Monte-Carlo Tree Search and Minimax Hybrids

Hendrik Baier and Mark H.M. Winands
Games and AI Group, Department of Knowledge Engineering
Faculty of Humanities and Sciences, Maastricht University
Maastricht, The Netherlands
Email: {hendrik.baier,m.winands}@maastrichtuniversity.nl

***Abstract*—*Monte-Carlo Tree Search* is a sampling-based search algorithm that has been successfully applied to a variety of games. Monte-Carlo rollouts allow it to take distant consequences of moves into account, giving it a strategic advantage in many domains over traditional depth-limited minimax search with alpha-beta pruning. However, MCTS builds a highly selective tree and can therefore miss crucial moves and fall into traps in tactical situations. Full-width minimax search does not suffer from this weakness.**

This paper proposes *MCTS-minimax hybrids* that employ shallow minimax searches within the MCTS framework. The three proposed approaches use minimax in the selection/expansion phase, the rollout phase, and the backpropagation phase of MCTS. Without requiring domain knowledge in the form of evaluation functions, these hybrid algorithms are a first step at combining the strategic strength of MCTS and the tactical strength of minimax. We investigate their effectiveness in the test domains of Connect-4 and Breakthrough.

I. Introduction

Monte-Carlo Tree Search (MCTS) [1], [2] is a best-first tree search algorithm that evaluates each state by the average result of simulations from that state. Instead of considering all possible moves from a state as traditional minimax search algorithms [3] would, it samples moves and can therefore handle large search spaces with high branching factors. Instead of depending on a static heuristic evaluation function to compare non-terminal states as in the minimax approach, it uses Monte-Carlo simulations that can take long-term rewards into account.

MCTS has been successfully applied in a variety of domains from the games of Go, Amazons, and Lines of Action, to General Game Playing, planning, and optimization [4]. While MCTS has shown considerable success, there are still a number of games such as Chess and Checkers in which the traditional approach to adversarial planning, minimax search with alpha-beta pruning [5], remains superior. Since MCTS builds a highly selective search tree, focusing only on the most promising lines of play, it has been conjectured that it could be less appropriate than traditional, full-width minimax search in search spaces containing a large number of *shallow traps* [6]. In trap situations such as those frequent in Chess, precise tactical play is required to avoid immediate loss. MCTS methods based on sampling could easily miss a crucial move or underestimate the significance of an encountered terminal state due to averaging value backups. Conversely, MCTS could be more effective in domains such as Go, where terminal positions and potential traps are rare or do not occur until the latest stage of the game. MCTS can here fully play out its strategic and positional understanding resulting from Monte-Carlo simulations of entire games.

In this paper, we explore ways of *combining* the strategic strength of MCTS and the tactical strength of minimax in order to produce more universally useful hybrid search algorithms. We do not assume the existence of evaluation functions, allowing the MCTS-minimax hybrids to be applied in any domain where plain MCTS is used.

This paper is structured as follows. Section II provides some background on MCTS as the baseline algorithm. Section III gives a brief overview of related work on the relative strengths of minimax and MCTS, as well as results with combining or nesting tree search algorithms. Section IV describes different ways of incorporating shallow-depth minimax searches into the different parts of the MCTS framework, and Section V shows experimental results of these MCTS-minimax hybrids in the test domains of Connect-4 and Breakthrough. Conclusions and future research follow in Section VI.

II. Background

In this section, we shortly review the baseline algorithm used in this paper: MCTS with the MCTS-Solver extension.

A. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [1], [2] is a best-first tree search algorithm using statistical sampling to evaluate states. MCTS works by repeating the following four-phase loop until computation time runs out [7]. Each iteration represents one simulated game.

Phase one: *selection*. The tree is traversed from the root to one of the leaves, using a selection policy to choose the move to sample from each state. Critical is a balance of exploitation of states with high value estimates and exploration of states with uncertain value estimates. In this work, we use the popular UCT variant of MCTS, with the UCB1 policy as selection policy [8].

Phase two: *expansion*. When a leaf has been reached, one or more of its successors are added to the tree. In this paper, we always add the one successor played in the current iteration.

Phase three: *rollout*. A rollout (also called *playout*) policy is used to choose moves starting from the state represented by the newly added node until the simulated game ends. The simplest choice of uniformly random moves is sufficient to achieve convergence of MCTS to the optimal move in the limit. We use uniformly random moves except for Subsection IV-A, where rollout moves are chosen with the help of minimax.

Phase four: *backpropagation*. The result of the finished game is used to update value estimates of all states traversed during the simulation.

Our implementation also takes transpositions into account, i.e. builds a rooted directed acyclic graph instead of a tree [9]. In games where transpositions occur, nodes can have more than one parent.

B. MCTS-Solver

In this paper, we do not assume the availability of heuristic evaluation functions. Therefore, minimax search can only distinguish terminal and non-terminal game states, potentially producing search results such as *proven win* or *proven loss* through minimax backup. In order to be able to handle these proven values, we use MCTS with the *MCTS-Solver* extension [10] as the baseline algorithm.

The basic idea of MCTS-Solver is allowing for the backpropagation of not only regular simulation outcomes such as lost or won games, but also game-theoretic values such as proven losses and proven wins whenever terminal states are encountered by the search tree. First, whenever a move from a given game state s has been marked as a proven win for player A, the move leading to s can be marked as a proven loss for the opposing player B. Second, whenever *all* moves from a given state s have been marked as proven losses for A, the move leading to s can be marked as a proven win for B. If at least one move from s has not been proven to be a loss yet, the move leading to s is only updated with a regular rollout win in this backpropagation phase. (We do not prove draws in this paper. Draws are backpropagated as half a win.)

