Game-Tree Properties and MCTS Performance

Hilmar Finnsson and **Yngvi Björnsson**
School of Computer Science
Reykjavík University, Iceland
{hif,yngvi}@ru.is

Abstract

In recent years Monte-Carlo Tree Search (MCTS) has become a popular and effective search method in games, surpassing traditional $\alpha\beta$ methods in many domains. The question of why MCTS does better in some domains than in others remains, however, relatively open. In here we identify some general properties that are encountered in game trees and measure how these properties affect the success of MCTS. We do this by running it on custom-made games that allow us to parameterize various game properties in order for trends to be discovered. Our experiments show how MCTS can favor either deep, wide or balanced game trees. They also show how the level of game progression relates to playing strength and how disruptive Optimistic Move can be.

Introduction

Monte-Carlo simulations play out random sequences of actions in order to make an informed decision based on aggregation of the simulations end results. The most appealing aspect of this approach is the absence of heuristic knowledge for evaluating game states. Monte-Carlo Tree Search (MCTS) applies Monte-Carlo simulations to tree-search problems, and has nowadays become a fully matured search method with well defined parts and many extensions. In recent years MCTS has done remarkably well in many domains. The reason for its now diverse application in games is mostly due to its successful application in the game of *Go* (Gelly et al. 2006; Enzenberger and Müller 2009), a game where traditional search methods were ineffective in elevating the state-of-the-art to the level of human experts.

Other triumphs of MCTS include games such as *Amazons* (Lorentz 2008), *Lines-of-Action* (Winands, Björnsson, and Saito 2010), *Chinese Checkers* (Sturtevant 2008), *Spades and Hearts* (Sturtevant 2008) and *Settlers of Catan* (Szita, Chaslot, and Spronck 2009).

MCTS is also applied successfully in *General Game Playing(GGP)* (Finnsson and Björnsson 2008), where it outplays more traditional heuristic-based players in many types of games. However, in other type of games, such as many chess-like variants, the MCTS-based GGP agents are hopeless in comparison to their $\alpha\beta$-based counterparts.

This raises the questions of which game-tree properties inherit to the game at hand make the game suited for MCTS or not. Having some idea of the answers to these questions can be helpful in deciding if a problem should be attacked with MCTS, and then using which algorithmic enhancements.

In this paper we identify high level properties that are commonly found in game trees to a varying degree and measure how they affect the performance of MCTS. As a testbed we use simple custom made games that both highlight and allow us to vary the properties of interest. The main contribution of the paper is the insight it gives into how MCTS behaves when faced with different game properties.

The paper is structured as follows. In the next section we give a brief overview of MCTS and go on to describing the game-tree properties investigated. Thereafter we detail the experiments we conducted and discuss the results. Finally related work is acknowledged before concluding.

Monte-Carlo Tree Search

MCTS runs as many Monte-Carlo simulations as possible during its deliberation time in order to form knowledge derived from their end results. This knowledge is collected into a tree-like evaluation function mimicking a manageable part of the game-tree. In this paper we talk about the game tree when referring to the game itself, but when we talk about the MCTS tree we are referring to the tree MCTS builds in memory. When time is up the action at the root which is estimated to yield the highest reward is chosen for play.

MCTS breaks its simulation down into a combination of four well defined strategic phases or steps, each with a distinct functionality that may be extended independent of the other phases. These phases are: *selection, playout, expansion,* and *back-propagation*.

Selection - When the first simulation is started there is no MCTS tree in memory, but it builds up as the simulations run. The Selection phase lasts while the simulation is still at a game tree node which is a part of the MCTS tree. In this phase informed action selection can be made as we have the information already gathered in the tree. For action selection we use a search policy which has the main focus of balancing exploration/exploitation tradeoff. One can select from multiple methods to implement here, but one of the more popular has been the Upper Confidence Bounds applied to Trees (UCT) algorithm (Kocsis and Szepesvári 2006). UCT selects action a in state s from the set of available actions

$A(s)$ given the formula:

$$a^* = argmax_{a \in A(s)} \left\{ Avg(s,a) + 2 * C_p \sqrt{\frac{\ln Cnt(s)}{Cnt(s,a)}} \right\}$$

Here $Avg(s,a)$ gives the average reward observed when simulations have included taking action a in state s. The function Cnt returns the number of times state s has been visited on one hand and how often action a has been selected in state s during simulation on the other hand. In (Kocsis and Szepesvári 2006) the authors show that when the rewards lie in the interval $[0,1]$ having $C_p = 1/\sqrt{2}$ gives the least regret for the exploration/exploitation tradeoff.

