Small and large MCTS playouts applied to Chinese Dark Chess stochastic game

Nicolas Jouandeau, Tristan Cazenave

Small and large MCTS playouts applied to Chinese Dark Chess stochastic game

Nicolas Jouandeau[1] and Tristan Cazenave[2]

[1] LIASD, Université de Paris 8, France
n@ai.univ-paris8.fr
[2] LAMSADE, Université Paris-Dauphine, France
cazenave@lamsade.dauphine.fr

Abstract. Monte-Carlo tree search is a powerful paradigm for full information games. We present various changes applied to this algorithm to deal with the stochastic game Chinese Dark Chess. We experimented with group-nodes and chance-nodes using various configurations: with different playout policies, with different playout size, with true or evaluated wins. Results show that extending playout size over the real draw condition is beneficial to group-nodes and to chance-nodes. It also shows that using evaluation function can reduce the number of draw games with group-nodes and can be increased with chance-nodes.

1 Introduction

Chinese Dark Chess (CDC) is a popular stochastic two players game in Asia that is often played on 4x8 rectangular board where players do not know flipping moves' payoff. The two player (called black and red) start with the same set of pieces: one king, two guards, two bishop, two knights, two rooks, two cannons and five pawns (pieces that are similar to Chinese Chess). Before the first move, players do not know their color. The first player move defines the first player color. In classical games, pieces evolve on squares and can move vertically and horizontally from one square to an adjacent free square (*i.e* up, down, left and right). A piece can capture another piece according to pieces value. Captures are done on vertical and horizontal adjacent squares except for cannons that capture pieces by jumping over another piece. Such jump is done over a piece (called the jumping piece) and to the target piece. Free spaces can stand between its initial position and the jumping piece and between the jumping piece and the target position. Even if flipping moves imply multiple board possibilities, classical moves can lead to similar positions during the game and capturing rules are different for each piece, MonteCarlo Tree Search (MCTS) programs have been recently improved as best. We show in this paper two different MCTS implementations that can be further improved using longer playouts, playing policies playout and heuristic playout.

Section 2 describes related works. Section 3 presents two MCTS implementations. Section 4 shows experimental results achieved with different playout size, playout policies and playout evaluation functions. At the end, section 5 concludes.

2 Related works

In this section we expose related works on CDC. Even if there are few works on this game, previous works concern *alpha-beta*, move policies, endgame databases and MCTS.

Chen *et al.* [1] used *alpha-beta* algorithm with different revealing policies combined with a *initial-depth flipping* method to reduce the branching factor. They distinguished opening, middle and endgame to apply different policies.

Chen *et al.* [2] built an endgame databases with retrograd analysis. Created databases are done for each first move color, up to 8 revealed pieces. They used 2TB of memory to represent 10^{12} positions. Positions status are stored as win, lost or draw.

Yen *et al.* [3] presented a non-deterministic Monte Carlo tree search model by combining chance nodes [4] and MCTS. They create shorter simulation by moderating the three policies named *Capture First*, *Capture Stronger Piece First* and *Capture and Escape Stronger Piece First*. As draw rate decreases, win rate increases and simulations are more meaningful for MCTS.

Chang and Hsu [5] solved the 2x4 variant. They created a *Oracle* variant where every pieces are known. Comparing the *Oracle* variant and the classical variant shows that the first move is crucial on 2x4 board.

Chen and Hsu [6] presented a policy-oriented search method to build opening books in case of very large state space as it is in CDC. Attack, defend, claim or discover territory are compared according to player's turn. Results show that player's level is a little stronger with openings. As flipping moves can completely change the game issue, they showed that enhancements provided are probabilistically acquired.

Safidine *et al.* [7] exploit pieces combinations to reduce endgame databases. By combining material symmetry identified by relations between pieces and endgames building with retrograd analysis, winning positions are recorded in databases. This general method has been applied to Skat, Dominoes and CDC. Even if relationship between pieces in CDC creates intricate symmetries, they reduced the size of 4 elements endgame tables by 9.92 and the size of 8 elements endgame tables by 3.68.

