Monte-Carlo Tree Reductions for Stochastic Games

Nicolas Jouandeau, Tristan Cazenave

Monte-Carlo Tree Reductions for Stochastic Games

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 (MCTS) is a powerful paradigm for perfect information games. When considering stochastic games, the tree model that represents the game has to take chance and a huge branching factor into account. As effectiveness of MCTS may decrease in such a setting, tree reductions may be useful. Chance-nodes are a way to deal with random events. Move-groups are another way to deal efficiently with a large branching factor by regrouping nodes. Group-nodes are regrouping only reveal moves and enable a choice between reveal moves and classical moves. We present various policies to use such reductions for the stochastic game Chinese Dark Chess. Move-groups, chance-nodes and group-nodes are compared.

1 Introduction

Chinese Dark Chess (CDC) is a popular stochastic two player game in Asia that is most commonly played on a 4x8 rectangular board where players do not know payoff of reveal moves. The 2 players (called black and red) start with the same set of pieces. Before the first move, players do not know their colors. The first player move defines the first player color. Then pieces can capture other pieces according to their values and their positions. Even if reveal moves imply a huge number of possible boards, classical moves can lead to similar positions during the game and capturing rules are different for each piece [7]. As Monte Carlo Tree Search (MCTS) techniques deal with nodes statistics, blindness goes along with branching factor. MCTS programs seem to be promising in CDC. In TCGA 2012, one participant was a MCTS program. In TCGA 2013, five participants, including the winner called *DarkKnight*, were MCTS programs. In TCGA 2014, the alpha-beta program *Yahari* won the competition ahead of *DarkKnight*. As CDC has a huge branching due to the revealing moves, we try to reduce the revealing moves dependency by applying different ways of regrouping nodes. In the context of stochastic games, we believe that a better understanding of MCTS behavior is needed. We show in this paper 3 different MCTS implementations, called move-groups, chance-nodes and group-nodes, that are using longer playouts, the same playout policy and no heuristic playouts.

Section 2 describes related works. Section 3 presents move-groups, chance-nodes and group-nodes principles applied to MCTS algorithms and different regrouping policies. Section 4 shows experimental results. At the end, section 5 concludes.

2 Related works

In this section we expose related works on creating nodes and regrouping them in CDC and in stochastic games.

Most previous works related to CDC consider openings building [2], endgames building [3–5], sub-problems resolution [6]. Due to a long expertise in *alpha-beta*, most programs use the minimax extension to games of chance called *expecti-max* [7] with its common pruning extensions *Star1* and *Star2* [8]. It remains that MCTS programs are highly sensitive to their parameters [1].

Move-groups have been proposed in [10] 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 [11] 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 [12].

Chen *et al.* [7] used *alpha-beta* algorithm with different revealing policies combined with the *initial-depth flipping* method to reduce the branching factor.

Yen *et al.* [1] presented a non-deterministic MCTS with chance-nodes [13]. They create shorter simulation by moderating the three policies named *Capture First*, *Capture Stronger Piece First* and *Capture and Escape Stronger Piece First*.

Jouandeau and Cazenave [9] presented MCTS influence of various playout size, of basic or advanced policies and with heuristic playouts. They studied group constitution related to position's pieces. They showed that relevant playout sizes, policies and heuristic playouts are not equivalent for chance-nodes and group-nodes.

More generally, MCTS has been successfully applied in the past to other multi-player perfect information stochastic games with different regrouping optimizations.

In Backgammon, Monte Carlo playouts have been parallelized to evaluate a position. As playing a position depends on dices, random dices sequences have been also evaluate in parallel. As the number of possible moves increases when dices are doubles, states evaluation can be divided in 2 sub-problems [14] : the first without double dices and the second with double dices. In other words, they distinguished states in 2 groups : light states that have small branching factor and heavy states that have huge branching factor.

In Scrabble, simulations are restricted inside a directed acyclic graph to produce existing words [15].

In Poker, betting strategies depends on the gamestate [16] (that can be Pre-Flop, Flop, River). During these gamestates, players' hands are consistent with their actions. Thus simulations are limited to special tracks that are defined by players' hands.

These contributions in other stochastic games show that biased sampling according to particular things of gamestates and regrouping possibilities has been used to settle efficient MCTS playouts.

3 Regrouping nodes

In this section, we present the use of chance-nodes, move-groups and group-nodes principles applied to MCTS.

