Monte Carlo Tree Search with Heuristic Evaluations using Implicit Minimax Backups

Authors

Abstract

Monte Carlo Tree Search (MCTS) has improved the performance of game-playing engines in domains such as Go, Hex, and general-game playing. MCTS has been shown to outperform classic minimax search in games where good heuristic evaluations are difficult to obtain. In recent years, combining ideas from traditional minimax search in MCTS has been shown to be advantageous in some domains, such as Lines of Action, Amazons, Breakthrough, Connect Four, and Kalah. In this paper, we propose a new way to use heuristic evaluations to guide the MCTS search by storing the two sources of information, estimated win rates and heuristic evaluations, separately. Rather than using the heuristic evaluations to replace the playouts, our technique backs them up *implicity* during its MCTS simulations. These learned evaluation values are then used to guide future simulations. Compared to current techniques, we show that using implicit minimax backups reduces regret in an example game and leads to stronger play performance in Breakthrough, Lines of Action, and Kalah, (...and Hearts?).

Introduction

Monte Carlo Tree Search (MCTS) (Coulom 2007; Kocsis and Szepesvári 2006) is a simulation-based best-first search paradigm that has been shown to increase performane in domains such as turn-taking games, general-game playing, real-time strategy games, single-agent planning, and more (Browne et al. 2012). While the initial applications have been to games where heursitic evaluations are difficult to obtain, progress in MCTS research has shown that heuristics can be effectively be combined in MCTS, even in games where class minimax search has traditonally been preferred.

The most popular Monte Carlo tree search algorithm is UCT (Kocsis and Szepesvári 2006), an algorithm which performs a single simulation from the root of the search tree to a terminal state at each iteration. During the iterative process, a model of the game tree is incrementally built by adding a new leaf node to the tree on each iteration. The nodes in the tree are used to store statistical information regarding wins and losses, and backpropagation policies are used to update parent estimates, and these improving estimates are then used to guide future simulations. When the simulation reaches parts of the game tree not included in the model, a default playout policy is used to simulate to a terminal state where a win or loss is determined.

In this work, we propose to augment MCTS with the help of an implicitly-computed minimax search which uses heuristic evaluations. Unlike previous work, these heuristic evaluations are used as *separate source of information*, and backed up in the same way as in classic minimax search. These minimax-style backups are done *implicitly* as a simple extra step during the standard simulation, and always maintained separately from win rate estimates obtained from playouts. These two separate information sources are then used to guide the MCTS search. We show that combining heuristic evaluations in this way reduces simple regret and improves performance in four different games.

Related Work

Several techniques for minimax-influenced backup rules in the simulation-based MCTS framework have already been proposed. The first was Coulom's original *maximum backpropagation* (Coulom 2007). This method of backpropagation suggests, after a number of simulations to a node has been reached, to switch to propagating the maximum value instead of the simulated (average) value. The rationale behind this choice is that after a certain point, the search algorithm should consider the node *converged* and return an estimate of the best value. Maximum backpropagation has also been used in other Monte Carlo tree search algorithms and demonstrated success in probabilistic planning, as an alternative type of forecaster in BRUE (Feldman and Domshlak 2013) and as Bellman backups for online dynamic programming in Trial-based Heuristic Tree Search (Keller and Helmert 2013).

The first use of enhancing MCTS using prior knowledge was in Computer Go (Gelly and Silver 2007). In this work, offline-learned knowledge initialized values of expanded nodes increased performance against significantly against strong benchmark player. This technique was also confirmed to be advantageous in Breakthrough (Lorentz and Horey 2013). Another way to introduce prior knowledge is a progressive bias during selection (Chaslot et al. 2008), which also showed increased performance in Go play strength.

In games where minimax search performs well, such

as Kalah[1] modifying MCTS to use minimax-style backups and heuristic values instead to replace playouts offers a worthwhile trade-off under different search time settings (Ramanujan and Selman 2011). Similarly, there is further evidence suggesting not replacing the playout entirely, but terminating them early using heuristic evaluations, has increased the performance in Lines of Action (LOA) (Winands, Björnsson, and Saito 2010), Amazons (Kloetzer 2010; Lorentz 2008), and Breakthrough (Lorentz and Horey 2013). In LOA and Amazons, the MCTS players enhanced with evaluation functions outperform their minimax counterparts using the same evaluation function.

