Optimal Player Information MCTS applied to Chinese Dark Chess

Nicolas Jouandeau

Optimal Player Information MCTS applied to Chinese Dark Chess

Nicolas Jouandeau
LIASD
Paris 8 University
Saint-Denis, France
Email: n@ai.univ-paris8.fr

***Abstract*—Alpha-beta and Monte-Carlo Tree Search (MCTS) are two powerful paradigms useful in computer games. When considering imperfect information, the tree that represents the game has to deal with chance. When facing such games that presents increasing branching factor, MCTS may consider, as alpha-beta do, pruning to keep efficiency. We present a modified version of MCTS-Solver algorithm, called OPI-MCTS as Optimal Player Information MCTS, that adds game state information to exploit logical reasoning during backpropagation and that influences selection and expansion. OPI-MCTS is experimented in Chinese Dark Chess, which is a imperfect information game. OPI-MCTS is compared with classical MCTS.**

I. Introduction

Alpha-beta pruning algorithm is a powerful game-tree search successfully applied in many domains. By considering the 2 players as min-player and max-player, it aims to get the best move according to : (1) an evaluation function applied to leaf nodes; (2) with the backpropagation of values towards the root; (3) with cutting possibilities depending on parents and children values. Even if alpha-beta cuts improves Minimax a lot (with a pruning that does not change the final result), their success rate highly depends on the evaluation function accuracy.

Monte Carlo Tree Search (MCTS) is another powerful game-tree search, newer than alpha-beta and also successfully applied in many domains [1]. The first MCTS algorithm is UCT [2] that combines UCB formula [3] with tree search. The great benefit of UCT is to be able of being used without expert domain knowledge.

Chinese Dark Chess (CDC) is a two-players imperfect information game played with a 4x8 rectangular board where players do not know the location of their pieces at the beginning. Even if they have the same initial material, their flipping moves will reveal pieces (own ones or opponent's ones) that will define player's color and pieces positions. Capture are only allowed from a revealed piece to another revealed piece or to an empty position. Pieces of each player are divided between 7 different types, that have to consider different constraints while they are moving. Even if flipping moves imply multiple board possibilities, classical moves can lead to similar positions during the game. CDC tournaments (mainly at Computer Olympiad (CO), Technologies and Applications of Artificial Intelligence (TAAI) and Taiwan Computer Game Association Computer Game Workshop (TCGA)) are using a Swiss system, with rounds of two games per player. The first player unable to play loses. To avoid infinite games, 3-times repetition of a position leads to draw game. When no flipping or capturing move arises within a fixed number of plies, it is also leading to draw game. The number of plies depends on teams involved, commonly 180 plies are considered.

As MCTS deals with statistics without expert knowledge, efficiency remains linked to branching factor. In TCGA and TAAI 2012, the MCTS with chance nodes program Diablo won the competition. In TCGA 2013, the MCTS program DarkKnight won the competition ahead 4 other MCTS programs. In CO 2013, DarkKnight on again. In TAAI 2013, the alpha-beta program Yahari won the competition. In TCGA and TAAI 2014, the alpha-beta program Yahari won the competition, leaving the second place to DarkKnight during TCGA. In CO 2015, DarkKnight returned and won. As CDC has a huge branching due to the revealing moves, chances nodes allow alpha-beta to reduce the branching factor dependency. Even if chance nodes have even been used with MCTS by Diablo, we believe that a better understanding of MCTS behavior and CDC complexity [4] are needed.

As adding chance nodes seems to reduce the efficiency of MCTS by sealing it in added chance layers, we propose to add game state information that could be exploited by a logical decision process to guide next MCTS iterations, to prune useless evaluations and also to backpropagate this information in the tree. We propose to add a termination status that allows logical backpropagation to define parents termination. In this way, we present a modified version of MCTS-Solver [5] applied to Chinese Dark Chess, with new selection, expansion and backpropagation functions. These new algorithm is called OPI-MCTS, as Optimal Player Information MCTS. Unlike the initial MCTS-Solver proposal, we modify the statistics of nodes according to their termination status without biasing the formula with expert knowledge. The number of playouts backpropagated differes depending on nodes types.

This paper is organized as follows: Section II describes related works in CDC with alpha-beta, MCTS and other improvements. Section III presents our OPI-MCTS algorithm. Section IV shows experimental results. Section V concludes.

II. Related works

In this section we expose related works on CDC. Recent works on this game concern alpha-beta pruning, MCTS improvements, openings and endgame databases.

Winands *et al.* [5] presented the MCTS variant called MCTS-Solver designed to prove the value of positions in

the tree. They applied it to Lines of Action, that is a 2-players perfect information game with sudden-death situations. Depending on the rules, positions that validate each players goal can lead to a win or a draw. They used a modified version of UCT formula with Progressive Bias based on pre-computed knowledge of the game. Hsueh *et al.* [6] used MCTS with Early Playout Terminations (EPT), Implicit Minimax Backups (IMB) and Quality-Based Rewards (QBR). They showed that combining these three techniques improves CDC MCTS programs.

