation length was calculated as follows:

$$R_L = R + sign(R) \times a \times b(\lambda_L) \tag{4}$$

where $R$ is the reward (e.g., 1 for a win and -1 for a loss), $a$ the influence for this quality, $b(\lambda)$ a sigmoid function as shown in Formula (5) which scales the values to

[-1, 1], $\lambda_L$ a normalized value in Formula (6), $k$ a constant to be tuned and $l$ is the length of that simulation.

$$b(\lambda) = -1 + \frac{2}{1 + e^{-k\lambda}} \tag{5}$$

$$\lambda_L = \frac{\mu_L^p - l}{\sigma_L^p} \tag{6}$$

For terminal state quality, a function describing quality of terminal states was needed. For example, for Breakthrough, the piece difference between the winning and losing players was used to measure the terminal state quality, which was scaled to [0, 1]. Let $\mu_T^p$ and $\sigma_T^p$ respectively denote sample mean and sample standard deviation of terminal state quality for player $p$. Similarly, the rewards $R_T$ for terminal state quality was calculated in a way similar to Formula (4) as follows.

$$R_T = R + sign(R) \times a \times b(\lambda_T) \tag{7}$$

where $\lambda_T$ is a normalized value in Formula (8), and $t$ is the terminal state quality.

$$\lambda_T = \frac{t - \mu_T^p}{\sigma_T^p} \tag{8}$$

In Formula (4) and (7), $a$ is a constant or is calculated according to the data accumulated online. Their experimental results showed not much difference.

### 3.3.2 Our Work

We also incorporate into DARKKNIGHT the above two quality assessments, simulation length and terminal state quality, to adjust the rewards from wins and losses. For draws, the simulations are not sampled. The two quality assessments are measured when simulations end. Without early playout terminations, MCTS simulations end when one player wins. With early playout terminations, MCTS simulations also end when one obtains a likely outcome with win or loss.

At terminal states, simulation length is simply the same as the one described in [35], and terminal state quality is obtained in the following way. First, simply count the remaining pieces of the winner. Then, incorporate domain knowledge like piece weights. Namely, we use the following formula:

$$\sum_{i \in S} (1 + c_w \times \omega_i) = |S| + c_w \sum_{i \in S} \omega_i \tag{9}$$

where $S$ is the set of remaining pieces of the winner, $|S|$ the size of $S$, $c_w$ a coefficient, and $\omega_i$ the weight of piece $i$ as in Table 1. The larger the coefficient $c_w$ is, the higher the influence of the piece weights. All values of terminal state quality are scaled to [-1, 1] according to a sigmoid function. Note that we do not consider the remaining pieces of the loser, if any.

## 3.4 Progressive Bias

This subsection reviews the previous work on progressive bias and then describes our work to incorporate it into the baseline program.

### 3.4.1 Previous Work

Without any knowledge, MCTS will still converge to the best move in a sufficient number of simulations eventually [28]. However, with some more knowledge, MCTS can use it to differentiate the newly generated nodes and then guide the search to converge more efficiently. *Progressive bias* [8][9][10][23][24][34] is one of such techniques, proposed to guide selections based on given knowledge. The researchers incorporated knowledge by adding additional heuristics for the games Go and Othello. In [8][10], they proposed to extend $UCB_i$ (Formula (1)) to the following:

$$Q_i + C\sqrt{\frac{\ln N}{n_i}} + c_p \times \frac{H_i}{n_i} \tag{10}$$

where $H_i$ is the heuristic score for the move corresponding to node $i$ and $c_p$ a coefficient representing the influence of the heuristic. Nodes with higher $H_i$ will be selected earlier. Thus, $H_i$ is move evaluation that affects move selection, while the heuristic in Subsection 3.2 (for implicit minimax backups) is position evaluation that affects the estimated values of search trees.

The extended term (the third) is the heuristics used to bias the search to favor selecting nodes with higher $H_i$. It gets smaller as $n_i$ grows. That is, the heuristics weigh less for larger $n_i$. This implies to trust more for the win rates obtained by MCTS when the nodes are visited more times. In [8], the experimental results showed increased win rates of 25.1%, 21.5%, and 4.8% respectively in 9×9, 13×13, and 19×19 Go. This technique, with different forms from the above formula, was also successfully applied to Go [9][23][24] and Othello [34]. In [36], the researchers proposed *Predictor+UCB* (*PUCB*) which also extended $UCB_i$ to introduce biases over moves. The regret bound for PUCB was also studied. Other than the work mentioned above, some researchers [18][19] proposed to initialize win rates and visit counts by some default values based on prior knowledge.

### 3.4.2 Our Work

This paper proposes some heuristics for obtaining $H_i$ for CDC in DARKKNIGHT and then incorporates the heuristics into progressive bias by modifying Formula (10) as follows:

$$Q_i + C\sqrt{\frac{\ln N}{n_i}} + c_p \times \frac{H_i}{count_i} \tag{11}$$

More specifically, $n_i$ in the third term of Formula (10) is changed to a general $count_i$. In practice, $count_i$ can be set to many functions, such as $n_i$, $\sqrt{n_i}$, $\ln(n_i + 1)$, $l_i$, $\sqrt{l_i}$, and $\ln(l_i + 1)$, where $l_i$ is the loss count of node $i$, which is the summation of the number of losses and half the number of draws for CDC in this paper. In [24][34], their formula included the heuristics similar to the one with square root, and in [9] theirs similar to the one with logarithm. However, all the past work [8][9][10][23][24][34] considered $n_i$, not $l_i$.

