Exploiting Game Decompositions in Monte Carlo Tree Search

Aline Hufschmitt, Jean-Noël Vittaut, Nicolas Jouandeau

Exploiting Game Decompositions in Monte Carlo Tree Search

Aline Hufschmitt[1], Jean-Noël Vittaut[2], and Nicolas Jouandeau[2]

[1] CREC Saint-Cyr, Écoles de Coëtquidan, Guer, France
aline.hufschmitt@st-cyr.terre-net.defense.gouv.fr
[2] Computer Science Lab, University Paris 8, France
{jnv,n}@up8.edu

Abstract. In this paper, we propose a variation of the MCTS framework to perform a search in several trees to exploit game decompositions. Our Multiple Tree MCTS (MT-MCTS) approach builds simultaneously multiple MCTS trees corresponding to the different sub-games and allows, like MCTS algorithms, to evaluate moves while playing. We apply MT-MCTS on decomposed games in the *General Game Playing* framework. We present encouraging results on single player games showing that this approach is promising and opens new avenues for further research in the domain of decomposition exploitation. Complex compound games are solved from 2 times faster (*Incredible*) up to 25 times faster (*Nonogram*).

Keywords: Monte Carlo Tree Search · General Game Playing · Decomposition.

1 Introduction

General Game Playing (GGP) is a branch of Artificial Intelligence with the aim of achieving versatile programs capable of playing any game without human intervention. Game specific algorithms cannot be used in a general game player as the game played should not be known in advance. An important aspect of GGP research is the development of automated rule analysis techniques to speedup the search.

Among the games considered in GGP, some are composed of different independent sub-games assembled sequentially or in parallel and played synchronously or asynchronously [3]. A player program able to identify the sub-games, solve them individually and synthesize the resulting solutions, can greatly reduce the search cost [4,2]. Some approaches have been proposed to decompose single [5] and multi-player games [17,6,7]. Using these decompositions two different strategies have been proposed to solve the global game in the GGP framework. The first approach, inspired from hierarchical planning and named *Concept Decomposition Search* [5], aims at solving single player games. The search algorithm is split into two stages: local search collecting all local plans and global search trying to interleave local plans from different sub-games to find the best global plan.

These two steps are embedded into an iterative deepening framework. *Concept Decomposition Search* is extended to multi-player games [17] using *turn-move sequences* (TMSeqs) as a result of the local search. A TMSeq indicates both moves and players who performed them. Global search is based on standard search techniques but uses TMSeqs instead of legal moves, which results in a much smaller game tree compared to full search. The second approach [3] is based on Answer Set Programming (ASP) to solve decomposed games. Local plans are combined as and when they are calculated to find a global plan. This search is used on single player games. How to generalize this approach to multi-player games is not clear and remains an open problem. In these previous works, if a global plan is not found in the allocated time, the search returns nothing.

In this paper, we propose a new approach to solve decomposed games based on Monte Carlo Tree Search (MCTS). MCTS is the state of the art framework for GGP players but also for specialized player like Alpha Go [15], for video games and non-game domains [1]. Our Multiple Tree MCTS (MT-MCTS) approach builds simultaneously multiple MCTS trees corresponding to different sub-games. Instead of producing a global plan, MT-MCTS allows to evaluate moves in the different states of the game. A player program using MT-MCTS can therefore begin to play while continuing to explore the game and identify the next best moves. We compared the performance of our MT-MCTS algorithm with that of an MCTS search using the Upper Confidence Bound applied to Trees (UCT) policy and transposition tables on single player games.

The remainder of this paper is organized as follows. In the next section, we briefly review the MCTS framework, the UCT policy and common optimizations. In section 3, we present our MT-MCTS approach using multiple trees to solve decomposed games. We present experimental results on several single-player games in section 4. In section 5, we discuss these results, the challenges risen by a simultaneous search in several trees, the possible extensions of our algorithm and open problems. We conclude in section 6.