Playout - This phase begins when the simulations exit the MCTS tree and have no longer any gathered information to lean on. A common strategy here is just to play uniformly at random, but methods utilizing heuristics or generalized information from the MCTS tree exist to bias the this section of the simulation towards a more descriptive and believable path. This results in more focused simulations and is done in effort to speed up the convergence towards more accurate values within the MCTS tree.

Expansion - Expansion refers here to the in-memory tree MCTS is building. The common strategy here is just to add the first node encountered in the playout step (Coulom 2006). This way the tree grows most where the selection strategy believes it will encounter its best line of play.

Back-Propagation - This phase controls how the end result of the MC simulations are used to update the knowledge stored in the MCTS tree. This is usually just maintaining the average expected reward in each node that was traversed during the course of the simulation. It is easy to expand on this with things like discounting to discriminate between equal rewards reachable at different distances.

Game-Tree Properties

Following are tree properties we identified as being important for MCTS performance and are general enough to be found in a wide variety of games. It is by no means a complete list.

Tree Depth vs. Branching Factor

The most general and distinct properties of game trees are their depth and their width, so the first property we investigate is the balance between the tree depth and the branching factor. These are properties that can quickly be estimated as simulations are run. With increasing depth the simulations become longer and therefore decrease the number of samples that make up the aggregated values at the root. Also longer simulations are in more danger of resulting in improbable lines of simulation play. Increasing the branching factor results in a wider tree, decreasing the proportion of lines of play tried. Relative to the number of nodes in the trees the depth and width can be varied, allowing us to answer the question if MCTS favors one over the other.

Progression

Some games progress towards a natural termination with every move made while other allow moves that maintain a status quo. Examples of naturally progressive games are *Connect 4*, *Othello* and *Quarto*, while on the other end of the spectrum we have games like *Skirmish*, *Chess* and *Bomberman*. Games that can go on infinitely often have some maximum length imposed on them. When reaching this length either the game results in a draw or is scored based on the current position. This is especially common in GGP games. When such artificial termination is applied, progression is affected because some percentage of simulations do not yield useful results. This is especially true when all artificially terminated positions are scored as a draw.

Optimistic Moves

Optimistic moves is a name we have given moves that achieve very good result for its player assuming that the opponent does not realize that this seemingly excellent move can be refuted right away. The refutation is usually accomplished by capturing a piece that MCTS thinks is on its way to ensure victory for the player. This situation arises when the opponent's best response gets lost among the other moves available to the simulation action selection policies. In the worst case this causes the player to actually play the optimistic move and lose its piece for nothing. Given enough simulations MCTS eventually becomes vice to the fact that this move is not a good idea, but at the cost of running many simulations to rule out this move as an interesting one. This can work both ways as the simulations can also detect such a move for the opponent and thus waste simulations on a preventive move when one is not needed.

Empirical Evaluation

We used custom made games for evaluating the aforementioned properties, as described in the following setup subsection. This is followed by subsections detailing the individual game property experiments and their results.

Setup

All games have players named White and Black and are turn-taking with White going first. The experiments were run on Linux based dual processor Intel(R) Xeon(TM) 3GHz and 3.20GHz CPU computers with 2GB of RAM. Each experiment used a single processor.

All games have a scoring interval of $[0,1]$ and MCTS uses $C_p = 1/\sqrt{2}$ with an uniform random playout strategy. The node expansion strategy adds only the first new node encountered to the MCTS tree and neither a discount factor nor other modifiers are used in the back-propagation step. The players only deliberate during their own turn. A custom-made tool is used to create all games and agents. This tool allows games to be setup as FEN strings[1] for boards of any size and by extending the notation one can select from custom predefined piece types. Additional parameters are used to set game options like goals (capture all opponents or reach the back rank of the opponent), artificial termination depth and scoring policy, and whether squares can inflict penalty points.

