Quality-based Rewards for Monte-Carlo Tree Search Simulations

Abstract. Monte-Carlo Tree Search is a best-first search technique based on sampling the state space of a domain. In gameplay, positions are scored based on the rewards of numerous randomized play-outs. Generally, play-out rewards are defined discretely, e.g. $r \in \{-1, 0, 1\}$ and backpropagated from the expanded leaf to the root node. However, a play-out may provide additional information beside the loss/draw/win state of the terminal position. Therefore, we introduce measures for assessing the a posteriori quality of Monte-Carlo simulations. We show that altering the rewards of simulated play-outs based on their assessed quality improves results in six distinct two-player games, and in the General Gampe-playing agent CADIAPLAYER. To achieve these results we propose two enhancements, the *Relative Bonus* and *Qualitative Bonus*. Both are used as control variates, a variance reduction method for statistical simulation. The former is based on the number of moves made during a simulation, whereas the latter relies on a domain-dependent assessment of the game's terminal state. The proposed enhancements lead to a performance increase in the domains discussed.

1 INTRODUCTION

Monte-Carlo Tree Search (MCTS) is a best-first search technique based on random sampling of the state space for a specified domain [7, 11]. In gameplay, this means that decisions are made based on the results of random play-outs. MCTS has been successfully applied to various two-player games games such as Go [14], Lines of Action [24], and Hex [1]. Moreover, MCTS has recently seen successes in other domains such as real-time strategy games [5], arcade games such as Ms Pac-Man [12] and the Physical Travelling Salesman problem [13], but also in real-life domains such as optimization, scheduling and security [5].

In the past, several techniques for determining the quality of simulations have been proposed [22], where play-outs are cut-off early and their state heuristically evaluated. Furthermore, evaluating the final *score* of a game has shown to improve results in games that base the winning player on the one with the highest score [16]. However, for some domains a heuristic evaluation may not be available or too time-consuming, and certainly not all games determine the winning player on the highest scoring player. Nonetheless, by merely using the straightforward discrete reward r, any information other than the loss/draw/win state of the play-out's final position is disregarded. For these reasons, we propose assessing the rewards of play-outs based on any information available at a terminal state.

In this paper, two techniques are proposed for determining the quality of a simulation, based on properties of the play-out. The first, Relative Bonus, assesses the quality of a simulation based on its duration. The second, Qualitative Bonus, considers a quality assessment of the terminal state. We show that adjusting results in a specific way using these quantities leads to increased performance in six distinct two-player games. Furthermore, we determine the advantages of using the Relative Bonus in the General Game-playing agent CADIAPLAYER [4], which won the International GGP competition in 2007, 2008, and 2012

The paper is structured as follows. First, the general MCTS framework is discussed in Section 2. Next, two different techniques for assessing the quality of play-outs are detailed in Section 3. Section 4 explains how rewards can be altered using the quality measures from the previous section. Followed by pseudo-code outlining the proposed algorithm. Finally the performance of the proposed enhancements is determined in Section 6, accompanied by a discussion and conclusion in Section 7.

2 MONTE-CARLO TREE SEARCH

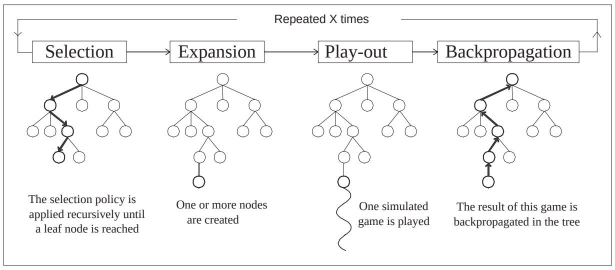
Repeated X times
Selection
Expansion
Play-out
Backpropagation
The selection policy is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 1. Strategic steps of Monte-Carlo Tree Search [6].

Monte-Carlo Tree Search (MCTS) is a search method based on random sampling of a domain [7, 11]. MCTS grows a search tree online by selecting nodes to expand based on a selection policy. Rewards stored at nodes are averaged over the results of numerous simulations. Each simulation consist of two parts, 1) the selection step, where moves are selected and played inside the tree, according to the selection policy, and 2) the play-out step, where moves are played according to a simulation strategy, outside the tree. At the end of each play-out a terminal state is reached and the result r, usually expressed numerically in some discrete range, e.g. $r \in \{-1, 0, 1\}$ representing a loss, draw or win, respectively, is backpropagated along the tree from the expanded leaf to the root node. All rewards are colleced at the nodes on the first ply, on which the final move to play is based. The move is selected based on either the node with the highest number of visits, the highest average reward, or a combination [6].

MCTS searches through possible actions by building a tree incrementally over time. An average is maintained at each node corresponding to the rewards collected each time the node was visited.

The root of the tree corresponds to the game's current position. The basic version of MCTS consists of four steps, which are performed iteratively until a computational threshold is reached, i.e. a set number of iterations, an upper limit on memory usage, or a time constraint. The four steps (depicted in Figure 1) at each iteration are [6]:

- **Selection.** Starting at the root node, children are chosen according to a selection policy described in Subsection 2.1. When a leaf is reached that does not represent a terminal state it is selected for expansion.
- **Expansion.** All children are added to the selected leaf node given available moves.
- **Play-out.** A simulated play-out is run, starting from the state of the added node. Moves are performed randomly or according to a simulation policy until a terminal state is reached.
- **Backpropagation.** The result of the simulated play-out is propagated from the expanded node back up to the root. Statistics are updated along the tree for each node selected during the selection step and visit counts are increased accordingly.

