Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Sufficiency-Based Selection Strategy for MCTS *

Stefan Freyr Gudmundsson and Yngvi Björnsson
School of Computer Science
Reykjavik University, Iceland
{stefang10,yngvi}@ru.is

Abstract

Monte-Carlo Tree Search (MCTS) has proved a remarkably effective decision mechanism in many different game domains, including computer Go and general game playing (GGP). However, in GGP, where many disparate games are played, certain type of games have proved to be particularly problematic for MCTS. One of the problems are game trees with so-called optimistic moves, that is, bad moves that superficially look good but potentially require much simulation effort to prove otherwise. Such scenarios can be difficult to identify in real time and can lead to suboptimal or even harmful decisions. In this paper we investigate a selection strategy for MCTS to alleviate this problem. The strategy, called *sufficiency threshold*, concentrates simulation effort better for resolving potential optimistic move scenarios. The improved strategy is evaluated empirically in an n-arm-bandit test domain for highlighting its properties as well as in a state-of-the-art GGP agent to demonstrate its effectiveness in practice. The new strategy shows significant improvements in both domains.

1 Introduction

From the inception of the field of Artificial Intelligence (AI), over half a century ago, games have played an important role as a testbed for advancements in the field, resulting in game-playing systems that have reached or surpassed humans in many games. A notable milestone was reached when IBM's chess program Deep Blue [Campbell *et al.*, 2002] won a match against the number one chess player in the world, Garry Kasparov, in 1997. The 'brain' of Deep Blue relied heavily on both an efficient minimax-based game-tree search algorithm for thinking ahead and sophisticated knowledge-based evaluation of game positions, using human chess knowledge accumulated over centuries of play. A similar approach has been used to build world-class programs for many other deterministic games, including Checkers [Schaeffer, 2009] and Othello [Buro, 1999].

*The support of Icelandic Centre for Research (RANNIS) is acknowledged.

For non-deterministic games, in which moves may be subject to chance, Monte-Carlo sampling methods have additionally been used to further improve decision quality. To accurately evaluate a position and the move options available, one plays out (or samples) a large number of games as a part of the evaluation process. Backgammon is one example of a non-deterministic game, where possible moves are determined by rolls of dice, for which such an approach led to world-class computer programs (e.g., TD-Gammon [Tesauro, 1994]).

In recent years, a new simulation-based paradigm for game-tree search has emerged, Monte-Carlo Tree Search (MCTS) [Coulom, 2006; Kocsis and Szepesvári, 2006]. MCTS combines elements from both traditional game-tree search and Monte-Carlo simulations to form a full-fledged best-first search procedure. Many games, both non-deterministic and deterministic, lend themselves well to the MCTS approach. As an example, MCTS has in the past few years greatly enhanced the state of the art of computer Go [Enzenberger and Müller, 2009], a game that has eluded computer based approaches so far.

MCTS has also been used successfully in General Game Playing (GGP) [Genesereth *et al.*, 2005]. The goal there is to create intelligent agents that can automatically learn how to skillfully play a wide variety of games, given only the descriptions of the game rules (in a language called GDL [Love *et al.*, 2008]). This requires that the agents learn diverse game-playing strategies without any game-specific knowledge being provided by their developers. Most of the strongest GGP agents are now MCTS-based, such as ARY [Méhat and Cazenave, 2011], CADIAPLAYER [Björnsson and Finnsson, 2009; Finnsson and Björnsson, 2011a], MALIGNE [Kirci *et al.*, 2011], and TURBOTURTLE. Although MCTS have proved on average more effective than traditional heuristic-based game-tree search in GGP, there is still a large number of game domains where it does not work nearly as well, for example in non-progressing or highly tactical (chess-like) games. The more general concept of *optimistic actions*, encapsulating among other things tactical traps, is by and large problematic for MCTS [Ramanujan *et al.*, 2010; Finnsson and Björnsson, 2011b].

In this paper we propose an improved selection strategy for MCTS, *sufficiency-threshold*, that is more effective in domains troubled with optimistic actions and, generally speaking, more robust on the whole. We also take steps towards

better understanding how determinism and discrete game outcomes affect the action-selection mechanism of MCTS, and then empirically evaluate the sufficiency-threshold strategy in such domains, where it shows significant improvements.