[1]Forsyth-Edwards Notation. http://en.wikipedia.org/wiki/FEN

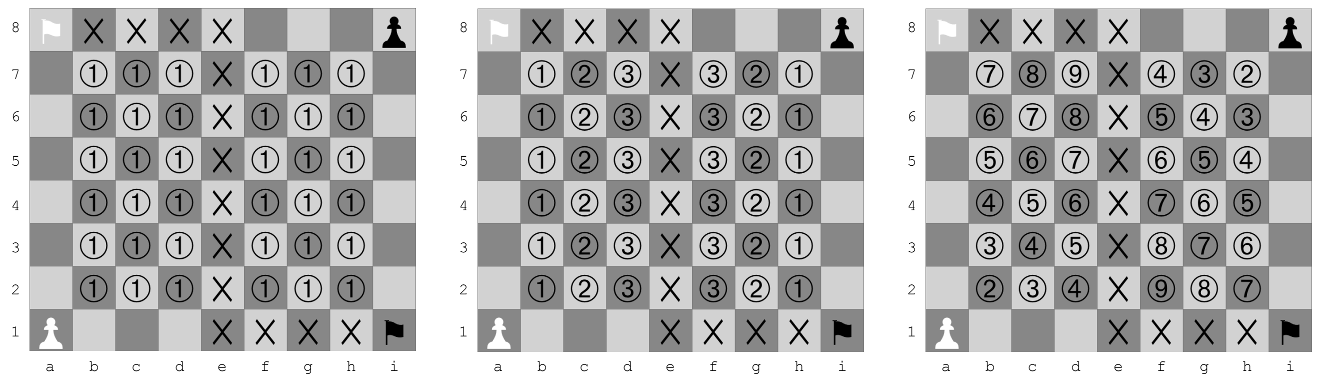


Figure 1: (a)Penalties Game (b)Shock Step Game (c)Punishment Game

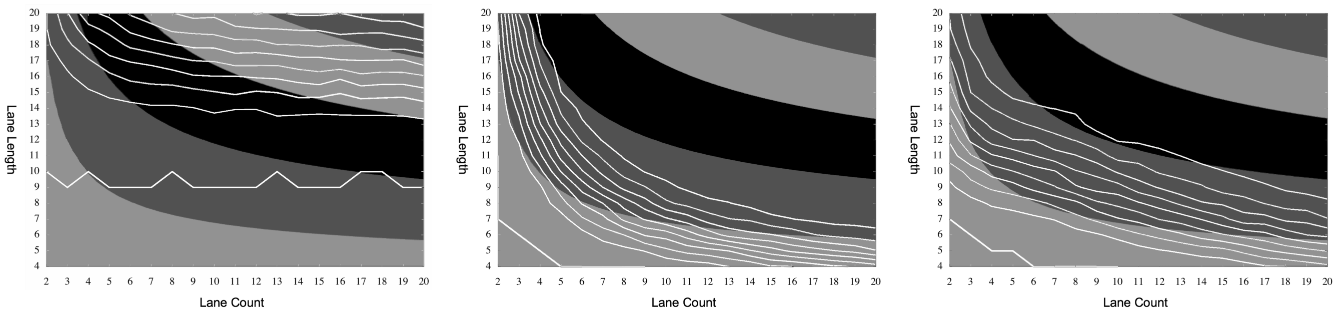


Figure 2: (a)Penalties Results (b)Shock Step Results (c)Punishment results

Tree Depth vs. Branching Factor

The games created for this experiment can be thought of as navigating runners through an obstacle course where the obstacles inflict penalty points. We experimented with three different setups for the penalties as shown in Figure 1. The pawns are the runners, the corresponding colored flags their goal and the big X's walls that the runners cannot go through. The numbered squares indicate the penalty inflicted when stepped on. White and Black each control a single runner that can take one step forward each turn. The board is divided by the walls so the runners will never collide with each other. For every step the runner takes the players can additionally select to make the runner move to any other lane on their side of the wall. For example, on its first move in the setups in Figure 1 White could choose from the moves *a1-a2*, *a1-b2*, *a1-c2* and *a1-d2*. All but one of the lanes available to each player incur one or more penalty points. The game is set up as a turn taking game but both players must make equal amount of moves and therefore both will have reached the goal before the game terminates. This helps in keeping the size of the tree more constant. The winner is the one that has fewer penalty points upon game termination. The optimal play for White is to always move on lane *a*, resulting in finishing with no penalty points, while for Black the optimal lane is always the one furthest to the right. This game setup allows the depth of the tree to be tuned by setting the lanes to a different length. The branching factor is tuned through the number of lanes per player. To ensure that the amount of tree nodes does not collapse with all the transpositions possible in this game, the game engine produces state ids that depend on the path taken to the state it represents. Therefore states that are identical will be perceived as different ones by the MCTS algorithm if reached through different paths. This state id scheme was only used for the experiments in this subsection.