Chen *et al.* [8] present equivalence classes computation for endgames with unrevealed pieces. Boards are identified by threats and pieces' positions that are compared in a multiple steps algorithm. Compression rates of material has been studied from 3 to 32 pieces. Endgames database has been computed with 3 to 8 pieces and its number of element is reduced by 17.20.

Move groups have been proposed in [9] to address the problem of MCTS in games with a high branching factor. When there are too many moves, it can be a good heuristic to regroup the statistics of moves that share some properties. For example in the game of Amazons where the branching factor can be of the order of a thousand of moves a natural way to reduce the branching factor is to separate the queen move from the stone placement. In the imperfect information game Dou Di Zhu, Information Set [10] has been combined with move groups: the player first chooses the base move and then the kicker, as two separate

consecutive decision nodes in the tree. Move groups have also been analyzed on an abstract game [11].

3 Stochastic MCTS

In this section, we present the use of chance-nodes and of group-nodes with MCTS. Group-nodes are used to reduce the branching factor created by flipping moves. In a similar manner to move groups that regroup moves, we define group-nodes to consider all revealing moves at the same position in a single node. Apart from that, chance-nodes are the classical way to manage stochastic information in trees. We present the main loop and the selection function of MCTS with group-nodes and with chance-nodes. Both algorithms are presented as anytime interruptible process, that tend to produce better solution over computing time and that are instantaneously interrupt when time is up or when a winning move is found.

3.1 With group-nodes

During the game, applying a flipping move can conduct to different boards. At the beginning of the game, the first player has 32 possible moves that correspond to $4 * 10^{36}$ possibilities. During the game, the number of possible boards linked to flipping moves decreases with the number of unrevealed pieces. For the penultimate flipping move, 2 boards are possible and for the last flipping move, only one board. But as the number of possible boards stays over 120 for more than 5 different pieces, the number of possible boards remains important at most of the time before endgame. To reduce the number of children node produced by flipping moves, all possible boards that arise from a flipping move are gathered in single group-node. Therefore, at the beginning of the game, the root node is followed by 32 children that are group-nodes. Then other actions (moves, jumps and captures) are represented in the tree with classical nodes. The main loop of MCTS with group-nodes is presented in Alg 1. The selection phase (line 2) returns a node q to expand, its corresponding board and $\mathcal{L}$ the list of moves to apply for expansion. By default, all nodes inserted in the tree have $UNSET$ winning color. If this node is known as winning position, the process is interrupted (line 3). Otherwise each element of $\mathcal{L}$ creates a new child to q and store it in $\mathcal{N}$. For each element of $\mathcal{N}$, the board is modified, a new simulation is applied from the modified board and a new result is backpropagate from the new node up to the *root_node*. At the end, the `bestNext` function selects the best next node q_{best} of the *root_node* that defines the best next move.

The selection function of MCTS with group-nodes is presented in Alg 2. The process iterates to find the best node q, its corresponding board and its moves $\mathcal{M}$. There are 4 different cases:

- q is known as winning position. Then the process is interrupted (line 5).
- q is leading a player to a new winning position. Then it returns q with the corresponding *board* and an empty set (line 7).

Algorithm 1 MCTS with group-nodes

```
1: while not-interrupted do
2:    (q, board, L) ← select ( )
3:    if q.winning_color != UNSET then break
4:    N ← expandAll (q, L)
5:    for each e ∈ N do
6:       board' ←applyMove (e, board)
7:       simulate (e, board')
8:       backpropagate (e)
9:    end for
10: end while
11: q ← bestNext (root_node)
12: return q
```

- q is a new node that has not been extended previously. Then it returns q with the corresponding *board* and new moves that corresponds to the *board* (lines 9 to 11).
- q is a group-node previously evaluated with a different flipping outlet that produces new moves that are not yet considered in q children. Then it returns q with the corresponding *board* and new moves that corresponds to only previously unconsidered moves in q group-node (line 13 to 15).

