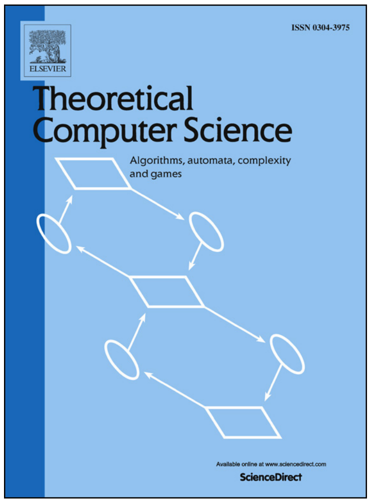
Accepted Manuscript



Using Evaluation Functions in Monte-Carlo Tree Search

Richard Lorentz

PII: S0304-3975(16)30271-7
DOI: http://dx.doi.org/10.1016/j.tcs.2016.06.026
Reference: TCS 10829

To appear in: *Theoretical Computer Science*

Received date: 22 January 2016
Revised date: 6 May 2016
Accepted date: 20 June 2016

Please cite this article in press as: R. Lorentz, Using Evaluation Functions in Monte-Carlo Tree Search, *Theoret. Comput. Sci.* (2016), http://dx.doi.org/10.1016/j.tcs.2016.06.026

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Using Evaluation Functions in Monte-Carlo Tree Search

Richard Lorentz

Department of Computer Science, California State University, Northridge, CA 91330-8281 USA

Abstract

For decades the game playing algorithms of choice have been based on the mini-max algorithm and have had considerable success in many games, e.g., chess and checkers. Recently a new algorithmic paradigm called Monte-Carlo Tree Search (MCTS) has been discovered and has proven to perform well in games where mini-max has failed, most notably in the game of Go. Many view mini-max and MCTS based searches as competing and incompatible approaches. However, a hybrid technique using features of both mini-max and MCTS is possible. We call this algorithm MCTS-EPT (MCTS with early playout termination) and study it from the context of three different games: Amazons, Breakthrough, and Havannah. This paper expands and elaborates on work presented in [1] and [2].

Keywords: MCTS MCTS-EPT Breakthrough Havannah Amazons

1. Introduction

From the first days of game programming mini-max enhanced with alpha-beta pruning has been the algorithm of choice. See, for example, [3, 4]. All of this changed in 2006 when the first experiments with Monte-Carlo Tree Search (MCTS) began to appear, see [5, 6] (for a good history and survey, see [7]), and research on MCTS has exploded in the past decade.

MCTS differs from classical mini-max game tree search in two major ways. Firstly, no evaluation function is needed in MCTS. Instead, random game play-outs, sometimes called simulations or rollouts, 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. Random playouts are essentially random games played from a given position until the end of the game with the goal of trying to judge who has the advantage in that position. The random games are often skewed so that more reasonable moves are more likely to be made [6].

Email address: lorentz@csun.edu (Richard Lorentz)

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Secondly, MCTS builds the search tree so that more promising lines of play are more thoroughly explored in the tree than less promising ones. This differs from mini-max algorithms that examine different lines of play in a much more uniform fashion.

We have learned that MCTS can dramatically outperform mini-max based search engines in games where evaluation functions are difficult to obtain, and especially in games with large branching factors [7, 8].

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 more likely to win. Whereas evaluation functions are typically not used at all in MCTS, they are fundamental in mini-max searches in that they provide the values that are propagated up the search tree in order to determine the best move.

We call this hybrid approach MCTS with early playout termination (MCTS-EPT, or simply EPT). A number of successful programs have been written using EPT. See, for example, [9, 1, 10, 11].

We have performed a systematic study of MCTS-EPT using the games of Amazons, Breakthrough, and Havannah. We have written EPT programs for each of these games and have studied the properties of EPT using these programs. We present our results as follows. In Section 2 we give the rules of the three games and briefly describe some of the their major features.

In Section 3 we discuss the evolution of our three programs. In Section 4 we explain in detail exactly what the MCTS-EPT algorithm is, how it is implemented for each of the programs, what the similarities are between them, and how they differ with the goal of showing the general power and limitations of EPT. Section 5 summarizes the results and suggests a number of different avenues for future work.

2. Amazons, Breakthrough, and Havannah

Amazons, Breakthrough, and Havannah are all two-person, perfect information, strategy games with no random elements. Roughly speaking, Amazons can be classified as a territory oriented game, Breakthrough is a race game, and Havannah is a connection game. Despite the conspicuous differences in these three games we will see that MCTS-EPT can deal quite successfully with all three. The rules and basic information about the games follow.