The first game we call Penalties and can be seen in Figure 1 (a). Here all lanes except for the safe one have all steps giving a penalty of one. The second one we call Shock Step and is depicted in Figure 1 (b). Now each non-safe lane has the same amount of penalty in every step but this penalty is equal to the distance from the safe lane. The third one called Punishment is shown in Figure 1 (c). The penalty amount is now as in the Shock Step game except now it gets progressively larger the further the runner has advanced.

We set up races for the three games with all combinations of lanes of length 4 to 20 squares and number of lanes from 2 to 20. We ran 1000 games for each data point. MCTS runs all races as White against an optimal opponent that always selects the move that will traverse the course without any penalties. MCTS was allowed 5000 node expansions per move for all setups. The results from these experiments are shown in Figure 2. The background depicts the trend in how many nodes there are in the game trees related to number of lanes and their length. The borders where the shaded areas meet are node equivalent lines, that is, along each border all points represent the same node count. When moving from the bottom left corner towards the top right one we are increasing the node count exponentially. The lines, called win lines, overlaid are the data points gathered from running the MCTS experiments. The line closest to the bottom left corner represent the 50% win border (remember the opponent is perfect and a draw is the best MCTS can get). Each borderline after that shows a 5% lower win ratio from the previous one. This means that if MCTS only cares how many nodes there are in the game tree and its shape has no bearing on the outcome, then the win lines should follow the trend of the background plot exactly.

The three game setups all show different behaviors related to how depth and branching factor influence the strength of MCTS. When the penalties of any of the sub-optimal moves are minimal as in the first setup, bigger branching factor seems to have almost no effect on how well the player does. This is seen by the fact that when the number of nodes in the game tree increases due to more lanes, the win lines do not follow the trend of the node count which moves down. They stay almost stationary at the same depth.

As soon as the moves can do more damage as in the second game setup we start to see quite a different trend. Not only does the branching factor drag the performance down, it does so at a faster rate than the node count in the game tree is maintained. This means that MCTS is now preferring more depth over bigger branching factor. Note that as the branching factor goes up so does the maximum penalty possible.

In the third game the change in branching factor keeps on having the same effect as in the second one. In addition, now that more depth also raises the penalties, MCTS also declines in strength if the depth becomes responsible for the majority of game tree nodes. This is like allowing the players to make bigger and bigger mistakes the closer they get to the goal. This gives us the third trend where MCTS seems to favor a balance between the tree depth and the branching factor.

To summarize MCTS does not have a definite favorite when it comes to depth and branching factor and its strength cannot be predicted from those properties only. It appears to be dependent on the rules of the game being played. We show that games can have big branching factors that pose no problem for MCTS. Still with very simple alterations to our abstract game we can see how MCTS does worse with increasing branching factor and can even prefer a balance between it and the tree depth.

Progression

For experimenting with the progression property we created a racing game similar to the one used in the tree depth vs. width experiments. Here, however, the size of the board is kept constant (20 lanes $\times$ 10 rows) and the runners are confined to their original lane by not being allowed to move sideways. Each player, White and Black, has two types of runners, ten in total, initially set up as shown in Figure 3. The former type, named *active* runner and depicted as a pawn, moves one step forward when played whereas the second, named *inactive* runner and depicted by circular arrows, stays on its original square when played. In the context of

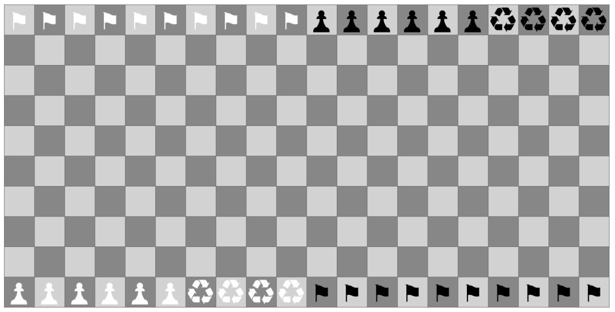


Figure 3: Progression Game

GGP each inactive runner has only a single *noop* move available for play. By changing the ratio between runner types a player has, one can alter the progression property of the game: the more active runners there are, the faster the game progresses (given imperfect play). In the example shown in the figure each player has 6 active and 4 inactive runners. The game terminates with a win once a player's runner reaches a goal square (a square with the same colored flag).