This solving extension to plain MCTS has been successfully used e.g. in Lines of Action [10], Hex [11], [12], Havannah [13], Shogi [14], Tron [15], and Focus [16]. It has been generalized to more than two game outcomes in a way that allows for alpha-beta style pruning [17], and to simultaneous move games in the concept of General Game Playing [18].

MCTS-Solver handles game-theoretic values better than MCTS without the extension because it avoids wasting time on the re-sampling of proven game states. However, it still suffers from the MCTS weakness that such game-theoretic values need time to propagate all the way up through the tree in order to influence the move decision at the root. MCTS-Solver may for example need to keep sampling a state many times until it has proved *all* moves from this state to be losses, such that it can backpropagate a proven win to the next-higher level of the tree. In Subsection IV-C of this paper we describe how we use shallow-depth, exhaustive minimax searches to speed up this process and guide MCTS more effectively.

III. Related Work

Ramanujan *et al.*'s work [6], [19], [20] has repeatedly dealt with characterizing search space properties that influence the performance of MCTS relative to minimax search. *Shallow traps* were identified in [6] as a feature of domains that are problematic for MCTS, in particular Chess. Informally, Ramanujan *et al.* define a *level-k search trap* as the possibility of a player to choose an unfortunate move such that *after* executing the move, the opponent has a guaranteed winning strategy at most k plies deep. While such traps at shallow depths of 3 to 7 are not found in e.g. Go until the latest part of the endgame, they are relatively frequent in Chess games even at grandmaster level [6], partly explaining the problems of MCTS in this domain. A resulting hypothesis is that in regions of a search space containing no or very few terminal positions, shallow traps should be rare and MCTS variants should make comparatively better decisions, which was confirmed in [19] for the game of Mancala. In [20] finally, a synthetic tree model was used to explore the dependence of MCTS performance on the density of traps in the search space. A similar problem to shallow traps was presented by Finnsson and Björnsson [21] under the name of *optimistic moves*—seemingly strong moves that can be refuted right away by the opponent, but take MCTS prohibitively many simulations to find the refutation. One of the motivations of this paper was to employ shallow-depth minimax searches within MCTS to increase the visibility of shallow traps and allow MCTS to avoid them more effectively.

In the context of General Game Playing, Clune [22] compared the performance of minimax with alpha-beta pruning and MCTS and, restricted to the class of turn-taking, two-player, zero-sum games we are addressing here, identified a stable and accurate evaluation function as well as a relatively low branching factor as advantages for minimax over MCTS. In this paper, we explore the use of minimax within the MCTS framework even when no evaluation function is available.

One method of combining different tree search algorithms that was proposed in the literature is the use of shallow minimax searches in every step of the MCTS *rollout phase*. This was typically restricted to a 1-ply lookahead, as in [23] and [13] for the game of Havannah. 2-ply searches have been applied to the rollout phase in Lines of Action [24], Chess [25], as well as various multi-player games [26]. However, the existence of an evaluation function was assumed here. A different hybrid algorithm $UCTMAX_H$ was proposed in [19], employing minimax backups in an MCTS framework. However, again a strong heuristic was assumed as a prerequisite. In our work, we explore the use of minimax searches of various depths without any domain knowledge beyond the recognition of terminal states. Minimax in the rollout phase is covered in Section IV-A.

In the framework of proof-number search (PNS [27]), 1- and 3-ply minimax searches have been applied in the expansion phase of PNS [28]. In [29], nodes proven by PNS in a first search phase were stored and reused by alpha-beta in a second search phase. In [30], Monte-Carlo playouts were used to initialize the proof and disproof numbers at newly expanded nodes.

Furthermore, the idea of nesting search algorithms has been used in [31] and [32] to create Nested Monte-Carlo Tree Search and Nested Monte-Carlo Search, respectively. In this paper, we are not using search algorithms recursively, but nesting two different algorithms in order to combine their strengths: MCTS and minimax.

IV. Hybrid Algorithms

In this section, we describe the three different approaches for applying minimax with alpha-beta pruning within MCTS

that we explore in this work.

A. Minimax in the Rollout Phase

While uniformly random move choices in the rollout are sufficient to guarantee the convergence of MCTS to the optimal policy, more informed rollout strategies typically greatly improve performance [33]. For this reason, it seems natural to use fixed-depth minimax searches for choosing rollout moves. Since we do not use evaluation functions in this paper, minimax can only find forced wins and avoid forced losses, if possible, within its search horizon. If minimax does not find a win or loss, we return a random move.

This strategy thus improves the quality of play in the rollouts by avoiding certain types of blunders. It informs tree growth by providing more accurate rollout returns. We call this strategy *MCTS-MR* for *MCTS with Minimax Rollouts*.

B. Minimax in the Selection and Expansion Phases

Minimax searches can also be embedded in the phases of MCTS that are concerned with traversing the tree from root to leaf: the selection and expansion phases. This strategy can use a variety of possible criteria to choose whether or not to trigger a minimax search at any state encountered during the traversal. In this paper, we experimented with starting a minimax search as soon as a state has reached a given number of visits (for 0 visits, this would include the expansion phase). Other possible criteria include e.g. starting a minimax search for a loss as soon as a given number of moves from a state have already been proven to be losses, or starting a minimax search for a loss as soon as average returns from a node fall below a given threshold (or searching for a win as soon as returns exceed a given threshold, conversely), or starting a minimax search whenever average rollout lengths from a node are short, suggesting proximity of terminal states. These are left as future work.

This strategy improves MCTS search by performing shallow-depth, full-width checks of the immediate descendants of a subset of tree nodes. It informs tree growth by avoiding shallow losses, as well as detecting shallow wins, within or close to the MCTS tree. We call this strategy *MCTS-MS* for *MCTS with Minimax Selection*.