One may want to combine minimax backups or searches without using an evaluation function. The prime example is MCTS-Solver (Winands, Björnsson, and Saito 2008). When using MCTS-Solver, proven wins and losses are backpropagated as extra information in MCTS. When a node is proven to be a win or a loss, it no longer needs to be searched. This simple, domain-independent modification greatly enhances MCTS in many games (some examples). Score-bounded MCTS extends this idea to games with multiple outcomes, leading to $\alpha\beta$ style pruning in the tree (Cazenave and Saffidine 2010). Finally, one can use hybrid minimax searches in the tree to initialize nodes during, enhance the playout, or to help MCTS-Solver in backpropagation (Baier and Winands 2013).

Finally, recent work has attempted to explain and identify some of the shortcomings that arise from estimates in MCTS, specifically compared to situations where classic minimax search has historically performed well (Ramanujan, Sabharwal, and Selman 2010b; 2010a). Attempts have been made to overcome the problem of *traps* or *optimistic moves* (strategic tactial lines that are problematic for MCTS) such as sufficiency thresholds during selection (Gudmundsson and Björnsson 2013) and shallow minimax searches (Baier and Winands 2013). We show that implicit minimax backups also improve performance in domains with tactical short-term goals.

Adversarial Search in Turn-based Games

A finite deterministic Markov Decision Process (MDP) is 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. Here, $\mathcal{S}$ is a finite non-empty set of *states*. $\mathcal{A}$ is a finite non-empty set of *actions*, where we denote $\mathcal{A}(s) \subseteq \mathcal{A}$ the set of available actions at state s. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \Delta\mathcal{S}$ is a *transition function* mapping each state and action to a distribution over successor states. Finally, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \Re$ is a *reward function* mapping (state, action, successor state) triplets to numerical rewards.

A two-player perfect information game is an MDP with a specific form. Denote $\mathcal{Z} = \{s \in \mathcal{S} : \mathcal{A}(s) = \emptyset\} \subset \mathcal{S}$ the set of *terminal states*. In addition, for all nonterminal states $s' \in \mathcal{S} - \mathcal{Z}$, $\mathcal{R}(s, a, s') = 0$. There is a *player identity function* $\tau : \mathcal{S} - \mathcal{Z} \mapsto \{1, 2\}$. In this paper, we assume fully deterministic domains, so $\mathcal{T}(s, a)$ maps to a single state. When it is clear from the context and unless otherwise stated, de denote $s' = \mathcal{T}(s, a)$. However, the ideas proposed can be easily extended to domains with stochastic transitions.

Monte Carlo Tree Search is a simulation-based best-first search algorithm that incrementally builds a model of the game $\mathcal{G}$, in memory. Each search starts with from a *root state* $s_0 \in \mathcal{S} - \mathcal{Z}$, and initially sets $\mathcal{G} = \emptyset$. Each simulation samples a trajectory $\rho = (s_0, a_0, s_1, a_1, \cdots, s_n)$, where $s_n \in \mathcal{Z}$. The portion of the ρ where $s_i \in \mathcal{G}$ is called the *tree portion* and the remaining portion is called the *playout portion*. In the tree portion, actions are chosen according to some *selection policy*. The first state encountered in the playout portion is *expanded* (added to $\mathcal{G}$.) The actions chosen in the playout portion are determined by a specific *playout policy*. States $s \in \mathcal{G}$ are referred to as *nodes* and statistics are maintained for each node s: the cumulative reward, r_s, and visit count, n_s. By popular convention, we define $r_{s,a} = r_{s'}$ where $s' = \mathcal{T}(s, a)$, and similarly $n_{s,a} = n_{s'}$. Also, we use r_s^τ to denote the reward at state s *with respect to player* $\tau(s)$.

Let $\hat{V}(s, a)$ be an estimator for node s and $\hat{Q}(s, a)$ for the state-action pair. For example, one popular estimator is the observed mean over all simulations $\bar{Q}(s, a) = r_{s,a}^\tau / n_{s,a}$. The most widely-used is based on a bandit algorithm called Upper Confidence Bounds (UCB) (Auer, Cesa-Bianchi, and Fischer 2002) in an algorithm adapted for MCTS called UCT (Kocsis and Szepesvári 2006), which selects action a' using

$$a' = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \left\{ \hat{Q}(s, a) + C\sqrt{\frac{\ln n_s}{n_{s,a}}} \right\}, \qquad (1)$$

where C is parameter determining the weight of exploration.