We also impose an upper limit on the number of moves a game can last. A game is terminated artificially and scored as a tie if neither player has reached a goal within the upper limit of moves. By changing the limit one can affect the progression property of the game: the longer a game is allowed to last the more likely it is to end in a naturally resulting goal rather than being depth terminated, thus progressing better. We modify this upper limit of moves in fixed step sizes of 18, which is the minimum number of moves it takes Black to reach a goal (Black can first reach a flag on its 9th move, which is the 18th move of the game as White goes first). A *depth factor* of one thus represents an upper limit of 18 moves, depth factor of two 36 moves, etc.

In the experiments that follow we run multiple matches of different progression, one for each combination of the number of active runners ([1-10]) and the depth factor ([1-16]). Each match consists of 2000 games where MCTS plays White against an optimal Black player always moving the same active runner. The computing resources of MCTS is restricted to 100,000 node expansions per move.

The result is shown in Figure 4, with the winning percentage of MCTS plotted against both the depth factor (left) and percentage of simulations ending naturally (right). Each curve represents a game setup using different number of active runners.[2] The overall shape of both plots show the same trend, reinforcing that changing the depth factor is a good model for indirectly altering the number of simulations that terminate naturally (which is not easy to change directly in our game setup). When looking at each curve in an isolation we see that as the depth factor increases, so does MCTS's performance initially, but then it starts to decrease again. Increasing the depth factor means longer, and thus fewer, simulations because the number of node expansions per move

[2]We omit the 5, 7, and 9 active runners curves from all plots to make them less cluttered; the omitted curves follow the same trend as the neighboring ones.