In the rest of this subsection, we describe three evaluation methods for heuristic

scores $H_i$ based on CDC domain-specific knowledge. The first evaluation calculates $H_i$ as follows. For a capture move that captures a piece $p$, add $\omega_p$ to $H_i$, where $\omega_p$ is the weight of $p$ as listed in Table 1. For an escape move that avoid a piece $p$ from being captured, add $\omega_p$ to $H_i$. Finally, for a suicide move that has a safe piece $p$ be moved to an unsafe square (to be captured by the opponent), subtract $\omega_p$ from $H_i$.

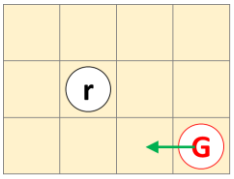



**Fig. 2.** An example of a curb move

The second evaluation calculates $H_i$ as above, but add one more heuristic for curb moves. A *curb move* is to move a piece $p$ to a diagonal square of an opponent piece $q$ which is capturable by $p$, such that the opponent piece has less mobility. As shown in Fig. 2, moving `G` left curbs `r` such that `r` cannot move right and down (otherwise, `r` will be captured). This feature often occurs in CDC games and is important especially in endgames where the winning players usually use continuous curb moves to force a win. For a curb move, add $c_{curb} \times \omega_q$ to $H_i$, where $c_{curb}$ is a coefficient normally less than one since $q$ has not been really captured yet. In the case that a move curbs two (or more) pieces at the same time, just consider the one with the highest weight. Note that if a curb move is also a suicide move, it is viewed as a suicide move only without extra bonus from curb.

The third evaluation calculates $H_i$ as the second, but add one more heuristic for flipping. For flipping actions, we consider the following four cases. In the first two cases, the flipped piece belongs to the player to move. In the first case, the flipped piece $p$ can be captured immediately by some opponent piece. In this case, simply subtract $P(p) \times \omega_p$ from $H_i$, where $P(p)$ is the probability of flipping $p$. For example, for a flipped piece, say `M`, if one opponent `k` or `g` is neighboring to it, then `M` will be captured immediately. Hence, $P(M) \times \omega_M$ is subtracted from $H_i$. Note that we disregard whether $p$ is protected since it is hard to have an efficient and accurate method to identify it. In the second case, the flipped piece $p$ can capture some opponent piece $q$ in the next move. For example, `M` for $p$ and `r` for $q$. Similarly, we disregard whether $q$ can escape, since it is also hard to have an efficient and accurate method to calculate the escape. In this case, add $c_{flip} \times P(p) \times \omega_q$ to $H_i$, where $c_{flip}$ is a coefficient normally much less than one due to the uncertainty of the escape of $q$.

In the next two cases, the flipped piece belongs to the opponent. In the third case, the flipped piece $p$ can capture some piece $q$ of the player immediately. In this case, simply subtract $P(p) \times \omega_q$ from $H_i$, similar to the first case. For example, for a flipped piece, say `m`, if `R` is neighboring to it, then `m` can capture `R` immediately. In the fourth case, the flipped piece $p$ can be captured by some piece $q$ of the player. For example, `m` for $p$ and `G` for $q$. Similarly, we disregard whether $p$ can escape. In this

case, add $c_{flip} \times P(p) \times \omega_p$ to $H_i$, where $c_{flip}$ is used for the same reason as that in the second case.

For the above, if there are more than one captured pieces, simply choose the one with the highest weight with a reason similar to that for curb moves.

## 4 Experiments

As mentioned in Subsection 2.2, the number of chance nodes decreases as flipping actions performed. Fig. 3 shows some statistical data for CDC games in average, including the numbers of legal moves, chance nodes, and revealed pieces on a board. The statistical data were collected from 1 million self-play games by the original DARKKNIGHT with 10,000 simulations per move. From Fig. 3, all pieces are revealed around the 100th plies, and the most number of revealed pieces on a board is around 14 at the 35th plies.

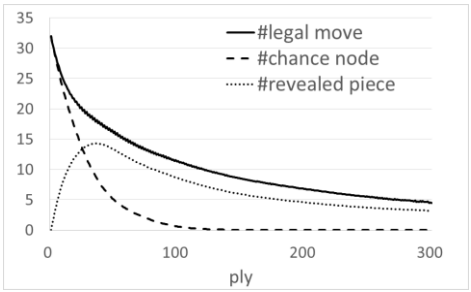



**Fig. 3.** Statistical data for CDC

In our experiments, each modified version played 1,000 games against the baseline, the original DARKKNIGHT, unless explicitly specified. Among the 1,000 games, one played 500 games as the first and the other 500 as the second. For each game, one scored 1 point for a win, 0 for a loss and 0.5 for a draw. For a given version, the average score of 1,000 games was the win rate against the baseline. Also, we show 95% confidence interval for all win rates.

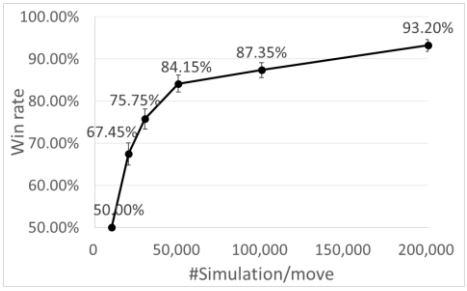



**Fig. 4.** Baseline with different numbers of simulations per move against that with 10,000

Initially, we performed a strength test for the baseline with different numbers of simulations per move against the one with 10,000, as shown in Fig. 4. In the rest of experiments, unless explicitly specified, we chose the one with 30,000, which was reasonable in the sense of both strength and computation time. Namely, it had a win rate of 75.75% (±2.39%), while it took about 3.2 minutes to run a game with one thread on machines equipped with Intel (R) Core(TM) i5-3570K CPU, 3.40GHz .

