AIIDE 2012 Doctoral Consortium
AAAI Technical Report WS-12-18

Research Summary

Hendrik Baier
Games and AI Group
Department of Knowledge Engineering
Maastricht University, The Netherlands
hendrik.baier@maastrichtuniversity.nl

Introduction

Monte-Carlo Tree Search (MCTS) (Coulom 2007; Kocsis and Szepesvári 2006) is an online planning algorithm that combines the ideas of best-first tree search and Monte-Carlo evaluation. Since MCTS is based on sampling, it does not require a transition function in explicit form, but only a generative model of the domain. Because it grows a highly selective search tree guided by its samples, it can handle huge search spaces with large branching factors. By using Monte-Carlo playouts, MCTS can take long-term rewards into account even with distant horizons. Combined with multi-armed bandit algorithms to trade off exploration and exploitation, MCTS has been shown to guarantee asymptotic convergence to the optimal policy (Kocsis and Szepesvári 2006), while providing approximations when stopped at any time.

The relatively new MCTS approach has started a revolution in computer Go and increased the level of play of computer Go programs substantially (Gelly and Silver 2008; Lee et al. 2009). Furthermore, it has achieved considerable success in domains as diverse as the games of Hex (Arneson, Hayward, and Henderson 2010), Amazons (Lorentz 2008), LOA (Winands, Björnsson, and Saito 2010), and Ms. Pacman (Ikehata and Ito 2011); in General Game Playing (Finnsson and Björnsson 2008), planning (Nakhost and Müller 2009; Silver and Veness 2010), and optimization (de Mesmay et al. 2009; Sabharwal and Samulowitz 2011). See (Browne et al. 2012) for a survey of MCTS applications and enhancements.

Whereas in the first instance the focus of MCTS research has been on the practical application, current research begins to address the problem of understanding the nature, the underlying principles, of MCTS. A careful understanding of the nature of MCTS will lead to more effective search algorithms. Hence, my two interrelated research questions are:

1. How can we formulate models that increase our understanding of how Monte-Carlo Tree Search (MCTS) works? and
2. How can we use the developed MCTS models to create effective search algorithms?

In the near future, increased research effort into these directions might enable us to win an even game against a Go professional; and in the longer term, maybe even against a professional StarCraft player (Balla and Fern 2009). Applications outside the field of games are numerous (Browne et al. 2012).

Background

Monte-Carlo Tree Search (MCTS) (Coulom 2007; Kocsis and Szepesvári 2006) is a best-first search algorithm using statistical sampling to evaluate states. For each move decision of the player, MCTS constructs a search tree T, starting from the current game state as root. This tree is selectively deepened into the direction of the most promising moves, which are determined by the success of simulated games starting with these moves. After n simulated games, the tree contains $n + 1$ states, for which distinct value estimates are maintained.

MCTS works by repeating the following four-phase loop until computation time runs out (Chaslot et al. 2008). Each loop represents one simulated game.

Phase one: *selection*. The tree is traversed from the root to one of the leaves. At each node, MCTS uses a selection policy to choose the move to sample from this state. Critical is a balance of exploitation of states with high value estimates and exploration of states with uncertain value estimates.

Phase two: *expansion*. When a leaf has been reached, one (or more) of its successors are added to the tree.

Phase three: *playout*. A playout (also called *rollout*) policy is used to choose moves until the game ends. Uniformly random move choices can be sufficient to achieve convergence of MCTS to the optimal move in the limit, but playout policies utilizing basic domain knowledge often improve convergence speed considerably.

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

Listing 1 shows pseudocode of the basic MCTS algorithm.

In a variety of applications, a variant of MCTS called *Upper Confidence Bounds for Trees* (UCT) (Kocsis and Szepesvári 2006) has shown excellent performance. UCT uses the UCB1 formula, originally developed for the multi-armed bandit problem (Auer, Cesa-Bianchi, and Fischer

```
MCTS(startState) {
   while(there is time left) {
      currentState ← startState
      simulation ← {currentState}
      while(currentState ∈ Tree) {
         currentState ← selectionPolicy(currentState)
         simulation ← simulation + currentState
      }
      addToTree(currentState)
      while(simulation not ended) {
         currentState ← playoutPolicy(currentState)
         simulation ← simulation + currentState
      }
      result = cumulativeReward(simulation)
      forall(state ∈ simulation) {
         state.value ← backPropagate(state.value, result)
      }
   }
   return action(bestChild(startState))
}
```

Listing 1: MCTS pseudocode.