C. Minimax in the Backpropagation Phase

As mentioned in Subsection II-B, MCTS-Solver tries to propagate game-theoretic values (*proven win* and *proven loss*) as far up the tree as possible, starting from the terminal state visited in the current simulation. It has to switch to regular rollout returns (*win* and *loss*) as soon as at least one sibling of a proven loss move is not marked as proven loss itself. Therefore, we employ shallow minimax searches whenever this happens, actively searching for proven losses instead of hoping for MCTS-Solver to find them in future simulations. If minimax succeeds at proving all moves from a given state s to be losses, we can backpropagate a *proven win* instead of just a *win* for the opponent player to the move leading to s. Since these minimax searches can both consider the values of terminal states as well as states already proven in previous rollouts, it is possible to get different results for repeated minimax searches starting from the same state at different times during the search.

This strategy improves MCTS-Solver by providing the backpropagation step with helpful information whenever possible, which allows for quicker proving and exclusion of moves from further MCTS sampling. Other than the strategies described in IV-A and IV-B, it only triggers when a terminal position has been found in the tree and the MCTS-Solver extension applies. For this reason, it avoids wasting computation time on minimax searches in regions of the search space with no or very few terminal positions. We call this strategy *MCTS-MB* for *MCTS with Minimax Backpropagation*.

V. Experimental Results

We tested the MCTS-minimax hybrids in two different domains: The two-player, zero-sum games of *Connect-4* and *Breakthrough*. These games, popular in the General Game Playing community [34]–[36], were chosen due to their simple rules and bounded game length, while still providing search spaces of non-trivial size and complexity. In all experimental conditions, we compared the hybrids against regular MCTS-Solver as the baseline. Optimal UCT parameters such as the exploration factor were determined once for MCTS-Solver in both games and then kept constant for both MCTS-Solver and the MCTS-minimax hybrids during testing. Draws, which are possible in Connect-4, were counted as half a win for both players. We used minimax with alpha-beta pruning, but no other search enhancements. Unless stated otherwise, computation time was 1 second per move.

A. Games

1) Connect-4: The variant of Connect-4 we are using is played on a 7×6 board. It has been proven to be a win for the first player [37].

At the beginning of the game, the board is empty. The two players alternatingly place white and black tokens in one of the seven columns, always filling the lowest available space of the chosen column. The game is won by the player who succeeds first at connecting four tokens of his own color either vertically, horizontally, or diagonally. If the board is filled completely without any player reaching this goal, the game ends in a draw.

2) Breakthrough: The variant of Breakthrough we are using is played on a 6×6 board. The game was originally described as being played on a 7×7 board, but other sizes such as 8×8 are popular as well, and the 6×6 board preserves an interesting search space.

At the beginning of the game, the first two rows of the board are occupied by twelve white pieces, and the last two rows are occupied by twelve black pieces. The two players alternatingly move one of their pieces straight or diagonally forward, onto an empty square of the board. Two pieces cannot occupy the same square. However, players can capture the opponent's pieces by moving onto their square in diagonal direction only. The game is won by the player who succeeds first at reaching the home row of his opponent, i.e. reaching the first row as Black or reaching the last row as White, with one piece.

B. Existence of Shallow Traps

In order to measure an effect of employing shallow minimax searches without an evaluation function within MCTS, terminal states need to be present in sufficient density throughout the search space, in particular the part of the search space relevant at our level of play. We played 1000 self-play games of MCTS-Solver in both Connect-4 and Breakthrough to test this property, using $50,000$ rollouts per move. At each turn, we determined whether there exists at least one trap at depth (up to) 3 for the player to move. The same methodology was used in [6]. Figures 1 and 2 show that shallow traps are indeed found throughout both domains, which suggests improving the ability of MCTS to identify and avoid such traps is worthwhile. Next, we see that in contrast to Breakthrough the density of traps for both players differs significantly in Connect-4. Finally, we note that Breakthrough rarely last longer than 40 turns, which explains why the data become more noisy.

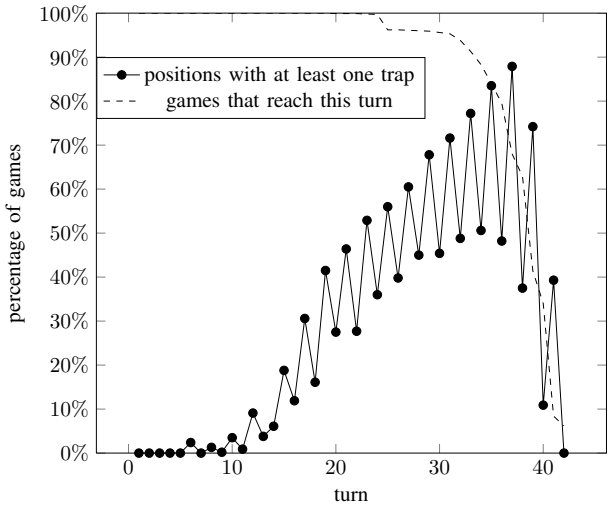

Fig. 1. Density of level-3 search traps in Connect-4.