The `next` function (Alg 2 line 6) returns all current next nodes of q. The `move` function (line 8) returns all next moves of the current *board*. If $\mathcal{M}$ is empty (line 9), the corresponding node is noted as winning node for the opponent player of the current *board* turn. The test applied line 13, checks if the current situation fits with the current group-node. If it fits, then the `best_next` function is applied to select the best nodes inside $\mathcal{M}$ (line 17). If it does not fit, then the selection process is stopped and $\mathcal{M}$ becomes a set of previously unconsidered moves (line 14).

3.2 With chance-nodes

During the game, moves can conduct to the creation of chance-nodes. As different pieces are unknown, each flipping move is represented with a chance-node. Other board modifications, like moves, jumps and captures, are represented with classical nodes. Chance-nodes are composed of classical nodes. At a chance-node, each flipping possibility corresponds to a new child. The main loop of MCTS with chance-nodes is presented in Alg 3.

According to UCT formulae, this process applies iteratively selection, simulation and backpropagation. From a selected node q, the simulation leads to a new node q_{new} from which the result of the last simulation is backpropagate toward the *root_node*. At the end, the `bestNext` function selects the best next node q_{best} of the *root_node* that defines the best next move. The selection of a chance node can lead to different boards and the selection of a classical node leads to an expected situation. At the beginning, all nodes are inserted in the

Algorithm 2 `select` ( ) with group-nodes

```
1: q ← root
2: board ← root_board
3: M ← ∅
4: while not-interrupted do
5:    if q.winning_color != UNSET then break
6:    N ← next (q)
7:    if size (N) = 0 then break
8:    M ← moves (board)
9:    if size (M) = 0 then
10:      q.winning_color ← opponent (board.turn)
11:      break
12:   end if
13:   if ∃ e ∈ M with e ∉ N then
14:      M ← M − (M ∩ N)
15:      break
16:   end if
17:   (q, board) ← best_next (q, M)
18: end while
19: return (q, board, M)
```

Algorithm 3 MCTS with chance-nodes

```
1: while not-interrupted do
2:    (q, board) ← select ( )
3:    if q_new.winning_color != UNSET then break
4:    q_new ← simulate (q, board)
5:    backpropagate (q_new)
6: end while
7: q_best ← bestNext (root_node)
8: return q_best
```

tree without winning color information. If the selected node is a winning node, the process can be immediately interrupted (line 3 ALG 3 and line 4 ALG 4).

Algorithm 4 `select` ( ) with chance-nodes

```
1: q ← root
2: board ← root_board
3: while not-interrupted do
4:    if q.winning_color != UNSET then break
5:    M ← moves (board)
6:    if size (M) = 0 then
7:       q.winning_color ← opponent (board.turn)
8:       return (q, board)
9:    end if
10:   if (pos_i, pos_f) ← newMove (q, board, M) then
11:      if (pos_i = pos_f) then
12:         q' ← addChanceNode (q)
13:         q_new ← addNode (q')
14:      else
15:         q_new ← addNode (q)
16:      end if
17:      board ← play (pos_i, pos_f)
18:      return (q_new, board)
19:   end if
20:   (q, board) ← best (q, board, M)
21: end while
```

The selection function of MCTS with chance-nodes is presented in ALG 4. The process iterates to find the best node q and its corresponding board. From the *root_node* and the *root_board*, the current *board* is updated according to the best move. At each iteration, a set of moves $\mathcal{M}$ is defined according to the selected board position and its turn. If this set is empty (line 6), the corresponding node is noted as winning node for the opponent player of the current *board* turn. If the best move is a new flipping move (line 11), a new chance node and a new node are added in the tree. If initial and final positions differ, a simple classical node is added.

4 Experiments

In this section, we present various experiments to select fastest policies, to reduce draw endgames, with varying playout size, with or without evaluation function.

4.1 Fastest policies

In this section, we present various policies used to enhance playouts. We present basic and advanced ones and evaluate them to be useful in MCTS with as fast

simulations as possible. The fastest policies are considered as most promising and are selected to continue our study.

We have used 4 basic playout policies, that are natural to use in CDC :

- Random, where players play randomly.
- Capture, where players try to capture opponent pieces.
- Avoid, where players try to avoid opponent's capture.
- Trap, where players try to minimize opponents moves.

