Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

MCTS-Minimax Hybrids with State Evaluations (Extended Abstract)*

Hendrik Baier[1] and **Mark H. M. Winands**[2]
[1]Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
[2]Maastricht University, Maastricht, The Netherlands
hendrik.baier@cwi.nl, m.winands@maastrichtuniversity.nl

Abstract

Monte-Carlo Tree Search (MCTS) has been found to show weaker play than minimax-based search in some tactical game domains. In order to combine the tactical strength of minimax and the strategic strength of MCTS, *MCTS-minimax hybrids* have been proposed in prior work. This article continues this line of research for the case where heuristic state evaluation functions are available. Three different approaches are considered, employing minimax in the rollout phase of MCTS, as a replacement for the rollout phase, and as a node prior to bias move selection. The latter two approaches are newly proposed. Results show that the use of enhanced minimax for computing node priors results in the strongest MCTS-minimax hybrid in the three test domains of Othello, Breakthrough, and Catch the Lion. This hybrid also outperforms enhanced minimax as a standalone player in Breakthrough, demonstrating that at least in this domain, MCTS and minimax can be combined to an algorithm stronger than its parts.

1 Introduction

Monte-Carlo Tree Search (MCTS) [Coulom, 2007; Kocsis and Szepesvári, 2006] is a best-first tree search algorithm based on Monte-Carlo simulations for state evaluation. It has shown considerable success in a variety of domains—see [Silver *et al.*, 2016] for the prominent recent example of Google DeepMind's AlphaGo, and [Browne *et al.*, 2012] for an earlier literature survey. However, there are still a number of adversarial domains such as the games of Chess and (International) Checkers in which the traditional approach to adversarial planning, minimax search with $\alpha\beta$ pruning [Knuth and Moore, 1975], remains superior. Part of the reason could be the selectivity of MCTS, its focusing on only the most promising lines of play. In tactical games such as Chess, a large number of terminal states and *shallow traps* exist in the search space [Ramanujan *et al.*, 2010a]. These require precise play to avoid immediate loss, and the selective sampling and averaging value backups of MCTS can easily miss or underestimate an important move. Conversely, MCTS could be more effective in domains such as Go, where terminal states and potential traps do not occur until the latest stage of the game. Here, MCTS can fully play out its strategic and positional understanding resulting from Monte-Carlo simulations of entire games.

Note that the recent publication of AlphaZero [Silver *et al.*, 2017] points towards MCTS possibly becoming the dominant approach even for Chess-like games. Deep reinforcement learning seems to be able to produce state evaluators that avoid traps surprisingly well on their own. Regardless of the state evaluator however, MCTS is still more susceptible to traps than alpha-beta, and deep neural networks may give a boost to alpha-beta as well. While AlphaZero opens new questions, this article could give an indication as to how the strengths of MCTS and alpha-beta can be combined using any evaluation function, hand-coded or learned.

In previous work [Baier and Winands, 2015], *MCTS-minimax hybrids* have been introduced, embedding shallow minimax searches into the MCTS framework. This was a first step towards combining the strategic strength of MCTS with the tactical strength of minimax. The results of the hybrid algorithms MCTS-MR, MCTS-MS, and MCTS-MB have been promising even without making use of domain knowledge such as heuristic evaluation functions. However, their inability to evaluate non-terminal states makes them ineffective in games with very few or no terminal states throughout most of the search space, such as the game of Othello. Furthermore, some form of state evaluation is often available in practice—as AlphaGo [Silver *et al.*, 2016] demonstrated, deep learning makes this possible even for domains where hand-crafting evaluation functions has traditionally been considered very difficult. This article therefore continues this line of research by addressing the case where domain knowledge is available.