Implicit Minimax Backups in MCTS

Now, suppose we are given an evaluation function $v_0(s)$ whose range is the same as that of the reward function $\mathcal{R}$. Assuming $v_0(s)$ is a sensible indicator of the reward, we would like that this added source of information strictly benefits MCTS. To make use of this information in MCTS, we add another value to maintain at each node, the *implicit minimax evaluation with respect to player* $\tau(s)$, v_s^τ, and define $v_{s,a}^\tau$ as before. During backpropagation, r_s and n_s are updated in the usual way, and additionally v_s^τ is updated a minimax backup rule based on children values. Also, rather than using $\hat{Q} = \bar{Q}$ for selection in Equation 1, we use

$$\hat{Q}^{IM}(s, a) = (1 - \alpha)\frac{r_{s,a}^\tau}{n_{s,a}} + \alpha v_{s,a}^\tau. \qquad (2)$$

Pseudo-code is presented in Algorithm 1. There are three simple additions to vanilla MCTS, which are located on lines 2, 8, and 13. During selection, $\hat{Q}^{IM}$ from Equation 2 replaces $\bar{Q}$ in Equation 1. During backpropagation, the implicit minimax evaluations v_s^τ are updated based on the children's values. For simplicity, a single $\max$ operator is used here since the evaluations are assumed to be in view of player $\tau(s)$. Depending on how the game is modeled, the implementation may require keeping track of or accounting for signs of rewards, for example a negamax model would

[1] Kalah has been weakly solved for several different starting configurations (Irving, Donkers, and Uiterwijk 2000).

include a sign switch from the returned child on line 8. The function $\alpha(n_s)$ will determine how much weight to attribute to these evaluations. Finally, after a node expansion, on line 13, the implicit minimax value is initialized to its heuristic evaluation $v_s^\tau \leftarrow v_0^\tau(s)$.

```
1  SELECT(s):
2      Let A' be the set of actions a ∈ A(s) maximizing
       Q̂^IM(s,a) + C sqrt(ln n_s / n_{s,a})
3      return a' ~UNIFORM(A')
4
5  UPDATE(s, r):
6      r_s ← r_s + r
7      n_s ← n_s + 1
8      v_s^τ ← max_{a∈A(s)} v_{s,a}^τ
9
10 SIMULATE(s_prev, a_prev, s):
11     if s ∉ G then
12         EXPAND(s)
13         v_s^τ ← v_0^τ(s)
14         r ←PLAYOUT(s)
15         UPDATE(s, r)
16         return r
17     else
18         if s ∈ Z then return R(s_prev, a_prev, s)
19         a ←SELECT(s)
20         s' ← T(s, a)
21         r ←SIMULATE(s, a, s')
22         UPDATE(s, r)
23         return r
24
25 MCTS(s_0):
26     while time left do SIMULATE(−, −, s_0)
27     return argmax_{a∈A(s_0)} n_{s_0,a}
```

Algorithm 1: Pseudo-code of MCTS with implicit minimax backups.

In essence, MCTS with implicit minimax backups acts like a heuristic approximation of MCTS-Solver for the portion of the search tree that has not reached terminal states. However, unlike MCTS-Solver and minimax hybrids, these modifications are based on heuristic evaluations rather than proven wins and losses.

Empirical Evaluation

■ [ML]$_1$:We'll need to modify how this paragraph reads so as to not give away our identities. Results for different alpha values in LOA are shown in Figure 3. A particularly good enhancement in LOA is the progressive bias from move categories (see (Winands, Björnsson, and Saito 2010)). We test how implicit minimax backups perform against the strongest known LOA player (MC-LOA) with and without progressive bias from move categories enabled. MC-LOA also uses MCTS-Solver, (dynamic) early playout terminations, and more.

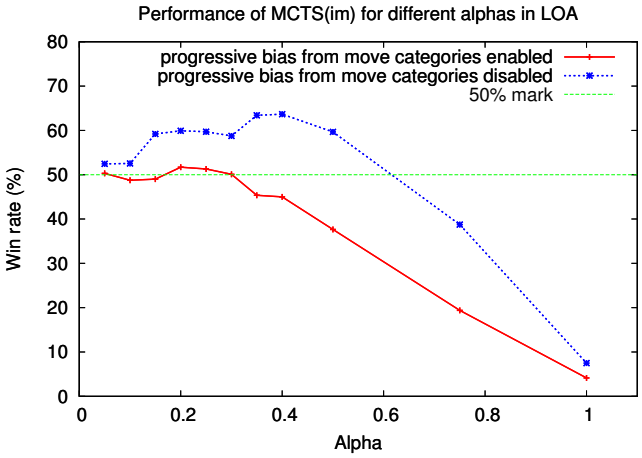

Figure 1: Results in MC-LOA. Each data point represents 1000 games with 1 second of search time.