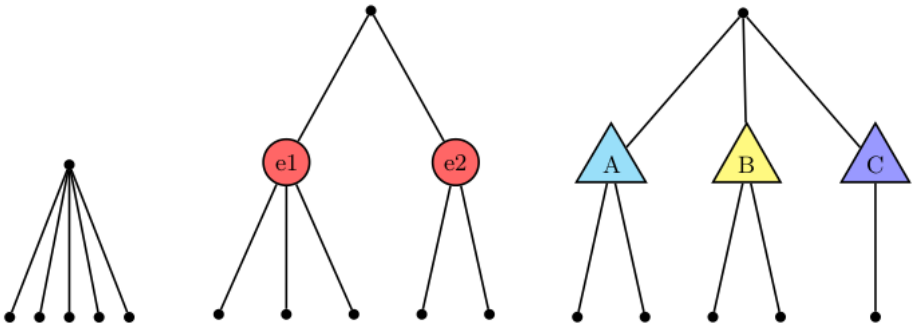

Fig. 1: Common, chance-nodes and move-groups representations.

Figure 1 show the differences between representations of a common tree, a tree with chance-nodes and a tree with move-groups. With chance-nodes and move-groups, branching factor reductions can arise. With chance-nodes (related to a game where possible moves are partly defined by rolling a dice), events *e1* and *e2* are selecting possible next nodes. With move-groups, children are divided between categories (here 3 categories A, B and C) to perform a smart descent towards the best leaf. When these categories are defined by the moves coordinates in the board, move-groups are called group-nodes.

The main loop of MCTS applied to perfect information games is presented in Alg. 1. It is a statistical search based on 3 steps called *selection*, *expansion* and *backpropagation*, also guided by random games (*i.e.* playouts). The tree expansion is performed to evaluate the current most promising node. The main loop presented is limited to *nbPlayouts* iterations but it can be an anytime process. It can start from an empty tree with its root node only or from a tree

filled by previous MCTS loops. The use of such procedure is consistent if and only if the root node is not a *endgame* position (*i.e.* root is not a solved problem). This process can lead to the insertion of 1 to *nbPlayouts* nodes. The `select` function takes the current tree $\mathcal{T}$ as input and performs descent toward the best node q and returns the move m to produce a new node. Then the `expand` function applies m to q and produces q_{new}. Starting from this new node q_{new}, a playout helps to collect statistical information of q_{new}. Then `backpropagation` function insert q_{new} in the tree $\mathcal{T}$ as statistically informed node and backpropagates q_{new} results of its parents. If during the descent, this q_{new} node is a winning leaf, this leaf is updated and future tree expansions are expected to perform different descents and select different nodes. According to this exclusion of endgames inside $\mathcal{T}$, the `select` function detailed in Alg. 2 always selects node q with at most 1 move. It selects the best node q_{best} from all possible moves m from q or it breaks at the first node $(q + m)$ not in $\mathcal{T}$.

input : $\mathcal{T}$ the current tree
output: $\mathcal{T}$ expanded with 1 to *nbPlayouts* nodes

for *nbPlayouts* **do**
  (q , m) ← `select` ($\mathcal{T}$) ;
  q_{new} ← `expand` (q , m) ;
  res ← `playout` (q_{new}) ;
  `backpropagate` (q_{new} , res) ;

Algorithm 1: Classical MCTS applied to perfect information games.

input : $\mathcal{T}$ the current tree
output: (q, m) where $q \in \mathcal{T}$ and q to expand by applying a move m

$q \leftarrow root$;
while *true* **do**
  $q_{best} \leftarrow \{\emptyset\}$;
  foreach *move m from q* **do**
    if $(q + m) \notin \mathcal{T}$ **then return** (q, m) ;
    q_{best} ← `best` (q_{best} , $(q + m)$) ;
  $q \leftarrow q_{best}$;

Algorithm 2: `Select` function of classical MCTS.

Considering stochastic games and modifications of tree's representation that are arising with chance-nodes and move-groups usage :

– the `select` function may distinguish types of nodes.

– the endgame shortcut assertion is no more true. Thus endgames can be selected inside `select` function. It implies modifications in main loop and in `select` function.

```
input  : 𝒯 the current tree
output: 𝒯 expanded with nbPlayouts iterations
for nbPlayouts do
    ( q , m ) ← select ( 𝒯 ) ;
    if m ≠ {∅} then
        q_new ← expand ( q , m ) ;
        res ← playout ( q_new ) ;
        backpropagate ( q_new , res ) ;
    else
        res ← result ( q ) ;
        backpropagate ( q , res ) ;
```

Algorithm 3: MCTS applied to stochastic games.