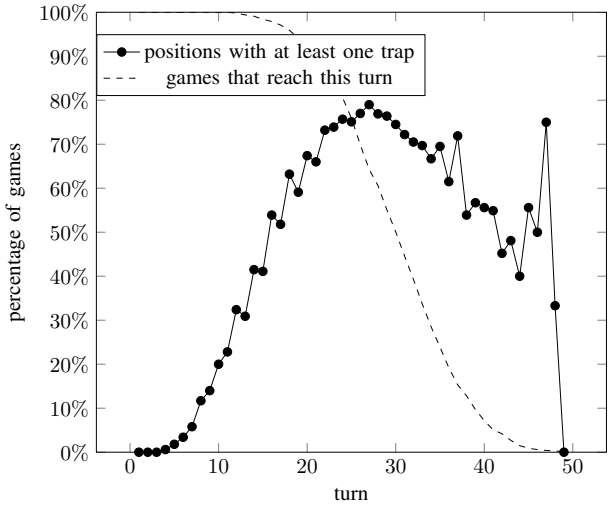

Fig. 2. Density of level-3 search traps in Breakthrough.

C. Connect-4

In this subsection, we summarize the experimental results in the game of Connect-4. Our baseline MCTS-Solver implementation performs about $91,000$ simulations per second when averaged over an entire game.

1) Minimax in the Rollout Phase: We tested minimax at search depths 1 ply to 4 plies in the rollout phase of a Connect-4 MCTS-Solver player. Each resulting player, abbreviated as MCTS-MR-1 to MCTS-MR-4, played 1000 games against regular MCTS-Solver with uniformly random rollouts. Figure 3 presents the results.

Minimax is computationally more costly than a random rollout policy. MCTS-MR-1 finishes about 65% as many simulations per second as the baseline, MCTS-MR-2 about 18% as many, MCTS-MR-3 about 6% as many, MCTS-MR-4 about 2% as many when searching the start position of Connect-4 for one second. This typical speed-knowledge trade-off explains the decreasing performance of MCTS-MR for higher minimax search depths, although the quality of rollouts increases. Remarkably, MCTS-MR-1 performs significantly worse than the baseline. This also held when we performed the comparison using equal numbers of MCTS iterations $(100,000)$ per move instead of equal time (1 second) per move for both players. In this scenario, we found MCTS-MR-1 to achieve a win rate of 35.7% in 1000 games against the baseline. It remains to be shown in future work whether this is e.g. due to some specific imbalance in Connect-4 rollouts with depth-1 minimax.

In our Connect-4 experiments, MCTS-MR-2 outperformed all other conditions. Over an entire game, it completed about $30,000$ simulations per second on average. In an additional 2000 games against the baseline, it won 73.2% of games, which is a significant improvement ($p<0.001$).

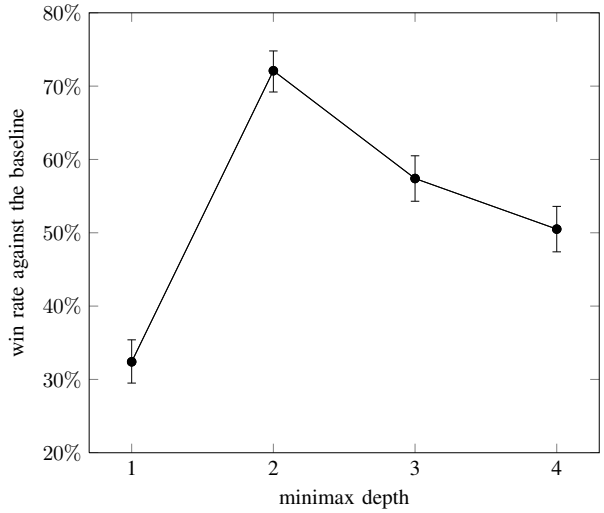

Fig. 3. Performance of MCTS-MR in Connect-4.

2) Minimax in the Selection and Expansion Phases: The variant of MCTS-MS we tested starts a minimax search from a state in the tree if that state has reached a fixed number of visits when encountered by the selection policy. We call this variant, using a minimax search of depth d when reaching v visits,

MCTS-MS-d-Visit-v. If the visit limit is set to 0, this means every tree node is searched immediately in the expansion phase even before it is added to the tree.

We tested MCTS-MS-d-Visit-v for $d \in \{2, 4\}$ and $v \in \{0, 1, 2, 5, 10, 20, 50, 100\}$. We found it to be most effective to set the alpha-beta search window such that minimax was only used to detect forced losses (traps). Since suicide is impossible in Connect-4, we only searched for even depths. Furthermore, we started independent minimax searches for each legal move from the node in question, which allows to store found losses for individual moves even if the node itself cannot be proven to be a loss. Each condition consisted of 1000 games against the baseline player. The results are shown in Figure 4. Low values of v result in too many minimax searches being triggered, which slows down MCTS. High values of v mean that the tree below the node in question has already been expanded to a certain degree, and minimax might not be able to gain much new information. Additionally, high values of v result in too few minimax searches, such that they have little effect.

MCTS-MS-2-Visit-2 was the most successful condition. It played about $83,000$ simulations per second on average over an entire game. There were 5000 additional games played against the baseline and a total win rate of 53.4% was achieved, which is a significantly higher performance ($p<0.001$).

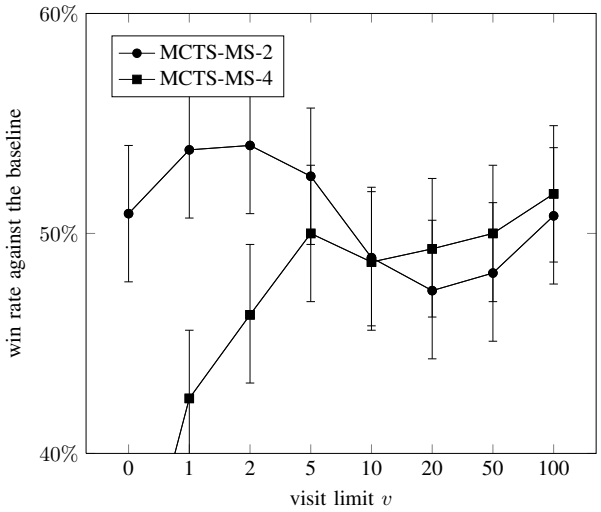

Fig. 4. Performance of MCTS-MS in Connect-4.