Lorentz [7] presented MCTS improvements with EPT in many games with different characteristics. EPT has been applied successfully to Amazons, Breakthrough and Havannah.

Lanctot *et al.* [8] proposed a heuristic function to guide MCTS playouts. This principle, called IMB, has been applied successfully to Kalah, Breakthrough and Lines of Action.

Pepels *et al.* [9] proposed to modify playouts rewards with QBR according to playout length and playout termination. QBR has been applied successfully to Amazons, Breakthrough, Cannon, Checkers and Chinese Checkers. Improvements seems to be stuck for some games like Pentalah that blinds the playout length indicator.

Chen *et al.* [10] used alpha-beta algorithm on CDC with policies to deal with revealing pieces. They also present a way to reduce the branching factor by linking flipping moves to depth. Different policies are defined according to the advancement in game.

Baier and Winands [11] proposed different modifications of MCTS with minimax-like enhancements during the selection, playout and backpropagation phases to combine minimax short term evaluation and MCTS long term evaluation. Experiments on Connect-4 and Breakthrough show improvements without adding specific game expert knowledge according to depth and choosen enhanced phase.

Saffidine *et al.* [12] used buffering and reevaluation to solve the problem of chunked data that could come from UCT parallelisation. The buffering hold statistics inside nodes and the reevaluation recalculates nodes statistics before a new selection. Thus nodes statistics are divided between two parts (*i.e.* unpropagated and propagated values) and the number of playouts considered differes from the real ones when buffering is on.

Yen *et al.* [13], [14] proposed a way to combine MCTS and chance nodes [15]. They shown useful policies called *Capture First*, *Capture Stronger Piece First* and *Capture and Escape Stronger Piece First*, that improved the program's level.

Jouandeau and Cazenave [16] presented MCTS influence of various playout length, playout policy with heuristic board evaluation. They studied group constitution related to pieces position. They compared MCTS with chance-nodes and with group-nodes. It shows that playout lengths, policies and heuristic values are dependant to these MCTS variants.

Jouandeau and Cazenave [17] compared tree reduction issues for games with large branching factor. They compared different regrouping policies and nodes selection in CDC. It shows that chance-nodes regrouping policy is better without heuristic evaluation and with Minimax termination.

Chang and Hsu [18] solved the 2x4 variant and demonstrate that the first move is crucial on 2x4 board. They used a *Oracle* variant where every pieces are known to compare solutions with optimal moves.

Chen and Hsu [19] built opening databases. As flipping moves can reverse the situation, they showed that enhancements provided are probabilistically acquired.

Chen *et al.* [20] built an endgame databases with retrograd analysis, up to 8 revealed pieces, using 2TB of memory to represent 10^{12} positions. Positions status are stored as win, lost or draw.

Saffidine *et al.* [21] built endgame databases with retrograd analysis and pieces combinations pieces exploitation. By combining material, identified symmetries reduce the size of 4 pieces endgame tables by 9.92 and the size of 8 pieces endgame tables by 3.68.

Chen *et al.* [22] 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.

III. Pruning in MCTS

In this section, we present OPI-MCTS algorithm that combines classical MCTS with a full expansion phase, add states to prooved nodes and at last, apply different UCT formula to these nodes with virtual numbers of playouts.

A. Classical MCTS with a full expansion

Fig. 1 presents the main loop of classical MCTS algorithm. It consists in iteratively applying the 3 phases called selection, expansion and backpropagation, illustrated here respectively by the 3 functions `select`, `expand` and `backpropagate`. This main loop tries to get the best next move from the $current\_board$. The search is initialized with the $current\_board$ that defines the $root$ node of the tree. At each iteration, the selection phase returns the best sequence $\mathcal{S}$ to start with. Then the expansion phase applies $\mathcal{S}$ to find the part of the tree that will grow, getting new results that will be backpropagated towards shallowest nodes, hoping of getting a more promising $\mathcal{S}$ during next call. The number of new nodes created during an expansion phase depends on the state of the last element of $\mathcal{S}$ and on its number of children. If the last element of $\mathcal{S}$ is terminal (Fig. 3 lines 2 to 4) then a single win, draw or lost is added to statistics of q_i node. Otherwise, one playout is played for each next move (Fig. 3 line 7). Thus the number of new nodes added in the tree corresponds to the number of children of q_i. To improve the confidence in values in the tree, we choose to develop all children at each iteration. It allows us to get more accurate estimates of parent nodes. At the end, the best next node of the $root$ node defines the best move.