2.1. Amazons

Amazons is played on a 10 × 10 board. Each player is given 4 amazons positioned on the board as seen on the left of Figure 1[1]. A player's move comprises two parts: (1) First an amazon moves like a chess queen without

[1]All figures are taken from [12].

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

passing over any other amazons or marked squares; (2) After the amazon has landed it throws a marker to a square, essentially blocking that square, where again the possible squares the marker may be placed are those that can be reached unimpeded in the directions a chess queen can move. The last player to make a legal move is the winner and White makes the first move.

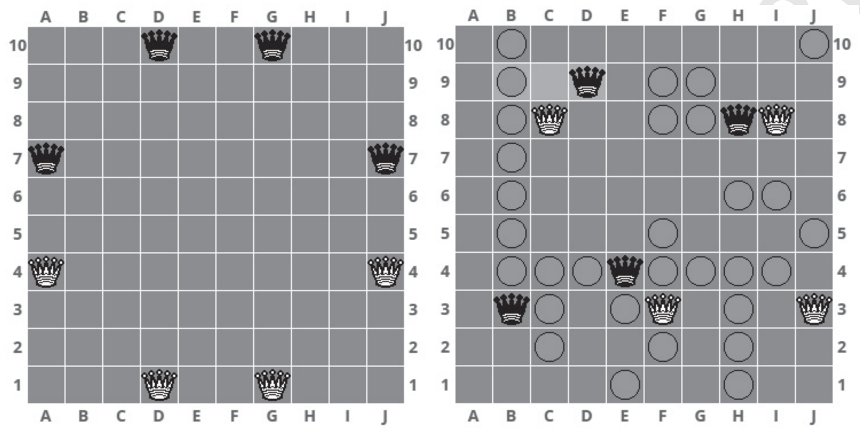


Figure 1: Sample Amazons positions.

The right of Figure 1 shows a position after 30 moves. Black has just moved an amazon from B9 to D9 and has thrown the marker back to B9. Since the winner is the last player to move, the goal is try to give your amazons the most freedom to move while trying to restrict your opponent's amazons. In this same position we see that Black's amazon on B3 will probably move about 12 times (each move throws a marker restricting subsequent moves). This is the territorial aspect of the game. The amazons on H8 and I8 are in a battle to try to restrict each other while still providing freedom for themselves. Black ultimately won the game by making the last move while having 6 additional moves to spare.

2.2. Breakthrough

Breakthrough is played on an 8×8 board. Each player begins with 16 pieces as shown on the left of Figure 2. White pieces move one square at a time either diagonally or vertically to unoccupied squares and towards row 8. White may capture a black piece if the piece is located where a diagonal legal move could be made – as a chess pawn might capture. A white piece cannot move forward if that square is occupied by either a white or a black piece. Black moves similarly in the other direction. The first player to reach the last row or capture all of the enemy pieces is the winner. White plays first.

The right of Figure 2 shows a game after 60 moves. A few strategic points of note: (1) White has 3 pieces on row 4 indicating a possible edge in the race to the end; (2) Row 1 is still well protected by White because a single black

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

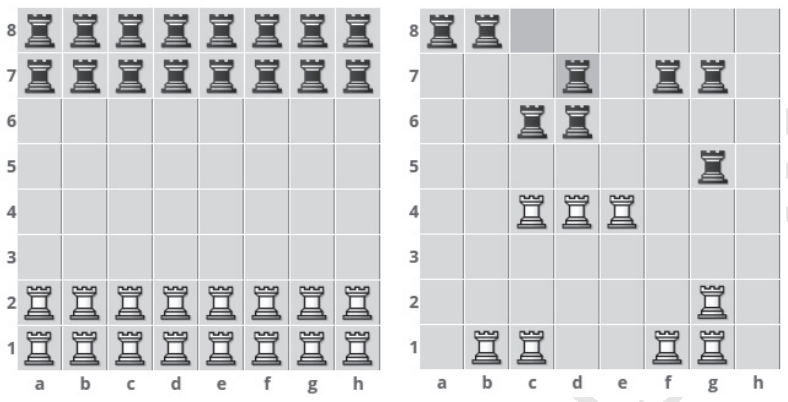


Figure 2: Sample Breakthrough positions.

piece cannot make it to row 1; (3) Black has exposed row 8 more then White has exposed row 1. The question is does White have enough power with those three advanced pieces to break through and win before Black can trade them off and use his remaining advanced pieces to win? It is a race to see who can reach their goal first. White ended up winning this game in another 10 moves.