According to these basics policies, we settled 4 advanced policies declined from the basics:

- Capture and avoid, where players try first to capture one opponent, try second to avoid opponents and otherwise play randomly.
- Avoid and capture, where players try first to avoid opponents, try second to capture one opponent and otherwise play randomly
- Capture and trap, where players try first to capture one opponent and otherwise to trap opponents.
- Capture avoid and trap, where players try first to capture an opponent, try second to avoid opponents and otherwise to trap opponents.

All these policies have been tested for 2000 playouts at the beginning, the middle and the end of the game. Results are shown in Fig. 1-3 and in Tab. 1.

Fig. 1 show the board at the beginning of the game, when player colors are unknown. Best moves are colored in gray on the board and are bold in Tab. 1.

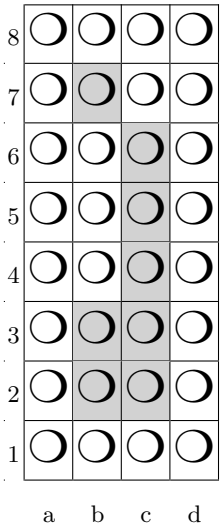

Fig. 1: Beginning.

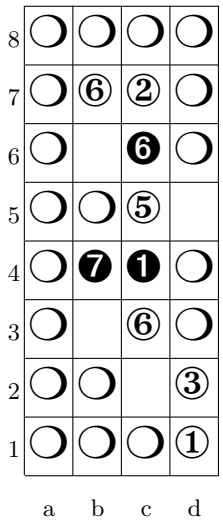

Fig. 2: Middlegame.

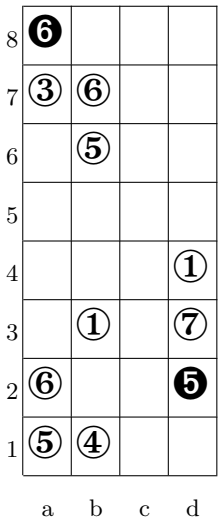

Fig. 3: Endgame.

Table 1: Playing 2000 playouts at beginning, middle game and endgame.

Policy	Time	Rem. pieces	Playout size	Without eval. fun. Best	W	L	D	With eval. fun. Best	W	L	D
At beginning											
Random	0.660	12.16 (2.31)	106.37 (9.25)	d7	-	-	100	**b7**	69	30	01
Capture	0.375	5.67 (2.20)	100.01 (17.69)	**c2**	25	22	53	**b3**	52	48	-
Avoid	5.973	16.78 (2.88)	207.41 (74.49)	**b2**	-	-	100	c8	57	43	-
Capture and avoid	0.598	7.24 (2.25)	107.27 (18.05)	b1	34	13	53	c8	55	45	-
Avoid and capture	1.049	7.69 (2.20)	123.02 (20.15)	**c4**	18	06	76	**c6**	38	60	02
Trap	140.457	8.95 (2.15)	579.69 (115.79)	d7	-	-	100	d6	86	14	-
Capture and trap	99.753	15.07 (3.07)	610.96 (111.26)	**c3**	02	-	98	**c5**	52	48	-
Capture avoid and trap	39.799	7.12 (2.52)	435.53 (177.59)	d3	31	25	44	d1	64	36	-
At midlle game											
Random	0.606	12.18 (2.24)	91.89 (9.20)	d8	-	-	100	c6-c7	43	57	-
Capture	0.348	6.91 (2.17)	75.60 (19.26)	**c6-c7**	48	02	50	**c6-c7**	90	10	-
Avoid	5.072	16.60 (2.72)	194.68 (87.18)	d4	-	-	100	d4	55	44	01
Capture and avoid	0.585	7.53 (1.96)	89.06 (17.93)	**c6-c7**	34	02	64	**c6-c7**	84	16	-
Avoid and capture	0.885	7.55 (2.02)	103.68 (15.79)	**c6-c7**	04	14	82	**c6-c7**	30	69	01
Trap	157.911	9.68 (1.86)	704.73 (62.78)	d7	-	-	100	d8	97	03	-
Capture and trap	123.055	14.49 (2.89)	711.58 (45.01)	b8	01	-	99	d4	76	24	-
Capture avoid and trap	34.201	9.92 (1.59)	370.29 (177.85)	**c6-c7**	54	00	46	**c6-c7**	100	-	-
At endgame											
Random	0.504	8.23 (1.14)	42.72 (1.46)	a8-a7	-	-	100	a8-a7	-	100	-
Capture	0.469	6.26 (0.94)	41.93 (6.02)	a8-a7	00	22	78	a8-a7	-	100	-
Avoid	0.781	10.12 (0.54)	40.66 (1.55)	d2-c2	-	02	98	d2-c2	-	100	-
Capture and avoid	0.733	8.00 (0.00)	43.00 (0.00)	a8-a7	-	-	100	a8-a7	-	100	-
Avoid and capture	0.817	9.80 (0.53)	40.34 (4.36)	d2-c2	-	04	96	d2-c2	-	100	-
Trap	11.138	7.00 (0.00)	44.00 (0.00)	d2-c2	-	-	100	d2-c2	-	100	-
Capture and trap	7.389	11.00 (0.00)	40.00 (0.00)	d2-c2	-	-	100	d2-c2	-	100	-
Capture avoid and trap	6.030	8.0 (0.00)	43.00 (0.00)	a8-a7	-	-	100	a8-a7	-	100	-