Fig. 2 presents the classical MCTS selection function. This function recursively selects the `bestChild` node according to statistics while nodes have children.

mcts ($current\_board$)

1: $root \leftarrow$ `init`($current\_board$)
2: **while** not-interrupted **do**
3: $\mathcal{S} \leftarrow \{root\}$
4: $\mathcal{S} \leftarrow$ `select` ($\mathcal{S}$)
5: (q, w, d, l) $\leftarrow$ `expand` ($\mathcal{S}$)
6: `backpropagate` (q, w, d, l)
7: **end while**
8: $q \leftarrow$ `best` ($root$)
9: **return** q

Fig. 1. Classical MCTS algorithm main loop that returns a node q with the best move from a current board.

select ($\mathcal{S}$)

1: $q \leftarrow$ `last`($\mathcal{S}$)
2: **if** q has no child **then return** $\mathcal{S}$
3: $q_{best} \leftarrow$ `bestChild` (q)
4: **return** `select` ($\mathcal{S} + q_{best}$)

Fig. 2. Classical MCTS `select` function.

Fig. 3 presents a full MCTS expansion function. Usually, the selection is applied until positions of the tree are selected and then only one node is added. To improve the confidence in values in the tree, our expandion phase develop all children of the current node q_i, to get an accurate estimates of q_i. It starts with retrieving the node q_i that corresponds to following the sequence $\mathcal{S}$ from the $current\_board$. Then it checks the state of the game, and breaks if it is a win, a draw or a lost. A new result is also added to the current node (Fig. 3 lines 2 to 4). Otherwise new moves are considered from q_i, to add new nodes in the tree and to return sums of wins, draws and loses of all q_i children (Fig. 3 line 13). The `playout` function should distinguish issues of a random game. Then res should take its values in $\{win, draw, lost\}$. As presented in related works, the playout function can be enhanced with EPT and end-game databases. q_f scores that are always set to 1 here, can be enhanced with QBR.

expand ($\mathcal{S}$)

1: $q_i \leftarrow$ `apply` ($current\_board$, $\mathcal{S}$)
2: **if** q_i is a win **then return** (q_i, 1, 0, 0)
3: **if** q_i is a $draw$ **then return** (q_i, 0, 1, 0)
4: **if** q_i is a $lost$ **then return** (q_i, 0, 0, 1)
5: **for** `each move` m from q_i **do**
6: $new\_board \leftarrow$ `apply` (q_i, m)
7: $res \leftarrow$ `playout` (new_board)
8: **if** res is a win **then** $q_f \leftarrow (m, 1, 0, 0)$
9: **if** res is a $draw$ **then** $q_f \leftarrow (m, 0, 1, 0)$
10: **if** res is a $lost$ **then** $q_f \leftarrow (m, 0, 0, 1)$
11: add q_f as child of q_i
12: **end for**
13: **return** (q_i, $\sum win$, $\sum draw$, $\sum lost$)

Fig. 3. Classical MCTS `expand` function.

Fig. 4 presents the classical MCTS backpropagation function. As playout simulations have been achieved in children, N value is truly set to allow children to update their UCT value with the `setUctVal` function. The *upper confidence bounds* (Eq. (1)) is mostly used to define UCT values with win_i winning playouts at node i, N_i playouts at node i and N playouts at the parent. As presented in related works, this formula can be enhanced with IMB.

$$argmax_i \left(\frac{win_i}{N_i} + C\sqrt{\frac{log(N)}{N_i}} \right) \quad (1)$$

At last, the `best` function (Fig. 1 line 8) defines the child of $root$ that maximizes win_i/N_i.

backpropagate (q, w, d, l)

1: $q.N$ += $w + d + l$
2: $q.win$ += w
3: $q.draw$ += d
4: $q.lost$ += l
5: **for** `each child` e `of` q **do**
6: `setUctVal` (e)
7: **end for**
8: **if** $q.parent \neq \emptyset$ **then**
9: `backpropagate` ($q.parent$, $q.win$, $q.draw$, $q.lost$)
10: **end if**

Fig. 4. Classical MCTS `backpropagate` function.

To store more information in nodes, we added a game state information that contains the outcome of the game for optimal players. Corresponding to the common results of two-players games, we defined 3 values that are *WIN*, *DRAW* and *LOST*. The game tree is then composed with classical nodes and with nodes that hold game outcome information. As this information defines the optimal player outcome, our OPI-MCTS algorithm is carrying OPI-nodes. This allows us to settle an UCT improvements that will certainly play the optimal move according to the number of iterations performed from the current state. Improvements consist in exploiting OPI-nodes during selection, modifying UCT formula for OPI-nodes and taking into account OPI-nodes during backpropagation. Therefore, the main loop of our improved MCTS algorithm stops as soon as the root node is an OPI-node. In the whole process, the goal is to avoid the selection of OPI-nodes, to overestimate UCT values of *WIN* OPI-nodes, to underestimate UCT values of *DRAW* and *LOST* OPI-nodes.

