MCTS: Improved Action Selection Techniques for Deterministic Games*

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

Abstract

The simulation-based principles behind *Monte-Carlo Tree Search (MCTS)* have their roots in non-deterministic domains. In game domains, MCTS has nonetheless proved successful in both non-deterministic and deterministic games. This has been achieved by using more or less identical selection mechanisms for choosing actions, thus potentially not fully exploiting the inherent differences of deterministic and non-deterministic games. In this paper we take steps towards better understanding how determinism and discrete game outcomes may influence how action selection is best done in the selection step in MCTS. We use a simple n-arm-bandit test domain to show that action selection can be improved by taking into account whether a game is deterministic and has only few discrete game outcomes possible. We introduce two methods in this context to do so: *moving average* return function and *sufficiency threshold* and evaluate them empirically in the n-arm-bandit test domain, as well as providing preliminary results in a GGP inspired game. Both methods offer significant improvement over the standard UCT action-selection mechanism.

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 1997) and Othello (Buro 1999).

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 has 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. The reason for the success of MCTS in this domain is in part because of the difficulty in automatically generating effective state evaluation heuristics. 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 2011), and MALIGNE (Kirci *et al.* 2011), however, with the notable exception of FLUXPLAYER (Schiffel and Thielscher 2007; Haufe *et al.* 2011), which employs a traditional heuristic-based game-tree search. Currently only finite, deterministic, perfect-information games can be expressed in the GDL language, so the focus of GGP agents has so far been on such games. However, extensions to GDL were recently proposed for augmenting the language to also express non-deterministic, imperfect information games (Thielscher 2010). Subsequently, a relevant question to ask is whether the deterministic vs. non-deterministic nature of a game affects how MCTS is best employed (in GGP). The simulation-

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

based principles behind MCTS have their roots in non-deterministic domains, and although MCTS has been used successfully in both deterministic and non-deterministic game domains, the inherent difference of such types of games has potentially not been fully exploited. For example, actions selection (at non-chance nodes) is currently done the same way in both domains.

In this paper we take steps towards better understanding how determinism and discrete game outcomes can affect the action selection mechanism of MCTS. We show that by taking such properties into consideration action selection can be improved. We introduce two methods for doing so: *moving average* return function and *sufficiency threshold*. Both methods show significant improvement over the standard action-selection mechanism.

The paper is structured as follows. In the next section we give a brief overview of MCTS, followed by a discussion of how determinism and discrete game outcomes influence action selection in MCTS. Next we introduce the two techniques for exploiting the differences and evaluate them empirically. Finally, we conclude and discuss future work.

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. The four steps are depicted in Figure 1. 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:

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

$N(s)$ 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.

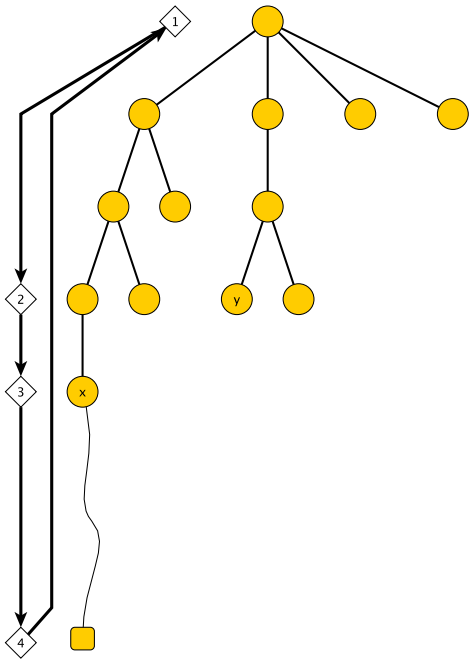


Figure 1: The four steps of MCTS: (1) *selection* step when traversing the tree from the root to a leaf; (2) *expansion* step, where node x is added; (3) *playout* from the expanded node until a terminal position is reached; (4) *back-propagation* step where the game outcome is backed up to the root, updating statistics as appropriate.