Fig. 2 shows the resulting board situation after the following 10 turns. Unknown pieces are represented with white circles. The 10 moves played are (columns are annotated with letters and rows are annotated with numbers. Flipping moves indicate a revealed piece under parenthesis. Moving and capturing moves indicate two coordinates.) :

```
c4(k) d1(P) ; d2(N) d5(p) ; c2(G) c3(c) ; c5(C) c5-c3 ;
c4(p) c6(M) ; b3(r) c6-c5 ; b3-c3 c2-c3 ; c7(C) c5-d5 ;
b6(g) d5-c5 ; b6-c6 b7(G) ;
```

It is now first player's turn to play. 19 reveal moves remain. First player is black and its non-flipping possible moves are :

```
b4-b3 ; c6-b6 ; c6-c7 ; c6-c5 ;
```

Second player is white (*i.e.* red) and its non-flipping possible moves are :

```
b7-b6 ; c5-c4 ; c5-d5 ; c3-b3 ; c3-c2 ; c3-c4 ; d2-c2 ;
```

First player has captured only one `C` piece and second player has captured 3 pieces that are `p c r`. Good move for black is `c6-c5`, or reveal `c8` and `d7`.

Fig. 3 shows an endgame board, where everything is known. Black has clearly lost the game.

As CDC games can end in a draw, we settled an evaluation function that can evaluate a draw board. This evaluation is based on the material that is remaining on the board for one player over its opponent. It allows to assign numerical value to draw endgame boards. As we do not know if playing first is an advantage, we allow this evaluation to reply draw if material are equivalent.

In some specific cases, draw depends on pieces position and then this function gives false win detection. In order, pieces {K G M R N C P} are associated to {0.15, 0.1, 0.07, 0.05, 0.03, 0.05, 0.05}.

Tab. 1 compares the time needed by basic and advanced policies to achieve 2000 playouts. This table also gives the average of remaining pieces and its standard deviation, the average playout size and its standard deviation, best moves with and without evaluation function.

It shows that some policies are too slow (*i.e.* Avoid, Avoid and capture, Trap, Capture and trap, Capture avoid and trap) to play a significant number of playouts to be simply used in MCTS. Remaining pieces and playout size showed us that longer playout could be interesting.

4.2 Reducing draw endgames

In this section, we select the best policies by checking their ability to create as less draw endgames as possible. Results are presented according to various draw conditions.

As some policies are considered as too slow, we only kept *Random* (*i.e.* `RND`), *Capture* (*i.e.* `CAP_RND`), *Capture and avoid* (*i.e.* `CAP_AVD_RND`) and *Avoid and Capture* (*i.e.* `AVD_CAP_RND`) (where these three last also call random if nothing has been done first).