The paper is structured as follows. In the next section we provide necessary background material, then we discuss sufficiently good moves and the sufficiency-threshold strategy. This is followed by an empirical evaluation of the strategy in both an n-arm-bandit setup and GGP. Finally, we conclude and discuss future work.

2 Background

Before we introduce our new selection strategy we first provide the necessary background on MCTS, optimistic actions, and n-arm-bandits (which we use as a part of the experimental evaluation of the new selection strategy).

2.1 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a simulation-based search technique that extends Monte-Carlo simulations to be better suited for (adversary) games. It starts by running a pure Monte-Carlo simulation, but gradually builds a game tree in memory with each new simulation. This allows for a more informed mechanism where each simulation consists of four strategic steps: *selection*, *expansion*, *playout*, and *back-propagation*. In the *selection* step, the tree is traversed from the root of the game tree until a leave node is reached, where the *expansion* step expands the leave by one level (typically adding only a single node). From the newly added node a regular Monte-Carlo *playout* is run until the end of the game (or when some other terminating condition is met), at which point the result is *back-propagated* back up to the root modifying the statistics stored in the game tree as appropriate. MCTS continues to run such four step simulations until deliberation time is up, at which point the most promising action of the root node is played.

In this paper we are mainly concerned with the selection step, where *Upper Confidence-bounds applied to Trees (UCT)* [Kocsis and Szepesvári, 2006] is widely used for action selection. At each internal node in the game tree an action a^* to simulate is chosen as follows:

$$a^* = argmax_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \quad (1)$$

$N(s)$ stands for the number of samples gathered in state s and $N(s,a)$ for number of samples gathered when taking action a in state s. $A(s)$ is the set of possible actions in state s and $Q(s,a)$ is the expected return for action a in state s, usually the arithmetic mean of the $N(s,a)$ samples gathered for action a. The term added to $Q(s,a)$ decides how much we are willing to explore, where the constant C dictates how much effect the exploration term has versus exploitation. With $C = 0$ our samples would be gathered greedily, always selecting the top-rated action for each playout. When we have values of $N(s,a)$ which are not defined, we consider the exploration term as being infinity.

Although MCTS is effective in many game domains, it has difficulties in other common game structures. Several such properties have been identified, including traps, the non-progression property and optimistic actions [Ramanujan *et al.*, 2010; Finnsson and Björnsson, 2011b].

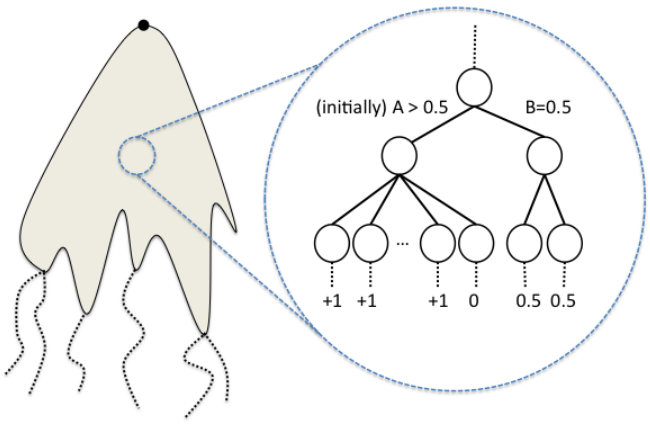


Figure 1: An example of an optimistic action in a MCTS tree: move A initially, but incorrectly, looks much better than move B, because the one move that refutes A (scored as 0) has many siblings that all are winning (for example, in chess, the refutation move could be a trivial recapture of a piece, however, by not recapturing the game is lost).

2.2 Optimistic Actions

Optimistic actions are moves that upon initial investigation look promising, even leading to a win, but are easily refuted in practice. A common source of this problem in simulation-based search are moves leading to positions where the opponent has many legal replies but with only one (or a very few) of them being a refutation. It takes many simulations to explore all the opponent's options and establish the true refutation. Thus, most of the simulations return a positive feedback to start with, labeling the move path leading to that position with a too optimistic score. This can happen at any level in the MCTS game tree, as depicted in Figure 1. Such scenarios are common in many games, for example, recapturing a piece in chess (or other material-dominant games).

