Early Playout Termination in MCTS

Richard Lorentz(✉)

Department of Computer Science, California State University,
Northridge, CA 91330-8281, USA
lorentz@csun.edu

Abstract. Many researchers view mini-max and MCTS-based searches as competing and incompatible approaches. For example, it is generally agreed that chess and checkers require a mini-max approach while Go and Havannah require MCTS. However, a hybrid technique is possible that has features of both mini-max and MCTS. It works by stopping the random MCTS playouts early and using an evaluation function to determine the winner of the playout. We call this algorithm MCTS-EPT (MCTS with early playout termination) and study it using MCTS-EPT programs we have written for Amazons, Havannah, and Breakthrough.

1 Introduction

Monte-Carlo Tree Search (MCTS) differs from "classical" mini-max game-tree search in two major ways. First, no evaluation function is needed in MCTS. Instead, the random playouts in the MCTS act as a kind of sampling of the possible outcomes from various board positions, which in turn can be used to rate (evaluate) these different positions. Also, MCTS builds the search tree so that more promising lines of play are more thoroughly explored in the tree than less promising ones. As a result, we have learned that MCTS can drastically outperform mini-max based search engines in games where evaluation functions are difficult to obtain, and especially in games with large branching factors [1,2].

A hybrid approach to MCTS is possible, however. Instead of allowing the random playout to run until the end of the game we can instead terminate the playout early and then apply an evaluation function to the position to determine which side is likely to win. We call this approach MCTS with early playout termination (MCTS-EPT, or simply EPT). A number of successful programs have been written using EPT. See, for example, [5–7,9].

We have written EPT programs that play the games of Amazons, Breakthrough, and Havannah and we will refer to them as Amabot, Breakbot, and Havbot. Amabot was originally written using mini-max techniques and, playing under the name Invader, was one of the top Amazons programs at the Computer Olympiads from 2001–2005 [11], but never finished in first place. After converting from mini-max to EPT, Amabot has won each of the last five Computer Olympiads it has entered.

Breakbot is a more recent program. In contrast to Amabot, it was originally written as an MCTS program and then was migrated over to the MCTS-EPT approach. The pure MCTS version played a fairly average game, whereas

A. Plaat et al. (Eds.): ACG 2015, LNCS 9525, pp. 12–19, 2015.
DOI: 10.1007/978-3-319-27992-3_2

the EPT incarnation is very strong, being one of the top 3 players on the Little Golem game-playing Web site [10], where it plays under the name Wanderer.

Havbot was also originally pure MCTS [8] that has recently been converted to EPT. Havbot was also a moderately strong MCTS program, but is only slightly stronger using EPT. Like Breakbot, Havbot also plays under the name Wanderer. It has played in a number of Computer Olympiads and also plays on the Little Golem Web site.

Though creating an EPT program is straightforward, we will explain in detail (1) the requirements and difficulties of producing a strong EPT program from the perspective of our success with Amabot and Breakbot and (2) our difficulties with Havbot.

2 History

We begin with a brief history of our research into MCTS-EPT. By 2007 Amabot had performed well in a number of Computer Olympiads, but had never managed to win one. Johan de Koning's program, 8qp, was the five-time winner of the event and we could not seem to reach its level of play. Also in 2007 the MCTS revolution was in full swing, so we wondered what MCTS could offer us beyond what our mini-max program was providing. The mini-max program was using a sophisticated evaluation function so we had little hope that MCTS would be able to achieve the same level of play without using all the knowledge that was available to the evaluation function. Unknown to us at the time, Julien Kloetzer was doing the same research under the guidance of Hiroyuki Iida [5]. As it turns out we independently came to the same conclusion, namely, random playouts were insufficient. We needed to use the large amount of knowledge that was coded in the evaluation function. We also both discovered that the evaluation function can best be used as EPT rather than, say, to help guide the random playouts. In the case of Amabot, we were ultimately able to achieve a win rate of 80 % using EPT over the mini-max based program. We then went on to win the next five Computer Olympiads using EPT. We believe Kloetzer's program had the potential for similar results, but he did not have the luxury of a pre-existing, strong evaluation function, leaving him at a disadvantage.

In 2009, motivated in part by the "Havannah Challenge" [12], a number of projects began to develop an Havannah playing program, including our Havbot project. Havannah seemed to be a perfect candidate for MCTS because of its high move branching factor, its very large state space, and the fact that a good evaluation function seems very hard to find. With but one exception, all known Havannah playing programs use MCTS. The one exception is a mini-max based program written by the talented game programmer Johan de Koning. The one time he entered it in the Computer Olympiad it lost every game it played against the other two programs, providing strong evidence that MCTS is the approach of choice.