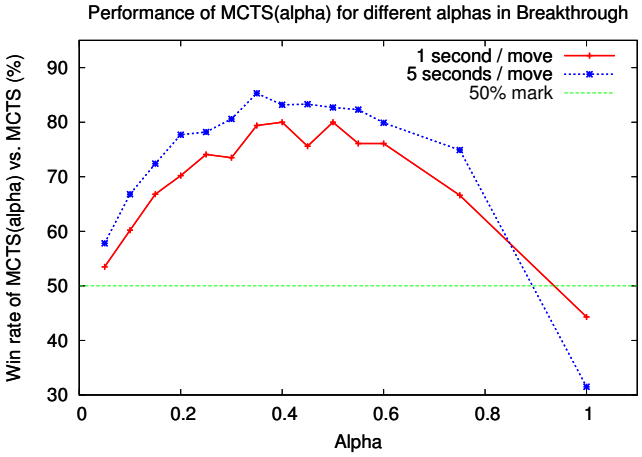

Figure 2: Results in Breakthrough. Each data point represents 1000 games with 1 second of search time.

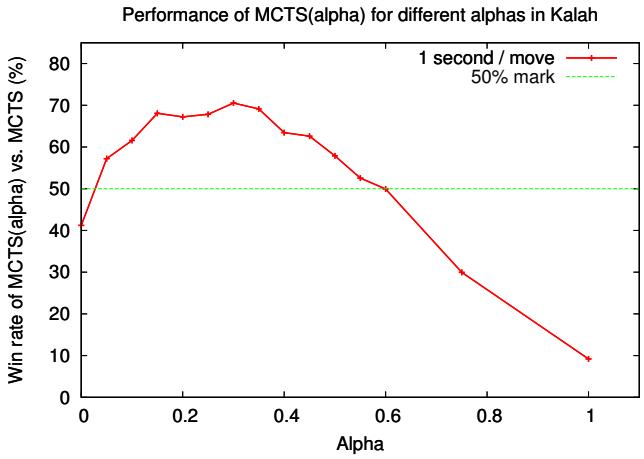

Figure 3: Results in Kalah. Each data point is $\pm$ 3% at the moment (still running), with 1 second of search time. Same evaluation function as (Ramanujan and Selman 2011) normalized to $[0, 1]$ using tanh and an (optimal) pdepth of 4.

Some important things we might want to consider for the experiments and/or message:

- **Early playout terminations**. When a playout is started from state s, descend d plies by choosing actions according to a playout policy to reach state s^d, then return the value of the $v_0^\tau(s^d)$. In Kalah, UCT$_H$ uses early terminations with $d = 0$. In previous work in Amazons and Breakthrough, and some domains with chance, best performance is achieved for $d > 0$.
- **ϵ-greedy playouts**. At state s, the playout policy chooses randomly with probability ϵ and chooses an action $a \in \mathcal{A}(s)$ that maximizes $v_0^\tau(\mathcal{T}(s, a))$.
- **Constant/Progressive bias.** Constant or progressive bias defines $\hat{Q}$ as in Equation 2 except replaces $v_{s,a}^\tau$ with $v_0^\tau(s')$ or with $v_0^\tau(s')/(n_{s,a} + 1)$, respectively.
- **Node priors** (Lorentz paper claims this make large difference in Breakthrough)
- Probably the most similar competitor is UCTMAX$_H$ from (Ramanujan and Selman 2011). UCT$_H$ is standard UCT with early playout terminations ($d = 0$). UCTMAX$_H$ is the same, except maximum backpropagation is used. Note that UCTMAX$_H$ is different from implicit minimax; implicit minimax keeps the average computed by the playouts and minimax values separately, benefiting from both sources of information. I expect implicit minimax to work better than UCTMAX$_H$, but it might be nice to show this (esp. in Kalah).
- The extension to games with chance nodes is straightforward. It might be nice to show it, though. There is a tactical candidate game growing in popularity (Chinese Dark Chess). Probably not going to happen – just not enough time : (
- Implicit max^n in Hearts. If this works well and we want to include the results, we'd change the formulation a bit to talk about the > 2 player case (Sturtevant 2008).
- For Chinese Checkers we should also try Progressive History (Nijssen and Winands 2011; Nijssen 2013) since it was shown to work quite well there. See also (Roschke and Sturtevant 2013).
- Simple regret / observed error? We need a small game that's solveable so we can compute the optimal minimax values. Small Chinese Checkers?
- Seems like, at least from our observations of watching experiments in Breakthrough, implicit minimax could be better at detecting/defending "fortresses" (Guid and Bratko 2012).