When such positions occur in the MCTS game-tree they continue to back-propagate false (too optimistic) values until enough simulations have been performed. In such scenarios it may be better to concentrate simulations on the suspected optimistic move to correct its value. A related scenario is when there are several good looking moves in a given position with a similar value. The standard UCT selection strategy would distribute the simulation effort among them somewhat equally to try to establish a single best move. This may be problematic in the presence of optimistic moves. A better strategy may be to instead commit to one of those sufficiently good moves, at least in discrete outcome deterministic games as we show. This has the benefit of increasing the certainty of the returned value and potentially avoid the optimistic move fallacy. Once the refutation reply has been identified subsequent simulations start to return a radically different value, resulting in the mean score values decreasing.

2.3 N-arm-bandit and the Mean's Path

To simulate a decision process for choosing moves in a game we can think of a one-arm-bandit problem. We stand in front of a number of one-arm-bandits, slot machines, with coins to play with. Each bandit has its own probability distribution which is unknown to us. The question is, how do we maximize our profit? We start by playing the bandits to gather some information. Then, we have to decide where to put our limited amount of coins. This becomes a matter of balancing between exploiting and exploring, i.e. play greedily and try less promising bandits. The selection formula (Equation 1) is derived from this setup [Kocsis and Szepesvári, 2006]. Instead of n one-arm-bandits we can equally talk about one n-arm-bandit and we will use that terminology henceforth.

What do we mean by the bandit's probability distribution? If we only consider the slot machine and discard the game-tree connection we are likely to identify the distribution with its mean, which we believe to be a constant value. With increasing number of samples we gather from the bandit the more the sampled mean approaches the bandit's mean. This happens with more and more certainty thanks to the central limit theorem. Let us re-connect with the game tree. How well does this approach describe what happens in a game tree? In a previous section we defined the optimistic move, i.e. a move which looks promising to begin with when simulations are scarce. Let us assume we are deciding which move to play in a given position and one move shows a very high score after 100 simulations but the scores drops significantly after 1000 simulations, e.g. when we have discovered its refutation further down the game tree. If we play the same position repeatedly, starting from scratch, and measure the score for this move after 100 simulations each time, it would always have a high score. The average of the move for the repeated position would approach the move's correct mean for 100 simulations. However, with additional simulations we would approach a different mean. This is because in a game tree the mean can be moving as new promising moves get established. Therefore, when using an n-arm-bandit to model the behavior of a simulation-based search in a game tree, it is more accurate to accompany each bandit with a path that its mean will follow as opposed to a constant mean. This path we call the mean's path and picture it as a function somewhat related to a discretized random walk.

When dealing with game trees and a selection mechanism such as MCTS the mean can truly change as the Monte-Carlo tree grows larger. For example, in adversary games the MCTS gradually builds up a minimax line of play and discovering a strong reply can drastically change a high sample mean. An important part of Kocsis and Szepesvari's work [Kocsis and Szepesvári, 2006] is that they prove that the selection formula (1) will in the end find the true, game theoretic, value of a position. For the mean's path this equals a stability will be reached after enough number of steps — or simulations. What are then the possible stable values? In deterministic games the true value of a position can only be one of the terminal values, e.g. in a game with binary results, win or loss, the mean's path will only stabilize at the win or loss value. Deterministic games with a few (e.g. two or three) possible terminal values will therefore have the same few possible stable values. This can be exploited as we will show. Non-deterministic games have a different nature as the chance nodes can lead to true values unlike the terminal values.

In [Kocsis and Szepesvári, 2006] the goal is to minimize regret, i.e. we want to minimize our loss of playing the bandits. Using a simple regret would better describe the process of choosing a move in a game [Tolpin and Shimony, 2012]. We can look at it as an n-arm-bandit where we have a fixed amount of coins to use to gather information after which we have to choose one arm to gamble all our money on and the outcome is dictated by the bandit's stable or true value. We only consider simple regret here.