### 4.1 Experiments for Incorporating Individual Techniques

In this subsection, we incorporate techniques mentioned in Section 3 individually into the baseline to see how much they can improve.

#### 4.1.1 Early Playout Terminations

As mentioned for early playout termination (EPT) in Subsection 3.1, we used material combinations to detect whether a playout reaches a terminal state earlier with a likely outcome, one of win, loss or draw. The experimental result showed a significant improvement with a win rate of 60.75% (±2.76%) against the baseline program. The result indicated that the accuracy of the simulated results was indeed increased with the help of the likely outcomes returned from EPT. In addition, the program with EPT also ran faster at 34,730 simulations per second from the initial board, than the one without EPT, at 27,356. The speedup came from the reduction of nearly one third of the playout length. The program with EPT averagely played 101.7 moves in a playout (from the initial board) while the one without EPT played 148.4 moves. It is expected that the improvement is even greater during middle and end games.

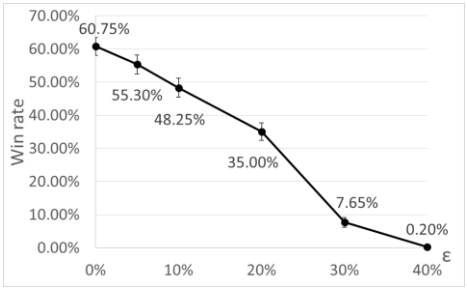



**Fig. 5.** Win rates for EPT with different $\varepsilon$

An experiment was performed to show how the simulation accuracy influences playing strength. In the experiment, instead of returning likely outcomes, wrong outcomes were returned with a probability of $\varepsilon$. More specifically in our experiments, likely wins return losses with a probability of $\varepsilon$, and *vice versa*. The results in Fig. 5 shows that the playing strength got worse as $\varepsilon$ increased and went down nearly to 0 when $\varepsilon$ is 40%.

#### 4.1.2 Implicit Minimax Backups

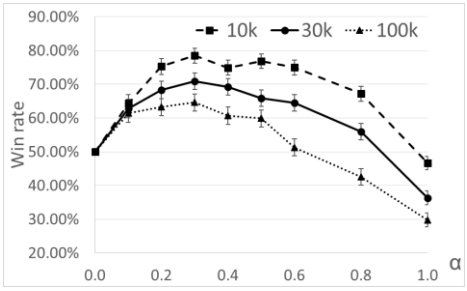
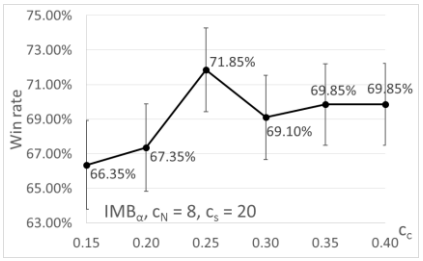



**Fig. 6.** Win rates for IMB with different $\alpha$ and different numbers of simulations per move

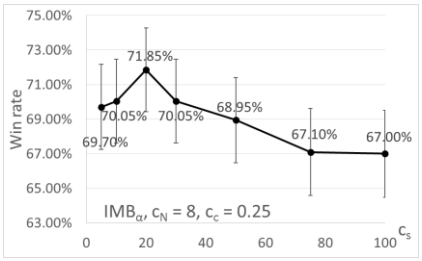
As mentioned for implicit minimax backups (IMB) in Subsection 3.2, we used minimax scores from heuristic evaluations to help guide the selections of MCTS more accurately. In our experiments, we firstly tested different weights of minimax score, $\alpha$ (fixed as constant), and different numbers of simulations per move. The experimental results are shown in Fig. 6 which includes three lines, respectively, for 10,000, 30,000 and 100,000 simulations per move. For fairness, the corresponding baselines also ran the same numbers of simulations.

Fig. 6 shows that the win rates are the highest when $\alpha = 0.3$ for the three lines, and 78.45% (±2.18%) for the one with 10,000 simulations per move, 70.90% (±2.42%) for 30,000 and 64.60% (±2.51%) for 100,000. The figure shows that IMB did significantly improve the playing strength. On one hand, in the case that $\alpha$ was too high, the win rates went down for the following reason. The heuristic evaluations weighted too much higher than online estimation which is usually more accurate than heuristic evaluations for a sufficiently large number of simulations. On the other hand, in the case that $\alpha$ was too low, the win rates also went down for the following reason. Less heuristic information was used to help guide the selections of MCTS accurately, since short-term tactical information provided by the minimax scores was missing, as explained in [29].

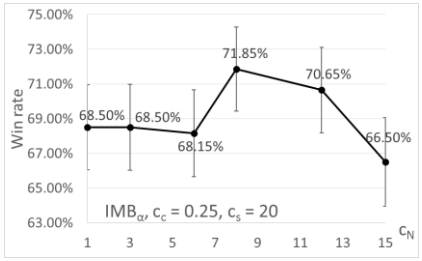
Fig. 6 also has the following implication. For a higher number of simulations per move, the improvement was relatively smaller. This hints that IMB has less improvement or becomes worse for a sufficiently large number of simulations per move. The reason is: the help of minimax scores decreases, since simulated results become more accurate with more simulations.