The algorithms discussed in this article make use of *state evaluations*. These state evaluations can either be the result of simple evaluation function calls, or the result of minimax searches using the same evaluation function at the leaves. Three different approaches for integrating state evaluations into MCTS are considered. The first approach uses state evaluations to choose rollout moves (MCTS-*IR* for *informed rollouts*). The second approach uses state evaluations to terminate rollouts early (MCTS-*IC* for *informed cutoffs*). The third approach uses state evaluations to bias the selection of moves in the MCTS tree (MCTS-*IP* for *informed priors*). Using min-

*This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Baier and Winands, in press].

imax with $\alpha\beta$ to compute state evaluations means accepting longer computation times in favor of typically more accurate evaluations as compared to simple evaluation function calls. Only in the case of MCTS-IR, minimax has been applied before [Ramanujan *et al.*, 2010b]; the use of minimax for the other two approaches is newly proposed in the form described here. The MCTS-minimax hybrids are tested and compared to their counterparts using evaluation functions without minimax in the domains of Othello, Breakthrough, and Catch the Lion.

After the branching factor of a domain is identified as a limiting factor of the hybrids' performance, further experiments are conducted using domain knowledge not only for state evaluation, but also for *move ordering*. Move ordering reduces the average size of $\alpha\beta$ trees, and furthermore allows to restrict the effective branching factor of $\alpha\beta$ to only the k most promising moves in any given state (k*-best pruning*). Again, this has only been done for MCTS with minimax rollouts before [Winands *et al.*, 2010]. The enhanced MCTS-minimax hybrids with move ordering and k-best pruning are tested and compared to the unenhanced hybrids as well as the equivalent algorithms using static evaluations in all three domains. They are also tested against each other to determine the relative strongest hybrid, compared across domains, studied at different time settings and different branching factors, combined with each other, and finally compared to an $\alpha\beta$ baseline.

2 Hybrid Algorithms

This section describes the three different approaches for employing heuristic knowledge within MCTS that we explore in this article. For each approach, a variant using simple evaluation function calls and a hybrid variant using shallow minimax searches is considered. Two of the three hybrids are newly proposed in the form described here.

2.1 MCTS with Informed Rollouts (MCTS-IR)

The convergence of MCTS to the optimal policy is guaranteed even with uniformly random move choices in the rollouts. However, more informed rollout policies can greatly improve performance [Gelly *et al.*, 2006], and preserve convergence if they explore sufficiently. When a heuristic evaluation function is available, it can be used in every rollout step to compare the states each legal move would lead to, and choose the most promising one. Instead of choosing this *greedy* move, it is effective in some domains to choose a uniformly random move with a low probability ϵ, so as to avoid determinism and preserve diversity in the rollouts. Our implementation additionally ensures non-deterministic behavior even for $\epsilon = 0$ by picking moves with equal values at random both in the selection and in the rollout phase of MCTS. The resulting rollout policy is typically called ϵ*-greedy* [Sturtevant, 2008]. In the context of this work, we call this approach *MCTS-IR-E* (MCTS with informed rollouts using an evaluation function).

The depth-one lookahead of an ϵ-greedy policy can be extended in a natural way to a depth-d minimax search for every rollout move [Winands and Björnsson, 2011; Nijssen and Winands, 2012]. We use a random move ordering in minimax as well in order to preserve non-determinism. In contrast to [Winands and Björnsson, 2011] and [Nijssen and Winands, 2012] where several enhancements such as move ordering, k-best pruning, and killer moves were added to $\alpha\beta$, we first test unenhanced $\alpha\beta$ search in Subsection 3.1. We are interested in its performance before introducing additional improvements, especially since our test domains have smaller branching factors than e.g. the games Lines of Action (around 30) or Chinese Checkers (around 25-30) used in [Winands and Björnsson, 2011] and [Nijssen and Winands, 2012], respectively. Move ordering and k-best pruning are then added in Subsection 3.2. Using a depth-d minimax search for every rollout move aims at stronger move choices in the rollouts, which make rollout returns more accurate and can therefore help to guide the growth of the MCTS tree. We call this approach *MCTS-IR-M* (MCTS with informed rollouts using minimax).