2.3. Havannah

Havannah is played on an hexagonal board with anywhere from four to ten hexagons per side. Figure 3 shows two positions from a size six game.

Players alternate placing pieces on hexagons where Black plays first. Havannah is a connection type game where each side is trying to be the first to obtain either a ring, a bridge, or a fork. A ring is a continuous collection of hexagons all occupied by the same color that form a ring. For example, in the second board in Figure 3 if the last piece in the second column were white instead of black, White would have a ring. A bridge is a continuous collection of hexagons of the same color that connect two corners. For example, in the same board if the last piece in the last column were white, then White would have a bridge. A fork is a collection of pieces that connect three different edges, not including corners. Again, looking at the same position we can see that White has an unstoppable fork in two moves. He can place a piece on the first column in the fourth or fifth position to connect to that edge, then connect those pieces to the ones on the top right with one more move giving a connection to the left, top-left, and right edges. White will win this game in two moves.

The position on the left shows the same Havannah game after only ten moves. One can note that Black was trying to connect two corners at the bottom left, but White occupied one of the corners, preventing this. White, perhaps, has a similar plan with the top left corners but we can see in the right diagram that Black prevented this and White, instead, worked to connect three edges.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

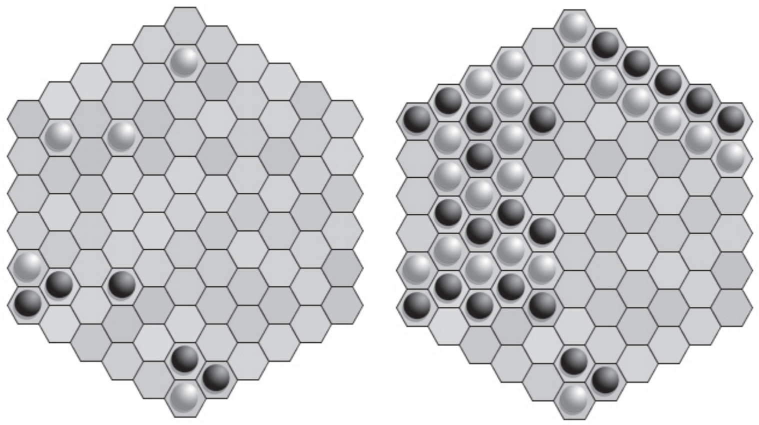


Figure 3: Sample Havannah positions.

3. Three EPT programs

We have written programs for these three games to study and better understand the properties of the MCTS-EPT algorithm. Our programs for Amazons, Breakthrough, and Havannah are called Amabot, Breakbot, and Havbot respectively. We summarize the history, properties, and structure of the three programs.

3.1. Amabot

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 [13], but never finished in first place. 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 [9]. 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 both discovered that the evaluation function can best be used for 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 six Computer Olympiads using EPT. We believe Kloetzer's program had the potential for similar results, but he did

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

not have the luxury of a pre-existing, strong evaluation function, leaving him at a disadvantage.

Amabot also taught us that simply using a powerful evaluation function was not sufficient with EPT. A faster evaluation evaluation function was needed for EPT. With any game playing algorithm there is always a tradeoff between the time spent evaluating positions and the number of nodes examined in the game tree. In this case, the mini-max algorithm with the very sophisticated but slow evaluation outperforms a version with a faster but less accurate evaluation function with about a 62% winning advantage. On the other hand, the EPT algorithm performed better with the faster evaluation with an approximately 60% winning advantage. In some ways these different results are not too surprising since the value of the evaluation is crucial in determining the value of a position in mini-max, while in the case of EPT it is really only the sign of the evaluation that is critical. Of course in general it is not a simple matter to determine what features of an evaluation can be abbreviated to save time. In the case of Amabot the territory calculations were greatly simplified.

3.2. Havbot

In 2009, motivated in part by the "Havannah Challenge" [14], a number of projects began to develop Havannah playing programs, 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 hard to find. It is difficult for an evaluation function to judge how easily a connection and which connection can be made from a given position. With but one exception, all known Havannah playing programs use MCTS. The one exception is a mini-max based program written by 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 proper choice.

The question was can we use EPT to improve Havbot? We developed a very crude evaluation function that attempts to estimate how many moves it will require a player to make a connection. It is obviously very difficult to determine this with a static evaluation function, but even with this imprecise evaluation function we are still able to report some positive results. The EPT version outperforms the pure MCTS version (by a small margin) and this strongly suggests an improvement in the evaluation should produce a corresponding improvement in Havbot.