Candidate domains at the moment: Breakthrough, Lines of Action, Kalah, and Hearts. In terms of choosing domains, it seems like this work on "somewhat tactical games" (where minimaxing will help).

A comparison to standard minimax seems appropriate. It'd be nice to find a game where MCTS with implicit minimax backups is preferred over all of the others (minimax, MCTS, UCTMAX$_H$).

References

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. 2013. Monte-Carlo tree search and minimax hybrids. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 129–136.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; 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.

Cazenave, T., and Saffidine, A. 2010. Score bounded Monte Carlo Tree Search. In van den Herik, H.; Iida, H.; and Plaat, A., eds., *Proceedings of the 7th International Conference on Computers and Games (CG 2010)*, volume 6515 of *LNCS*. Kanazawa, Japan: Springer. 93–104.

Chaslot, G. M. J.-B.; Winands, M. H. M.; Uiterwijk, J. W. H. M.; van den Herik, H. J.; 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 *Proceedings of the 5th International Conference on Computers and Games (CG'06)*, volume 4630 of *LNCS*, 72–83. Berlin, Heidelberg: Springer-Verlag.

Feldman, Z., and Domshlak, C. 2013. Monte-Carlo planning: Theoretically fast convergence meets practical efficiency. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in uct. In *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*.

Gudmundsson, S., and Björnsson, Y. 2013. Sufficiency-based selection strategy for mcts. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*.

Guid, M., and Bratko, I. 2012. Detecting fortresses in chess. *Elektrotehniški Vestnik* 79(1–2):35–40.

Irving, G.; Donkers, H.; and Uiterwijk, J. 2000. Solving Kalah. *ICGA Journal* 23(3):139–148.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*.

Kloetzer, J. 2010. *Monte-Carlo Techniques: Applications to the Game of Amazons*. Ph.D. Dissertation, School of Information Science, Japan Institute of Science and Technology, Ishikawa, Japan.

Kocsis, L., and Szepesvári, C. 2006. Bandit-based Monte Carlo planning. In *15th European Conference on Machine Learning*, volume 4212 of *LNCS*, 282–293.

Lorentz, R., and Horey, T. 2013. Programming breakthrough. In *Proceedings of the 8th International Conference on Computers and Games (CG)*.

Lorentz, R. 2008. Amazons discover Monte-Carlo. In *Proceedings of the 6th International Conference on Computers and Games (CG)*, volume 5131 of *LNCS*, 13–24.

Nijssen, J., and Winands, M. 2011. Enhancements for multi-player monte-carlo tree search. In *Proceedings of the 7th International Conference on Computers and Games (CG 2010)*, volume 6515 of *LNCS*, 238–249.

Nijssen, J. 2013. *Monte-Carlo Tree Search for Multi-player Games*. Ph.D. Dissertation, Maastricht University, Maastricht, The Netherlands.

Ramanujan, R., and Selman, B. 2011. Trade-offs in sampling-based adversarial planning. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS)*, 202–209.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2010a. On adversarial search spaces and sampling-based planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 242–245.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2010b. Understanding sampling style adversarial search methods. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, 474–483.

Roschke, M., and Sturtevant, N. 2013. UCT enhancements in chinese checkers using an endgame database. In *Proceedings of the IJCAI Workshop on Computer Games*.

Sturtevant, N. R. 2008. An analysis of UCT in multi-player games. *International Computer Games Journal* 31(4):195–208.

Winands, M.; Björnsson, Y.; and Saito, J.-T. 2008. Monte-Carlo tree search solver. In *Computers and Games (CG 2008)*, volume 5131 of *LNCS*, 25–36. Springer, Berlin Heidelberg.

Winands, 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.