2.2 MCTS with Informed Cutoffs (MCTS-IC)

The idea of rollout cutoffs is an early termination of the rollout in case the rollout winner, or the player who is at an advantage, can be reasonably well predicted with the help of an evaluation function. The statistical noise introduced by further rollout moves can then be avoided by stopping the rollout, evaluating the current state of the simulation, and backpropagating the evaluation result instead of the result of a full rollout to the end of the game [Lorentz, 2008; Winands *et al.*, 2010]. If on average, the evaluation function is computationally cheaper than playing out the rest of the rollout, this method can also result in an increased sampling speed as measured in rollouts per second. A fixed number m of rollout moves can be played before evaluating in order to introduce more non-determinism and get more diverse rollout returns. If $m = 0$, the evaluation function is called directly at the newly expanded node of the tree. As in MCTS-IR, our MCTS-IC implementation avoids deterministic gameplay through randomly choosing among equally valued moves in the selection policy. We scale all evaluation values to $[0, 1]$. In the following, we call this approach *MCTS-IC-E* (MCTS with informed cutoffs using an evaluation function).

We propose an extension of this method using a depth-d minimax search at cutoff time in order to determine the value to be backpropagated. In contrast to the integrated approach taken in [Winands and Björnsson, 2011], we do not assume MCTS-IR-M as rollout policy and backpropagate a win or a loss whenever the searches of this policy return a value above or below two given thresholds. Instead, we play rollout moves with an arbitrary policy (uniformly random unless specified otherwise), call minimax when a fixed number of rollout moves has been reached, and backpropagate the heuristic value returned by this search. Like MCTS-IR-M, this strategy tries to backpropagate more accurate rollout returns, but by computing them directly instead of playing out the rollout. We call this approach *MCTS-IC-M* (MCTS with informed cutoffs using minimax).

2.3 MCTS with Informed Priors (MCTS-IP)

Node priors [Gelly and Silver, 2007] represent one method for supporting the selection policy of MCTS with heuristic information. When a new node is added to the tree, or after it has been visited n times, the heuristic evaluation h of the corresponding state is stored in this node. This is done in the

form of virtual wins and virtual losses, weighted by a prior weight parameter γ. The following formulas show how to update the win (w) and visit (v) counters of the node at hand.

$$v \longleftarrow v + \gamma \tag{1a}$$

$$w \longleftarrow w + \gamma h \tag{1b}$$

We assume $h \in [0, 1]$. If the evaluation value h is 0.6 and the weight γ is 100, for example, 60 wins and 100 visits are added to the node at hand. This is equivalent to 60 virtual wins and $100 - 60 = 40$ virtual losses. Since heuristic evaluations are typically more reliable than the MCTS value estimates resulting from only a few samples, this prior helps to guide tree growth into a promising direction. If the node is visited frequently however, the influence of the prior progressively decreases over time, as the virtual rollout returns represent a smaller and smaller percentage of the total rollout returns stored in the node. Thus, MCTS rollouts progressively override the heuristic evaluation. We call this approach *MCTS-IP-E* (MCTS with informed priors using an evaluation function) in this article.

We propose to extend MCTS-IP with a depth-d minimax search in order to compute the prior value to be stored. This approach aims at guiding the selection policy through more accurate prior information in the nodes of the MCTS tree. We call this approach *MCTS-IP-M* (MCTS with informed priors using minimax).

3 Experimental Results

We tested the algorithms in three different domains: *Othello*, *Catch the Lion*, and *6×6 Breakthrough*. These are all deterministic perfect-information turn-taking zero-sum games. As baseline algorithm for comparison, we used $MCTS-Solver$, an MCTS variant that is able to handle proven game-theoretic values [Winands *et al.*, 2008].

We first briefly summarize the main takeaways of our experimental results for MCTS-IR-E, MCTS-IC-E, and MCTS-IP-E, as well as MCTS-IR-M, MCTS-IC-M, and MCTS-IP-M using unenhanced $\alpha\beta$. After identifying the branching factor of a domain as an important limiting factor for algorithm performance, we then provide the main findings of our improved results for MCTS-IR-M-k, MCTS-IC-M-k, and MCTS-IP-M-k using $\alpha\beta$ with move ordering and k-best pruning. Please refer to the full article for detailed results, and for a description of the heuristic board evaluation functions used in the three test domains.