2 MCTS and UCT

The MCTS framework allows to build a game tree in an incremental and asymmetric manner [1]. Each node represents a state of the game, and each edge represents a joint move[3]. MCTS begins from a node representing the current state and then repeats four steps for a given number of iterations or for a given time: the selection of a path in the game tree according to some *tree policy*; the expansion of the tree with the creation of a new node; a game simulation (*playout*) according to a *default policy*; the back-propagation of the playout result to update nodes statistics. The number of playouts in MCTS is a key factor for the quality of the evaluation of a game tree node [9].

The *default policy* usually consists in playing random moves until a terminal state is reached. For the selection, UCT is the most common *tree policy*. It

[3] In the GGP framework, player moves are always simultaneous. In alternate move games, all players but one play a useless move to skip the turn

provides a balance between the exploitation of the best moves and the exploration of the whole tree. A common optimization consists in using *transposition tables*: identical states are represented by a single node in the tree. In the GGP domain, many games that involve cycles use a *stepper*, a counter used to avoid infinite games. Identical states present at different depths in the game tree are then differentiated by this *stepper* and transpositions can only occur at a same depth. Using transpositions, a GGP game tree becomes a directed acyclic graph. When using transpositions, the evaluation of the different moves is commonly stored in the outgoing edges instead of the child nodes [14]. The number of visits remains stored in the parent node. Another common optimization used to guide the search is the pruning of fully explored branches [11,16]. During the selection step, instead of returning in branches that are completely evaluated, the mean score of the branch is computed and used for the back-propagation step.

3 Multiple Tree MCTS (MT-MCTS)

The decomposition of a game into several sub-games produces sub-states, in which available moves depend on the sub-states combination, and then provides a difficulty for legal moves computation. The decomposition also poses a problem for the identification of terminal sub-states. For example, the game *Incredible* is decomposed into a labyrinth (*Maze*), a game of cubes (*Blocks*), a *stepper* and a set of useless contemplation actions. The game is over if the *stepper* reaches 20 or if the *Maze* sub-game is over. But, in *Blocks*, a sub-state is never terminal by itself. We can also imagine a game where two sub-states are both non-terminal but their conjunction is terminal and must be avoided in a global plan. The decomposition also raises an issue for evaluating sub-states where scores can result from a timely combination with other sub-states. More specifically in GGP, the score described by the `goal` predicate is reliable only if the current state is terminal [10]. These two facts make the score function less reliable in sub-trees. At last, the decomposition raises the problem of its reliability. If the decomposition is inconsistent, the evaluation of legal moves can be wrong, leading the player program to choose illegal moves and compute inconsistent sub-states.

To avoid all these problems, we propose the following approach: doing simulations in the global game and building a sub-tree for each sub-game. Legal moves, next states and the end of game can be evaluated for the global game in which the real score is known. Move selection is performed according to the evaluation of the sub-states in the sub-trees. An inconsistency of the decomposition is detected if during two simulations, the same move from the same sub-state leads to different following sub-states. A partial[4] but consistent decomposition allows to play by the rules, although exploration may be less effective.

When a *stepper* is separated from the rest of a game, cycles can occur in some sub-games and in sub-state transpositions a move evaluation may differ according to the game depth. This problem is referred as the *graph history in-*

[4] A decomposition is *partial* if a sub-game can be further decomposed.

teraction problem [12]. A general solution for games with binary scores is available [8]. However, in the GGP framework, the scores are more graduated and this general solution is therefore not applicable. To exploit some transpositions while avoiding the *graph history interaction problem*, the current version of our MT-MCTS considers transpositions only at a same depth in the sub-trees i.e. sub-games are represented by rooted directed acyclic graphs.

Global simulations and sub-trees building: our MT-MCTS iterates four steps like MCTS (algo. 1) except that the selection step is composed of alternated local and global selections[5].