3) Minimax in the Backpropagation Phase: MCTS-Solver with minimax in the backpropagation phase was tested with minimax search depths 1 ply to 6 plies. Contrary to MCTS-MS as described in V-C2, we experimentally determined it to be most effective to use MCTS-MB with a full minimax search window in order to detect both wins and losses. We therefore included odd search depths. Again, all moves from a given node were searched independently in order to be able to prove their individual game-theoretic values. The resulting players were abbreviated as MCTS-MB-1 to MCTS-MB-6 and played 1000 games each against the regular MCTS-Solver baseline. The results are shown in Figure 5.

MCTS-MB-2 as the best-performing variant played 5000 additional games against the baseline and won 52.1% of them, which shows a significant improvement ($p<0.05$). It played about $90,000$ simulations per second when averaged over the whole game.

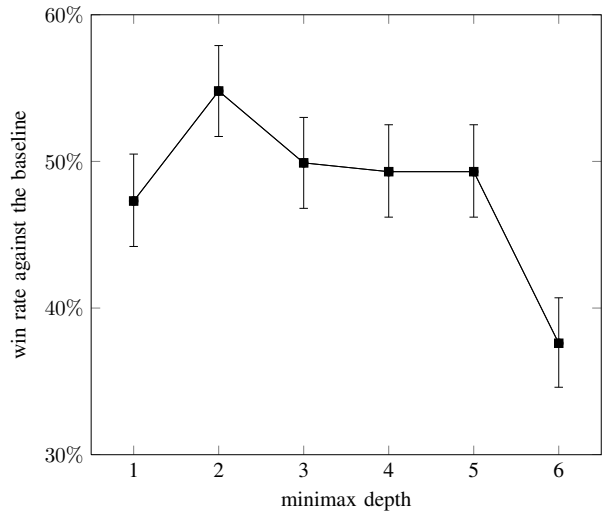

Fig. 5. Performance of MCTS-MB in Connect-4.

D. Breakthrough

The experimental results in the Breakthrough domain are described in this subsection. Our baseline MCTS-Solver implementation plays about $61,000$ simulations per second on average.

1) Minimax in the Rollout Phase: As in Connect-4, we tested 1-ply to 4-ply minimax searches in the rollout phase of a Breakthrough MCTS-Solver player. The resulting players MCTS-MR-1 to MCTS-MR-4 played 1000 games each against regular MCTS-Solver with uniformly random rollouts. The results are presented in Figure 6.

Interestingly, all MCTS-MR players were significantly weaker than the baseline ($p<0.001$). The advantage of a 1- to 4-ply lookahead in rollouts does not seem to outweigh the computational cost in Breakthrough, possibly due to the larger branching factor, longer rollouts, and more time-consuming move generation than in Connect-4. MCTS-MR-1 searches only about 7.3% as fast as the baseline, MCTS-MR-2 about 1.3% as fast, MCTS-MR-3 about 0.2% as fast, MCTS-MR-4 about 0.03% as fast when measured in simulations completed in a one-second search of the empty Connect-4 board. When comparing with equal numbers of MCTS iterations ($10,000$) per move instead of equal time (1 second) per move for both players, MCTS-MR-1 achieved a win rate of 61.5% in 1000 games against the baseline. MCTS-MR-2 won 83.5% of 1000 games under the same conditions. It may be possible to optimize our Breakthrough implementation—however, as the following subsections indicate, application of minimax in other phases of MCTS seems to be the more promising approach in this game.

2) Minimax in the Selection and Expansion Phases: We tested the same variants of MCTS-MS for Breakthrough as for Connect-4: MCTS-MS-d-Visit-v for $d \in \{2, 4\}$ and $v \in \{0, 1, 2, 5, 10, 20, 50, 100\}$. 2000 games against the baseline

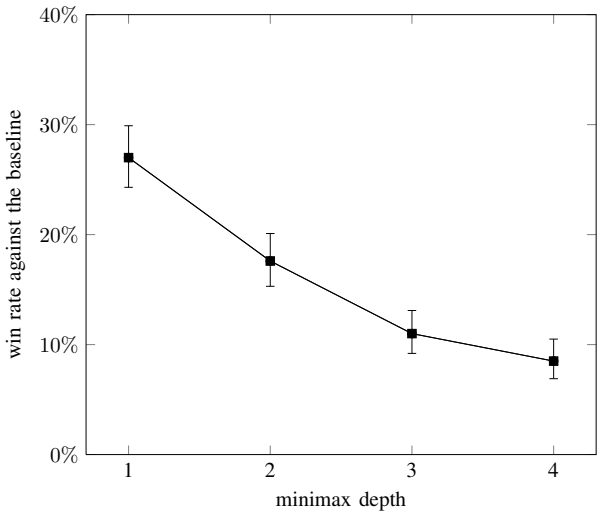

Fig. 6. Performance of MCTS-MR in Breakthrough.

player were played for each experimental condition. Figure 7 shows the results.

Just as in Connect-4, MCTS-MS-2-Visit-2 turned out to be the most effective variant. When averaged over the whole game, it performed about $47,000$ simulations per second. 2000 additional games against the baseline confirmed a significant increase in strength ($p<0.001$) with a win rate of 62.2%.