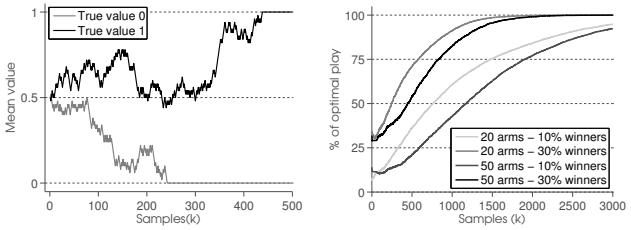
We will not spend many words on the variance of the probability distribution of each arm. The volatility of a position could be evaluated in some games which could be reflected in the value of the standard deviation.

When discussing the action selection for n-arm-bandits we usually talk about UCB (*Upper Confidence Bound*) [Auer *et al.*, 2002] and UCT when working with trees. Avoiding ambiguity we will talk about UCT for both scenarios throughout this paper.

3 Sufficiently Good Moves

Assume that after running a fixed number of simulations in a game, two of the available actions in a position have established themselves as substantially better than the others, say scoring 0.85 and 0.88 respectively where the scoring is between 0 (loss) and 1 win. In a non-deterministic game with a substantial chance element, or in a game where the outcome is scored on a fine grained scale, one might consider spending additional simulations to truly establish which one of the two actions is indeed the better one before committing to either one to play. In contrast, in a deterministic game with a few outcomes this is not necessarily the case. Both moves are likely to lead to a win and no matter which one is played the true outcome is preserved. So, instead of spending additional simulations on deciding between two inconsequential decisions, the resources could be used more effectively. Generally speaking, if there are only win or loss outcomes possible in a deterministic game then once the $Q(s, a)$ values become sufficiently close to a legitimate outcome based on enough simulations, spending additional simulations to distinguish between close values is not necessarily wise use of computing resources. This is even more so true in games suffering from suspected optimistic moves. As mentioned earlier a deterministic game with only win and loss outcomes has only two stable values for the mean's path. We want to take advantage of situations where the possible stable values are easily distinguished and the sample means are close to one of the values. On the other hand when the stable values are unpredictable or close to each others it is possibly better to use other methods [Tolpin and Shimony, 2012; Auer and Ortner, 2000] to gain more accurate estimates of the perceived best moves. We expect this to happen more often in non-deterministic games and deterministic games with many possible outcomes.

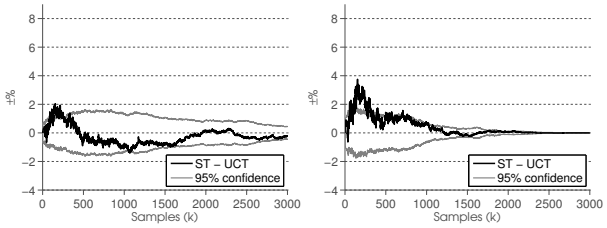
To better understand this concept think of a position in chess where a player can capture a rook or a knight. After a



(a) Two types of mean's path following a random walk

(b) UCT with various number of arms and winners

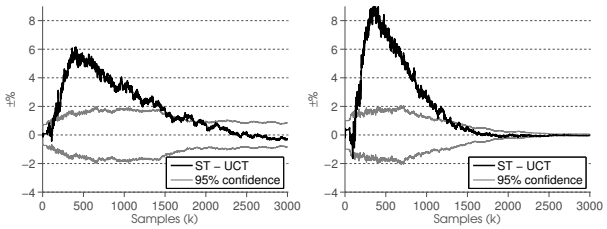
Figure 2: Two examples of mean's paths and ratio of optimal play using UCT



(a) ST vs UCT (10% win)

(b) ST vs UCT (30% win)

Figure 3: 20 arms



(a) ST vs UCT (10% win)

(b) ST vs UCT (30% win)

Figure 4: 50 arms

few simulations we get high estimates for both moves. Probabilities are that both lead to a win, i.e. both might have the same true value as 1. For humans it is possibly easier to secure the victory by capturing the rook but we are more interested in knowing whether there is a dangerous reply lurking just beyond our horizon, i.e. whether one of the moves is an optimistic move. We argue that at this point it is more important to get more reliable information about one of the moves instead of trying to distinguish between, possibly, close to equally good moves. Either our estimate of one of the moves stays high or even gets higher and our confidence increases or the estimate drops and we have proven the original estimate wrong which can be equally important. We introduce a sufficiency threshold α such that whenever we have an estimate $Q(s,a) > \alpha$ from (1) we say that this move is sufficiently good and therefore unplug the exploration. To do so we replace C in Equation (1) by $\hat{C}$ as follows:

$$\hat{C} = \begin{cases} C & \text{when all } Q(s,a) \leq \alpha, \\ 0 & \text{when any } Q(s,a) > \alpha. \end{cases} \quad (2)$$