3.1 Results with Unenhanced $\alpha\beta$

In all experimental conditions, we compared the hybrids (ending -M) as well as their counterparts using heuristics without minimax (ending -E) against regular MCTS-Solver as the baseline. Rollouts were uniformly random except for MCTS-IR. Optimal MCTS parameters such as the exploration factor C were determined once for MCTS-Solver in each game and then kept constant for both MCTS-Solver and the MCTS-minimax hybrids during testing. Draws, which are possible in Othello, were counted as half a win for both players. We used minimax with $\alpha\beta$ pruning, but no other search enhancements. Computation time was 1 second per move.

Main Findings

- MCTS-IR-E works well in all three domains. However, MCTS-IR-M does not improve on it in any domain. The reason is the high computational cost of calling minimax many times in each MCTS simulation.
- MCTS-IC-E works well in all domains but Breakthrough (the evaluation function we used in Breakthrough may not be accurate enough for MCTS to fully rely on it instead of rollouts). However, MCTS-IC-M does not improve on MCTS-IC-E in any domain, the reason again being the high computational cost of calling minimax even just once in every MCTS simulation.
- MCTS-IP-E works well in all domains. MCTS-IP-M improves on it in all domains but Breakthrough. The reason is probably that the parameters of MCTS-IP make it easier to control the computational cost, but the branching factor of Breakthrough (15.5 compared to 10.5 in Catch the Lion and 8 in Othello) is still causing problems.
- In conclusion, MCTS-IP-M seems to be a promising hybrid, but the sensitivity to the branching factor of the given domain is a problem for the hybrids in general.

3.2 Results with Move Ordering and k-best Pruning

In Subsection 3.1, $\alpha\beta$ search was used in its basic, unenhanced form. This was sufficient to improve MCTS-IP in Othello and Catch the Lion, but too computationally expensive for MCTS-IR and MCTS-IC in these two domains, as well as for all hybrids in Breakthrough. The performance difference between the hybrids can be explained by the fact that MCTS-IP allows to control the frequency of minimax calls with the parameter n, while MCTS-IC needs to call minimax once in every simulation, and MCTS-IR even for every single simulation move. This makes it easier for MCTS-IP to trade off the computational cost of embedded minimax searches against their advantages over static evaluation function calls. The performance difference between the domains can be explained by the larger branching factor of Breakthrough compared to Othello and Catch the Lion, which affects full-width minimax more strongly than for example the sampling-based MCTS. The main problem of MCTS-minimax hybrids seems to be their sensitivity to the branching factor of the domain.

We therefore conduct further experiments applying limited domain knowledge not only for state evaluation, but also for move ordering. The application of move ordering is known to strongly improve the performance of $\alpha\beta$ through a reduction of the average size of the search tree [Knuth and Moore, 1975]. Additionally, with a good move ordering heuristic one can restrict $\alpha\beta$ to only searching the k moves in each state that seem most promising to the heuristic (*k-best pruning*). The number of promising moves k is subject to empirical optimization. Move ordering and k-best pruning could make all MCTS-minimax hybrids viable in domains with much higher branching factors, including the newly proposed MCTS-IC-M and MCTS-IP-M. We call the hybrids with activated move ordering and k-best pruning *enhanced hybrids* or *MCTS-IR-M-k*, *MCTS-IC-M-k*, and *MCTS-IP-M-k*, respectively. Please refer to the full article for detailed results, for a description of

the move ordering functions used, and for an analysis of their effectiveness at reducing effective branching factors.