The state of Havannah programming is still in its early stages. 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. Retooling Havbot to use EPT is moving us in the right direction. 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

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

noticeably higher levels of play. It currently beats the mini-max version of Havbot about 60% of the time.

3.3. Breakbot

Breakbot was originally written as an MCTS program and then was migrated over to the MCST-EPT approach. The pure MCTS version played a fairly average game, whereas the EPT incarnation is very strong, being one of the top 3 players on the Little Golem game playing Web site [12], where it plays under the name Wanderer.

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 its level of play languished in the low intermediate range. With the introduction of EPT its level rose considerably and quickly, and after considerable effort on the evaluation function and some of the techniques described in Section 4, 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 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.

4. Details

We now consider implementation details for MCTS-EPT. Our observations and conclusions concerning these details are drawn not only from the experiments described but from many years of competing the three different programs in 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 [12]. Some features seem to span most EPT situations while others apply to more specific settings.

4.1. 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 of the game. Anecdotal evidence indicated 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 promising line of play and did not have the needed time to find a better move. This, of course,

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

is a general problem with MCTS, and can be fatal near the end game when a missed winning line or a failed proper defense 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 a standard 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 [15]. 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 no longer uses any mini-max and has been exclusively an EPT program for the last 6 years.

4.2. 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 [16]. Alternatively, node initialization (sometimes referred to as *priors* [17]) 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 in excess of 2,000 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 progressive widening. The actual cutoff values used in progressive widening are rather ad hoc, having been determined experimentally. Specifically, if *nodect* is the number of times a node has been visited then the following code fragment shows how *cutoff*, the number of children of the node that are actually considered, is calculated.

ACCEPTED MANUSCRIPT

```
if (nodect <= 500)
    node_cutoff = 5;
else if (nodect <= 2500)
    node_cutoff = 25;
else if (nodect <= 5000)
    node_cutoff = nodect / 100;
else
    node_cutoff = 25 + nodect / 200;
```

Notice that this implies that a node needs to be visited more than 150,000 times before the hard cutoff value of 750 becomes a factor. Amabot wins approximately 60% of its games against a version that does not use 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 these two cases the winning advantage when using node initialization is significant, being over 75%.

4.3. 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 programs the optimal termination point was quite early and nearly at the same point, namely, after around six 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.

There is the related question as to why fewer than the optimal number of playouts also under-performs. To help us better understand where the cutoff should be and why it is where it is we focused our attention on Havbot and 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 Figure 4. where we show the results of a white Breakbot player with an EPT termination point of 4 playing against Breakbot versions playing as black that were modified to have different EPT 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

ACCEPTED MANUSCRIPT

just how quickly it degrades. Incidentally, note that the second row of the figure shows that the best version of Breakbot wins 57% of the games when playing as white.

Termination moves	Winning result against a basic version	Number of different moves chosen
1	33%	12
4	43%	7
6	27%	11
12	10%	17

Figure 4: Playout Termination Points in Breakbot.

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 games where moves were generated by two versions of Breakbot, the basic version and a version with the corresponding playout termination. The average number of differences found per game is shown in the third column of Figure 4. where the average number of moves per game is 55. Notice that the second row tells us that simply by the nature of MCTS the move selected is different 7 out of 55 times, or about 13% of the time. However this number is greater in all the other rows.

For example, the first row of the table suggests that about 5 times a game, or roughly 9% 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. The other rows show similar results with the differences increasing as the win rates decrease. Visual examination of the differing 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.

A similar test was performed with Havbot and the results are shown in Figure 5.

In this case the optimal termination point is seen to be around 8. Though slightly larger than it was for Breakbot, the average length of our Havannah games was 65, we still find it surprisingly small. Also, we again see the number of different moves chosen to increase roughly as the win rates decrease. Also of note is the first row of this table which again shows a large penalty both in terms of win rate and different moves chosen for playing only one random move. It is clear that it is important to have some random moves in the playouts, but not too many.

It is difficult to provide a definitive explanation as to exactly what causes these different, and usually weaker, moves during a game. Why would fewer random moves in a playout hinder performance? Observational evidence sug-

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Termination moves	Winning result against a basic version	Number of different moves chosen
1	39%	9
4	48%	6
8	58%	5
12	51%	6
16	56%	7
20	54%	8
28	45%	9
36	37%	10

Figure 5: Playout Termination Points in Havbot.

gests 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 but after too many random moves a position may lose the essence of the original 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 Breakthrough this is approximately 4 moves, for Havannah 8 moves, and it turns out for Amazons it is approximately 6 moves.