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.

Exploiting the Determinism

Assume that after running a fixed number of simulations in the tree in Figure 1 two of the available actions at the root have established themselves as substantially better than the others, say scoring 0.85 and 0.88 respectively. In a non-deterministic game with a substantial chance element, or in a game where the outcome is scored on a fine grained scale (e.g., [0.00,1.00]), 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 only win (=1) and loss (=0) 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 in 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 legiti-

mate outcome based on enough simulations, spending additional simulations to distinguish between close values is not necessarily wise use of computing resources.

Furthermore, with only win and loss (and draw) outcomes in a deterministic game, once winning lines are found, it may take many simulations for that information to propagate up the tree. This is because the $Q(s,a)$ values are average based. For example, assume that node y in Figure 1 is found to be a win. This implies that the second action at the root is a win (because the second player has no alternative to switch to). Provided that there is already a large number of simulations behind the $Q(s,a)$ value of the second root move, it will take many more simulations to raise the action's value to a win. Although the same is true in games with many possible outcomes, then we are much less likely to see such drastic changes in an evaluation, and if a game is non-deterministic, then the effects of such changes will typically be dampened as they back up the tree because of chance nodes.

Generally speaking, in a non-deterministic game, or a game with a fine grained spread of outcomes, the $Q(s,a)$ values may be considered as reasonable estimators of the true game outcome, whereas in deterministic games with only a few discrete outcomes the value is not directly estimating the true outcome. For example, consider a game like Othello. How do we treat an estimate of, say 0.7, in a given position? We know that the true value of the position is either 0 or 1. Figure 2 shows the nature of the sampling averages when the samples we gather approach infinity. In the infinity we reach the true value (Kocsis and Szepesvári 2006). The figure shows how the estimates approach the true values as we gather more samples. For deterministic games with only two outcomes (DGs) the estimates approach either 0 or 1 but for non-deterministic games (or games with a "continuos" range of outcomes) (NDGs) the true values can take number of values between 0 and 1. If there is no additional information at hand we would expect the true value for an NDG in this situation to follow a β-distribution with mean 0.7. What we propose is that, for a DG, it should be possible to exploit the determinism. In a DG we know that an estimate approaches one of few known values. Our first attempt to exploit this are two methods we call *sufficiency threshold* (ST) and *moving average* return function (MA). We introduce them in the next two sections, respectively.

Sufficiency Threshold

In an NDG we may have two best moves with true values as 0.7 and 0.75 representing a 70% and 75% changes of winning, respectively. Early on in our search their estimates may not differ much and the MCTS agent spends most of its resources, i.e. samples, on deciding which is the better move - occasionally sampling from other moves. This is very important as finding the move with true value as 0.75 increases our winning changes by 5%. In DGs this is not as relevant. Let us take an example of a position in chess where white can capture a rook or a knight. After 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. We argue that at this point it is more important to get more

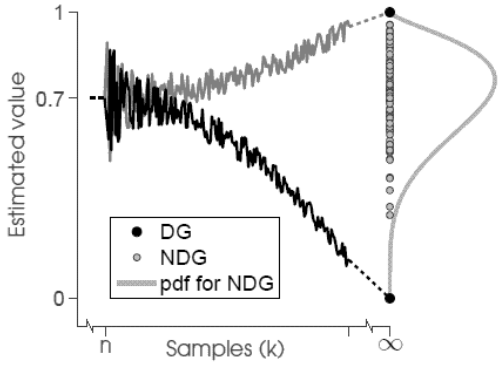


Figure 2: The estimated value is 0.7 after n samples. With infinite samples a DG can only take the values 0 or 1, whereas an NDG can take values spread over the interval $[0,1]$ following a β-distribution with mean 0.7.

reliable information about one of the moves instead of trying to distinguish between, possibly, 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 $Q(s,a) > \alpha$ we unplug the exploration. To do so we replace C in Equation (1) by $\hat{C}$ as follows:

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

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, the ST agent differs from the UCT agent in temporarily rearranging the order of moves to sample. After such a rearrangement the agents more or less couple back to selecting the same moves to sample from.