We call this method *sufficiency threshold (ST)*. When our estimates drop below the sufficiency threshold we go back to the original UCT method. For unclear or bad positions where estimates are less than α most of the time, showing occasional spikes, this approach differs from UCT in temporarily rearranging the order of moves to sample. After such a rearrangement the methods more or less couple back to selecting the same moves to sample from.

4 Experiments

We empirically evaluated the ST selection strategy in three different scenarios: an n-arm-bandit to clearly demonstrate its properties, a sample game position demonstrating its potentials in alleviating problems caused by optimistic moves, and finally in a simulation-based GGP agent to show its effectiveness on a variety of games.

4.1 N-arm-bandit Experiments

In our n-arm-bandits experiment we consider only true values 0 and 1. With each sample we gather for a bandit we move one step further along the mean's path.

Our setup is related to Sutton and Barto's approach (1998) but adapted for deterministic environments. Once a path reaches 0 or 1 it has found its true value and does not change after that. This way we get closer to the true value of a bandit the more samples we gather from it. Figure 2a shows possible paths hitting 0 or 1. We let $M_i(k_i)$ be the mean value for bandit i after k_i samples. The total number of samples is $k = \sum_i k_i$. We use the results from the samples to evaluate the expected rewards of the bandits. Let $Q_i(k)$ play the same role as $Q(s,a)$ in (1), i.e. be the expected reward for bandit i after k samples. For each k we record which arm we would choose for our final bet, i.e. with the highest $Q_i(k)$ value.

We experiment with a bandit as follows. Pulling an arm once is a *sample*. A single *task* is to gather information for k samples, $k \in [1, 3000]$. For each sample we measure which action an agent would take at that point, i.e. which bandit would we gamble all our money on with current information available to us. Let $V(k)$ be the value of the action taken after gathering k samples. $V(k) = 1$ if the chosen bandit has a true value of 1 and $V(k) = 0$ otherwise. A *trial* consists of running t tasks and calculate the mean value of $V(k)$ for each $k \in [1, 3000]$. This gives us one measurement, $\bar{V}(k)$, which measures the ratio of optimal play for an agent with respect to k. There is always at least one bandit with a true value of 1. Each trial is for a single n-arm-bandit, representing one type of a position in a game. In the experiments that follow we compare the performance of ST to UCT, using parameter settings of $C = 0.4$ and $\alpha = 0.6$.

We run experiments on 50 different bandits (models) generated randomly as follows. All the arms start with $M_i(1) = 0.5$ and have randomly generated mean's paths although constrained such that they hit loss (0) or win (1) before taking 500 steps. The step size is 0.02 and each step is equally likely to be positive or negative. One trial consisting of 200 tasks is

run for each bandit, giving us 50 measurements of $\bar{V}(k)$ for each agent and each $k \in [1, 3000]$. In the following charts we show a 95% confidence interval over the models.

In the experiments two dimensions of the models are varied: first the number of arms are either 20 or 50, and second, either 10% or 30% of the arms lead to a win (the remaining to a loss). Figure 2b shows $\bar{V}(k)$ for UCT, which we use as a benchmark. Figures 3 and 4 show the performance of ST relative to UCT when using 20 and 50 arms, respectively. The figures show the increase or decrease in the ratio of optimal play for each k.

ST is overall doing significantly better than UCT except when we have 20 arms and 10% winners. With 50 arms the ST agent is much better than UCT. The general trend is that to begin with there is simply not enough information for neither ST nor UCT to figure out which moves are promising and which are not. After a while ST starts to perform better and only after many more simulations is UCT able to catch up.