2002), as a selection policy for balancing exploration and exploitation. UCB1 guarantees that the expected loss due to not always selecting the best move (*regret* growth) stays within a constant factor of the best possible bound.

Described in the framework of reinforcement learning, there are two interacting processes within MCTS.

Policy evaluation: In the backpropagation phase after each simulated game, the result of that game is used to update the value estimates of each visited state $s \in T$.

$$n_s \longleftarrow n_s + 1 \tag{1a}$$

$$\hat{V}^\pi(s) \longleftarrow \hat{V}^\pi(s) + \frac{r - \hat{V}^\pi(s)}{n_s} \tag{1b}$$

where n_s is the number of times state s has been traversed in all simulations so far, r is the reward received at the end of the current simulation, and $\hat{V}^\pi(s)$ is the current estimate of the value of state s for a player following policy π.

Policy improvement: During each simulation, the policy adapts to the current value estimates. In case of MCTS using UCB1 in the selection phase (UCT), and a uniformly random policy in the playout phase:

$$\pi(s) = \begin{cases} \underset{m \in A(s)}{\operatorname{argmax}} \left(\hat{V}^\pi\big(P_m(s)\big) + E \times \sqrt{\frac{2\ln(n_s)}{n_{s.m}}} \right) & \text{if } s \in T \\ \operatorname{random}(s) & \text{otherwise} \end{cases}$$

where $A(s)$ is the set of available moves in state s, $P_m(s)$ is the position reached from state s with move m, $n_{s,m}$ is the number of times move m has been chosen in state s in all simulations so far, $\operatorname{random}(s)$ chooses one of the moves available in s with uniform probability, and E is an exploration coefficient, which has a domain-dependent optimum.

A crucial result of (Kocsis and Szepesvári 2006) is that as the number of simulations grows to infinity, the probability of UCT selecting a suboptimal move at the root of the tree converges to zero.

Since their initial success in the game of Go, the ideas of MCTS has been applied and adapted to many other domains such as one-player games and planning tasks, multi-player games, stochastic or partially observable environments, continous state or action spaces, games with simultaneous moves, or real-time games. Enhancements include the integration of expert heuristics, parallelization techniques, the addition of solving capabilities, move pruning strategies, and many more (cf. (Browne et al. 2012)).

Research Plan

In the context outlined above, my Ph.D. research revolves around the following objectives:

1. While MCTS has shown noteworthy results in many applications, its underlying principles are not fully understood. In a similar way to how Don Beal's analysis of minimax (Beal 1999) benefitted minimax algorithms and enhancements, I expect an investigation of the *nature* of MCTS to be valuable for research on sample-based search algorithms. An example for this is the analysis of the properties of search spaces that have beneficial or detrimental effects on the performance of MCTS. On the theoretical side, it is therefore my goal to formulate models that increase our understanding of the fundamentals of Monte-Carlo Tree Search.
2. On the applied side, I expect that increased understanding of MCTS will lead to algorithmic improvements, and help overcome some known shortcomings of MCTS. One example is the difficulty MCTS has with finding precise move sequences—its tactical weakness as opposed to strategic strength. Another one is the horizon effect arising when a negative outcome is found in one branch of the tree, but is continually pushed beyond the search tree as long as other, distracting branches are available to explore. In conclusion, I plan to devise, implement and test improved search algorithms related to or based on MCTS.

Finished Work

In this section, I present my publications as first author. The first two subsections deal with research in the two-player game of Go, while the last two subsections are concerned with work on one-player games (puzzles).

Opening Books for MCTS

MCTS is nowadays the dominant approach for playing the game of Go. However, MCTS does not perform well in the opening phase of the game, as the branching factor is high and consequences of moves can be far delayed. Human knowledge about Go openings is typically captured in *joseki*, local sequences of moves that are considered optimal for both players. The choice of the correct joseki in a given whole-board position, however, is difficult to formalize.

The paper "Active Opening Book Application for Monte-Carlo Tree Search in 19x19 Go" (Baier and Winands 2011) presents an approach to successfully apply global as well as

local opening moves, extracted from databases of high-level game records, in the MCTS framework. Instead of blindly playing moves that match local joseki patterns (*passive* opening book application), knowledge about these moves is integrated into the search algorithm by the techniques of move pruning and move biasing (*active* opening book application). Thus, the opening book serves to nudge the search into the direction of tried and tested local moves, while the search is able to filter out locally optimal, but globally problematic move choices.