Moving Average Return Function

Figure 2 shows how the reward function $Q(s,a)$ either approaches 0 or 1 in a DG with two possible outcomes. From our standpoint we sample more often from moves with high estimates and when we choose the opponents moves we sample more often moves which give us lower estimates. Thus we gradually build up lines of play similar to the minimax approach. Our first samples may give a misleading impression of a move but as we gather more samples the estimate $Q(s,a)$ gets more reliable and moves towards 0 or 1. This is a recipe for thinking about moving averages instead of the customary arithmetic mean used to evaluate $Q(s,a)$. The contribution of old samples in the arithmetic mean can overshadow the increase or decrease in $Q(s,a)$. We want to slowly forget older samples as we gather new ones. We do this using λ such that the update rule for $Q(s,a)$ is

$$Q(s,a) = Q_{old}(s,a) + \lambda(r - Q_{old}(s,a)) \quad (3)$$

where r is the result from the simulation. This method is sometimes called recency-weighted average and results in a weighted average of past rewards and the initial estimate of $Q(s, a)$ (Sutton and Barto 1998). Since the initial estimate plays a major role we do not start using the moving average until we have gathered some minimum amount of samples, M. With fewer samples we use the arithmetic mean and λ becomes

$$\lambda = \begin{cases} 1/N(s,a) & \text{when } N(s,a) \leq M, \\ \lambda_0 & \text{when } N(s,a) > M. \end{cases} \quad (4)$$

where λ_0 is a constant.

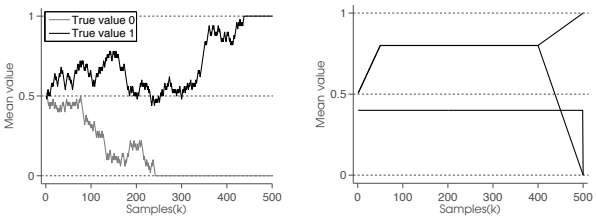
Experiments

In the following we first describe the experimental setup. In the experiments that follow we contrast moving average's (MA's) and sufficiency threshold's (ST's) performance to that of UCT. First we give an intuitional example using a single model and show the effect of adding noise to the model. This is followed by experiments representing average performance over a wide range of models. We also look at how increasing the number of actions affects performance. Finally, we provide preliminary results when using MA and ST in an actual game.

Setup

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). With each bandit having its own probability distribution the randomness is inherited. Therefore it is reasonable to simulate an NDG as n one-arm-bandits. Instead of n one-arm-bandits we can equally talk about one n-arm-bandit and we will use that terminology henceforth. To more accurately simulate a DG we alter the n-arm-bandit. We continue to give each bandit a probability distribution, but now we focus on a mean value for each bandit. Playing a bandit means that we choose a number uniformly at random from the interval $[0, 1]$. If the number is below the bandit's mean value the outcome of the play is a win, 1, otherwise it is a loss, 0. Furthermore we let the mean values move towards the possible true values for the DG we are simulating, following the path of a random walk[1]. One can argue that the mean values follow the path of a correlated random walk (Goldstein 1951), since paths in the game-tree are often highly correlated. With use of the Central Limit Theorem we can approximate a correlated random walk with a random walk when the observation time, i.e. the number of samples, is long enough. It might be interesting to experiment with a correlated random walk coupled with a limited number of samples. This, however, adds complexity to the model which does not serve our purposes at this time. In our experiments 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.

[1] By the path of a random walk we mean a discretized path with constant step size with equal probabilities of taking a step in the positive or negative direction.