However, progress in Havannah programming has not progressed as we might have hoped. Though the top programs do play at a reasonable level, about the

level of somebody who has played the game for 6 months or a year, they still play with a very unnatural style, and often win their games by virtue of tactical shots missed by the human opponent. Our feeling is that Havannah programs cannot be expected to play at an elite level until they learn to play a more natural, human-like game. Towards this end, we have retooled HAVBOT to use EPT. Evidence is still inconclusive, and more details will be provided below, but we feel that its current style of play is more natural and has the potential to improve to noticeably higher levels of play. It currently beats the mini-max version of HAVBOT about 60 % of the time.

BREAKBOT, like HAVBOT, was written initially using MCTS but we fully expected to transition to EPT. As was the case with the MCTS version of AMABOT, without an evaluation function it's level of play languished in the low intermediate range. With the introduction of EPT it's level rose considerably and quickly, where after quite a bit of work, it is, at the time of this writing, the third highest rated player on Little Golem, and the second highest rated active player. The evidence that BREAKBOT with EPT outperforms MCTS is convincing. What is not quite so obvious is if it is better than mini-max based programs. The evidence we have to support this viewpoint is that there are two other programs playing on Little Golem, both of them are mini-max based, and BREAKBOT has won the majority of the encounters, though against the stronger of the two, LUFFYBOT, all of the games have been very close. We may conclude that EPT stands up well against mini-max and even though many of the games have been close, BREAKBOT ultimately outperforms the mini-max based ones.

3 Details

We now consider implementation details for MCTS-EPT. Our conclusions concerning these details are drawn from many years of experimenting (beginning in 2007) with the three different programs across two different playing situations (real time as played in the Computer Olympiads and very slow as played on the turn-based Web site Little Golem). As such some features seem to span most EPT situations while others apply to more specific settings.

3.1 Blending Mini-Max and EPT

It would seem natural that certain phases of a game would lend themselves to mini-max analysis while others to EPT. In fact, for many years AMABOT was written so that EPT was used throughout the majority of the game, and then switched over to mini-max near the end. Evidence seemed to indicate that the breadth-first nature of mini-max was superior near the end of the game because it would be less likely to miss a tactical shot that EPT (and MCTS in general) might miss because EPT had gotten stuck on a "good" line of play and did not have the time to find a better move. This, of course, is a general problem with MCTS, and can be fatal near the end game when a missed winning line or a failed proper defence can quickly and permanently turn a game around.

We now believe this is not true for two reasons. First, it is easy to incorporate solvers into MCTS, and therefore EPT, by propagating wins and losses up the MCTS tree in the usual and/or fashion. The advantage of being able to prove nodes outweighs anything lost by the tendency of MCTS to get stuck on a suboptimal line of play. Further, the solver can accelerate the exit from a bad line of play because winning and losing positions propagate immediately up the tree rather than requiring many simulations to reach the same conclusion.

Secondly, it is simply the case that the strengths of MCTS extend well to all aspects of the game. A good example is seen when dealing with defective territory in Amazons, a problem that turns up near the end of the game. This has always been a bit of a sticky issue with programs because the overhead necessary to deal with defects, typically done either by using patterns or other computationally expensive procedures in the evaluation function, does not seem to be worth the cost. In the case of EPT, however, defective territory is easily detected. In the presence of defective territory the MCTS tree accurately assesses the defect because the random playouts show that the territory cannot be properly filled. As a result, Amabot has not used any mini-max and has been exclusively an EPT program for the last 5 years.

3.2 Progressive Widening

It is usually necessary to assist EPT by focusing on promising moves above and beyond what the MCTS rules suggest. In nodes with very few visits it can be difficult to distinguish among the many children. Two apparently different techniques have been developed that accomplish essentially the same thing. Progressive widening restricts access to nodes with low evaluation values and low visit counts and gradually phases them in as the parent node gets more visits [4]. Alternatively, node initialization (sometimes referred to as *priors* [3]) initializes the win and visit counts of nodes at the time of their creation with win values that reflect the strength of the node, again determined by the evaluation function.

In all three of our programs we have seen that it is necessary to use one of these techniques. In the case of Amabot, progressive widening is used. In fact, since Amabot possesses such a mature and accurate evaluation function and since Amazons allows so many legal moves, especially in the early parts of the game, we push progressive widening a bit further and do some forward pruning. Amazons positions can have more than 2000 legal moves. When building the EPT tree, we evaluate all possible children of a node and only put the top 750 in the tree and then from these we proceed with the usual progressive widening.

With Havbot and Breakbot we use the evaluation function to initialize win values in new nodes. Considerable tuning is necessary to find good initial values because, as is so common with MCTS related algorithms, we must find the proper balance so that the tree grows without inappropriate bias. In all three cases the winning advantage when using these techniques is significant, being over 75 %.