In our experiments, active book application outperforms passive book application and plain MCTS in 19×19 Go.

Time Management for MCTS

While MCTS allows for fine-grained time control, little has been published on time management for MCTS programs under tournament conditions. The paper "Time Management for Monte-Carlo Tree Search in Go" (Baier and Winands 2012c) investigates the effects that various time-management strategies have on the playing strength in Go. We consider strategies taken from the literature as well as newly proposed and improved ones. We investigate both *semi-dynamic* strategies that decide about time allocation for each search before it is started, and *dynamic* strategies that influence the duration of each move search while it is already running. In our experiments, two domain-independent enhanced strategies, EARLY-C and CLOSE-N, each provide a significant improvement over the state of the art.

Nested Monte-Carlo Tree Search

MCTS is state of the art for online planning in large MDPs. Its playouts are typically random or directed by a simple, domain-dependent heuristic. In the paper "Nested Monte-Carlo Tree Search for Online Planning in Large MDPs" (Baier and Winands 2012b), we propose *Nested Monte-Carlo Tree Search* (NMCTS), in which MCTS itself is recursively used to provide a playout policy for higher-level searches. In three large-scale MDPs, SameGame, Clickomania and Bubble Breaker, we show that NMCTS is significantly more effective than regular MCTS at equal time controls, both using random and heuristic playouts at the base level. Experiments also suggest superior performance to Nested Monte-Carlo Search (NMCS) in some domains.

Beam Monte-Carlo Tree Search

MCTS has successfully been applied to various multi- and one-player games (puzzles). *Beam search* is a search method that only expands a limited number of promising nodes per tree level, thus restricting the space complexity of the underlying search algorithm to linear in the tree depth. The paper "Beam Monte-Carlo Tree Search" (Baier and Winands 2012a) presents *Beam Monte-Carlo Tree Search* (BMCTS), combining the ideas of MCTS and beam search. Like MCTS, BMCTS builds a search tree using Monte-Carlo simulations as state evaluations. When a predetermined number of simulations has traversed the nodes of a given tree depth, these nodes are sorted by their estimated value, and only a fixed number of them is selected for further exploration. In our experiments with the puzzles SameGame, Clickomania and Bubble Breaker, BMCTS significantly outperforms MCTS at equal time controls. We show that the improvement is equivalent to an up to four-fold increase in computing time for MCTS.

Work in Progress

I am currently working on the following topics.

Hybrid Search Algorithms

While MCTS has shown considerable success in games such as Go, Hex and others, 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, remains superior. This weakness of MCTS cannot always be explained by the existence of effective evaluation functions, as evaluation functions have been successfully combined with MCTS to produce strong players in e.g. Amazons and Lines of Action.

Since MCTS builds an asymmetric and sparse search tree, focusing only on the most promising lines of play, it has been conjectured that traditional, full-width alpha-beta search could be more appropriate in search spaces containing a large number of *traps* (Ramanujan, Sabharwal, and Selman 2010)—situations in which very precise play is required to avoid loss, and sampling-based methods with averaged value estimates can easily be lead astray. I am interested in testing this hypothesis about search space properties favoring MCTS or alpha-beta. Furthermore, I am aiming at developing hybrid approaches which could combine the strengths of both MCTS and alpha-beta to produce more universally successful search algorithms.

Change Point Detection

MCTS uses the averages of simulation returns to compute value estimates for each state in the search tree. While these estimates can be shown to converge to the optimal (minimax) values in the limit, convergence in practice can be slow due to the difficulty of recovering from an early, over-optimistic estimate. It would therefore be advantageous to detect sudden changes in the simulation returns from a certain node, in order to adjust this node's value estimate more quickly.

Change point detection (Basseville 1988) is the task of discovering abrupt changes in the properties of sequential data. I hope to successfully apply statistical methods from this field, which have already been used in the classic multi-armed bandit setting (Gelly and Wang 2006), to improve MCTS.

Adaptive Playouts

Learning in MCTS has until now been generally restricted to the tree, where value estimates for positions in the player's immediate future are constructed from sample returns. In the playout phase however it is still common to rely solely on static knowledge. Playout policies are generally handcrafted or acquired offline through machine learning techniques. The inability to improve playouts while searching

leads to systematically biased estimates in many critical situations.