Table 2: Applying 2000 playouts from beginning board.

Policy	time[sec]	draw	Rem. pieces	Playout size	Draw condition
RND	0.630	1.00	12.62	106.07	40
	1.067	0.99	6.03	232.11	160
	1.801	**0.67**	3.24	622.37	640
CAP_RND	0.376	0.70	5.66	103.71	40
	0.522	0.46	4.59	171.77	160
	0.802	**0.31**	4.29	335.92	640
CAP_AVD_RND	0.598	0.73	7.15	110.45	40
	0.953	0.53	6.07	185.41	160
	2.139	**0.43**	5.87	403.44	640
AVD_CAP_RND	1.012	0.87	7.77	124.29	40
	1.707	0.69	6.19	223.51	160
	4.353	**0.59**	5.85	550.41	640

Tab. 2 shows the time used to generate 2000 playouts from the beginning board. Next columns show the draw ratio, the average number of pieces at the end, the playout size according to different draw conditions. In a normal game, the draw condition is equal to 40 moves without capture or reveal. It shows that the number of draw can be reduced significantly by increasing the draw condition. In the same time, the number of remaining pieces are reduced. It shows that `CAP_RND` is really efficient for very important value of draw condition. We decided to eliminate the `AVD_CAP_RND` policy that increases the time with fewer draw than simple `RND` policy.

4.3 Group-nodes vs. chance-nodes

In this section, we challenge MCTS players by facing group-nodes against chance-nodes, with different playout size, with most promising policies. For each combination, we also checked the evaluation influence in helping MCTS formulae to select better player. All the player are tested against a reference player, that simply plays randomly when pieces are unrevealed and otherwise applies minimax to find the best move. Results are shown in Tab. 3 for 500 games with half as first player and half as second player. Each player has 1 sec to generate a new move. The corresponding number of draw games are shown in Fig. 4 for MCTS with group-nodes and in Fig 5 for MCTS with chance-nodes.

Table 3: Playing 500 games against reference player.

	playout size	40		160		640		2560	
		win	lost	win	lost	win	lost	win	lost
chance-nodes	C2-R	0	255	4	298	213	109	164	72
	C2-R-h	9	137	104	64	116	43	135	42
	C2-CR	**244**	**16**	242	21	245	15	149	21
	C2-CR-h	131	15	143	14	138	23	129	23
	C2-CAR	199	33	**245**	**27**	239	46	159	47
	C2-CAR-h	129	16	125	25	122	9	104	30
	playout size	40		160		640		2560	
		win	lost	win	lost	win	lost	win	lost
group-nodes	C1-R	10	249	51	185	78	187	64	233
	C1-R-h	60	269	103	231	80	219	85	263
	C1-CR	35	46	101	52	146	61	137	66
	C1-CR-h	120	87	167	98	**182**	**88**	147	127
	C1-CAR	41	35	103	52	128	93	72	170
	C1-CAR-h	109	100	167	86	**175**	**97**	130	143

The `RND` policy is now abbreviate with `R`, the `CAP_RND` policy is abbreviate `CR` and the `CAP_AVD_RND`) policy is abbreviate `CAR`. `C1` stands for group-nodes and `C2` stands for chance-nodes. `h` mentions evaluation function usage and no `h` means that only true victories are considered inside playouts.

Results shows that chance-nodes are less effective with evaluation function where group-nodes are more effective with evaluation function. It further shows that chance-nodes are more dependant on playout size than group-nodes. Best group-nodes players achieved 182 and 175 victories when chance-nodes achieved 244 and 245 victories.

For each policy, Fig 4 shows that evaluation function reduces the number of draw games with group-nodes. Nonetheless the number of draw games is equal or more important with chance-nodes (see Fig 5).

Table 4: Tournament between UCT players.