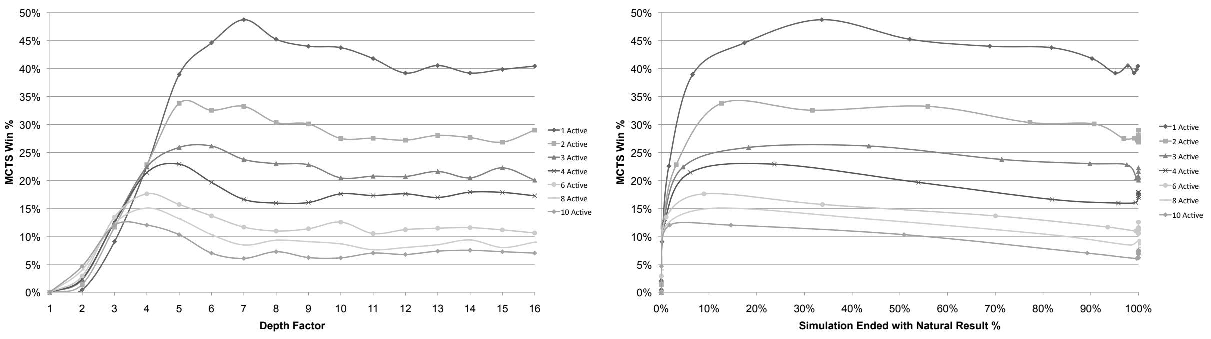


Figure 4: Progression Depth Factor: Fixed Node Expansion Count

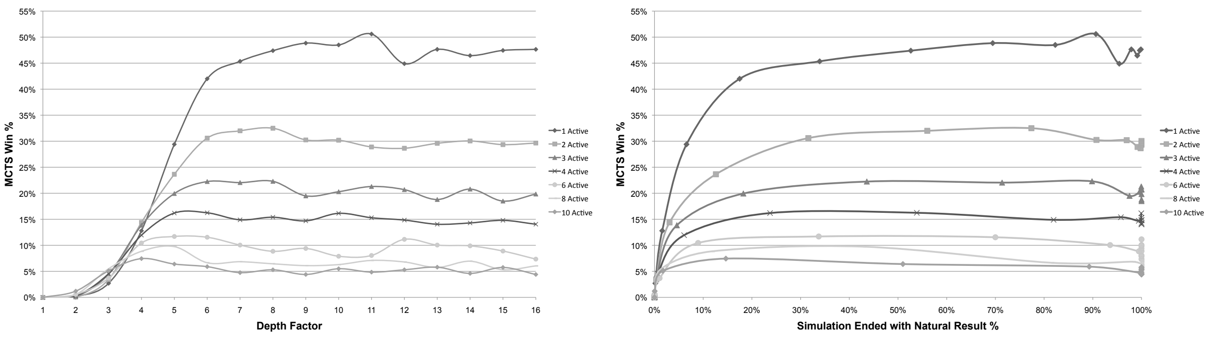


Figure 5: Progression Depth Factor: Fixed Simulation Count

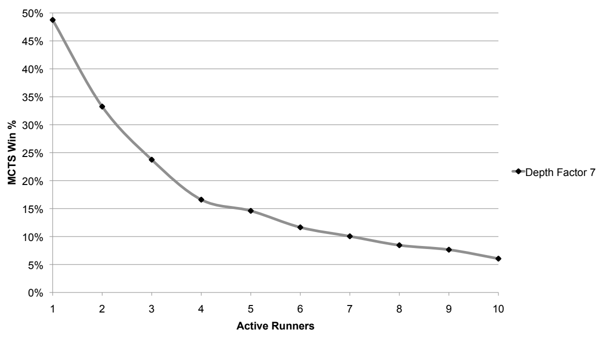
is fixed. The decremental effect can thus be explained by fewer simulations. This is better seen in Figure 5 where the result of identical experiments as in the previous figure is given, except now the number of simulations —as opposed to node expansions— is kept fixed (at 1000).

The above results show that progression is an important property for MCTS, however, what is somewhat surprising is how quickly MCTS's performance improves as the percentage of simulations ending at true terminal states goes up. In our testbed it already reaches close to peak performance as early as 30%. This shows promise for MCTS even in games where most paths may be non-progressive, as long as a somewhat healthy ratio of the simulations terminate in useful game outcomes. Additionally, in GGP one could take advantage of this in games where many lines end with the step counter reaching the upper limit, by curtailing the simulations even earlier. Although this would result in a somewhat lower ratio of simulations returning useful game outcomes, it would result in more simulations potentially resulting in a better quality tradeoff (as in Figure 4).

We can see the effects of changing the other dimension — number of active runners a player has— by contrasting the different curves in the plots. As the number of active runners increases, so does the percentage of simulations ending in true terminal game outcomes, however, instead of resulting in an improved performance, it decreases sharply. This performance drop is seen clearly in Figure 6 when plotted against the number of active runners (for demonstration, only a single depth factor curve is shown). This behavior, however, instead of being a counter argument against progression, is an artifact of our experimental setup. In the game setup, if White makes even a single mistake, i.e. not moving the most advanced runner, the game is lost. When there are more good runners to choose from, as happens when the number of active runners go up, so does the likelihood of inadvertently picking a wrong runner to move. This game property of winning only by committing to any single out of many possible good strategies, is clearly important in the context of MCTS. We suspect that in games with this property MCTS might be more prone to switching strategies than traditional $\alpha\beta$ search, because of the inherent variance in simulation-based move evaluation. Although we did not set out to investigate this now apparently important game property, it clearly deservers further investigation in future work.

Optimistic Moves

For this experiment we observe how MCTS handles a position in a special variation of Breakthrough which accentuates this property. Breakthrough is a turn-taking game played with pawns that can only move one square at a time either straight or diagonally forward. When moving diagonally they are allowed to capture an opponent pawn should one reside on the square they are moving onto. The player who is first to move a pawn onto the opponent's back rank wins. The variation and the position we set up is shown in Figure **??**. The big X's are walls that the pawns cannot move onto. There is a clear cut winning strategy for White