Algorithm 1 `MtMcts`(nbPlayouts)

1: **for** nbPlayouts **do**
2: S ← *current_state*
3: $\{s_1, ..., s_n\}$ ← `getSubstates`(S)
4: $\{val_1, ..., val_n\}$ ← `selectionWithExpansion`($\{s_1, ..., s_n\}$)
5: `simulationWithBackProp`($\{val_1, ..., val_n\}$)

Global variables are:

S : the current global state
$\mathscr{M}$: moves played during selection and expansion
$\mathscr{S}$: sets of sub-states visited during selection and expansion
$\{val_1, ..., val_n\}$: a vector of evaluations, one for each sub-game.
$\{e_1, ..., e_n\}$: a set of booleans indicating whether sub-games are fully explored or not, initialized to $\{false, ..., false\}$
$\{d_1, ..., d_n\}$: depth where a revision of a "fully explored" move flag occurred

At each step of the selection (algo.2), the legal moves evaluation is processed in the global game and an expansion is attempted (l.8-9). To try an expansion (algo. 3), a random move is played (l.33). Each sub-game is informed of the new sub-state reached. A new transition, and possibly a new node, are added to the sub-tree if necessary (l.36-39). If the transition is already known, i.e. a previously visited action triggers the same transition, the transition is labeled with these different moves which form a *meta-action* [6]. If a legal move triggers an already known transition in each sub-game, it is already evaluated and it is not necessary to test it: this move leads to a combination of already evaluated sub-states. Then, another move is randomly chosen. Playing with decomposed games therefore allows to reduce significantly the search space size. If no legal move leads to an expansion in one of the sub-games, the selection continues.

In a game like *Incredible*, the *Maze* sub-game is quickly fully explored. Then it systematically recommends the sequence of moves leading to the maximum gain allowed by this sub-game. However, ending this sub-game terminates the global game prematurely. For a local selection algorithm based on a balance

[5] An informal presentation of MT-MCTS with an example has been published in Journées dIntelligence Artificielle Fondamentale (JIAF) 2019.

between exploration and exploitation, it takes a large number of visits of the *Maze* terminal move to guide the search towards the exploration of other possible moves (playing in the *Block* sub-game). To alleviate this problem, the terminal legal moves are evaluated (l.10). If a terminal move returns the maximum score possible for the current player, it is always selected (l.12-15), otherwise, the selection continues with the non-terminal moves.

Algorithm 2 `selectionWithExpansion`$(\{s_1, ..., s_n\})$

```
 6: loop
 7:     𝒮 := 𝒮 ∪ {s_1, ..., s_n}
 8:     L ← getLegalMoves(S)
 9:     if expansion({s_1, ..., s_n}, L) then return {∅, ..., ∅}
10:     T ← filterTerminalMoves(L)
11:     {d_1, ..., d_n} ← checkNotFullyExplored({s_1, ..., s_n}, T)
12:     {best, score} ← getBestMove(T)
13:     if score = maximum possible evaluation then
14:         ℳ ← ℳ ∪ best
15:         return maximum evaluation for each sub-game
16:     explored ← true
17:     for i in {1, ..., n} do
18:         if ∃m ∈ L - T: ¬ fullyExplored(s_i, m) then
19:             explored ← false
20:         else
21:             fullyExplored(last(𝒮), last(ℳ)) ← true
22:             e_i ← true if {s_1, ..., s_n} = initial_state
23:     if explored but ∃i : e_i = false then
24:         return mean evaluation for each sub-game
25:     for i in {1, ..., n} do
26:         selected_i ← localSelectPolicy(s_i, L - T)
27:     best ← globalSelectPolicy({selected_1, ..., selected_n})
28:     ℳ ← ℳ ∪ best
29:     S ← apply(S, best)
30:     if terminal(S) then return {∅, ..., ∅}
31:     {s_1, ..., s_n} ← getSubstates(S)
```

To avoid re-visiting fully explored branches unnecessarily, we flag them to encourage visits to nearby branches. However, in the case of sub-games, a sub-state is not always terminal depending on the rest of the overall state. It is then necessary to develop a specific approach to flag the fully explored branches in sub-trees. We solve this problem by simply revising the labeling during selection and expansion update (l.11,38). If an action was flagged terminal during previous descents in the tree but is not terminal in the current situation, or if a new transition is added, the labeling is revised and revisions are back-propagated along the descent path (l.54-55).