B. Exploiting OPI-nodes

One way to exploit OPI information should be to add virtual win, draw or lost to these nodes. Such issue is not easy due to the UCT formula that contains an exploitation part and an exploration part, where OPI concerns only the exploitation part. On the other way, we choose to keep statistics as they are and just remove these nodes from the MCTS selection phase.

Fig. 5 presents the OPI-MCTS selection function. This function recursively selects the `bestNonOpiChild` node according to statistics while nodes have children. As a node with only OPI-nodes children must be an OPI-node, we are free to select nodes without checking children existence as in the classical MCTS selection function.

Fig. 6 presents the OPI-MCTS expansion function. As in the OPI selection function, we can retrieve q_i without checking that it corresponds to a terminal game state. Compared to the classical selection function, this one is thus slightly simplified.

select ($\mathcal{S}$)

1: $q \leftarrow$ `last`($\mathcal{S}$)
2: $q_{best} \leftarrow$ `bestNonOpiChild` (q)
3: **return** `select` ($\mathcal{S} + q_{best}$)

Fig. 5. OPI-MCTS `select` function that avoids OPI-nodes encountered.

expand ($\mathcal{S}$)

1: $q_i \leftarrow$ `apply` ($current\_board$, $\mathcal{S}$)
2: **for** `each move` m from q_i **do**
3: $new\_board \leftarrow$ `apply` (q_i, m)
4: $res \leftarrow$ `playout` (new_board)
5: **if** res is a win **then** $q_f \leftarrow (m, 1, 0, 0)$
6: **if** res is a $draw$ **then** $q_f \leftarrow (m, 0, 1, 0)$
7: **if** res is a $lost$ **then** $q_f \leftarrow (m, 0, 0, 1)$
8: add q_f as child of q_i
9: **end for**
10: **return** $(q_i, \sum win, \sum draw, \sum lost)$

Fig. 6. OPI-MCTS `expand` function with OPI-nodes.

C. OPI-nodes UCT values

According to the fact that OPI-nodes are not fairly considered in the selection phase, their final UCT-value is different at the end. As they are no more selected as soon as they are known as OPI-node, their exploration term is more important than their siblings. To fix it, we defined modifications of UCT formula to compute their final UCT value:

- For *WIN* OPI-nodes, we used Eq. (2). Their exploitation term is overestimated 100% wins that ranked them first.
- For *DRAW* OPI-nodes, we used Eq. (3). It corresponds to the average of the number of playout performed in their siblings. As we do not want to promote moves that lead to draw games, we restrict their UCT value to an exploration term, to make as much as undefined nodes more interesting than these nodes, even if they have low success rate.
- For *LOST* OPI-nodes, we used Eq. (4). Their previous *DRAW* OPI-nodes value is reduced by a constant factor that corresponds to a minimum success rate of classical nodes before behind worst than *DRAW* OPI-nodes.

$$1.0 + C\sqrt{\frac{log(N)}{N_i}} \tag{2}$$

$$C\sqrt{\frac{log(N - N_i + \frac{N-N_i}{T-1})}{\frac{N-N_i}{T-1}}} \tag{3}$$

$$C\sqrt{C_1\frac{log(N - N_i + \frac{N-N_i}{T-1})}{\frac{N-N_i}{T-1}}} \tag{4}$$

In Eq. (2), (3) and (4), T defines the number of siblings of the node i. Fig. 7 presents UCT-values for classical nodes,

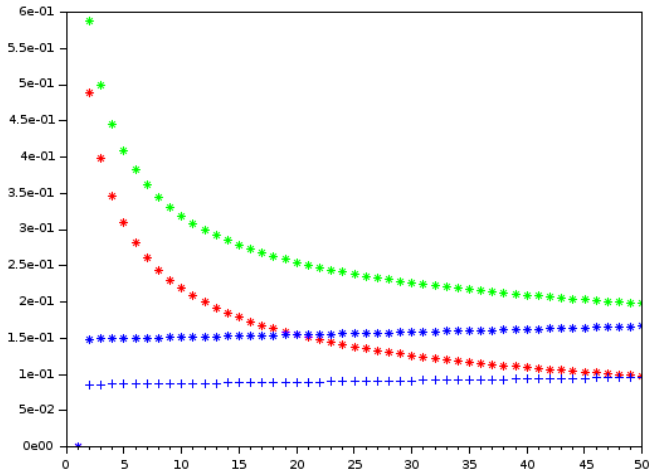

Fig. 7. Visualization of UCT values defined by Eq. (1), (3) and (4) for nodes with 10 siblings (*i.e.* $T = 10$) and various number of playouts. x-axis defines the number of playouts of the visualized node and y-axis shows the resulting UCT-value. Classical nodes (Eq. (1)) with a win success rate of 10% are in green, Classical nodes (Eq. (1)) with a win success rate of 0% are in red, *DRAW* OPI-nodes (Eq. (3)) are in blue * signs and *LOST* OPI-nodes (Eq. (4)) are in blue + signs. C_1 equals 0.33 in this case.