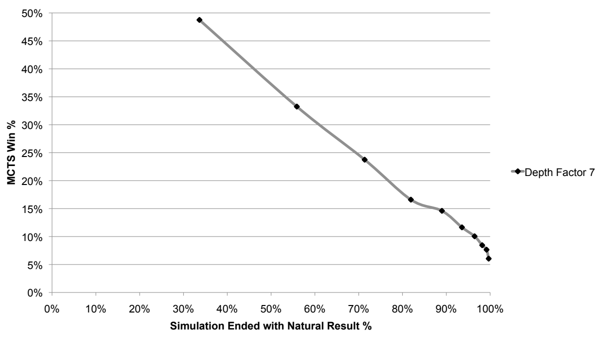


Figure 6: Progression Active Runners: Fixed Node Expansion Count

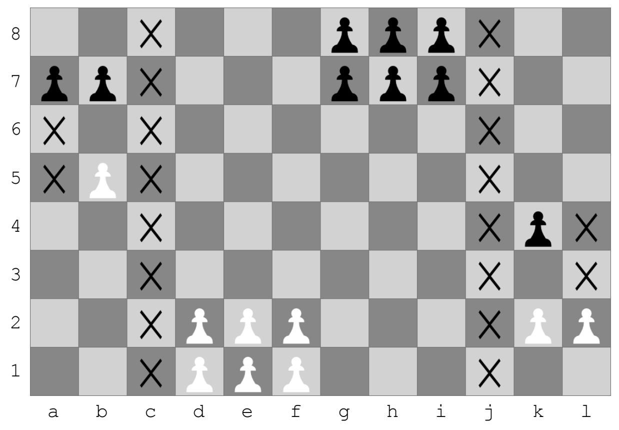


Figure 7: Optimistic Moves Game Results

on the board, namely moving any of the pawns in the midfield on the second rank along the wall to their left. The opponent only has enough moves to intercept with a single pawn which is not enough to prevent losing. This position has also built-in pitfalls presented by an optimistic move, for both White and Black, because of the setups on the *a* and *b* files and *k* and *l* files, respectively. For example, if White moves the *b* pawn forward he threatens to win against all but one Black reply. That is, capturing the pawn on *a7* and then win by stepping on the opponent's back rank. This move is optimistic because naturally the opponent responds right away by capturing the pawn and in addition, the opponent now has a guaranteed win if he keeps moving the capturing pawn forward from now on. Similar setup exists on the *k* file for Black. Still since it is one ply deeper in the tree it should not influence White before he deals with his own optimistic move. Yet it is much closer in the game tree than the actual best moves on the board.

We ran experiments showing what MCTS considered the best move after various amount of node expansions. We combined this with four setups with decreased branching factor. The branching factor was decreased by removing pawns from the middle section. The pawn setups used were the ones shown in Figure **??**, one with the all pawns removed from files *f* and *g*, one by additionally removing all pawns from files *e* and *h* and finally one where the midfield only contained the pawns on *d2* and *i7*. The results are in Table 1 and the row named "Six Pawns" refers to the setup in Figure **??**, that is, each player has six pawns in the midfield and so on. The columns then show the three most frequently selected moves after 1000 tries and how often they were selected by MCTS at the end of deliberation. The headers show the expansion counts given for move deliberation.

The setup showcases that optimistic moves are indeed a big problem for MCTS. Even at 50,000,000 node expansions the player faced with the biggest branching factor still erroneously believes that he must block the opponent's piece on the right wing before it is moved forward (the opponent's optimistic move). Taking away two pawns from each player thus lowering the branching factor makes it possible for the player to figure out the true best move (moving any of the front pawns in the midfield forward) in the end, but at the 10,000,000 node expansion mark he is still also clueless. The setup when each player only has two pawns each and only one that can make a best move, MCTS makes this realization somewhere between the 1,000,000 and 2,500,000 mark. Finally, in the setup which only has a single pawn per player in the midfield, MCTS has realized the correct course of action before the lowest node expansion count measured.

Clearly the bigger branching factors multiply up this problem. The simulations can be put to much better use if this problem could be avoided by pruning these optimistic moves early on. The discovery process of avoiding these moves can be sped up by more greedy simulations or biasing the playouts towards the (seemingly) winning moves when they are first discovered. Two general method of doing so are the MAST (Finnsson and Björnsson 2008) and RAVE (Gelly and Silver 2007) techniques, but much bigger improvements could be made if these moves could be identified when they are first encountered and from then on completely ignored.

Related Work

Comparison between Monte-Carlo and $\alpha\beta$ methods was done in (Clune 2008). There the author conjectures that $\alpha\beta$ methods do best compared to MCTS when: (1)The heuristic evaluation function is both stable and accurate, (2)The game

Table 1: Optimistic Moves Results

Nodes	500,000		1,000,000		2,500,000		5,000,000		10,000,000		25,000,000		50,000,000	
Six Pawns	b5-b6	1000	b5-b6	1000	b5-b6	926	b5-b6	734	b5-b6	945	l2-k3	519	k2-k3	507
					l2-k3	44	k2-k3	153	l2-k3	37	k2-k3	481	l2-k3	484
					k2-k3	30	l2-k3	113	k2-k3	18			f2-e3	9
Four Pawns	b5-b6	1000	b5-b6	1000	b5-b6	1000	b5-b6	996	l2-k3	441	e2-d3	535	e2-d3	546
							k2-k3	3	k2-k3	438	e2-e3	407	e2-e3	449
							l2-k3	1	b5-b6	121	b5-b6	46	e2-f3	5
Two Pawns	b5-b6	980	b5-b6	989	d2-d3	562	d2-d3	570	d2-d3	574	d2-d3	526	d2-d3	553
	l2-k3	13	k2-k3	6	d2-e3	437	d2-e3	430	d2-e3	426	d2-e3	474	d2-e3	447
	k2-k3	7	l2-k3	5	b5-b6	1								
One Pawn	d2-d3	768	d2-d3	768	d2-d3	781	d2-d3	761	d2-d3	791	d2-d3	750	d2-d3	791
	d2-e3	232	d2-e3	232	d2-e3	219	d2-e3	239	d2-e3	209	d2-e3	250	d2-e3	209

is two-player, (3) The game is turn-taking, (4) The game is zero-sum and (5) The branching factor is relatively low. Experiments using both real and randomly generated synthetic games are then administered to show that the further you deviate from theses settings the better Monte-Carlo does in relation to $\alpha\beta$.