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

(b) Three types of manually created mean's paths

Figure 3: Types of mean's paths

Our setup is related to Sutton and Barto's approach [1998] but adapted for deterministic environments. Once a path has reached 0 or 1 it has found its true value and it does not change after that. This way we get closer to the true value of a bandit the more samples we can gather from it. Here, we think of the simulation phase purely as information gathering. Instead of trying to maximize our profit from playing a fixed amount of coins we use a fixed amount of coins to gather information. With this information we choose one bandit to gamble all our money on in one bet and the outcome is dictated by the bandit's true value. We think this setup simulates well the process of using a Monte-Carlo sampling-based method in a DG. It is game-invariant and scales well with its parameters. Figure 3a shows possible paths hitting 0 or 1. We let $M_i(k_i)$ be the mean value for bandit i after k_i samples from it. We use the results from each sample to evaluate the expected reward of the bandits. Let $Q_i(k_i)$ be the expected reward for bandit i after k_i samples from it. The total number of samples is $k = \sum_i k_i$. The closer $Q_i(k_i)$ and $M_i(k_i)$ are to each other, the better is our estimate.

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 percentage for each agent of choosing the optimal action after k simulations. 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

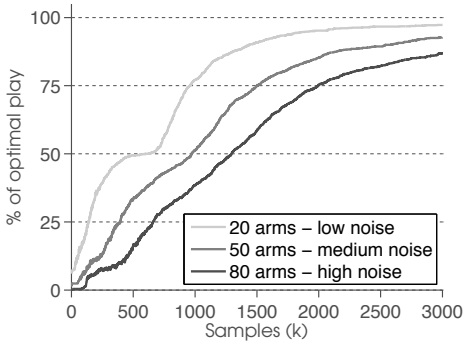
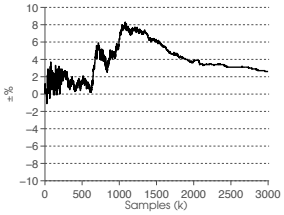
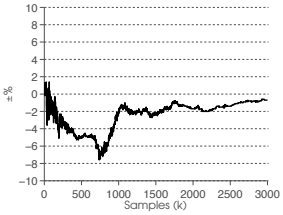


Figure 4: Manual setup using UCT with 1 winner

(a) MA vs UCT (b) ST vs UCT

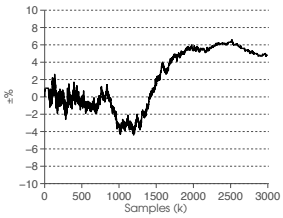
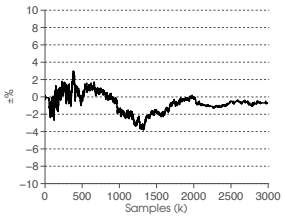
Figure 5: 20 arms - noise level low, 1 winner

game. In all our experiments we compare the performance of ST and MA to UCT.

Intuition Model and Noise Levels

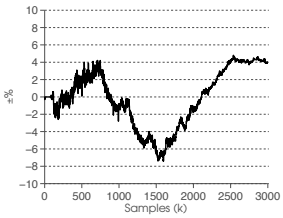
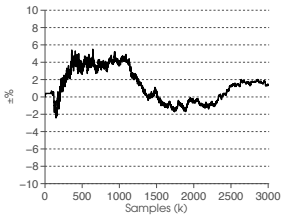
To get an intuition for this setup we start by manually building three types of mean's paths, as seen in Figure 3b. The first and second path are the same up to $k_i = 400$. After that the first path moves towards 1 and the second to 0. The third type of path has a constant mean value of 0.4 up to $k_i = 499$ and is 0 when $k_i = 500$. We build an n-arm-bandit, referred to as a *model*, having one arm of the first type, one arm of the second type and the remaining (varying number of) arms of the third type, representing noise.

In the following experiments we run a single trial of 1000 tasks using the above bandit. We set $C = 0.4$ in UCT (the value used for CADIAPLAYER), $\alpha = 0.6$ in ST, and $M = 30$ and $\lambda = 0.05$ in MA. We did not put much effort into tuning the parameters of ST and MA, so these values do by no mean represent optimal settings. The graph in Figure 4 serves as a baseline and shows the performance of UCT for different number of arms where all but 2 arms are noisy (type 3). More arms thus represent more noise. We can see how the UCT agent distinguishes the two promising arms from the others in the first few steps. The step shape of the low-noise trajectory shows this best. For a while the agent can not distinguish between the two promising arms and thus selects the winners 50% of the time. Thereafter, the agent is able to choose the winner gradually more often as we gather more samples.