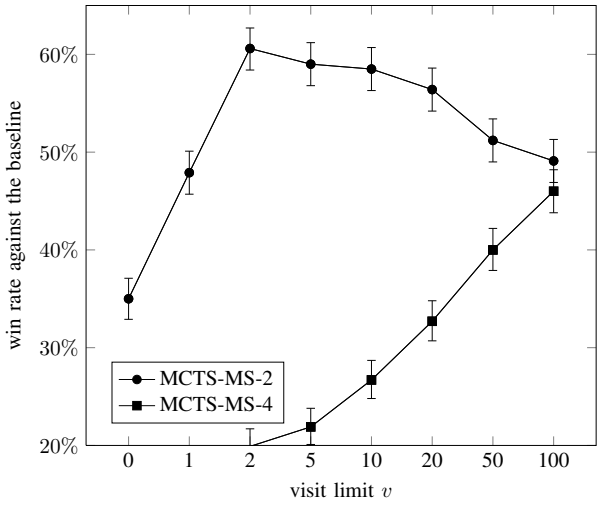

Fig. 7. Performance of MCTS-MS in Breakthrough.

3) Minimax in the Backpropagation Phase: MCTS-MB-2 to MCTS-MB-6 were tested analogously to Connect-4, playing 1000 games each against the regular MCTS-Solver baseline. Figure 8 presents the results.

The best-performing setting MCTS-MB-2 played 2000 additional games against the baseline and won 55.0% of them, which shows a significant improvement ($p<0.05$). It played about $60,000$ simulations per second on average.

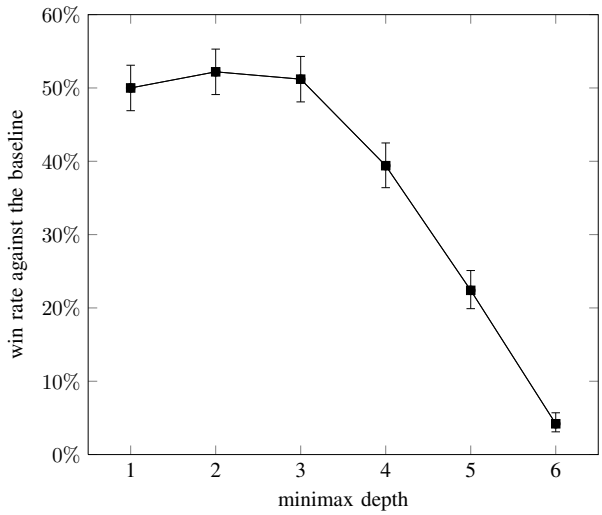

Fig. 8. Performance of MCTS-MB in Breakthrough.

E. Comparison of Algorithms

Sections V-C and V-D show the performance of MCTS-MR, MCTS-MS and MCTS-MB against the baseline player in both Connect-4 and Breakthrough. In order to facilitate comparison, we also tested the best-performing variants of these MCTS-minimax hybrids against each other. In Connect-4, MCTS-MR-2, MCTS-MS-2-Visit-2 and MCTS-MB-2 played in each possible pairing; in Breakthrough, MCTS-MR-1, MCTS-MS-2-Visit-2 and MCTS-MB-2 were chosen. 2000 games were played in each condition. Figure 9 presents the results.

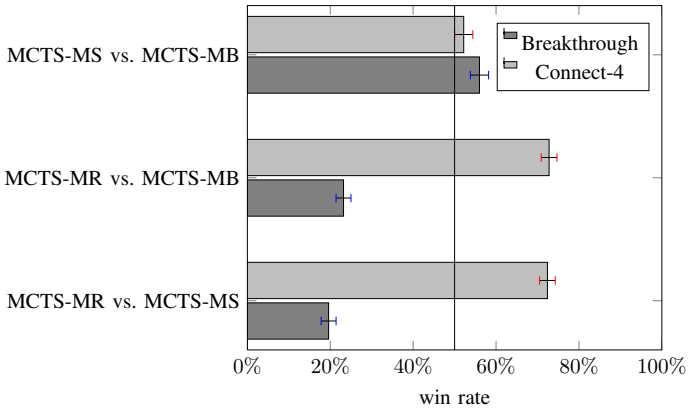

Fig. 9. Performance of MCTS-MR, MCTS-MS and MCTS-MB against each other in Connect-4 and Breakthrough.

Consistent with the results from the previous sections, MCTS-MS outperformed MCTS-MB in Breakthrough, while no significant difference could be shown in Connect-4. MCTS-MR was significantly stronger than the two other algorithms in Connect-4, but weaker than both in Breakthrough.

In a second experiment, the best-performing MCTS-minimax hybrids played against the baseline at different time settings from 250 ms per move to 5000 ms per move. The results are shown in Figure 10 for Connect-4, and Figure 11

for Breakthrough.

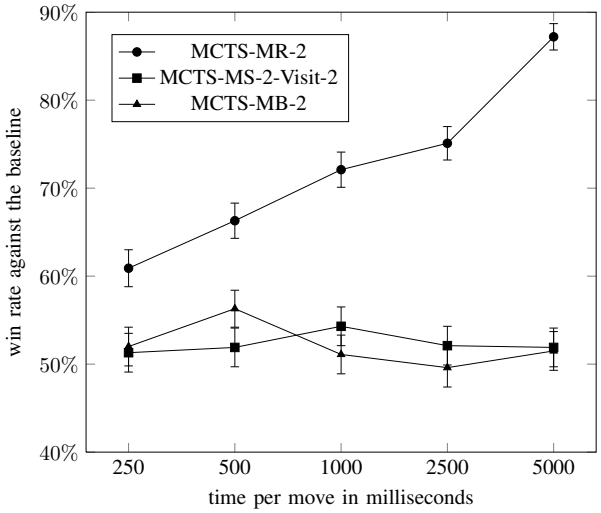

Fig. 10. Performance of MCTS-MR-2, MCTS-MS-2-Visit-2 and MCTS-MB-2 at different time settings in Connect-4.