4.2 Game Experiments

Using simplified models as we did in the aforementioned experiments is useful for showing the fundamental differences of the individual action-selection schemes. However, an important question to answer is whether the models fit real games. First, we try to get a clearer picture of the optimistic move and how the ST is able to guide the selection strategy out of its optimistic move traps. We have setup a position in the game Breakthrough, frequently played in different variants in GGP competitions. It is highly tactical deterministic game with only win and loss outcomes. It has proved challenging for MCTS to play accurately [Finnsson and Björnsson, 2008]. Each player starts with its first two backranks occupied by pawns of its own color and the goal is to advance a pawn to the opposite end of the board. The first player to achieve that wins. The pawns move forward, one square at a time, both straight and diagonally and can capture opponent's pawns with the diagonal moves.

The position in Figure 5, from a smaller board game variant (the regular game is played on an 8×8 board), showcases the problem at hand, and in a way resembles the types of arms described above. There are two promising moves which turn out to be bad, one that wins, and 10 other moves which do little. In the position, capturing a pawn on a7 or c7 with the pawn on b6 looks promising since all responses from black but one lose. Initially our samples give very high estimates of these two moves until black picks up on capturing back on a7 or c7. There is a forced win for white by playing a6. Black can not prevent white from moving to b7 in the next move, either with the pawn on a6 og b7. From b7 white can move to a8 or c8 and win.

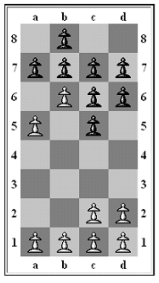


Figure 5: White wins with a5a6

Figure 6 shows how UCT and ST perform in the position in Figure 5. ST clearly outperforms UCT. This position demonstrates clearly the drawbacks of UCT. We are dealing with a problem with an optimistic move where UCT samples more or less equally often for each of the three promising moves. In Figure 6 we see how UCT reaches a plateau around 33%, where the optimal move is played approximately $1/3$ of the time as UCT needs more samples to distinguish between the three promising moves. Being able to disprove optimistic moves early is of a particular interest in GGP where reasoning is somewhat slow, often resulting in decisions being made based on relatively few simulations.

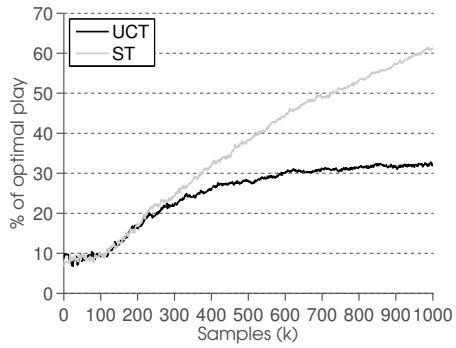


Figure 6: UCT and ST in the position in Figure 5

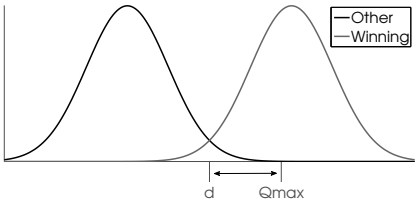


Figure 7: We gather statistics for the difference between d and $Qmax$

4.3 ST in GGP

We also try the ST selection strategy in a GGP environment using a world class GGP agent, CADIAPLAYER. We test it on four games, *Chomp*, *Runners*, *Connect-4*, and *Breakthrough* [1] with different numbers of simulations for the decision making. For ST to work well in a GGP environment it needs a few adjustments. The GGP agents need to be robust across many different games. Therefore, we need to soften the sufficiecy threshold a bit. First of all it can be difficult to decide an α threshold that works for all games. Also, the simulation results need not be at the correct scale and can be misleading in its absolute values. The strength of MCTS comes from ordering the possible actions reasonably, not necessarily with very accurate values - at least not until near the end of the game. What we want to do is to discover when our best perceived move, the highest $Q(s, a) = Qmax$, is 'close enough' to the winning value. The winning value is not necessarily 1; CADIAPLAYER, for example, discounts the result with the length of the simulation. We treat this as a classification problem, where simulations ending in a victory are labelled as the winning class opposed the other class for non-winning sim-