(a)




(b)




(c)

**Fig. 7.** The win rates for IMB$_\alpha$ with different (a) $c_c$, (b) $c_s$ and (c) $c_N$

Based on the results in Fig. 6, we selected some values for $c_c$, $c_s$ and $c_N$ to test IMB with dynamic $\alpha$ in Formula (3), denoted by IMB$_\alpha$. In the experiments, we varied one coefficient at a time, as shown in Fig. 7. When $c_c = 0.25$, $c_s = 20$ and $c_N = 8$, the highest win rate of 71.85% (±2.42%) was obtained, slightly better than fixing $\alpha = 0.3$ though it was not significant.

In [29], they mentioned that IMB had no significant improvement for Heart, also a PO game. Interestingly, our results show that IMB did significantly improve the playing strength for CDC.

### 4.1.3 Quality-Based Rewards

As mentioned in Subsection 3.3, we used simulation length (SL) and terminal state quality (TSQ) to adjust the rewards from simulations to favor those with shorter length and higher terminal state quality.

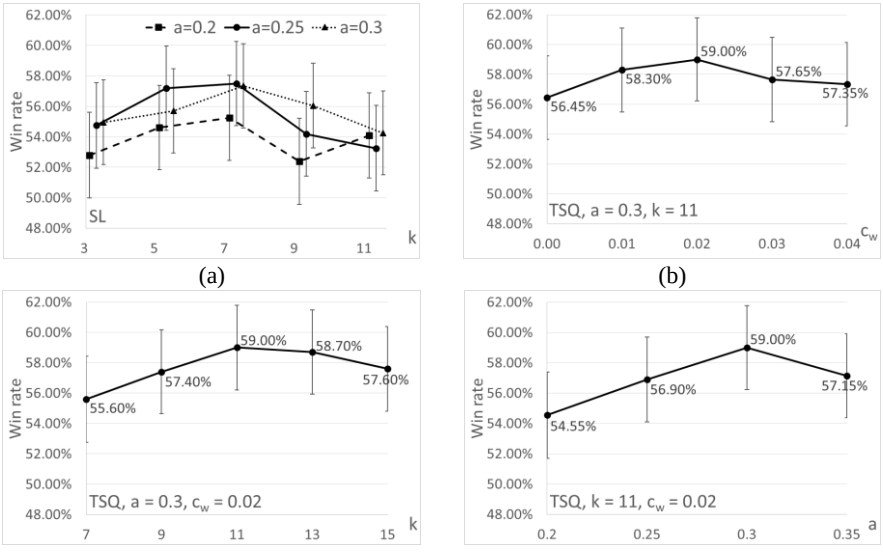



**Fig. 8.** The win rates for (a) SL, and for TSQ with different (b) $c_w$, (c) $k$ and (d) $a$

For SL, we tested two coefficients, the influence, $a$, and the sigmoid function constant, $k$, and obtained the results as shown in Fig. 8 (a), where the data with the same $k$ are staggered to show the confidence intervals. The highest win rate was 57.50% (±2.77%) when $a = 0.25$ and $k = 7$.

For TSQ, we needed one more coefficient, $c_w$, for piece weights when measuring terminal state quality. In our experiments, we obtained the highest win rate of 59.00% (±2.78%), when $a = 0.3$, $k = 11$ and $c_w = 0.02$. By fixing $a = 0.3$ and $k = 11$, we obtained the highest at $c_w = 0.02$, slightly better than the one at $c_w = 0.01$, as in Fig. 8 (b). Similarly, by fixing $a$ and $c_w$, we obtained the highest at $k = 11$ as in Fig. 8 (c). By fixing $k$ and $c_w$, we obtained the highest win rate at $a = 0.3$ as shown in Fig. 8 (d).

From above, quality-based rewards (SL and TSQ) improved the playing strength generally. However, the performances did not differ too much for different settings, due to relatively high confidence intervals.

### 4.1.4 Progressive Bias

As mentioned in Subsection 3.4 for progressive bias (PB), we used heuristics on moves to help guide selections for newly generated nodes. In our experiments, we firstly used the first evaluation for $H_i$, which considers capture moves, escape moves and suicide moves.

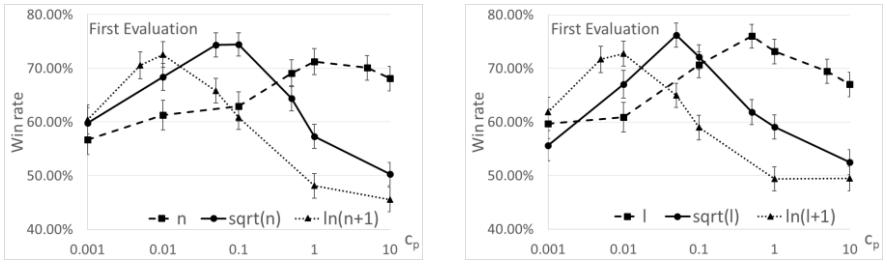
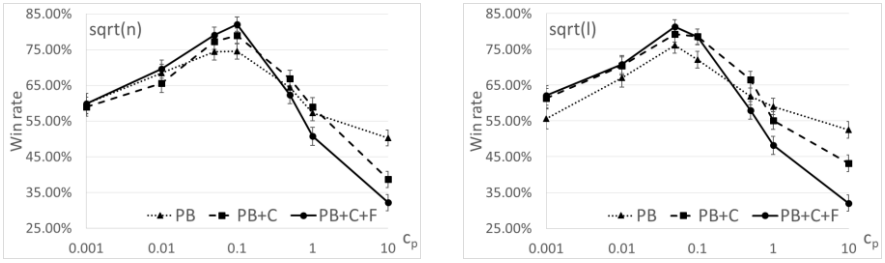