When all legal moves from a sub-state are terminal or fully explored, the previous move is also flagged "fully explored" (l.21). If the current state is the

initial state, the whole sub-tree is fully explored (l.22). When the current sub-state is fully explored in each subgame, the mean evaluation is computed and returned (l.24). Otherwise a local selection policy is applied in each sub-game on non-terminal legal moves (l.26). The best move is selected according to a global selection policy (l.27).

Algorithm 3 `expansion`($\{s_1, ..., s_n\}$, L)

```
32: while L ≠ ∅ do
33:     m ← popRandomMove(L)
34:     S' ← apply(S, m)
35:     {s'_1, ..., s'_n} ← getSsubstates(S')
36:     new_transition ← false
37:     for i in {1, ..., n} do
38:         if update(s_i, s'_i, m) then        ▷ possible inconsistency is detected here
39:             new_transition ← true
40:     if new_transition then
41:         M ← M ∪ m
42:         S ← S'
43:         return true
44: return false
```

Algorithm 4 `simulationWithBackProp`($\{val_1, ..., val_n\}$)

```
45: if {val_1, ..., val_n} = {∅, ..., ∅} then
46:     if ¬terminal(S) then
47:         S ← playoutWithDefaultPolicy(S)
48:     for i in {1, ..., n} do
49:         val_i ← {global score, max score for sub-game i with 3-valued logic}
50: for p in {1, ..., M.length} do
51:     {s_1, ..., s_n} ← S_p
52:     m ← M_p
53:     for i in {1, ..., n} do
54:         if p < d_i then
55:             fullyExplored(s_i, m) ← false
56:         t ← transition from s_i labeled with m
57:         update(N, n_t, w_t, w_t^max)        ▷ see local selection policy
```

If the selection ends on a non-evaluated state (algo. 4, l.45), a *playout* is done to reach a terminal state if necessary (l.47), then the state is evaluated (l.49). The details of this evaluation depend on the *local selection policy* and are explained below. The evaluation is back-propagated along the visited path (l.50-57) and the "fully explored" flags are revised if necessary.

The local selection policy chooses the best moves among legal non-terminal moves. The current sub-state is associated with a number of visits N. Each transition t from this sub-state is evaluated by a number of visits n_t and a

cumulated score w_t. The local selection returns a set of moves if there exists different transitions with the same evaluation or if the best transition is labeled with several moves.

We investigate different ways to perform this local selection. The first one is a standard application of the UCT policy. However, this approach is not satisfactory because a transition in a sub-game can receive a good evaluation without contributing to the global score: the evaluation was obtained thanks to a move sequence leading to a positive evaluation in another sub-game. Another more troublesome problem occurs in the case of binary scores: the score is always zero until a solution is found. The search is in this case reduced to a breadth first search and is not guided to the right combination of sub-states.

In the GGP framework, a game state is described by a finite set of instantiated fluents, some of which are true. The decomposition partitions these fluents in several groups which represent the sub-games states. In a global terminal state, it is possible to keep only the fluents corresponding to a sub-game, to give an *undefined* value to the other and to evaluate the logic rules of the game with a 3-valued logic. The true or undefined `goal` predicate instances represent the possible scores according to this sub-game state. The maximum `goal` score (*lmax*) corresponds to the maximum potential score that can be obtained if the best possible configuration is found in the other sub-games. The *lmax* score is a maximum indication, the true maximum score may not match exactly because using a 3-valued logic does not guarantee the most accurate information [13]. *lmax* evaluation is nevertheless a valuable indication of the sub-state value. It can be back-propagated in addition to the global score and cumulated in a w_t^{max} variable. For a given transition, w_t/n_t is a global score estimator and w_t^{max}/n_t is a local score estimator. These two estimators can be used in a new policy derived from UCT:

$$U = \alpha\frac{w_t}{n_t} + (1-\alpha)\frac{w_t^{max}}{n_t} + C * \sqrt{\frac{\log N}{n_t}} \tag{1}$$

with C the constant balancing the exploitation and exploration terms and $\alpha \in [0, 1]$ setting the balance between local and global score estimations.

To avoid going back into already fully explored sub-branches, the transitions corresponding to these branches are excluded from the local selection as long as there are transitions that are not fully explored.

The global selection of the best move is made depending on the moves recommended by the sub-games. In serial or synchronous parallel games, the intersection of the recommended move sets is always non-empty. The global selection is then straightforward: a move can be randomly selected in the intersection. However, in a parallel asynchronous game, different sub-games can propose disjoint move sets. It is then necessary to define a policy for the choice of a move at the global level. To define an any game policy, we propose a voting policy. Each sub-game recommending a move brings a vote. In the case of serial or synchronous parallel games, the best moves get as many votes as there are sub-games. In the case of parallel asynchronous games, each move may win a

maximum of one vote. Among the moves that received the most votes, those with the highest expected earnings are selected to direct the search towards moves that can lead to the best combination of sub-states. When the game goal is the conjunction of the sub-games goals, this highest expected earning for a move m among T sub-games is the product of the probability of gain in each sub-game s: $\prod_{s=1}^{T} w_m^s/n_m^s$. When the game goal is the disjunction of the sub-games goals, this highest expected earning is the sum of the probability: $\sum_{s=1}^{T} w_m^s/n_m^s$ with w_m^s the cumulated rewards earned during playouts and n_m^s the number of visits of the transition labeled with that move m in sub-game s. If several moves offer the greatest probability of gain, one of them is randomly selected.

4 Experiments

We present here experiments on MT-MCTS[6]. Firstly we evaluate different weighting of our local selection policy and secondly we compare the effectiveness of MT-MCTS against UCT and show that this approach can reduce the overall number of simulations and the solving time. We conducted our experiments on several single-player games: *Incredible*, different grids of *Nonogram* of size 5×5 and 6×6 and *Queens08lg*.

Incredible is an interesting game because it is possible to end the game prematurely with a suboptimal score. It is a usual test bed to evaluate players able to exploit decompositions. *Nonogram* is a logic puzzle in which cells in a grid must be colored or left blank according to numbers placed in front of columns and lines. The score is binary and UCT provides no improvement over depth-first or breadth-first search in this game. *Queens08lg* is on the contrary quickly solved by UCT. It is an *Eight Queens puzzle* in which it is illegal to place a queen in a position where it could capture another queen in one move. The game is over when a queen can no longer be placed. See [Anonymised Author PhD], for more information about these games and a detailled presentation of each Nonogram grid. We decompose these games with the statistical approach proposed by [7]. The decomposition time can vary slightly depending on the simulations done to collect statistical information. For each game and each configuration, we realized 10 tests and present the mean number of playouts and mean time necessary to solve the game. A game is considered solved when a maximum score leaf is found.