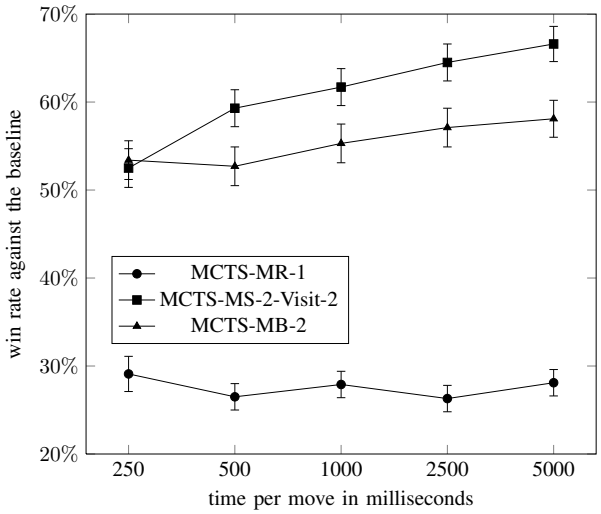

Fig. 11. Performance of MCTS-MR-1, MCTS-MS-2-Visit-2 and MCTS-MB-2 at different time settings in Breakthrough.

We can observe that at least up to 5 seconds per move, additional time makes the performance differences between algorithms more pronounced. While in Connect-4, it is MCTS-MR that profits most from additional time, we can see the same effect for MCTS-MS and MCTS-MB in Breakthrough.

VI. Conclusion and Future Research

The strategic strength of MCTS lies to a great extent in the Monte-Carlo simulations, allowing the search to observe even distant consequences of actions, if only through the observation of probabilities. The tactical strength of minimax lies largely in its exhaustive approach, guaranteeing to never miss a consequence of an action that lies within the search horizon, and backing up game-theoretic values from leaves with certainty and efficiency.

In this paper, we examined three knowledge-free ways of integrating minimax into MCTS: The application of minimax in the rollout phase with *MCTS-MR*, the selection and expansion phases with *MCTS-MS*, and the backpropagation phase with *MCTS-MB*. In both test domains of Connect-4 and Breakthrough, the newly proposed variants MCTS-MS and MCTS-MB significantly outperformed regular MCTS with the MCTS-Solver extension. The only way of integrating minimax into MCTS known from the literature (although typically used with an evaluation function), MCTS-MR, was quite strong in Connect-4 but weak in Breakthrough, suggesting it might be less robust with regard to differences between search spaces.

Note that in all experiments except those of Subsections V-D1 and V-C1, we used fast, uniformly random rollout policies. On the one hand, the overhead of our methods would be proportionally lower for any slower, informed rollout policies such as typically used in state-of-the-art programs. On the other hand, improvement on already strong policies might prove to be more difficult. Examining the influence of such MCTS implementation properties is a possible first direction of future research.

Second, the effect of properties of the games themselves deserves further study, e.g. average branching factor, average game length, trap density, and others. The successful application of MCTS-MB and MCTS-MS e.g. may depend on the frequency of terminal nodes found in the tree (or close to the tree, respectively) in the deciding phases of a game. Synthetic search spaces could be used to study these properties in isolation—the large number of differences between games like Connect-4 and Breakthrough potentially confounds many effects.

A third worthwhile direction of work is the incorporation of evaluation functions into the hybrid algorithms. This could make minimax potentially much more useful in regions of search spaces without or with very few terminal nodes. The main challenge for this approach is properly combining results of heuristic evaluation functions with the results of rollout returns, their averages and confidence intervals, as produced by MCTS.

Acknowledgment

This work is funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938.

References

[1] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *5th International Conference on Computers and Games (CG 2006). Revised Papers*, ser. Lecture Notes in Computer Science, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer, 2007, pp. 72–83.

[2] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *17th European Conference on Machine Learning, ECML 2006*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Springer, 2006, pp. 282–293.

[3] J. von Neumann and O. Morgenstern, *The Theory of Games and Economic Behavior*. Princeton: Princeton University Press, 1944.

[4] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[5] D. E. Knuth and R. W. Moore, "An Analysis of Alpha-Beta Pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[6] R. Ramanujan, A. Sabharwal, and B. Selman, "On Adversarial Search Spaces and Sampling-Based Planning," in *20th International Conference on Automated Planning and Scheduling, ICAPS 2010*, R. I. Brafman, H. Geffner, J. Hoffmann, and H. A. Kautz, Eds. AAAI, 2010, pp. 242–245.

[7] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008.

[8] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[9] B. E. Childs, J. H. Brodeur, and L. Kocsis, "Transpositions and Move Groups in Monte Carlo Tree Search," in *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, P. Hingston and L. Barone, Eds. IEEE, 2008, pp. 389–395.

[10] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, "Monte-Carlo Tree Search Solver," in *6th International Conference on Computers and Games, CG 2008*, ser. Lecture Notes in Computer Science, H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, Eds., vol. 5131. Springer, 2008, pp. 25–36.

[11] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[12] T. Cazenave and A. Saffidine, "Utilisation de la Recherche Arborescente Monte-Carlo au Hex," *Revue d'Intelligence Artificielle*, vol. 23, no. 2-3, pp. 183–202, 2009, in French.

[13] R. J. Lorentz, "Experiments with Monte-Carlo Tree Search in the Game of Havannah," *ICGA Journal*, vol. 34, no. 3, pp. 140–149, 2011.

[14] Y. Sato, D. Takahashi, and R. Grimbergen, "A Shogi Program Based on Monte-Carlo Tree Search," *ICGA Journal*, vol. 33, no. 2, pp. 80–92, 2010.