**Fig. 9.** Win rates for PB using the first evaluation with different $count_i$ based on (a) $n_i$ (b) $l_i$

The experiments tested different $count_i$ and $c_p$. In Fig. 9 (a), $n_i$, $\sqrt{n_i}$, and $\ln(n_i + 1)$ were used for $count_i$, while in Fig. 9 (b), $l_i$, $\sqrt{l_i}$, and $\ln(l_i + 1)$ were. In both figures, $c_p$ varied from 0.001 to 10. From the two figures, all kinds of $count_i$ showed win rates higher than 71% with proper settings of $c_p$. In Fig. 9 (a), the highest win rate was 74.45% (±2.18%) with $count_i = \sqrt{n_i}$ and $c_p = 0.1$. And in Fig. 9 (b), the highest win rate was 76.20% (±2.22%) with $count_i = \sqrt{l_i}$ and $c_p = 0.05$.

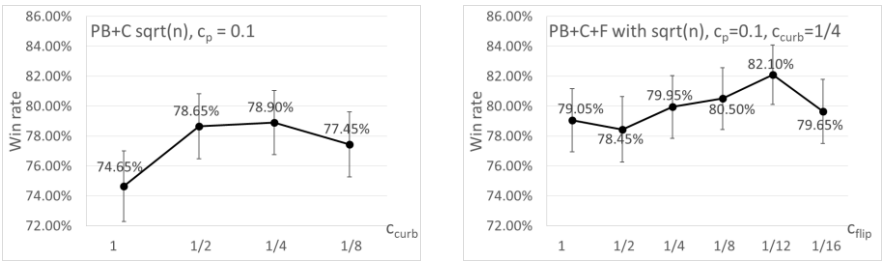
The above figures show that PB did significantly improve the playing strength, and that $count_i$ with square root, $\sqrt{n_i}$ and $\sqrt{l_i}$, performed better than the other two. Thus, the rest of our experiments considered the two cases only, though the one with $l_i$ is very close to $\sqrt{l_i}$.




(a) (b)

**Fig. 10.** Win rates for PB using different evaluations with $count_i$ based on (a) $\sqrt{n_i}$ (b) $\sqrt{l_i}$

Next, our experiments tested more heuristics, namely the second evaluation with curb moves, denoted by PB+C, and the third evaluation with curb moves and flipping, denoted by PB+C+F. These evaluations included three coefficients $c_p$, $c_{curb}$ and $c_{flip}$. First, we fixed both coefficients $c_{curb} = 1/4$ and $c_{flip} = 1/12$. In the case of $\sqrt{n_i}$ (for $count_i$), Fig. 10 (a) shows that PB+C+F performed best with the highest win rate of 82.10% (±1.98%), when $c_p = 0.1$. In the case of $\sqrt{l_i}$, Fig. 10 (b) shows that PB+C+F performed best with the highest win rate of 81.30% (±2.04%), when $c_p = 0.05$. Thus, the above shows that the playing strength can be improved with more heuristics with proper setting for these coefficients. Moreover, among all cases, PB+C+F performed best in the case that $count_i = \sqrt{n_i}$ and $c_p = 0.1$.




(a) (b)

**Fig. 11.** Win rates for (a) PB+C with different $c_{curb}$ and (b) PB+C+F with different $c_{flip}$

In order to confirm the best values for coefficients $c_{curb}$ and $c_{flip}$, our experiments chose $\sqrt{n_i}$ for $count_i$ and set $c_p = 0.1$ for PB+C and PB+C+F. Fig. 11 (a) shows that PB+C performed the best when $c_{curb} = 1/4$, though the one with $c_{curb} = 1/2$ had a very close win rate. Fig. 11 (b) shows the experimental results for PB+C+F with $c_{curb} = 1/4$. The results indicated that PB+C+F performed best when $c_{flip} = 1/12$.

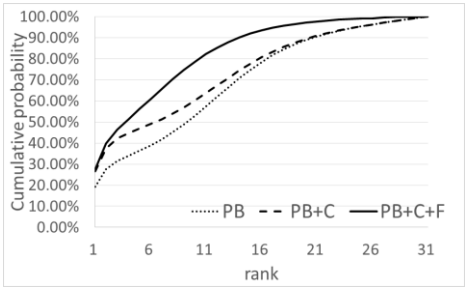



**Fig. 12.** Cumulative probability of finding the target move within a given rank

Fig. 12 shows the cumulative probability of finding the target move, or the so-called *prediction rate*, within a given rank for the three heuristics used in PB, PB+C and PB+C+F respectively. For a *target move*, we used the move selected by the best version of DARKKNIGHT (see Subsection 4.2) with 30,000 simulations per move. The results were collected from 1,000 games with 156,333 moves. From Fig. 12, the effectiveness of using more heuristics can also be obtained from the ability to find the target move. The prediction rate for PB+C+F was higher than that for PB+C, which in turn was higher than that for PB. From this, it is expected in CDC that the heuristics with higher prediction rates performs better in progressive bias.

### 4.2 Combinations of Techniques

In this subsection, we first summarize results in the previous subsection, and further improve the playing strength by combing the above techniques.