[1] All found in the games repository on the Dresden GGP server

Table 1: ST enhanced CADIAPLAYER vs. CADIAPLAYER

Simulations n	500	1 000	2 000	3 000	5 000	10 000
Runners	48.9 ± 4.8	53.6 ± 4.6	52.4 ± 4.3	$\mathbf{55.8 \pm 4.0}$	52.4 ± 3.7	50.3 ± 2.7
Chomp	49.0 ± 4.9	51.0 ± 4.9	49.5 ± 4.9	51.3 ± 4.9	49.5 ± 4.9	50.3 ± 4.9
Connect 4	47.0 ± 4.8	49.3 ± 4.8	48.4 ± 4.8	47.6 ± 4.8	49.1 ± 4.8	51.3 ± 4.6
Breakthrough	49.5 ± 4.9	$\mathbf{57.3 \pm 4.9}$	$\mathbf{55.0 \pm 4.9}$	$\mathbf{55.0 \pm 4.9}$	52.8 ± 4.9	–

ulations. Both classes form a distribution which then have a discriminant value, d, where it is equally likely for an unlabeled simulation result to belong to each class. The literature is rich in techniques of this sort (e.g., [Bishop, 2006]). Our approach, for simplicity, assumes the standard deviation of both distributions to be equal. Thus the discriminant value is only dependent on the average values of each class and the number of data points in each class

$$d = \frac{n_{other} \cdot \mu_{win} + n_{win} \cdot \mu_{other}}{n_{other} + n_{win}},$$

where n_{win} is the number of data in the winning class, μ_{win} is their average value and similarly n_{other} and μ_{other} are their counterparts for the non-winning class. The discriminant value, d, does not factor in where $Qmax$ is positioned relative to it. This can vary between games. Therefore, we measure the difference between d and $Qmax$ for each simulation, as depicted in Figure 7. The accumulation of these differences forms a distribution which we assume to be a normal distribution. We then use the statistical Q-function to measure the probability for a random variable from this distribution to have a value larger than the current difference $a = Qmax - d$. This probability is used directly as the probability of unplugging the exploration, although we set a floor of 10%, i.e. it is always at least a 10% chance of choosing an action to simulate with the traditional way. Perhaps, we do not need this floor but for the sake of robustness we chose such an ϵ-greedy approach. We consider all previous simulations as training data.

Table 1 shows the result between ST enhanced CADIAPLAYER versus standard CADIAPLAYER. We ran a match of 400 games between the agents, 200 as each side. We also ran it for different values n which is the fixed number of simulations the agents were given to decide each move. The n values range from 500 to $10\,000$. The time and space it needs are negligible in our GGP environment. The results suggest that we have windows of simulation counts where ST improves the player and outside these windows it does not seem harmful, as summarized for Breakthrough in Figure 8.

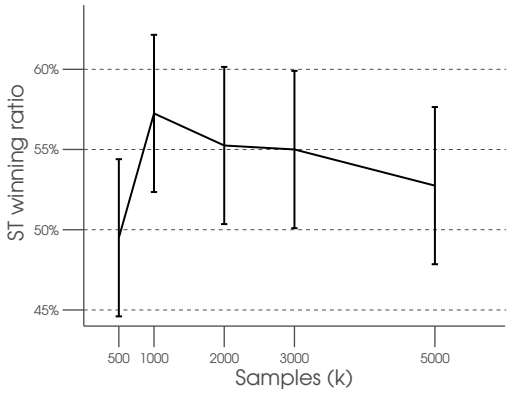


Figure 8: ST winning ratio for Breakthrough

5 Related Work

The current research focus on improving the selection phase in MCTS has been on incorporating domain knowledge to identify good actions earlier, materializing in enhanced schemes such as RAVE [Gelly and Silver, 2007] and Progressive Bias [Chaslot, 2010]. Accelerated UCT and discounted UCT [Hashimoto *et al.*, 2011] are two methods which try to solve the problem of moving mean's path. Although, they use a different terminology. Somewhat surprisingly discounted UCT has not produced positive results. In [Auer and Ortner, 2000] and [Tolpin and Shimony, 2012] statistical methods are used to guide the selection in favorable directions. Both assume a stable underlying mean's path and give significant improvements.