DRAW OPI-nodes, *LOST* OPI-nodes in similar situation. Their number of siblings is 10 and the number of playouts of their parent is 200 (that makes an average of 20 playouts per child). Green nodes are classical nodes with a success rate of 10% and red nodes are without win. Before the average of playouts per sibling, classical nodes are always better than OPI-nodes. *LOST* OPI-nodes are worst than *DRAW* OPI-nodes according to C_1 constant value.

D. OPI-node backpropagation

As we always develop all the children of a node during the expansion phase, it allows us to define the parent status by analysing children status. According to Eq. (2), (3) and (4), we backpropagate a virtual number of playouts from *DRAW* and *LOST* OPI-nodes that tries to avoid these nodes. As defined in Eq. (5), this virtual number of playouts is the maximum between the number of playouts N_i and the average number of playouts achieved in T sibling nodes. This number of virtual simulations is backpropagate until the root node.

$$max\left(N_i, \frac{N - N_i}{T - 1}\right) \tag{5}$$

Fig. 8 presents the OPI-MCTS backpropagation function for w wins, d draws and l losts at a node q . Even if UCT-values are backpropagated towards parent, OPI-value can be backpropagated to parent depending on logical conditions (lines 11 to 18):

- If one child is an *WIN* OPI-node, then the parent is also a *WIN* OPI-node (line 11).
- If all children are *LOST* OPI-nodes, then the parent is also a *LOST* OPI-node (line 13).
- If all children are *DRAW* OPI-nodes, then the parent is also a *DRAW* OPI-node (line 15).

- If all children are *LOST* and *DRAW* OPI-nodes, then the parent is also a *DRAW* OPI-node (line 17).

backpropagate (q, w, d, l)

1: $q.N$ += $w + d + l$
2: $q.win$ += w
3: $q.draw$ += d
4: $q.lost$ += l
5: $\mathcal{E} \leftarrow \emptyset$
6: **for** `each child` e `of` q **do**
7: `setUctVal` (e)
8: $\mathcal{E} \leftarrow \mathcal{E} + e$
9: **end for**
10: **if** $q.parent \neq \emptyset$ **then**
11: **if** $\exists e, e \in \mathcal{E} \wedge e$ is a *WIN* OPI-node
12: $q.parent \leftarrow$ *LOST* OPI-node
13: **if** $\forall e \in \mathcal{E}$, e is a *LOST* OPI-node
14: $q.parent \leftarrow$ *WIN* OPI-node
15: **if** $\forall e \in \mathcal{E}$, e is a *DRAW* OPI-node
16: $q.parent \leftarrow$ *DRAW* OPI-node
17: **if** $\forall e \in \mathcal{E}$, e is a *LOST* or *DRAW* OPI-node
18: $q.parent \leftarrow$ *DRAW* OPI-node
19: `backpropagate` ($q.parent$, $q.win$, $q.draw$, $q.lost$)
20: **end if**

Fig. 8. OPI-MCTS `backpropagate` function with OPI-nodes.

IV. Experiments

To compare MCTS and OPI-MCTS on CDC, we present results with four boards (Fig. 9) and with classical statistics of tournament between algorithms. All the pieces used are ⑦, ⑥, ⑤, ④, ③, ②, ①, ❼, ❻, ❺, ❹, ❸, ❷, ❶ where it corresponds respectively to pawn, cannon, knight, rook, minister, guard and king for each color (*e.g.* ⑦ is white pawn and ① is white king). ❍ represents any unrevealed piece. For all these experiments, we used an exploration constant K of 0.3. In the first board (Fig. 9a), we have limited the classical MCTS with the number of iterations used by OPI-MCTS to get a result. Usually the classical MCTS algorithm is always spending all allocated iterations. In the second board (Fig. 9b), we allowed 10[sec.] to give the best move. This query has been achieved for each color. In the third board (Fig. 9c), we allowed 10k iterations for each algorithm and in the fourth board (Fig. 9d), we allowed 20k iterations for each algorithm. At last, we performed classic games between MCTS and OPI-MCTS. We played 200 games and we allowed 300[sec.] per game. While playing in these games, we allowed to spend 3% of the remaining time, thereby varying from 9 to a little less than 1[sec.].