**Table 2.** The best win rates and speeds of EPT, IMB, SL, TSQ, and PB

| | Baseline | EPT | IMB | SL | TSQ | PB |
|---|---|---|---|---|---|---|
| Win rate | -- | 60.75% ±2.76% | 70.90% ±2.42% | 57.50% ±2.77% | 59.00% ±2.78% | 82.10% ±1.98% |
| Speed (#sim/sec) | 27,356 ±38 | 34,730 ±57 | 27,093 ±41 | 27,284 ±37 | 27,182 ±39 | 25,127 ±39 |

Table 2 summarizes the best results for incorporating each individual techniques, EPT, IMB, SL, TSQ, and PB, where each technique used the best settings as described above, except that we fixed $\alpha = 0.3$ for IMB for simplicity of analysis. These settings will be used in the rest of experiments. Table 2 also lists the speeds (from the initial board). Among all the techniques, EPT ran the fastest since playouts were terminated earlier with less overhead. IMB, SL, and TSQ updated data incrementally, so they only ran slightly slower than the baseline. Finally, for PB, since it calculated the expected evaluations for flipping actions, it ran the slowest, but improved the strength most.

**Table 3.** Win rates for the combinations of EPT, IMB, SL, TSQ, and PB

| | Individual | +SL | +TSQ | +EPT | +IMB |
|---|---|---|---|---|---|
| SL | 57.50% ±2.77% | -- | -- | -- | -- |
| TSQ | 59.00% ±2.78% | 61.70% ±2.68% | -- | -- | -- |
| EPT | 60.75% ±2.76% | 62.75% ±2.73% | 67.30% ±2.66% | -- | -- |
| IMB | 70.90% ±2.42% | 72.70% ±2.39% | 73.10% ±2.39% | 74.75% ±2.38% | -- |
| PB | 82.10% ±1.98% | 81.70% ±2.00% | 82.10% ±2.01% | 83.65% ±2.00% | 78.30% ±2.07% |

We firstly combined every two techniques, and the results for all the combinations are listed in Table 3. Generally, combining two techniques improved the playing strength, except for a few cases. Besides, the results also showed a tendency as follows. If a technique, say PB, had a better win rate (in the second column) than another, say IMB, all other techniques adding PB had better win rates than those adding IMB. However, it is interesting to see that the win rate for PB decreased with combining IMB. Our conjecture for this phenomenon is that the heuristics for both are correlated. For example, for a capture move, both weighted material sum in IMB and heuristic $H_i$ in PB increase at the same time. Therefore, the heuristic tends to weigh higher. The effect is just like to have a high $\alpha$ in IMB or a high $c_p$ in PB. This implies that the best individual settings of these coefficients, $\alpha$ and $c_p$, became less accurate when combined together.

Furthermore, we combined more techniques as the three combinations listed in Table 4. For the third combination, combining all the techniques together (in the rightmost column), the win rate reached up to 84.75% (±1.90%). The first combination, the best version in [22], was to remove PB from the third, while the second was to remove IMB. For the first combination, the win rate dropped to 76.70% (±2.31%). For the second combination, the win rate slightly dropped to 83.85% (±1.91%). This showed again that PB is more significant than IMB. We also did an experiment: using the first combination as baseline, the third combination had a win rate of 64.90% (±2.55%) which showed significant improvement over the best version in [22]. A version of DARKKNIGHT based on the above combined techniques won both CDC tournament champions in the 18th Computer Olympiad [21] and TCGA 2015 [40].

**Table 4.** Win rates for combining more techniques

| | +EPT +IMB +SL+TSQ | +EPT +SL+TSQ +PB | +EPT +IMB +SL+TSQ +PB |
|---|---|---|---|
| Win rate | 76.70% ±2.31% | 83.85% ±1.91% | 84.75% ±1.90% |

## 5 Conclusions

To our best knowledge, this paper is the first attempt to incorporate the four techniques, early playout terminations (EPT), implicit minimax backups (IMB), quality-based rewards (SL and TSQ), and progressive bias (PB) together and obtained significant improvement. We analyze the improvement through an MCTS-based CDC program, DARKKNIGHT. The experimental results showed that all of these techniques did significantly improve the playing strength with win rates of 60.75%, 71.85%, 57.50%, 59.00%, and 82.10% against the original DARKKNIGHT when incorporating EPT, IMB, SL, TSQ, and PB, respectively. The results also indicated that the improvement by progressive bias was most significant. By incorporating all together, the win rate reached up to 84.75%. The results demonstrated the effectiveness of the above techniques for an MCTS-based program playing a non-deterministic game CDC. The analysis provides some hints to games like CDC or even other non-deterministic problems, though the amount of improvement from each technique may be different from CDC's. An enhanced version of DARKKNIGHT with some fine tunings won the CDC tournament in the 18th Computer Olympiad.

**Acknowledgements.** The authors would like to thank Mark H. M. Winands for his suggestions of using loss counts. This work was supported by the Ministry of Science and Technology of the Republic of China (Taiwan) [contract numbers MOST 102-2221-E-009-069-MY2, 102-2221-E-009-080-MY2, 104-2221-E-009-127-MY2, and 104-2221-E-009-074-MY2].

## References

[1] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235-256 (2002), doi: 10.1023/A:1013689704352.

[2] Baier, H., Winands, M.H.M.: Monte-Carlo tree search and minimax hybrids with heuristic evaluation functions. In: Cazenave, T., Winands, M.H.M., Björnsson, Y. (eds.) Computer Games, vol. 504, pp. 45-63. Springer International Publishing (2014), doi: 10.1007/978-3-319-14923-3_4.