Main Findings

- MCTS-IR-M-k improves on MCTS-IR-E in all three domains. However, its strength comes from move ordering and k-best pruning alone, not from embedded minimax searches: The optimal minimax search depth is the minimal depth of 1.
- MCTS-IC-M-k improves on MCTS-IC-E in Othello and Breakthrough. However, its strength again comes from move ordering and k-best pruning, not from minimax – the optimal search depth for minimax is 1 here, too.
- MCTS-IP-M-k improves on MCTS-IP-E (and MCTS-IP-M) in all three domains, and truly profits from the embedded minimax searches (optimal minimax depths are larger than 1 in all test domains). It is significantly stronger than the best player found with unenhanced $\alpha\beta$ in all domains, and stronger than all other hybrids tested.
- A comparison of domains shows that all hybrids are most successful in Catch the Lion, probably due to the higher number of shallow hard traps in this Chess-like domain.
- MCTS-IP-M-k works well at all tested time settings from 250 ms per move to 5 s per move, but can overfit to the time settings it was tuned for.
- MCTS-IP-M-k and MCTS-IC-M-k, the hybrids proposed in this article, become considerably more effective as we increase the branching factor in Breakthrough by enlarging the board from 6×6 to 18×6. MCTS-IR-M-k cannot handle larger branching factors this well due to the much higher number of minimax calls.
- It is potentially useful to combine different ways of using identical domain knowledge in MCTS-minimax hybrids. MCTS-IP-M-k for example profits from the combination with MCTS-IR-M-k in Breakthrough and Catch the Lion.
- The combination of MCTS-IP-M-k and MCTS-IR-M-k outperforms both its MCTS part and its $\alpha\beta$ part in Breakthrough, demonstrating a successful combination of the advantages of the two search approaches. In Catch the Lion and Othello however, regular $\alpha\beta$ is still stronger than the best MCTS hybrids found for each domain.
- In conclusion, MCTS-IP-M-k is the strongest standalone MCTS-minimax hybrid investigated in all three tested domains. The use of enhanced minimax for computing node priors is therefore a promising new technique for integrating domain knowledge into an MCTS framework.

4 Conclusion and Future Research

In this article, we continued the research on MCTS-minimax hybrids for the case where domain knowledge in the form of heuristic evaluation functions is available. Three approaches for integrating such knowledge into MCTS were considered. MCTS-IR uses heuristic knowledge to improve the rollout policy. MCTS-IC uses heuristic knowledge to terminate rollouts early. MCTS-IP uses heuristic knowledge as prior for tree nodes. For all three approaches, we compared the computation of state evaluations through simple evaluation function calls (MCTS-IR-E, MCTS-IC-E, and MCTS-IP-E) to the computation of state evaluations through shallow-depth minimax searches using the same heuristic knowledge (MCTS-IR-M, MCTS-IC-M, and MCTS-IP-M).

Experiments with unenhanced $\alpha\beta$ in the domains of Othello, Breakthrough and Catch the Lion showed that the embedded minimax searches improve MCTS-IP in Othello and Catch the Lion, but are too computationally expensive for MCTS-IR and MCTS-IC in these two domains, as well as for all hybrids in Breakthrough. The main problem of MCTS-minimax hybrids with unenhanced $\alpha\beta$ seems to be the sensitivity to the branching factor of the domain at hand.

Further experiments introduced move ordering and k-best pruning to the hybrids in order to cope with this problem, resulting in the enhanced hybrid players called MCTS-IR-M-k, MCTS-IC-M-k, and MCTS-IP-M-k. Results showed that with these simple enhancements, MCTS-IP-M-k is the strongest standalone MCTS-minimax hybrid investigated in this article in all three tested domains. Because it does not have to call minimax in every rollout or even in every rollout move, it performs better than the other hybrids at low time settings when performance is most sensitive to a reduction in rollouts. It was also shown to work well at higher branching factors. Additionally, it was shown that the combination of MCTS-IP-M-k with minimax rollouts can lead to further improvements in Breakthrough and Catch the Lion. Moreover, the best-performing hybrid outperformed a simple $\alpha\beta$ implementation in Breakthrough, demonstrating that at least in this domain, MCTS and minimax can be combined to an algorithm stronger than its parts. MCTS-IP-M-k, the use of enhanced minimax for computing node priors, is therefore a promising new technique for integrating domain knowledge into an MCTS framework.

A first direction for future research is the application of additional $\alpha\beta$ enhancements. As a simple static move ordering has proven quite effective in all domains, one could for example experiment with dynamic move ordering techniques such as killer moves or the history heuristic.