The purpose of our first experiment is to compare different values of α in the local selection policy (eq.1). We use $C = 0.4$ which allow a good balance between exploration and exploitation in a majority of GGP games. We experimented on two games: *Incredible* and *Nonogram "checkerboard"*. The results are presented in figure 1. Using only the global score estimator ($\alpha = 1$) in the local selection policy does not allow to solve *Nonogram* due to its binary score. The estimated score is always zero in this game and the randomly chosen moves very unlikely

[6] The experiments are performed on one core of an Intel Core i7 2,7GHz with 8Go of 1.6GHz DDR3.

lead to the right combination of sub-states. On the contrary, the local score estimator ($\alpha = 0$) allows to guide the search in the sub-trees and solve this *Nonogram* in less than 5 seconds. However, the use of $\alpha = 1$ gives better results on *Incredible* while $\alpha = 0$ requires almost twice as much time to solve the game. By varying α, we notice that a small participation of the local score estimator ($\alpha = 0.75$) allows an even better result in *Incredible*. The weighting $\alpha = 0.25$ seems to allows the fastest resolution of *Nonogram*. However, the importance of the standard deviations, of the same order of magnitude as the resolution times or an order below, does not allow to identify a significant effect of the variation of this weighting. Considering these standard deviations, the resolution times are similar in the three tests mixing the local and global score estimators. More experiments will be necessary to verify the influence of unequal weighting of these two pieces of information.

Nevertheless, the association of both estimators appears desirable to constitute a polyvalent policy. In the following experiments, we used $\alpha = 0.5$ for the local selection since it gives overall the best result.

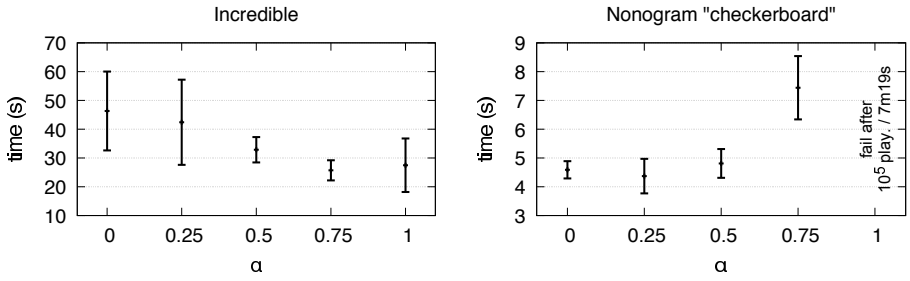

Fig. 1. Mean time on 10 tests to solve *Incredible* and *Nonogram "checkerboard"* with different values of α in the local selection policy of MT-MCTS (eq.1).

In a second experiment (fig.2) we compare the effectiveness of MT-MCTS against UCT in terms of number of playouts and time spent.

On *Incredible*, MT-MCTS is significantly better than UCT in terms of number of playouts necessary to solve the game. MT-MCTS uses twenty times less playouts but the move selection is significantly longer. In the end, the game is solved twice as fast.

Comparing our results with [5] (*Fluxplayer*) and [3] (ASP) is difficult because their approaches are totally different from UCT. *Fluxplayer* takes about 2 hours to solve *Incredible* by computing over 41 million states (compared to 280 thousand playouts for UCT[7]). Their decomposition method reduces this time to 45 seconds and 3212 calculated states. Their resolution time is greater than ours although fewer states are computed by their approach. The encoding of *Incredible* in ASP allows game resolution in 6.11 seconds. This time is reduced to 1.94 seconds by decomposition, a factor of 3. It should be noted that this approach is

[7] As each playout results in an expansion of the tree, we can compare the number of playouts with the number of calculated states.

optimized for solitary games. Since our approach requires 21 times less playouts for the resolution of the decomposed game, by optimizing the selection step of MT-MCTS, we hope to obtain a similar or even better improvement.

For a simple puzzle like *Queens08lg*, even if the resolution of the decomposed game requires 3 times less playouts, the decomposition time is too important compared to the gain that can be expected in the resolution.