[3] Björnsson, Y., Finnsson, H.: CadiaPlayer: a simulation-based general game player. IEEE Transactions on Computational Intelligence and AI in Games 1(1), 4-15 (2009), doi: 10.1109/TCIAIG.2009.2018702.

[4] Borsboom, J., Saito, J.-T., Chaslot, G., Uiterwijk, J.: A comparison of Monte-Carlo methods for phantom Go. In: Proc. BeNeLux Conf. Artif. Intell., pp. 57-64. Utrecht, Netherlands (2007)

[5] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte-Carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games 4(1), 1-43 (2012), doi: 10.1109/TCIAIG.2012.2186810.

[6] Chang, H.-J., Hsu, T.-S.: A quantitative study of 2 × 4 Chinese dark chess. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) Computers and Games, pp. 151-162. Springer International Publishing (2014), do: 10.1007/978-3-319-09165-5_13.

[7] Chang, H.-J., Hsueh, C.-W., Hsu, T.-S.: Convergence and correctness analysis of Monte-Carlo tree search algorithms: a case study of 2 by 4 Chinese dark chess. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG2015), pp. 260-266, Tainan, Taiwan (2015), doi: 10.1109/CIG.2015.7317963.

[8] Chaslot, G.: Monte-Carlo tree search. Ph. D. dissertation, Universiteit Maastricht, Maastricht, the Netherlands (2010)

[9] Chaslot, G., Fiter, C., Hoock, J.-B., Rimmel, A., Teytaud, O.: Adding expert knowledge and exploration in Monte-Carlo tree search. In: van den Herik, H.J., Spronck, P. (eds.) Advances in Computer Games, vol. 6048, pp. 1-13. Springer Berlin Heidelberg (2010), doi: 10.1007/978-3-642-12993-3_1.
[10] Chaslot, G., Winands, M.H., van den Herik, H., Uiterwijk, J.W., Bouzy, B.: Progressive strategies for Monte-Carlo tree search. New Mathematics and Natural Computation 4, (2008), doi: 10.1142/S1793005708001094.
[11] Chen, B.-N., Hsu, T.-S.: Automatic generation of opening books for dark chess. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) Computers and Games, pp. 221-232. Springer International Publishing (2014), doi: 10.1007/978-3-319-09165-5_19.
[12] Chen, B.-N., Shen, B.-J., Hsu, T.-S.: Chinese dark chess. ICGA Journal 33(2), 93-106 (2010)
[13] Chen, J.-C., Fan, G.-Y., Tsai, S.-Y., Lin, T.-Y., Hsu, T.-s.: Compressing Chinese dark chess endgame databases. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG2015), pp. 254-259, Tainan, Taiwan (2015), doi: 10.1109/CIG.2015.7317932.
[14] Chen, J.-C., Lin, T.-Y., Chen, B.-N., Hsu, T.-S.: Equivalence classes in Chinese dark chess endgames. IEEE Transactions on Computational Intelligence and AI in Games 7(2), 109-122 (2015), doi: 10.1109/TCIAIG.2014.2317832.
[15] Chen, J.-C., Lin, T.-Y., Hsu, S.-C., Hsu, T.-S.: Design and implementation of computer Chinese dark chess endgame database. In: Proceeding of TCGA Workshop 2012, pp. 5-9, Hualien, Taiwan (2012) (in Chinese)
[16] Enzenberger, M., Müller, M., Arneson, B., Segal, R.: Fuego: an open-source framework for board games and Go engine based on Monte-Carlo tree search. IEEE Transactions on Computational Intelligence and AI in Games 2(4), 259-270 (2010), doi: 10.1109/TCIAIG.2010.2083662.
[17] Finnsson, H.: Generalized Monte-Carlo tree search extensions for general game playing. In: The Twenty-Sixth AAAI Conference on Artificial Intelligence, pp. 1550–1556, Toronto, Canada (2012)
[18] Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Proceedings of the 24th international conference on Machine learning, pp. 273-280. ACM, Corvalis, Oregon (2007), doi: 10.1145/1273496.1273531.
[19] Gelly, S., Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence 175(11), 1856-1875 (2011), doi: 10.1016/j.artint.2011.03.007.
[20] Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo Go. Tech. rep., HAL - CCSd - CNRS, France (2006)
[21] Hsueh, C.-H., Wu, I.-C.: DarkKnight wins Chinese dark chess tournament. ICGA Journal 38(4), 249-251 (2016).
[22] Hsueh, C.-H., Wu, I.-C., Tseng, W.-J., Yen, S.-J., Chen, J.-C.: Strength improvement and analysis for an MCTS-based Chinese dark chess program. In the 14th conference on Advances in Computer Games (ACG2015), Leiden, the Netherlands (2015), doi: 10.1007/978-3-319-27992-3_4.
[23] Huang, S.-C.: New heuristics for Monte-Carlo tree search applied to the game of Go. Ph. D. dissertation, Nat. Taiwan Normal Univ., Taipei, Taiwan (2011)
[24] Ikeda, K., Viennot, S.: Efficiency of static knowledge bias in Monte-Carlo tree search. In: Computers and Games, pp. 26-38. Springer (2014), doi: 10.1007/978-3-319-09165-5_3.
[25] Jouandeau, N.: Varying complexity in CHINESE DARK CHESS stochastic game. In: Proceeding of TCGA Workshop 2014, pp. 86-90, Taipei, Taiwan (2014)
[26] Jouandeau, N., Cazenave, T.: Monte-Carlo tree reductions for stochastic games. In: Cheng, S.-M., Day, M.-Y. (eds.) Technologies and Applications of Artificial Intelligence, vol. 8916, pp. 228-238. Springer International Publishing (2014), doi: 10.1007/978-3-319-13987-6_22.
[27] Jouandeau, N., Cazenave, T.: Small and large MCTS playouts applied to Chinese dark chess stochastic game. In: Cazenave, T., Winands, M.H.M., Björnsson, Y. (eds.) Computer Games, vol. 504, pp. 78-89. Springer International Publishing (2014), doi: 10.1007/978-3-319-14923-3_6.
[28] Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006, vol. 4212, pp. 282-293. Springer Berlin Heidelberg (2006), doi: 10.1007/11871842_29.
[29] Lanctot, M., Winands, M.H.M., Pepels, T., Sturtevant, N.R.: Monte-Carlo tree search with heuristic evaluations using implicit minimax backups. In: 2014 IEEE Conference on Computational Intelligence and Games, CIG 2014. pp. 1-8 (2014), doi: 10.1109/CIG.2014.6932903.
[30] Lin, Y.-S., Wu, I.-C., Yen, S.-J.: TAAI 2011 computer-game tournaments. ICGA Journal 34(4), 248-250 (2011)