Second, some combinations of the hybrids play at a higher level than the hybrids in isolation, despite using the same heuristic knowledge. This may mean we have not yet found a way to fully and optimally exploit this knowledge, which should be investigated further.

Third, differences between test domains such as their density of terminal states, their density of hard and soft traps, or their progression property [Finnsson and Björnsson, 2011] could be studied in order to better understand the behavior of MCTS-minimax hybrids with heuristic evaluation functions, and how they compare to standalone MCTS and minimax.

Finally, recent work on AlphaZero [Silver *et al.*, 2017] has demonstrated the impressive success of using deep reinforcement learning to train a neural network which encodes domain knowledge. This network was then used within MCTS both for biasing the selection policy as well as for replacing the rollout policy with a state evaluation. It remains an interesting line of future work to see if powerful function approximators such as deep neural networks can also be even more effectively used in hybrid search algorithms such as those proposed here.

References

[Baier and Winands, 2015] Hendrik Baier and Mark H. M. Winands. MCTS-Minimax Hybrids. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2):167–179, 2015.

[Baier and Winands, in press] Hendrik Baier and Mark H. M. Winands. MCTS-Minimax Hybrids with State Evaluations. *Journal of Artificial Intelligence Research*, in press.

[Browne *et al.*, 2012] Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[Coulom, 2007] R. Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *5th International Conference on Computers and Games, CG 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83, 2007.

[Finnsson and Björnsson, 2011] Hilmar Finnsson and Yngvi Björnsson. Game-Tree Properties and MCTS Performance. In *IJCAI 2011 Workshop on General Intelligence in Game Playing Agents, GIGA'11*, pages 23–30, 2011.

[Gelly and Silver, 2007] Sylvain Gelly and David Silver. Combining Online and Offline Knowledge in UCT. In Z. Ghahramani, editor, *24th International Conference on Machine Learning, ICML 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 273–280, 2007.

[Gelly *et al.*, 2006] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with Patterns in Monte-Carlo Go. Technical report, HAL - CCSd - CNRS, France, 2006.

[Knuth and Moore, 1975] Donald E. Knuth and Ronald W. Moore. An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *17th European Conference on Machine Learning, ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293, 2006.

[Lorentz, 2008] Richard J. Lorentz. Amazons Discover Monte-Carlo. In van den Herik et al. [2008], pages 13–24.

[Nijssen and Winands, 2012] J. A. M. Nijssen and Mark H. M. Winands. Playout Search for Monte-Carlo Tree Search in Multi-player Games. In H. J. van den Herik and Aske Plaat, editors, *13th International Conference on Advances in Computer Games, ACG 2011*, volume 7168 of *Lecture Notes in Computer Science*, pages 72–83, 2012.

[Ramanujan *et al.*, 2010a] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On Adversarial Search Spaces and Sampling-Based Planning. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *20th International Conference on Automated Planning and Scheduling, ICAPS 2010*, pages 242–245, 2010.

[Ramanujan *et al.*, 2010b] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. Understanding Sampling Style Adversarial Search Methods. In Peter Grünwald and Peter Spirtes, editors, *26th Conference on Uncertainty in Artificial Intelligence, UAI 2010*, pages 474–483, 2010.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Silver *et al.*, 2017] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR*, abs/1712.01815, 2017.

[Sturtevant, 2008] Nathan R. Sturtevant. An Analysis of UCT in Multi-Player Games. *ICGA Journal*, 31(4):195–208, 2008.

[van den Herik *et al.*, 2008] H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, editors. volume 5131 of *Lecture Notes in Computer Science*, 2008.

[Winands and Björnsson, 2011] Mark H. M. Winands and Yngvi Björnsson. Alpha-Beta-based Play-outs in Monte-Carlo Tree Search. In Sung-Bae Cho, Simon M. Lucas, and Philip Hingston, editors, *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, pages 110–117, 2011.

[Winands *et al.*, 2008] Mark H. M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte-Carlo Tree Search Solver. In van den Herik et al. [2008], pages 25–36.

[Winands *et al.*, 2010] Mark H. M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239–250, 2010.