	C1-CAR-h-640	C2-CR-40	C2-CR-640	C2-CAR-160
C1-CR-h-640	**268** / 169	**282** / 97	176 / **252**	287 / 131
C1-CAR-h-640		**228** / 137	156 / **285**	236 / 175
C2-CR-40			149 / **220**	176 / 204

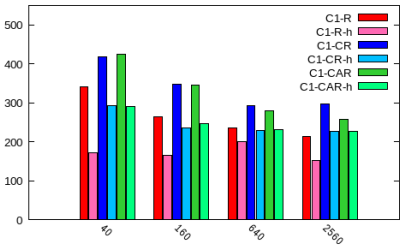

Fig. 4: Group-node draw of Tab 3

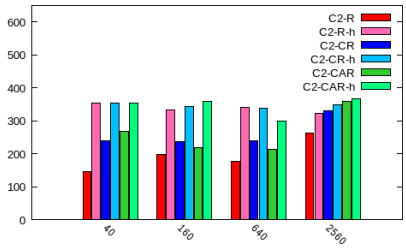

Fig. 5: Chance-node draw of Tab 3

Tab. 4 presents a tournament of best chance-nodes and group-nodes players. It shows that :

- Between group-nodes, `CR-h-640` (*i.e. Capture* with evaluation function and playout size of 640) is the best policy.
- Chance-nodes even simply with *Capture* policy are better than group-nodes.
- Between all players, the best is chance-nodes with *Capture*, without evaluation function and with playout size of 640.

It shows that sophisticated policies are better with group-nodes where the basic *Capture* policy is the best with chance-nodes. Extending playout size over the real draw condition is beneficial to group-nodes and to chance-nodes.

5 Conclusion

We have presented different Monte-Carlo tree search that deal with the stochastic game Chinese Dark Chess. We have shown relations with playout size, basic or advanced policies and evaluation function usage. While extending the playout size is useful to create more inform playouts, an evaluation function usage can increase or decrease player's effectiveness through modifying the number of draw possibilities.

References

1. B-N. Chen and B-J. Shen and T-S. Hsu. *Chinese Dark Chess*. ICGA Journal, 2010, vol. 33, num. 2, pp. 93.
2. Jr-C. Chen and T-Y. Lin and S-C. Hsu and T-S. Hsu, *Design and Implementation of Computer Chinese Dark Chess Endgame Database*. TCGA Computer Game Workshop (TCGA-2012).
3. S-J. Yen and C-W. Chou and Jr-C. Chen and I-C. Wu and K-Y. Kao, *The Art of the Chinese Dark Chess Program DIABLE*. Proc. of the Int. Computer Symposium (ICS-2012).
4. M. Lanctot and A. Saffidine and J. Veness and C. Archibald and M. Winands, *Monte Carlo *-Minimax Search*, 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI-2013).

5. H-J. Chang and T-S. Hsu. *A Quantitative Study of 24 Chinese Dark Chess*, Proc. of the 8th Int. Conf. on Computers and Games (CG-2013).
6. B-N. Chen and T-S. Hsu, *Automatic Generation of Chinese Dark Chess Opening Books*, Proc. of the 8th Int. Conf. on Computers and Games (CG-2013).
7. A. Saffidine and N. Jouandeau and C. Buron and T. Cazenave, *Material Symmetry to Partition Endgame Tables*, Proc. of the 8th Int. Conf. on Computers and Games (CG-2013).
8. Jr-C. Chen and T-Y.Lin and B-N. Chen and T-S. Hsu, *Equivalence Classes in Chinese Dark Chess Endgames*, IEEE Trans. on Computational Intelligence and AI in Games (2014).
9. B.E. Childs and J.H. Brodeur and L. Kocsis, *Transpositions and move groups in Monte Carlo tree search*, IEEE Symp. On Computational Intelligence and Games, pp 389–395 (CIG-2008).
10. P.I. Cowling and E.J. Powley and D. Whitehouse, *Information Set Monte Carlo Tree Search*, IEEE Trans. on Computational Intelligence and AI in Games, vol. 4, num. 2, pp 120–143, 2012.
11. G. Van Eyck and M. Müller, *Revisiting move groups in monte-carlo tree search*, Advances in Computer Games, pp 13–23, (ACG-2012).