Game	Algo.	# Playouts	Time (decomp.)	σ	# Fail
Incredible	UCT	280158	1m	14.3s	-
	MT-MCTS	13199	32.86s (2.50s)	4.4s	-
Queens08lg	UCT	67	0.01s	<0.1s	-
	MT-MCTS	22	1.30s (1.29s)	<0.1s	-
Nonogram "checkerboard"	UCT	432019	31.31s	16.5s	-
	MT-MCTS	776	4.81s (0.69s)	0.5s	-
Nonogram "G"	UCT	2988921	4m24s	4m8s	-
	MT-MCTS	9344	36.61s (0.77s)	16.45s	-
Nonogram "tetris"	UCT	4969111	7m20s	3m53s	1 *(after 10^7 play. / 15m17s)*
	MT-MCTS	3767	16.36s (1.01s)	6.5s	-
Nonogram "sitdog"	UCT	2552082	3m43s	2m50s	-
	MT-MCTS	5476	26.27s (0.98s)	14.92s	-
Nonogram "iG"	UCT	3086172	4m34s	3m9s	-
	MT-MCTS	10232	28.60s (0.85s)	12.40s	-
Nonogram "rally"	UCT	4284872	13m7s	6m41s	4 *(after 10^7 play. / 31m9s)*
	MT-MCTS	1762	49.26s (2.40s)	3.72s	-
Nonogram "cube"	UCT	21438731	1h17m23s	19m16s	8 *(after 5×10^7 play. / 3h4m)*
	MT-MCTS	358608	1h53m8s (2.75s)	45m7s	-

Fig. 2. Comparison of the resolution times of different games with UCT and MT-MCTS. The columns present the mean number of playouts and the mean time to solve the puzzles (failures excluded) over 10 tests. In parentheses is the time used for decomposition. The σ column indicates the standard deviations. The column "*# Fail*" indicates how many searches were stopped without finding the solution.

On *Nonograms* grids (5×5 and 6×6) MT-MCTS is significantly better than UCT. The resolution is 25 times faster for *"Tetris"*. This gain is not directly related to the number of sub-games identified: in *"Checkerboard"*, which has a larger number of sub-games, the resolution is only 6 times faster. The gain obtained for the resolution speed is directly related to the number of simulations needed: 300 times less for *"iG"* up to 2400 times less for *"rally"* (not to mention the 4 tests where UCT was interrupted after 10^7 playouts without finding any solution). The heavier selection step is largely mitigated here by the significant gain in the number of simulations. The time needed to solve *"cube"* with MT-MCTS is 2 hours on average. The resolution with UCT is sometimes successful in less than an 2 hours, but the majority of tests (8/10) failed to find the solution after 3 hours on average.

5 Discussion

During the expansion, some actions are not tested as we already have an evaluation for the resulting sub-states from previous descents in the sub-trees. There-

fore some combinations of sub-states may never be visited and splitting the search over several game trees offers in theory no guarantee of convergence toward a solution with an infinite number of playouts. In practice, a good selection policy allows to guide the search to find the right sequence of moves to reach the right combination of sub-states.

The problem in the GGP domain is that the optimal policy depends on the structure of the game. For example, excluding fully explored branches in local selection can quickly guide to the solution in *Incredible* as it avoids re-exploring a path leading to a suboptimal score. However, it can delay the resolution in a game like *Nonogram* where playing the already known good moves would have allowed to color some cells and guide the discovery of the following good moves.

Despite this delay, MT-MCTS is still more efficient than UCT on *Nonogram*. However, as an interesting specific combination of sub-states can remain unexplored for a long time, we assume that a fixed value for α in our policy may not be as effective on all GGP games. Further research is therefore needed. Many different policies could be considered to allow to go down rarely in the fully explored branches and to improve the selection step of MT-MCTS. Finding a policy for MT-MCTS that is proven effective on all games is an interesting open problem.

Nonogram naturally presents a composition of rules in rows and columns. The structure of MT-MCTS allows to explore a game decomposed in this way. Another avenue of research to consider is then the exploitation of different overlapping sub-games and, more generally, of non-disjoint sub-games.