(a) MA vs UCT (b) ST vs UCT

Figure 6: 50 arms - noise level medium, 1 winner

(a) MA vs UCT (b) ST vs UCT

Figure 7: 80 arms - noise level high, 1 winner

The graphs in Figures 5-7 show MA's and ST's performance relative to the UCT baseline (Figure 4), with different levels of noise. The y-axis indicates the difference in how often the optimal action is chosen compared to UCT (e.g., if UCT chooses the correct action 50% of the time, a value of +4 means that MA (or ST) chooses the correct action 54% of the time). Two trends can be read from the graphs. With low-noise the ST agent is doing worse than the UCT agent to begin with as it spends more resources on noisy arms giving false positive results. With increasing noise the ST agent surpasses the UCT agent when the UCT agent spends more resources on exploring all arms instead of proving a promising arm right or wrong. As the noise increases the MA agent needs more samples to surpass the other agents but eventually does so convincingly. In Figures 6a and 7a we notice a zig-zag behavior of the MA agent, scoring better than UCT, then worse and again better. This is a known behavior of moving averages when the step-size, λ_0, is too large (Sutton and Barto 1998).

Many Models and Actions

The aforementioned experiments provided intuition by using a single contrived model. In here 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. 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 calculate a 95% confidence interval over the models from $s \times t_{49}/\sqrt{50}$ with s as the sampled standard deviation and t_{49} from the Stu-

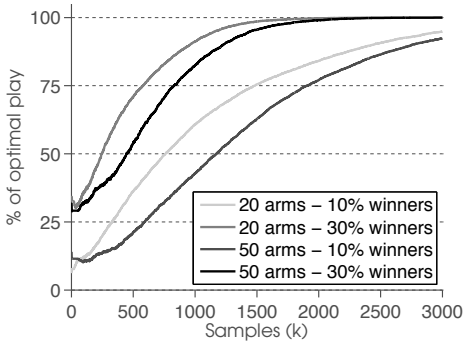
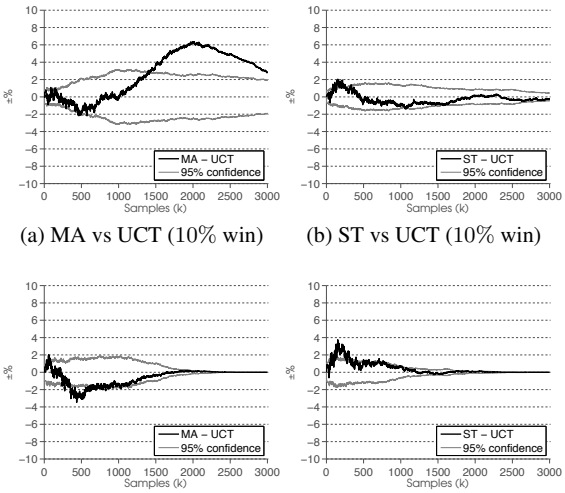


Figure 8: UCT with various number of arms and winners

dent's t-distribution with 49 degrees of freedom.