3.3 When to Terminate

Certainly a fundamental question is: when should the playout be terminated. The longer we delay the termination the more the behavior is like pure MCTS while sooner terminations put added emphasis on the evaluation function. We were surprised to find that in all three of our programs the optimal termination point was quite early and nearly at the same point in all three cases, namely, after around five moves. When the evaluation function is known to be quite reliable, as is the case with Amabot, and to a lesser extent Breakbot, it is not too surprising that an earlier termination should be preferred since additional random playouts before evaluating will only dilute the effect of the evaluation. However, in the case of Havbot, where the evaluation is still very much a work in progress and can be quite undependable, the optimal termination point is still about the same and later termination points degrade its behavior at a rate quite similar to what is observed in Amabot. In essence, it appears that even a weak evaluation function can compare favorably with a long random playout.

But what about termination points that are shorter than the optimal value? Since all three programs show similar results, let us focus on Breakbot. Though it stands to reason that shorter termination points might outperform longer ones when these termination points are reasonably large, it is not immediately obvious why the optimal value is not 1 or 0.

Consider Fig. 1 where we show the results of Breakbot playing as white against 4 other versions that were modified to have different termination points. Terminating after four random moves is optimal. Delaying the termination point beyond the optimal quickly degrades the performance and it is a bit surprising just how quickly it degrades.

Termination	Winning result
1	33%
4	43%
6	27%
12	10%

Fig. 1. Playout termination points in Breakbot.

But of particular interest is the first row that clearly shows that only 1 random move is not as good as 4. The values for 2 and 3 random moves degraded roughly uniformly. Why is it the case that a few random moves actually improve performance?

To help us understand this phenomenon we ran hundreds of games where at every position a move was generated by two versions of Breakbot, with termination points of 4 and 1. We found that on average the different versions disagreed on the best move about 12 times per game, where the average length of a game is 55 moves. It is important to point out, however, that a similar test performed on two copies of the same version of Breakbot (with termination

point 4) still disagreed an average of 7 times per game, simply because of the random nature of EPT. In general, this suggests that about 5 times a game, or roughly 10 % of the time, the termination-1 version selects a move that the termination-4 version probably would not, and presumably more often than not this is a weaker move. Visual examination of these moves, however, generally does not reveal major blunders. Rather, when differences are detectable at all, they are small and subtle. Of course, five minor mistakes a game is certainly sufficient to cause the observed drop in winning percentage.

But it is difficult to provide a definitive explanation as to exactly what causes these roughly five aberrations per game. Why would fewer random moves in a playout hinder performance? Observational evidence suggests it boils down to a trade off between the advantages of a deep evaluation and disadvantages of losing information from the randomness of a playout. In general, an evaluation near the end of the game is more reliable than one earlier on but after too many random moves a position may lose the essence of the starting position. We search for a happy medium where a few random moves take us closer to the end of the game, without having the random moves degrade the information too much. For all three games we are studying this cutoff seems to be around 4 or 5.

Related to this, we should mention the concept of improving the random playouts. This, of course, is an important technique for an MCTS program and is certainly one of the major reasons MCTS programs are so successful. In the case of EPT it appears to be of little or no help. On the one hand it is not too surprising given that the random playouts are only 4 or 5 moves deep, but on the other hand given how important it is for MCTS, we thought we could get some improvement in our programs by improving the playouts. Despite considerable effort on all three programs, we have never been able to demonstrate any advantage by introducing smart playouts.

Finally, we point out that in a game like Amazons the evaluation function can vary wildly depending on whose move it is. This is sometimes referred to as the parity effect. The evaluation function tends to heavily favor the player to move. To help stabilize EPT we can modify the random playouts so that they always terminate with the same player to move. In the case of Amazons we terminate the playout after either 5 or 4 moves, accordingly. This produces some small advantage in the case of Amabot, but in the cases of Havabot and Breakbot, where the evaluations do not display such a strong parity effect, adjusting the playouts this way does not seem to have any effect.

3.4 Miscellaneous

In this section we summarize a few other observations and techniques that we consider important.

It is generally the case that MCTS programs prefer to record wins and losses at the end of their playouts, rather than trying to somehow keep track of the margin of victory. We find the same is true with EPT. Rather than somehow use the value of the evaluation function, we have always obtained the best results

by simply treating the evaluation as a boolean function, reporting either a win or a loss.