[31] Lorentz, R.: Amazons discover Monte-Carlo. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) Computers and Games, vol. 5131, pp. 13-24. Springer Berlin Heidelberg (2008), doi: 10.1007/978-3-540-87608-3_2.
[32] Lorentz, R.: Early playout termination in MCTS. In the 14th conference on Advances in Computer Games (ACG2015), Leiden, the Netherlands (2015), doi: 10.1007/978-3-319-27992-3_2.
[33] Lorentz, R., Horey, T.: Programming breakthrough. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) Computers and Games, pp. 49-59. Springer International Publishing (2014), doi: 10.1007/978-3-319-09165-5_5.
[34] Nguyen, H., Viennot, S., Ikeda, K.: Fast optimization of the pattern shapes in board games with simulated annealing. Knowledge and Systems Engineering, pp. 325-337. Springer (2015), doi: 10.1007/978-3-319-11680-8_26.
[35] Pepels, T., Tak, M.J., Lanctot, M., Winands, M.H.M.: Quality-based rewards for Monte-Carlo tree search simulations. In: 21st European Conf. on Artif. Intell., Prague, Czech Republic (2014), doi: 10.3233/978-1-61499-419-0-705.
[36] Rosin, C.D.: Multi-armed bandits with episode context. Annals of Mathematics and Artificial Intelligence 61(3), 203-230 (2011), doi: 10.1007/s10472-011-9258-6.
[37] Russell, S., Norvig, P.: Artificial intelligence: a modern approach. Prentice Hall, New Jersey (1995)
[38] Saffidine, A., Jouandeau, N., Buron, C., Cazenave, T.: Material symmetry to partition endgame tables. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) Computers and Games, pp. 187-198. Springer International Publishing (2014), doi: 10.1007/978-3-319-09165-5_16.
[39] Su, T.-C., Yen, S.-J., Chen, J.-C., Wu, I.-C.: TAAI 2012 computer game tournaments. ICGA Journal 37(1), 33-35 (2014)
[40] TCGA 2015 computer game tournaments in Taoyuan, Taiwan, May 2015, available at `http://tcga2015.math.cycu.edu.tw/tournaments/eng/index.htm`
[41] Theory of computer games, a course in National Taiwan University taught by Tsu, T.-S., available at `http://www.iis.sinica.edu.tw/~tshsu/tcg/index.html`
[42] Tseng, W.-J., Chen, J.-C., Chen, L.-P., Yen, S.-J., Wu, I.-C.: TCGA 2013 computer game tournament report. ICGA Journal 36(3), 166-168 (2013)
[43] Van Lishout, F., Chaslot, G., Uiterwijk, J.W.: Monte-Carlo tree search in Backgammon. In: Computer Games Workshop, pp. 175-184, Amsterdam, the Netherlands (2007)
[44] Winands, M.H.M., Björnsson, Y., Saito, J.-T.: Monte-Carlo tree search in lines of action. IEEE Transactions on Computational Intelligence and AI in Games 2(4), 239-250 (2010), doi: 10.1109/TCIAIG.2010.2061050.
[45] Winands, M.H.M., Björnsson, Y., Saito, J.-T.: Monte-Carlo tree search solver. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) Computers and Games, vol. 5131, pp. 25-36. Springer Berlin Heidelberg (2008), doi: 10.1007/978-3-540-87608-3_3.
[46] Yang, J.-K., Su, T.-C., Wu, I.-C.: TCGA 2012 computer game tournament report. ICGA Journal 35(3), 178-180 (2012)
[47] Yen, S.-J., Chou, C.-W., Chen, J.-C., Wu, I.-C., Kao, K.-Y.: Design and implementation of Chinese dark chess programs. IEEE Transactions on Computational Intelligence and AI in Games 7(1), 66-74 (2015), doi: 10.1109/TCIAIG.2014.2329034.
[48] Yen, S.-J., Chen, J.-C., Chen, B.-N., Tseng, W.-J.: DarkKnight wins Chinese dark chess tournament. ICGA Journal 36(3), 175-176 (2013)
[49] Yen, S.-J., Su, T.-C., Wu, I.-C.: The TCGA 2011 computer-games tournament. ICGA Journal 34(2), 108-110 (2011)