The first board (Fig. 9a) is holding 2 pieces (one ② for the red-player and one ❷ for the black-player). The starting position of the black piece is fixed on the top right corner at $d-8$ and the starting position of the red piece is varying between positions annoted $p1$ to $p14$. The only position recorded in the game is the starting position. Then pieces can move everywhere on the board, according to the rules (including 3-cycle restriction). Each position has been tested for red-player turn with MCTS and OPI-MCTS. Results are respectively presented in Fig. 10 and Fig. 11. According to positions, columns of Fig. 10 and Fig. 11 give the number of iterations needed, the number of nodes added in the tree, the final best red-player move, the corresponding value, the depth of the tree and the length of the best sequence after this move. In the first, MCTS corresponding value is the classical UCT-value.

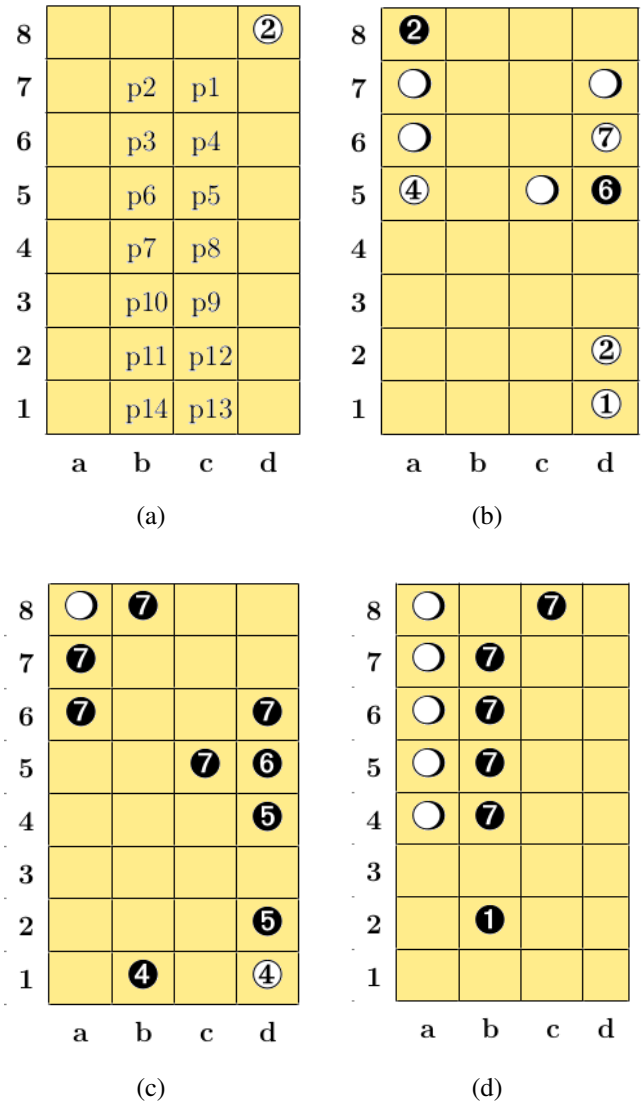

Fig. 9. Board examples for performance analysis from different positions with few pieces. Unrevealed pieces of board (b) are ❷ ❸ ❸ ⑤. Unrevealed piece of board (c) is a ②. Unrevealed pieces of (d) are five red pawns ⑦.

In the second board (Fig. 9b) , OPI-MCTS corresponding value is the game outcome or the modified UCT-value. On this first board, Fig. 10 and Fig. 11 shows that even if positions are close, their endings greatly differ in depth and in the outcome (that oscillates between WIN and LOST). Beyond giving guaranted termination, OPI-MCTS explores more nodes than classical MCTS. It shows that MCTS results are better for winning situations than for losing onces. MCTS tree depth is smaller than OPI-MCTS tree depth. Sizes of the sequence that correspond to best moves are shorter in MCTS than in OPI-MCTS. It could explain the failures of MCTS to find the optimal move.

The second board (Fig. 9b) is holding move pieces, for both sides and also with some unrevealed pieces (that are ❷

pos.	iter.	nodes	best	val.	depth	seq.
p1	3	11	d8-d7	0.204	2	2
p2	3926	2142	d8-d7	0.940	13	7
p3	83	238	d8-d7	0.063	7	6
p4	4106	1836	d8-d7	0.931	11	3
p5	123	300	d8-d7	0.056	7	4
p6	16317	9987	d8-d7	0.894	15	12
p7	392	889	d8-d7	0.035	10	6
p8	17784	10938	d8-d7	0.878	15	7
p9	4440	4054	d8-d7	0.015	14	10
p10	30815	30421	d8-d7	0.823	19	11
p11	20026	23622	d8-d7	0.007	19	14
p12	27062	31275	d8-d7	0.774	19	11
p13	59294	53609	d8-c8	0.005	21	12
p14	14866	19947	d8-d7	0.754	18	11