6 Conclusions and Future Work

We have shown that for certain types of games, where the stable values of the mean's path are predictable and far apart, we can improve the MCTS selection strategy with ST. It seems quite robust across many games, and was never harmful while proving particularly effective in domains suffering from the optimistic move syndrome, where it helps to expedite finding refutations. Furthermore, it seems more effective in games with a large branching factor, however, it also showed in practice promise in a low-branching factor game like Runners. This artifact could be explained by ST being able to search selected positions deeper because committing to a single move, thus finding wins and losses earlier. Furthermore, the ST method comes at little or no cost. It is easy to implement and the time and space it consumes are negligible (not measurable in our experiments).

It would be interesting to see whether we get multiple disjoint windows of this sort as the number of simulations increase. That falls under future work as well as running experiments with more games. It is also interesting to see how ST performs in agents designed for a specific games, such as Go. There, we should be able to figure out the sufficiency threshold offline so ST might come at very little computational cost. The dynamic version of ST needed for GGP could be improved with better classification tools, of which the machine-learning literature has plenty.

References

[Auer and Ortner, 2000] Peter Auer and Ronald Ortner. Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem, 2000.

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3):235–256, 2002.

[Bishop, 2006] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Björnsson and Finnsson, 2009] Yngvi Björnsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *IEEE Trans. Comput. Intellig. and AI in Games*, 1(1):4–15, 2009.

[Buro, 1999] Michael Buro. How machines have learned to play Othello. *IEEE Intelligent Systems*, 14(6):12–14, November/December 1999. Research Note.

[Campbell *et al.*, 2002] Murray Campbell, A. Joseph Hoane, Jr., and Feng-Hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1–2):57–83, 2002.

[Chaslot, 2010] Guillaume Chaslot. *Monte-Carlo Tree Search*. PhD dissertation, Maastricht University, The Netherlands, Department of Knowledge Engineering, 2010.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2006.

[Enzenberger and Müller, 2009] Markus Enzenberger and Martin Müller. 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, 2009.

[Finnsson and Björnsson, 2008] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 259–264, Cambridge, MA, USA, 2008. AAAI Press.

[Finnsson and Björnsson, 2011a] Hilmar Finnsson and Yngvi Björnsson. Cadiaplayer: Search-control techniques. *KI*, 25(1):9–16, 2011.

[Finnsson and Björnsson, 2011b] Hilmar Finnsson and Yngvi Björnsson. Game-tree properties and mcts performance. In *IJCAI'11 Workshop on General Intelligence in Game Playing Agents (GIGA'11)*, pages 23–30, 2011.

[Gelly and Silver, 2007] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 273–280, New York, NY, USA, 2007. ACM.

[Genesereth *et al.*, 2005] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General Game Playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[Hashimoto *et al.*, 2011] Junichi Hashimoto, Akihiro Kishimoto, Kazuki Yoshizoe, and Kokolo Ikeda. Accelerated UCT and its application to two-player games. In H. Jaap van den Herik and Aske Plaat, editors, *ACG*, volume 7168 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2011.

[Kirci *et al.*, 2011] Mesut Kirci, Nathan R. Sturtevant, and Jonathan Schaeffer. A GGP feature learning algorithm. *KI*, 25(1):35–42, 2011.

[Kocsis and Szepesvári, 2006] Levante Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293, Berlin / Heidelberg, 2006. Springer.

[Love *et al.*, 2008] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. Technical report, Stanford University, 2008. most recent version should be available at http://games.stanford.edu/.

[Méhat and Cazenave, 2011] Jean Méhat and Tristan Cazenave. A parallel general game player. *KI*, 25(1):43–47, 2011.

[Ramanujan *et al.*, 2010] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On adversarial search spaces and sampling-based planning. In *ICAPS*, pages 242–245, 2010.

[Schaeffer, 2009] Jonathan Schaeffer. *One Jump Ahead:Computer Perfection at Checkers*, volume 978-0-387-76575-4. 2009.

[Sutton and Barto, 1998] Richard Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT press, Cambridge, MA, USA, 1998.

[Tesauro, 1994] Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, 6(2):215–219, 1994.

[Tolpin and Shimony, 2012] David Tolpin and Solomon Eyal Shimony. Mcts based on simple regret. *CoRR*, abs/1207.5536, 2012.