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. In one sense it is not too surprising given that the random playouts are only 4 to 8 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 at best been able to demonstrate a minimal advantage by introducing smart playouts. Certainly nothing at the level as seen in the game of Go.

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 refereed 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 6 moves, accordingly. This produces a noticeable advantage in the case of Amabot, a winning rate improvement of about 5%, 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.

4.4. Miscellaneous

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

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

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

In Section 4.1 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 get sufficient visits. Even though this seems like a problem mainly for real time play, we find the problem carries over to turn-based play where we sometimes allow as much as 15 minutes 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 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 for that program. 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 the stage of the game (early, middle, or late) and the depth in the tree 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.

5. 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.

ACCEPTED MANUSCRIPT

Breakthrough has a smaller branching factor but evaluation functions tend to be rather primitive. Not many programs exist that play Breakthrough, but of the three others we are aware of, they are all mini-max based and 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 strength of EPT version is very close to the others. Even though the evaluation function for Havbot is still quite primitive the program is making 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 well.

References

[1] R. Lorentz, Amazons discover monte-carlo, in: H. van den Herik, X. Xu, Z. Ma, M. Winands (Eds.), Computers and Games, Vol. 5131 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 13–24.

[2] R. Lorentz, Early playout termination in mcts, in: A. Plaat, H. van den Herik, W. Kosters (Eds.), Advances in Computer Games, Theoretical Computer Science and General Issues, Springer International Publishing, 2015, pp. 12–19.

[3] D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning., Artif. Intell. 6 (4) (1975) 293–326.

[4] J. Schaeffer, A. Plaat, New advances in alpha-beta searching., in: R. E. Beck, M. L. Soffa (Eds.), ACM Conference on Computer Science, ACM, 1996, pp. 124–130.

[5] L. Kocsis, C. Szepesvári, Bandit based monte-carlo planning, in: In: ECML-06. Number 4212 in LNCS, Springer, 2006, pp. 282–293.

[6] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, B. Bouzy, Progressive strategies for monte-carlo tree search, New Mathematics and Natural Computation 4 (2008) 343–357. doi:10.1142/S1793005708001094.

[7] C. Browne, D. Powley, D. Whitehouse, S. Lucas, P. Cowling, T. Rohlfshagen, P., D. S., Perez, S. Samothrakis, C. Colton, A survey of monte carlo tree search methods, IEEE Transactions on Computational Intelligence and AI in Games 4 (1) (2012) 1–49.

[8] R. Coulom, Efficient selectivity and backup operators in monte-carlo tree search, in: Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers, 2006, pp. 72–83.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

[9] J. Kloetzer, H. Iida, B. Bouzy, The monte-carlo approach in amazons, in: Computer Games Workshop, Amsterdam, The Netherlands, 2007, pp. 113–124.

[10] R. Lorentz, T. Horey, Programming breakthrough, in: H. van den Herik, H. Iida, A. Plaat (Eds.), Computers and Games: 8th International Conference, CG 2013, Yokohama, Japan, August 13-15, 2013, Revised Selected Papers, Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 49–59.

[11] M. Winands, Y. Björnsson, Evaluation function based monte-carlo loa, in: H. van den Herik, P. Spronck (Eds.), Advances in Computer Games, Vol. 6048 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 33–44.

[12] Little golem web site.
URL `http://www.littlegolem.net/jsp/index.jsp`

[13] Invader at the computer olympiads.
URL `http://www.grappa.univ-lille3.fr/icga/program.php?id=249`

[14] The havannah challenge.
URL `https://chessprogramming.wikispaces.com/Havannah#The Havannah Challenge`

[15] J. Lieberum, An evaluation function for the game of amazons, Theor. Comput. Sci. 349 (2) (2005) 230–244. doi:10.1016/j.tcs.2005.09.048.

[16] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, B. Bouzy, Progressive strategies for monte-carlo tree search, New Mathematics and Natural Computation 4 (2008) 343–357. doi:10.1142/S1793005708001094.

[17] S. Gelly, D. Silver, Combining online and offline knowledge in uct, in: ICML '07: Proceedings of the 24th international conference on Machine learning, ACM Press, New York, NY, USA, 2007, pp. 273–280. doi:http://doi.acm.org/10.1145/1273496.1273531.

ACCEPTED MANUSCRIPT