Our version of MT-MCTS does not consider all transpositions to avoid games with cycles. Fifty-five percent of GGP games use a *stepper*, the development of a specific selection policy to take advantage of transpositions in games containing cycles is therefore an open and interesting search track that could significantly improve the level of GGP players.

6 Conclusion

In this paper we proposed an extension of MCTS to search in several trees representing the different parts of a decomposed problem. We tested this idea on several single player games in the *General Game Playing* domain. Playing with decomposed games allows to hope for a real change of scale in their resolution speed. Our tests with a weighted selection policy give promising results: the games are solved from 2 times (*Incredible*) to 25 times faster (*Nonogram*). Multiple Tree MCTS (MT-MCTS) can be extended to multi-player games such as conventional MCTS approaches and also allows non-independent sub-games to be exploited.

The new MT-MCTS approach opens different research tracks: the development of a selection policy efficient for the different types of compound games, the support of the specific case of games with cycles using a *stepper*, playing with overlapping sub-games and even the exploitation of incomplete or imperfect decompositions.

References

1. 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. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1), 1–43 (2012)
2. Cazenave, T.: Monte-Carlo Approximation of Temperature. Games of No Chance 4 **63** (2015)
3. Cerexhe, T., Rajaratnam, D., Saffidine, A., Thielscher, M.: A systematic solution to the (de-)composition problem in general game playing. In: Proceedings of the European Conference on Artificial Intelligence (ECAI) (2014)
4. Cox, E., Schkufza, E., Madsen, R., Genesereth, M.: Factoring general games using propositional automata. In: Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09). pp. 13–20 (2009)
5. Günther, M., Schiffel, S., Thielscher, M.: Factoring general games. In: Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09). pp. 27–33 (2009)
6. Hufschmitt, A., Méhat, J., Vittaut, J.N.: A general approach of game description decomposition for general game playing. In: Proceedings of the IJCAI-16 Workshop on General Game Playing (GIGA'16). pp. 23–29 (2016)
7. Hufschmitt, A., Vittaut, J.N., Jouandeau, N.: Statistical ggp games decomposition. In: Proceedings of the IJCAI-18 Workshop on Computer Games (CGW 2018). p. 19p. (2018)
8. Kishimoto, A., Müller, M.: A general solution to the graph history interaction problem. In: Nineteenth National Conference on Artificial Intelligence (AAAI 2004), San Jose, CA. pp. 644–649 (2004)
9. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Proceedings of the 17th European Conference on Machine Learning. pp. 282–293. Springer-Verlag, Berlin, Heidelberg (2006)
10. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General Game Playing: Game Description Language Specification. Tech. Rep. LG-2006-01, Stanford University (01 2006)
11. Mehat, J., Cazenave, T.: Combining uct and nested monte carlo search for single-player general game playing. IEEE Transactions on Computational Intelligence and AI in Games **2**(4), 271–277 (Dec 2011)
12. Palay, A.: Searching With Probabilities. Research Notes in Artificial Intelligence Series, Pitman Advanced Publishing Program (1985)
13. Reps, T.W., Loginov, A., Sagiv, S.: Semantic minimization of 3-valued propositional formulae. In: 17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings. p. 40 (2002)
14. Saffidine, A., Méhat, J., Cazenave, T.: Ucd: Upper confidence bound for rooted directed acyclic graphs. In: TAAI 2010. pp. 467–473. IEEE, Piscataway, NJ - (2010)
15. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of go without human knowledge. Nature **550**, 354–359 (10 2017)
16. Winands, M.H.M., Björnsson, Y., Saito, J.T.: Monte-carlo tree search solver. In: Proceedings of the 6th International Conference on Computers and Games. pp. 25–36. CG '08, Springer-Verlag, Berlin, Heidelberg (2008)
17. Zhao, D., Schiffel, S., Thielscher, M.: Decomposition of multi-player games. In: Proceedings of the Australasian Joint Conference onArtificial Intelligence. vol. 5866, pp. 475–484. Springer (2009)