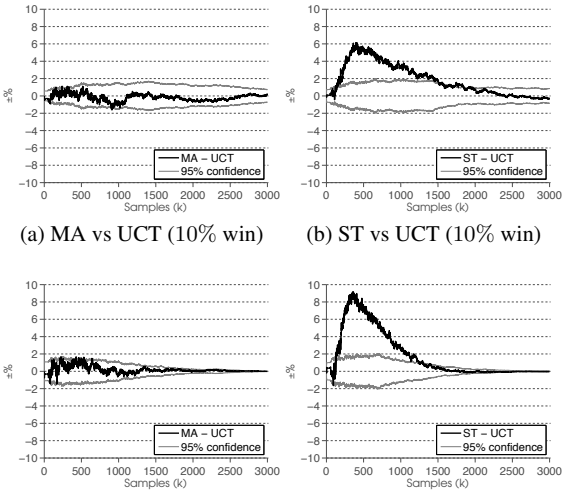
In the experiments two dimensions of the models are varied: first the number of arms are either 20 or 50, and second, either 10% or 30% of the arms lead to a win (the remaining to a loss). Figure 8 shows $\bar{V}(k)$ for UCT, which we use as a benchmark. Figures 9 and 10 show the performance of ST and MA relative to UCT when using 20 and 50 arms, respectively. In Figure 9, first we see how MA performs significantly better with 20 arms and 10% winners. Increasing the arms and winners seems to diminish MA's lead over the other agents. However, we should keep in mind that MA is sensitive to its parameters M and λ_0. It looks like the MA's parameters are tuned too greedily for more arms and winners. ST does not perform much better than UCT with little noise but with higher percentage of winners it performs significantly better than UCT when few samples have been gathered. With more noise the ST agent is much better than UCT. We changed the parameters to a more passive approach for MA with 50 arms by setting $M = 50$ and $\lambda_0 = \frac{1}{50} = 0.02$. There is no longer a significant difference between MA and UCT with 50 arms.



(a) MA vs UCT (10% win) (b) ST vs UCT (10% win)

(c) MA vs UCT (30% win) (d) ST vs UCT (30% win)

Figure 9: 20 arms



(a) MA vs UCT (10% win) (b) ST vs UCT (10% win)

(c) MA vs UCT (30% win) (d) ST vs UCT (30% win)

Figure 10: 50 arms

Breakthrough Game

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. In here we provide preliminary experimental results in a variant of the game Breakthrough[2], frequently played in different variants in GGP competitions. More in-depth experiments remain as future work. The goal of Breakthrough is to advance a pawn to the end of the board. The pawns move forward, one square at a time, both straight and diagonally and can capture opponents pawns with the diagonal moves. The position in Figure 11 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 and win.

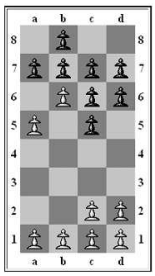


Figure 11: White wins with a5a6

[2]http://boardgames.about.com/od/free8x8games/a/breakthrough.htm

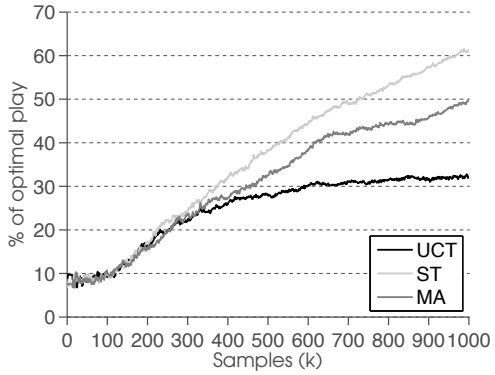


Figure 12: UCT, MA and ST in the position in Figure 11

Table 1: Frequency of a5a6 in 1000 trials

Samples	**UCT**	**ST**	**MA**
500	280	382 ($< 1\%$)	326 ($< 5\%$)
1000	323	611 ($< 1\%$)	497 ($< 1\%$)

Figure 12 shows how the three different methods, UCT, ST and MA, perform in the position in Figure 11. The threshold level for ST is 0.6 and for MA we had $M = 30$ and $\lambda = 0.05$. The exploration constant C was 0.4. Table 1 shows the frequency of the correct move, a5a6, and a measure of the probability that ST and MA are equal to UCT. Both ST and MA perform significantly better than UCT in this position. This position demonstrates clearly the drawbacks of UCT. To begin with, capturing on a7 and c7 are overestimated and playing a6 underestimated. By estimating the expected return with arithmetic mean these over- and underestimates get rectified rather slowly. Also, UCT wastes resources on exploring too much when the estimates are telling us we might have a forced win. We notice that in this real-game setup UCT behaves in a similar way as in the manual setup with low noise. In Figure 12 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.