Fig. 10. Classical MCTS evaluates of Fig. 9a black piece positions when it is red-player turn and there is a ❷ placed between p1 and p14.

pos.	iter.	nodes	best	val.	depth	seq.
p1	3	11	d8-d7	LOST	2	2
p2	3926	12979	d8-d7	WIN	22	16
p3	83	272	d8-c8	LOST	6	6
p4	4106	13933	d8-d7	WIN	21	19
p5	123	380	d8-c8	LOST	11	8
p6	16317	55095	d8-d7	WIN	27	23
p7	392	1283	d8-c8	LOST	12	12
p8	17784	59714	d8-d7	WIN	26	17
p9	4440	14575	d8-c8	LOST	26	22
p10	30815	103143	d8-d7	WIN	27	23
p11	20026	66162	d8-c8	LOST	26	20
p12	27062	89980	d8-d7	WIN	25	17
p13	59294	198736	d8-d7	LOST	24	22
p14	14866	49405	d8-d7	WIN	23	17

Fig. 11. OPI-MCTS evaluates of Fig. 9a black piece positions when it is red-player turn and there is a ❷ placed between p1 and p14.

col.	iter.	nodes	best	val.	depth	seq.
MCTS						
Red	9671	102227	a5-a4	0.59	9	5
Black	15830	145283	d5-d1	0.75	9	8
OPI-MCTS						
Red	9940	105088	a5-a4	0.59	9	4
Black	14936	136490	d5-d1	0.75	9	4

Fig. 12. MCTS and OPI-MCTS evaluates of Fig. 9b for black and red moves.

col.	iter.	nodes	best	val.	depth	seq.
MCTS						
Fig 9c	10k	95362	b8-c8	0.80	13	11
Fig 9d	20k	182776	c8-b8	0.93	13	13
OPI-MCTS						
Fig 9c	748	6510	a6-b6	WIN	7	4
Fig 9d	13936	127207	c8-b8	WIN	11	10

Fig. 13. MCTS and OPI-MCTS evaluates of Fig. 9c and Fig. 9d for black player.

❸ ❸ ⑤). Results are presented in Fig. 12 for each color. The difference between Red and Black comes from initial situations that are different for the two players. Black has less moves at the beginning due to the number of unrevealed pieces and more limited moves due to the position of its cannon (piece ❻ in the middle) and its guard (piece ❷ on top right corner). Results are similar for MCTS and OPI-MCTS, even in the number of iterations and nodes.

Third and fourth boards (Fig. 9c and 9d) are boards that can be solved with backtracking captures. In the third, the remaining piece is ② and black is to play. In the fourth, the remaining pieces are ⑦ ⑦ ⑦ ⑦ ⑦ and black is to play. Results of MCTS and OPI-MCTS are presented in Fig. 13. It shows that OPI-MCTS gets the right moves (and knows that it is a winning move) where MCTS gets a non-optimal move (In Fig. 9c, this move will allow to play one time more than a6-b6 for example) and where MCTS gets the right move with a rating success of 70%. As presented in the last column of Fig. 13, after playing the suggested move $a6-b6$ on the third board, a possible winning sequence of length 4 should be `a8; d2-c2; d1-d2; d5-d2;` and black wins as white has only one cannon ② without possible move.

At last, we achieved matches of MCTS-OPI agains MCTS. Results are presented in Fig 14. Independently of who is the first player, OPI-MCTS wins 63% against MCTS and lost 30% in average. The number of lost are reduced to get draw games when OPI-MCTS is the first player.

	win	lost	draw
As first player	128	46	26
As second player	124	74	2

Fig. 14. OPI-MCTS results against classical MCTS in 400 games of 300[sec.].

V. Conclusion

We have presented a new algorithm called OPI-MCTS that exploit termination information to guide MCTS selection and expansion. It comes from adding game state information to nodes and to apply a logical backpropagation of this game state information. As selection avoids known termination nodes, the expansion is fairly turned towards promising nodes and OPI-nodes are pruned during MCTS playouts. As possible, OPI-nodes are exploited to provide the final best move. As OPI-MCTS is general, it could be applied to many other games. OPI-MCTS improves classical MCTS as it wins 63% against a classical MCTS with common CDC tournament settings. It should provide better results by combining it with other improvements like EPT.

References

[1] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, *A Survey of Monte Carlo Tree Search Methods*. IEEE Trans. on Computational Intelligence and AI in Games, Vol. 4, num. 1, pp. 1–43 (TCIAIG-2012).

[2] L. Kocsis and C. Szepesvari, *Bandit based Monte-Carlo Planning*. 17th Euro. Conf. on Machine Learning, Springer, 2006, pp. 282–293 (ECML-2006).