The modified main loop is presented in Alg. 3. Even if a descent can lead to an endgame, stochastic games can lead to different events during the descent, that should lead to unevaluated parts of $\mathcal{T}$. The *selection* process is guided by nodes scores and by stochastic events. Thus the statistical scoring process can be applied systematically during *nbPlayouts* iterations, to insert 0 to *nbPlayouts* nodes in $\mathcal{T}$. In such games, *expansion*, *playout* and *backpropagation* are applied only if the selected move m differs from $\{\emptyset\}$.

The modified `select` function with chance-nodes is presented in Alg. 4. At each iteration, the state of the node is checked : if it is a chance-node, then the `dice` function adds an external event to q. If no move is available from q, then $\{\emptyset\}$ is returned as move m to apply. If q is not in $\mathcal{T}$, then q is inserted and the move returned to apply to q is its first move. Thus each time this node is selected, it will try to add another move from q before looking for the best children of q to descend one more time in $\mathcal{T}$.

The modified `select` function with move-groups is presented in Alg. 5. The process is divided between 3 cases :

– there is *no move from* q, that is equivalent to no group available. The process breaks and the tuple (q , $\{\emptyset\}$) is returned.
– 1 *move-group* g *exists from* q, which means that this group contains at least 1 move. In this case, the process tries to evaluate all possible moves of the move-group g.
– if 2 or more move-groups exists from q, then the process tries to select the first group without a move. If all groups have 1 or more moves, then the best group $\mathcal{M}_{best}$ is selected and 1 move from this group is considered. If this move is not in $\mathcal{T}$, then it returns the tuple (q , m). This process implies to

```
input  : 𝒯 the current tree
output: ( q, m ) where q ∈ 𝒯 and q to expand by applying a move m
q ← root;
while true do
    if q is a chance-node then q ← q + dice ( ) ;
    else
        if no move from q then break ;
        if  q ∉ 𝒯 then
            insert q in 𝒯 ;
            return ( q , first move from q ) ;
        else
            q_best ← {∅} ;
            foreach move m from q do
                if (q + m) ∉ 𝒯 then return ( q, m ) ;
                q_best ← best ( q_best , (q + m) ) ;
            q ← q_best ;
return ( q , {∅} ) ;
```

Algorithm 4: `Select` function with chance-nodes.

```
input  : 𝒯 the current tree
output: ( q, m ) where q ∈ 𝒯 and q to expand by applying a move m
q ← root;
while true do
    if no move from q then break ;
    if only 1 move-group g exists from q then
        q_best ← {∅} ;
        foreach move m of g from q do
            if (q + m) ∉ 𝒯 then return ( q, m ) ;
            q_best ← best ( q_best , (q + m) ) ;
        q ← q_best ;
    else
        ℳ_best ← {∅} ;
        foreach move-group ℳ from q do
            if ℳ ∩ 𝒯 = ∅ then
                add one-move of ℳ in 𝒯 ;
                return ( q , m ) ;
            ℳ_best ← best ( ℳ , ℳ_best ) ;
        m ← one-move of ℳ_best ;
        if (q + m) ∉ 𝒯 then return ( q, m ) ;
        q ← q + m ;
return ( q , {∅} ) ;
```

Algorithm 5: `Select` function with move-groups

```
input  : 𝒯 the current tree
output: ( q, m ) where q ∈ 𝒯 and q to expand by applying a move m
q ← root;
while true do
    if no move from q then break ;
    q_best ← {∅} ;
    foreach possible move m from q do
        if m is a classical move then
            if (q + m) ∉ 𝒯 then return ( q, m ) ;
            q_best ← best ( q_best , (q + m) ) ;
        else if m is a reveal move then
            q_new ← revealRandomlyAt ( q, m );
            if q_new ∉ 𝒯 then return ( q, m ) ;
            q_best ← best ( q_best , q_new ) ;
    q ← q_best ;
return ( q , {∅} ) ;
```

Algorithm 6: `Select` function with group-nodes

check the intersection between a move-group and $\mathcal{T}$. The `one-move` function defines the policy to generate and add new moves in $\mathcal{T}$.