Simulated games provide more information than their result: The complete sequence of moves leading to the result, even if relatively myopic and random due to simple and exploratory policies, can potentially be input to a learning process. The RAVE technique (Gelly and Silver 2007) serves as a convincing demonstration. An ongoing interest of mine since my Master's thesis is utilizing this information throughout entire simulated games, not only in the selection phase, but in particular in the playout phase.

Acknowledgement

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

References

Arneson, B.; Hayward, R. B.; and Henderson, P. 2010. Monte Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence in Games* 2(4):251–258.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47(2-3):235–256.

Baier, H., and Winands, M. H. M. 2011. Active Opening Book Application for Monte-Carlo Tree Search in 19×19 Go. In P. De Causmacker, J. Maervoet, T. M. K. V., and Vermeulen, T., eds., *23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, 3–10.

Baier, H., and Winands, M. H. M. 2012a. Beam Monte-Carlo Tree Search. In *2012 IEEE Conference on Computational Intelligence and Games (CIG 2012)*. Submitted.

Baier, H., and Winands, M. H. M. 2012b. Nested Monte-Carlo Tree Search for Online Planning in Large MDPs. In *20th European Conference on Artificial Intelligence (ECAI 2012)*. Accepted.

Baier, H., and Winands, M. H. M. 2012c. Time Management for Monte-Carlo Tree Search in Go. In van den Herik, H., and Plaat, A., eds., *13th International Conference on Advances in Computer Games (ACG 2011)*, volume 7168 of *Lecture Notes in Computer Science*. In press.

Balla, R.-K., and Fern, A. 2009. UCT for Tactical Assault Planning in Real-Time Strategy Games. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 40–45.

Basseville, M. 1988. Detecting changes in signals and systems - A survey. *Automatica* 24(3):309–326.

Beal, D. F. 1999. *The Nature of Minimax Search*. Ph.D. Dissertation, Maastricht University.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Chaslot, G. M. J.-B.; Winands, M. H. M.; van den Herik, H. J.; Uiterwijk, J. W. H. M.; and Bouzy, B. 2008. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation* 4(3):343–357.

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

de Mesmay, F.; Rimmel, A.; Voronenko, Y.; and Püschel, M. 2009. Bandit-Based Optimization on Graphs with Application to Library Performance Tuning. In Danyluk, A. P.; Bottou, L.; and Littman, M. L., eds., *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, 729–736.

Finnsson, H., and Björnsson, Y. 2008. Simulation-Based Approach to General Game Playing. In Fox, D., and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 259–264.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In Ghahramani, Z., ed., *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML 2007)*, volume 227 of *ACM International Conference Proceeding Series*, 273–280.

Gelly, S., and Silver, D. 2008. Achieving Master Level Play in 9 x 9 Computer Go. In Fox, D., and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 1537–1540.

Gelly, S., and Wang, Y. 2006. Exploration Exploitation in Go: UCT for Monte-Carlo Go. In *NIPS-2006: On-line trading of Exploration and Exploitation Workshop*.

Ikehata, N., and Ito, T. 2011. Monte-Carlo Tree Search in Ms. Pac-Man. In *2011 IEEE Conference on Computational Intelligence and Games (CIG 2011)*, 39–46.

Kocsis, L., and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, 282–293.

Lee, C.-S.; Wang, M.-H.; Chaslot, G. M. J.-B.; Hoock, J.-B.; Rimmel, A.; Teytaud, O.; Tsai, S.-R.; Hsu, S.-C.; and Hong, T.-P. 2009. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in Games* 1(1):73–89.

Lorentz, R. J. 2008. Amazons Discover Monte-Carlo. In van den Herik, H. J.; Xu, X.; Ma, Z.; and Winands, M. H. M., eds., *Proceedings of the 6th International Conference on Computers and Games (CG 2008)*, 13–24.

Nakhost, H., and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1766–1771.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2010. On Adversarial Search Spaces and Sampling-Based Planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 242–245.

Sabharwal, A., and Samulowitz, H. 2011. Guiding Combinatorial Optimization with UCT. In *ICAPS 2011 Workshop on Monte-Carlo Tree Search: Theory and Applications*.

Silver, D., and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In Lafferty, J. D.; Williams, C. K. I.; Shawe-Taylor, J.; Zemel, R. S.; and Culotta, A., eds., *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, 2164–2172.

Winands, M. H. M.; Björnsson, Y.; and Saito, J.-T. 2010. Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games* 2(4):239–250.