[3] P. Auer, N. Cesa-Bianchi and P. Fischer, *Finite-time Analysis of the Multiarmed Bandit Problem*. Machine Learning, Springer, 2002, Vol. 47, num. 2-3, pp. 235–256.

[4] N. Jouandeau, *Varying Complexity in CHINESE DARK CHESS Stochastic Game*. TCGA Workshop 2014, pp. 86–90 (TCGA-2014).

[5] M.H.M. Winands, Y. Bjornsson, J.T. Saito, *Monte Carlo Tree Search Slver*. Computers and Games, 5131, pp. 25–36 (CG-2008).

[6] C-H. Hsueh, I-C. Wu, W-J. Tseng, S.J. Yen and Jr-C. Chen, *Strength Improvement and Analysis for an MCTS-Based Chinese Dark Chess Program*. 14th Int. Conf. Advances in Computer Games (ACG-2015).

[7] R. Lorentz, *Early Playout Termination in MCTS*. 14th Int. Conf. Advances in Computer Games (ACG-2015).

[8] M. Lanctot, M.H.M. Winands, T. Pepels and N.R. Sturtevant, *Monte Carlo Tree Search with Heuristic Evaluations Using Implicit Minimax Backups*. IEEE Int. Conf. on Computational Intelligence and Games, pp. 1–8 (CIG-2014).

[9] T. Pepels, M.J. Tak, M. Lanctot and M.H.M. Winands, *Quality-Based Rewards for Monte-Carlo Tree Search Simulations*. 21st Euro. Conf. on Artificial Intelligence., pp. 1–6 (ECAI-2014).

[10] B-N. Chen, B-J. Shen and T-S. Hsu, *Chinese Dark Chess*. ICGA Journal, 2010, Vol. 33, num. 2, pp. 93–106.

[11] H. Baier and M.H.M. Winands, *Monte-Carlo Tree Search and minimax hybrids*. IEEE Int. Conf. on Computational Intelligence in Games, pp. 1–8 (CIG-2013).

[12] M. Chee, A. Saffidine and M. Thielscher, *A Principled Approach to the Problem of Chunking in UCT*. IJCAI Computer Games Workshop 2015, Springer CCIS, pp. 1–14 (IJCAI-CGW-2014).

[13] S-J. Yen, C-W. Chou, Jr-C. Chen, I-C. Wu and K-Y. Kao, *The Art of the Chinese Dark Chess Program DIABLE*. Advances in Intelligent Systems and Applications, Springer Berlin Heidelberg, 2013, Vol. 1, pp. 231–242.

[14] S-J. Yen, C-W. Chou, J-C. Chen, I-C. Wu, K-Y. Kao, *Design and Implementation of Chinese Dark Chess Programs*. IEEE Trans. on Computational Intelligence and AI in Games, Vol. 7, num. 1, pp. 1–9 (TCIAIG-2014).

[15] M. Lanctot, A. Saffidine, J. Veness, C. Archibald and M. Winands, *Monte Carlo *-Minimax Search*. 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI-2013).

[16] N. Jouandeau and T. Cazenave, *Small and Large MCTS Playouts Applied to Chinese Dark Chess Stochastic Game*. ECAI Computer Games Workshop 2014, Springer CCIS, Vol. 504, pp. 78–90 (ECAI-CGW-2014).

[17] N. Jouandeau and T. Cazenave, *Monte-Carlo Tree Reductions for Stochastic Games*. 19th Conf. on Technologies and Applications of Artificial Intelligence, Springer LNCS, Vol. 8916, ISBN 978-3-319-13986-9, pp. 228–238 (TAAI 2014).

[18] H-J. Chang and T-S. Hsu. *A Quantitative Study of 24 Chinese Dark Chess*. 8th Int. Conf. on Computers and Games, Springer, Vol. 8427, pp. 151–162 (CG-2013).

[19] B-N. Chen and T-S. Hsu, *Automatic Generation of Chinese Dark Chess Opening Books*. 8th Int. Conf. on Computers and Games, Springer, Vol. 8427, pp. 221–232 (CG-2013).

[20] Jr-C. Chen, T-Y. Lin, S-C. Hsu and T-S. Hsu, *Design and Implementation of Computer Chinese Dark Chess Endgame Database*. TCGA Computer Game Workshop, pp. 5–9 (TCGA-2012).

[21] A. Saffidine, N. Jouandeau, C. Buron and T. Cazenave, *Material Symmetry to Partition Endgame Tables*. 8th Int. Conf. on Computers and Games, Springer, Vol. 8427, pp. 221–232 (CG-2013).

[22] Jr-C. Chen, T-Y. Lin, B-N. Chen and T-S. Hsu, *Equivalence Classes in Chinese Dark Chess Endgames*. IEEE Trans. on Computational Intelligence and AI in Games, Vol. 7, num. 2, pp. 109–122 (TCIAIG-2014).