The modified `select` function with group-nodes is presented in Alg. 6. Revealing moves are regrouped by position. Each position to reveal leads to different boards. Thus a board with 3 known pieces with 4 possible moves each and with 10 unrevealed pieces will have 12 children for its known pieces and 10 children for its unrevealed pieces. As revealing positions can leads to different boards, possible moves are always recomputed with group-nodes. The `select` function returns the first unevaluated classical move or the first unevaluated reveal move from the current best node in the tree. The function `revealRandomlyAt` applies a random reveal at the position m. As revealed pieces will be different, subgroups will be also different. Thus the group-nodes regrouping policy produced an approximate evaluation of groups.

In this paper, we investigate the way that groups constitution influence MCTS performances in CDC stochastic game. To achieve this, we consider different regrouping policies and different generating policies inside groups:

- revealed group or unrevealed group : these 2 groups are simply defined on the board by revealed and unrevealed pieces. Using these 2 groups, we tried to generate randomly new moves (abrev. *move-group-random*) and to cycle over the considered move-group's elements (abrev. *move-group-cycle*).
- revealed pieces or unrevealed group : this is equivalent to group-nodes. Unrevealed pieces are considered randomly inside the unrevealed group and revealed pieces are considered individually (abrev. *group-nodes*).

4 Experiments

In the first experiment, we compare the 5 regrouping policies *move-groups-random*, *move-groups-cycle-R*, *move-groups-cycle-M*, *group-nodes* and *chance-nodes* to a random player and to a reference player *rand-mm*. The policies *move-groups-cycle-R* and *move-groups-cycle-M* are respectively starting by reveal moves and by real moves. Thus *move-groups-cycle-R* is more dependant on the number of revealing possibilities than *move-groups-cycle-M*. The reference player *rand-mm* simply plays randomly when pieces are unrevealed and otherwise applies minimax to find the best move. In our experiments, UCT (abrev. Upper Confidence bounds applied to Trees) values are computed with

$$(v/(v+d)) + \sqrt{K * log(n)/(v+d)}$$

with n simulations, v wins, d losses and K equals to 0.3. As capture has been proven to contribute in better MCTS evaluations [1, 9], captures are preferred to random moves inside playouts.

As playouts can finish with a draw endgame and are evaluated without heuristic function, we extended the draw rule inside playouts to 640 turns to produce more informed playouts. Results presented in all tables involve 500 games in which half are achieved with one player as first and half are achieved with the other player as first. Games are played with 0.01[sec] per move and with 1[sec] per move.

Policy	Against *rand*			Against *rand-mm*		
	win	lost	draw	win	lost	draw
with 0.01[sec] per move						
move-groups-random	194	0	306	90	95	315
move-groups-cycle-R	81	0	419	100	400	0
move-groups-cycle-M	202	0	298	100	150	250
group-nodes	314	0	186	1	238	261
chance-nodes	360	0	140	191	13	296
with 1[sec] per move						
move-groups-random	291	0	209	140	42	318
move-groups-cycle-R	64	0	436	0	437	63
move-groups-cycle-M	282	0	218	205	14	281
group-nodes	353	0	147	49	15	436
chance-nodes	393	0	107	249	3	248

Table 1: Games against random-player and random-minimax player.

Results of table 1 confirm that the *move-groups-cycle-R* policy of cycling on moves and starting by reveal moves is not a good policy. As similar results are obtained with *move-groups-random* and *move-groups-cycle-M*, the knowledge introduced with cycling and starting by known pieces is inefficient to do better than a random selection. Results show that *chance-nodes* are more effective

than others with simple playouts (*i.e.* no heuristic evaluation function inside playouts).

Policy	Against *rand-mm*		
	win	lost	draw
with 0.01[sec] per move			
move-group-random-mm	169	1	330
move-group-cycle-R-mm	186	2	312
move-group-cycle-M-mm	166	0	334
group-nodes-mm	24	5	471
chance-nodes-mm	240	0	260
with 1[sec] per move			
move-group-random-mm	265	0	235
move-group-cycle-R-mm	290	0	210
move-group-cycle-M-mm	293	0	207
group-nodes-mm	121	0	379
chance-nodes-mm	310	0	190

Table 2: Using minimax at the end.

In the second experiment, we enhanced these 5 policies by using minimax as the reference player *rand-mm* do. When all pieces are revealed, enhanced players apply minimax to find the best move. Policies are used when pieces are unrevealed and otherwise a minimax search is done. These modified players *move-groups-random-mm*, *move-groups-cycle-R-mm*, *move-groups-cycle-M-mm*, *group-nodes-mm* and *chance-nodes-mm* are compared to *rand-mm* player.