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).

A solver based variant of MCTS (Winands *et al.* 2008; Cazenave and Saffidine 2010) allows proven values to be propagated correctly in the MCTS game-tree, thus expediting how fast such values back-propagate up the tree. This offers similar benefits as MA does, however, only in the extreme cases of proven values.

We are not aware of any previous work specifically looking into how the selection step in MCTS can take advantage of the fact that it is playing a deterministic game with only discrete game outcomes.

Conclusions and Future Work

We have shown that the estimates of positions in DGs as a function of the number of samples follow a fundamentally different kind of path than in NDGs. Knowing that a function approaches either 0 or 1 in infinity can give us valuable information. Both our proposals to exploit this behavior, ST and MA, improve the traditional selection strategy in MCTS significantly. We should bear in mind that the MA agent is more sensitive to its parameters, M and λ_0, than the other agents and we did not put much effort in fine tuning the parameters. We have fixed the exploration factor to $C = 0.4$ for all agents. Different values of C could change the outcome and that needs to be investigated. Apart from that the only parameters we can change are in the ST and MA agents. The results we have for ST and MA can therefore be thought of as lower limits.

We believe the n-arm-bandit setup represents the behavior of simulation-based agents for DGs well. Taking into account that the we are not trying to maximize our profit as we gather samples, but using our samples to gather information to have the best probabilities of distinguishing a winning arm from the others. The results show significant improvements over UCT in all setups, for either or both of the agents ST and MA. The experiments are meant to mirror a wide range of possible positions in games with low and medium noise as well as few and many winners. Therefore we are confident that we have shown that our proposed methods, and more importantly our use of the determinism, improve the customary UCT selection method when used in DGs. Of special interest to GGP is the early lead of the ST agent over the others as GGP-players often have very limited time to play their moves.

We have pointed out two major parameters in describing a position of a game, i.e. how many moves are possible, the noise, and how many of them lead to a win. Increasing the noise seems to fit well for ST as well as increasing the number of winners. The MA does not perform as well with increased noise and winners but tuning the parameters in MA could improve it substantially. It is interesting to notice that the ratio of winners is not enough to explain the difference of the agents. It seems like the number of unsuccessful arms, or losers, does also play a part as they contribute to the number of false positive arms to begin with.

The Breakthrough experiment indicates that our methods fit well to GGP domains as ST and MA agents perform significantly better than UCT. Comparing the different methods in real games is the natural next step for future work. Furthermore, it would be interesting to try to characterize a real-game with parameters like noise, winners etc., e.g. the branching factor of the game-tree could be a representation of the noise. For each characterization we could then choose the correct method and parameters for the action selection phase. The methods we introduce here are intentionally simplistic, with the main purpose of demonstrating potentials. More sophisticated methods for handling moving averages and cutoff thresholds exist and will be investigated in future

work in this context. It is interesting to research in more detail the effect of noise, winners and losers on the agents and the parameters, C, α, M and λ_0, need to be investigated.

References

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.

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

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

Tristan Cazenave and Abdallah Saffidine. Score bounded monte-carlo tree search. In H. Jaap van den Herik, Hiroyuki Iida, and Aske Plaat, editors, *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2010.

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

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.

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.

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

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.

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

Sidney Goldstein. On diffusion by discontinuous movements and on telegraph equation. *Q. J. Mech. Appl. Math.*, IV:129–156, 1951.

Sebastian Haufe, Daniel Michulke, Stephan Schiffel, and Michael Thielscher. Knowledge-based general game playing. *KI*, 25(1):25–33, 2011.

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

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

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/.

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

Jonathan Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag New York, Inc., 1997.

Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1191–1196. AAAI Press, 2007.

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

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

Michael Thielscher. A general game description language for incomplete information games. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

Mark H. M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte-carlo tree search solver. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors, *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 - October 1, 2008. Proceedings*, volume 5131 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2008.