In (Ramanujan, Sabharwal, and Selman 2010) the authors identify *Shallow Traps*, i.e. when MCTS agent fails to realize that taking a certain action leads to a winning strategy for the opponent. Instead of this action getting a low ranking score, it looks like being close to or even as good as the best action available. The paper examines MCTS behavior faced with such traps 1, 3, 5 and 7 plies away. We believe there is some overlapping between our Optimistic Moves and these Shallow Traps.

MCTS performance in imperfect information games is studied in (**?**). For their experiment the authors use synthetic game trees where they can tune three properties: (1) *Leaf Correlation* - the probability of all siblings of a terminal node having the same payoff value, (2) *Bias* - the probability of one player winning the other and (3) *Disambiguation factor* - how quickly the information sets shrink. They then show how any combination of these three properties affect the strength of MCTS.

Conclusions an Future Work

In this paper we tried to gain insight into factors that influence MCTS performance by investigating how three different general game-tree properties influence its strength.

We found that it depends on the game itself whether MCTS prefers deep trees, big branching factor, or a balance between the two. Apparently small nuances in game rules and scoring systems, may alter the preferred game-tree structure. Consequently it is hard to generalize much about MCTS performance based on game tree depth and width. Progression is important to MCTS. However, our results suggests that MCTS may also be applied successfully in slow progressing games, as long as a relatively small percentage of the simulations provide useful outcomes. In GGP games one could potentially take advantage of how low ratio of real outcomes are needed, by curtailing potentially fruitless simulations early, thus increasing simulation throughput. Hints of MCTS having difficulty in committing to a strategy when faced with many good ones were also discovered. Optimistic Moves are a real problem for MCTS that escalates with an increased branching factor.

For future work we want to come up with methods for MCTS that help it in identifying these properties on the fly and take measures that either exploit or counteract what is discovered. This could be in the form new extensions, pruning techniques or even parameter tuning of known extension. Also more research needs to be done regarding the possible MCTS strategy commitment issues.

Acknowledgments

This research was supported by grants from The Icelandic Centre for Research (RANNÍS).

References

Clune, J. E. 2008. *Heuristic Evaluation Functions for General Game Playing*. PhD dissertation, University of California, Los Angeles, Department of Computer Science.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *The 5th International Conference on Computers and Games (CG2006)*, 72–83.

Enzenberger, M., and Müller, M. 2009. Fuego - an open-source framework for board games and go engine based on monte-carlo tree search. Technical Report 09-08, Dept. of Computing Science, University of Alberta.

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, Chicago, Illinois, USA, July 13-17, 2008*, 259–264. AAAI Press.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In Ghahramani, Z., ed., *ICML*, volume 227 of *ACM International Conference Proceeding Series*, 273–280. ACM.

Gelly, S.; Wang, Y.; Munos, R.; and Teytaud, O. 2006. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, 282–293.

Lorentz, R. J. 2008. Amazons discover monte-carlo. In *Proceedings of the 6th international conference on Computers and Games*, CG '08, 13–24. Berlin, Heidelberg: Springer-Verlag.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2010. On adversarial search spaces and sampling-based planning. In *ICAPS'10*, 242–245.

Sturtevant, N. R. 2008. An analysis of uct in multi-player games. In van den Herik, H. J.; Xu, X.; Ma, Z.; and Winands, M. H. M., eds., *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, 37–49. Springer.

Szita, I.; Chaslot, G.; and Spronck, P. 2009. Monte-carlo tree search in settlers of catan. In van den Herik, H. J., and Spronck, P., eds., *ACG*, volume 6048 of *Lecture Notes in Computer Science*, 21–32. Springer.

Winands, M. H. M.; Björnsson, Y.; and Saito, J.-T. 2010. Monte carlo tree search in lines of action. *IEEE Trans. Comput. Intellig. and AI in Games* 2(4):239–250.