Results of table 2 show that adding minimax search in the perfect information part of games improves all the players. This enhancement makes *chance-nodes-mm* the best player with 0.01[sec] and 1[sec] per move. In case of 1[sec] per move, *move-groups-cycle-R-mm* and *move-groups-cycle-M-mm* are closed to *chance-nodes-mm*. In all these experiments, *rand-mm* obtains quasi null scores.

In the third experiment, we evaluate the contribution of chance-nodes by gradually introducing random moves. The figure 2 shows the performance of *chance-nodes-mm* against himself. The second player is weakened with a random move during X first turns. Players are evaluated in 500 games, with 1[sec] per move, from 0 to 35 random moves. It shows that performances are equal when *chance-nodes-mm* plays randomly during its 10 first moves. After 10 first random moves, losts increase and draws decrease. It shows chance-nodes contribution while some unrevealed pieces remain. After 20 turns, the game has more chance to be fully revealed. It shows that similar gain is achieved in the perfect information part of the game. As there are 32 unrevealed positions at the beginning of the game, *chance-nodes* contributes effectively at least in managing 12 unrevealed pieces.

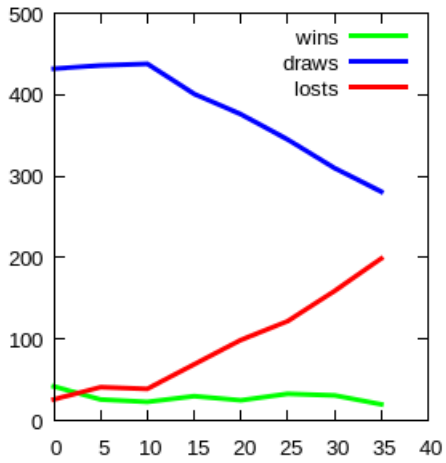

Fig. 2: *chance-nodes-mm* with X random plies against *chance-nodes-mm*.

5 Conclusion

Monte-Carlo Tree Search (MCTS) is a powerful paradigm for perfect information games. When considering stochastic games, the tree model that represents the game has to take chance and a huge branching factor into account. We have presented 3 ways to regroup nodes and their consequences to MCTS algorithm and the descent function. We have compared different regrouping policies and different generating policies in Chinese Dark Chess games. Experiments show that without heuristic function evaluation, *chance-nodes* regrouping policy is the best for the stochastic part of the game and that adding minimax search in the perfect information part of the game improves all players.

References

1. 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*, Int. Computer Symposium (ICS-2012).
2. B-N. Chen and T-S. Hsu, *Automatic Generation of Chinese Dark Chess Opening Books*, 8th Int. Conf. on Computers and Games (CG-2013).
3. 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).
4. A. Saffidine and N. Jouandeau and C. Buron and T. Cazenave, *Material Symmetry to Partition Endgame Tables*, 8th Int. Conf. on Computers and Games (CG-2013).
5. 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 (TCIAIG-2014).
6. H-J. Chang and T-S. Hsu. *A Quantitative Study of 24 Chinese Dark Chess*, 8th Int. Conf. on Computers and Games (CG-2013).
7. B-N. Chen and B-J. Shen and T-S. Hsu. *Chinese Dark Chess*. ICGA Journal, 2010, vol. 33, num. 2, pp. 93.
8. B.W. Ballard. *The *-minimax search procedure for trees containing chance nodes*. Artifical Intelligence, 21:327350, 1983.
9. N. Jouandeau and T. Cazenave. *Small and large MCTS playouts applied to Chinese Dark Chess stochastic game*, ECAI 3th Int. Computer Games Workshop 2014 (CGW-2014).
10. 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).
11. 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 (TCIAIG-2012).
12. G. Van Eyck and M. Müller, *Revisiting move groups in monte-carlo tree search*, Advances in Computer Games, pp 13–23, (ACG-2012).
13. 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).
14. F. Van Lishout and G. Chaslot and J.W.H.M. Uiterwijk, *Monte-Carlo Tree Search in Backgammon*, Int. Computer Games Workshop, pp 175–184, (CGW-2007).
15. B. Sheppard, *World-championship-caliber Scrabble*, Artificial Intelligence, vol 134, pp 241–275, 2002.
16. D. Billings and A. Davidson and J. Schaeffer and D. Szafron, *The Challenge of Poker*, Artificial Intelligence, vol 134, pp 201–240, 2002.