In Sect. 3.1 mention was made of the fact that EPT, as well as MCTS, can get stuck on a bad move simply because there is not enough time for the refutation of this weak move to achieve sufficient visits. Even though this seems like a problem mainly for real-time play, we find the problem carries over even for turn-based play where we sometimes allow as much as 15 min of thinking time. This problem can occur anywhere in the tree, but we have found that if we deal with it specifically at the root, we can get better results. What we do is we increase the exploitation constant only at the root so that exploration is encouraged, allowing more moves to be considered. Specifically, since all of our EPT programs are UCT based, we simply increase the UCT constant, typically by a factor of around 6. It does not make sense to uniformly change the UCT constant by this amount because the constant has already been optimized. But changing it only at the root has the very real effect of possibly allowing a move to be considered that might otherwise have been ignored. We have not been able to prove an advantage quantitatively, but we have seen quite a few cases of games on Little Golem where moves were found that were clearly better than those found without the adjustment while the reverse has yet to be observed. This technique, as well as the next, would probably apply to MCTS programs as well.

In the case of Breakbot we had to deal with the situation that early captures are almost always bad. We found no satisfactory way to deal with this in the evaluation because a capture by the first player usually requires a recapture by its opponent, so it balances out in the evaluation. Attempts to recognize the bad exchange after the fact in the evaluation had too many undesirable side effects. Our solution was to deal with this in the move selection process of the MCTS part of the search. Whenever a move was being selected for traversal or expansion in the MCTS tree, if it was a capture we hand tuned a penalty value for its winning percentage. This penalty is a function of (1) the stage of the game (early, middle, or late) and (2) the depth in the tree in which the move is being considered. This proved to be a successful way to deal with a problem that we were unable to deal with in the evaluation.

4 Conclusions

We have had considerable success with MCTS-EPT in games with a variety of features. Amazons is a game with a fairly large branching factor but it does allow for very precise and sophisticated evaluation functions. Still, EPT Amabot outperforms all mini-max based programs. Not only does Amabot do well in real-time play but it has played a number of games against some of the strongest players on turn based Little Golem and has never lost.

Breakthrough has a smaller branching factor but evaluation functions tend to be rather primitive. Not many programs exist that play Breakthrough, but of the two we are aware of (both play on the Little Golem site), we know that both are mini-max based, and both have losing records to Breakbot.

Havannah is a game, like Go, that has no strong mini-max based programs and not until the MCTS revolution did any reasonable programs exist. The three strongest Havannah playing programs all play on Little Golem and, though maybe slightly weaker than the other two, the MCTS version of HAVBOT plays a very similar game to the other two. Even though the evaluation function for HAVBOT is still quite primitive the program is making some promising looking moves and is outperforming its MCTS counterpart. As the evaluation continues to improve we feel there is great potential for this program.

As a side note, AMABOT and BREAKBOT are now so strong that progress comes very slowly. When deciding if a modification is an improvement sometimes simply running test games is not sufficient. If the tests are inconclusive, we must be ready to allow human intervention. How do the moves look to us? Do they seem to improve on the older version moves? How often do the moves look worse? We must be willing to make decisions based on answers to these kinds of questions, especially in the setting of turn based play, where results come at an agonizingly slow pace.

References

1. Browne, C., Powley, D., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, C.: A survey of monte carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–49 (2012)
2. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: 5th International Conference on Computers and Games, CG 2006, Turin, Italy, pp. 72–84 (2006)
3. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Ghahramani, Z. (ed.) Proceedings of the 24th International Conference on Machine Learning (ICML 2007), pp. 273–280. ACM, New York (2007)
4. Chaslot, G.M.J.-B., Winands, M.H.M., van den Herik, H.J., Uiterwijk, J.W.H.M., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Math. Nat. Comput. **4**(3), 343–357 (2008)
5. Kloetzer, J., Iida, H., Bouzy, B.: The monte-carlo approach in amazons. In: Computer Games Workshop, Amsterdam, The Netherlands, pp. 113–124 (2007)
6. Lorentz, R.J.: Amazons discover monte-carlo. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) CG 2008. LNCS, vol. 5131, pp. 13–24. Springer, Heidelberg (2008)
7. Lorentz, R., Horey, T.: Programming breakthrough. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) CG 2013. LNCS, vol. 8427, pp. 49–59. Springer, Heidelberg (2013)
8. Lorentz, R.: Experiments with monte-carlo tree search in the game of havannah. ICGA J. **34**(3), 140–150 (2011)
9. Winands, M.H.M., Björnsson, Y.: Evaluation function based monte-carlo LOA. In: van den Herik, H.J., Spronck, P. (eds.) ACG 2009. LNCS, vol. 6048, pp. 33–44. Springer, Heidelberg (2010)
10. http://www.littlegolem.net/jsp/index.jsp
11. http://www.grappa.univ-lille3.fr/icga/program.php?id=249
12. Havannah#The Havannah Challenge. https://chessprogramming.wikispaces.com/