[15] N. G. P. Den Teuling and M. H. M. Winands, "Monte-Carlo Tree Search for the Simultaneous Move Game Tron," in *Computer Games Workshop at ECAI 2012*, 2012, pp. 126–141.

[16] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for Multi-Player Monte-Carlo Tree Search," in *7th International Conference on Computers and Games, CG 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, H. J. van den Herik, H. Iida, and A. Plaat, Eds., vol. 6515. Springer, 2011, pp. 238–249.

[17] T. Cazenave and A. Saffidine, "Score Bounded Monte-Carlo Tree Search," in *7th International Conference on Computers and Games, CG 2010. Revised Selected Papers*, ser. Lecture Notes in Computer Science, H. J. van den Herik, H. Iida, and A. Plaat, Eds., vol. 6515. Springer, 2010, pp. 93–104.

[18] H. Finnsson, "Simulation-Based General Game Playing," Ph.D. dissertation, Reykjavík University, Reykjavík, Iceland, 2012.

[19] R. Ramanujan and B. Selman, "Trade-Offs in Sampling-Based Adversarial Planning," in *21st International Conference on Automated Planning and Scheduling, ICAPS 2011*, F. Bacchus, C. Domshlak, S. Edelkamp, and M. Helmert, Eds. AAAI, 2011.

[20] R. Ramanujan, A. Sabharwal, and B. Selman, "On the Behavior of UCT in Synthetic Search Spaces," in *ICAPS 2011 Workshop on Monte-Carlo Tree Search: Theory and Applications*, 2011.

[21] H. Finnsson and Y. Björnsson, "Game-Tree Properties and MCTS Performance," in *IJCAI'11 Workshop on General Intelligence in Game Playing Agents (GIGA'11)*, 2011, pp. 23–30.

[22] J. E. Clune, "Heuristic Evaluation Functions for General Game Playing," Ph.D. dissertation, University of California, Los Angeles, USA, 2008.

[23] F. Teytaud and O. Teytaud, "On the huge benefit of decisive moves in Monte-Carlo Tree Search algorithms," in *2010 IEEE Conference on Computational Intelligence and Games, CIG 2010*, G. N. Yannakakis and J. Togelius, Eds. IEEE, 2010, pp. 359–364.

[24] M. H. M. Winands and Y. Björnsson, "Alpha-Beta-based Play-outs in Monte-Carlo Tree Search," in *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, S.-B. Cho, S. M. Lucas, and P. Hingston, Eds. IEEE, 2011, pp. 110–117.

[25] R. Ramanujan, A. Sabharwal, and B. Selman, "Understanding Sampling Style Adversarial Search Methods," in *26th Conference on Uncertainty in Artificial Intelligence, UAI 2010*, P. Grünwald and P. Spirtes, Eds., 2010, pp. 474–483.

[26] J. A. M. Nijssen and M. H. M. Winands, "Playout Search for Monte-Carlo Tree Search in Multi-player Games," in *13th International Conference on Advances in Computer Games, ACG 2011*, ser. Lecture Notes in Computer Science, H. J. van den Herik and A. Plaat, Eds., vol. 7168. Springer, 2011, pp. 72–83.

[27] L. V. Allis, M. van der Meulen, and H. J. van den Herik, "Proof-Number Search," *Artificial Intelligence*, vol. 66, no. 1, pp. 91–124, 1994.

[28] T. Kaneko, T. Kanaka, K. Yamaguchi, and S. Kawai, "Df-pn with Fixed-Depth Search at Frontier Nodes," in *Proceedings of the 10th Game Programming Workshop, GPW05*, 2005, pp. 1–8, in Japanese.

[29] M. H. M. Winands, J. W. H. M. Uiterwijk, and H. J. van den Herik, "Combining Proof-Number Search with Alpha-Beta Search," in *13th Belgium-Netherlands Conference on Artificial Intelligence, BNAIC 2001*, B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, Eds., 2001, pp. 299–306.

[30] J.-T. Saito, G. Chaslot, J. W. H. M. Uiterwijk, and H. J. van den Herik, "Monte-Carlo Proof-Number Search for Computer Go," ser. Lecture Notes in Computer Science, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer, 2007, pp. 50–61.

[31] H. Baier and M. H. M. Winands, "Nested Monte-Carlo Tree Search for Online Planning in Large MDPs," in *20th European Conference on Artificial Intelligence, ECAI 2012*, ser. Frontiers in Artificial Intelligence and Applications, L. De Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, Eds., vol. 242. IOS Press, 2012, pp. 109–114.

[32] T. Cazenave, "Nested Monte-Carlo Search," in *21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, C. Boutilier, Ed., 2009, pp. 456–461.

[33] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," HAL - CCSd - CNRS, France, Tech. Rep., 2006.

[34] H. Finnsson and Y. Björnsson, "Simulation-Based Approach to General Game Playing," in *23rd AAAI Conference on Artificial Intelligence, AAAI 2008*, D. Fox and C. P. Gomes, Eds. AAAI Press, 2008, pp. 259–264.

[35] M. Kirci, J. Schaeffer, and N. Sturtevant, "Feature Learning Using State Differences," in *Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09)*, 2009, pp. 35–42.

[36] S. Sharma, Z. Kobti, and S. D. Goodwin, "Knowledge Generation for Improving Simulations in UCT for General Game Playing," in *21st Australasian Conference on Artificial Intelligence, AI 2008*, ser. Lecture Notes in Computer Science, W. Wobcke and M. Zhang, Eds., vol. 5360. Springer, 2008, pp. 49–55.

[37] L. V. Allis, "A Knowledge-Based Approach of Connect-Four," Master's thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, 1988.