The combination of moves selected during the selection and play-out steps form a single simulation. In its basic form, MCTS requires no evaluation function. Nonetheless, in most cases it is beneficial to add some domain knowledge for selecting moves to play during play-out. MCTS can be terminated anytime and select a move to play based on the number of visits or rewards collected on the first ply.

2.1 UCT

During the selection step, a policy is required to explore the tree for rewarding decisions and finally converge to the most rewarding one. The Upper Confidence Bound applied to Trees (UCT) [11] is derived from the UCB1 policy [2] for maximizing the rewards of a multi-armed bandit. UCT balances the exploitation of rewarding nodes whilst allowing exploration of lesser visited nodes. Consider a node p with children $I(p)$, then the policy determining which child i to select:

$$i^* = argmax_{i \in I(p)} \left\{ v_i + C\sqrt{\frac{\ln n_p}{n_i}} \right\} \quad (1)$$

where v_i is the score of the child i based on the average result of simulations that visited it. n_p is the visit count of the node and n_i the visit count of the current child. C is the exploration constant to be determined by experimentation.

3 ASSESSING SIMULATION QUALITY

In this section two measures by which the quality of the terminal state of a simulation can be assessed are discussed. First, in Subsection 3.1 the duration of a simulation is discussed as a measure of its quality. Second, in Subsection 3.2 a quality assessment of the terminal state of a match is considered. In the next section we establish how these quantities can be used to enhance the rewards of MCTS simulations.

3.1 Simulation Duration

The first, straightforward assessment of a simulation's quality is the duration of the simulated game played. Consider a single MCTS simulation as depicted in Figure 2, then we can define two seperate distances:

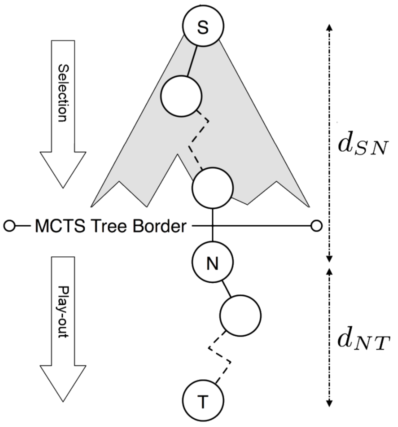

Figure 2. A single MCTS simulation [9].

1. The number of nodes between the root S to the expanded leaf N, d_{SN},
2. The number of moves required to reach T, the simulation's terminal state, from N during play-out d_{NT}.

The length of the simulation is then defined as the sum of these distances:

$$d = d_{SN} + d_{NT} \quad (2)$$

i.e. the total number of moves made by both players before reaching the terminal state of the game T from S, the root's game state. Moves played during play-out are selected by some simulation strategy. Generally this either a random strategy, or a rule-based, reactive strategy, combined with a source of randomness such as an ϵ-greedy selection [17, 18]. Various alternative methods have been proposed, such as using low-level $\alpha\beta$ searches [22], and methods that learn a strategy online, such as N-Grams [19] and the Last-Good-Reply policy [3], or the Move-average Sampling Technique (MAST) [8]. However, simulated moves are far from optimal. Because numerous simulations are to be made during the time allowed for search, any simulation strategy cannot be made overly computationally intensive. As such, each move played ultimately increases uncertainty with respect to the accuracy of the final result by some degree. Hence, the duration of the simulation may be regarded as an indicator of the accuracy of its result.

The main benefit of using simulation duration as a quality measure is that it is domain independent. Unless the number of moves in the game is fixed, the duration of a play-out in particular can be informative in determining its quality. Moreover, in certain games such as Chinese Checkers, simulation length has already been considered part of the evaluation function [15]. Thus it may be considered more than a mere property of the play-out.

3.2 Terminal State Quality

The second measure of a simulation's quality is based on a quality assessment of a match's terminal state. Although evaluation functions can be designed for most games, they are used to evaluate non-terminal states and assign them a specific value. However, this is contrary to MCTS, which generally performs a play-out until a terminal state is reached. Therefore, we are interested in evaluating the terminal state of a game rather than any intermediary states. Although this leaves potentially less informative features to be evaluated, it provides a direct application to MCTS. In some applications MCTS' performance is improved by using either a static, or early cut-off of the simulations, in this paper these methods are not considered and left for future research with respect to the intermediary quality assessment.

As before, consider a single MCTS simulation as depicted in Figure 2. When a terminal state is reached, a quality assessment function is called to evaluate the position with respect to the winning player. This measure q should reflect the quality of a terminal state. For instance, in a game with material such as Breakthrough, Chess or Checkers, an evaluation can be based on scoring the remaining material of the winning player. For a racing game such as Chinese Checkers, the inverse of the number of pieces the opponent has in his target base can be considered. As such, the quality is based on the a posteriori evaluation of the terminal state. Having witnessed the states and actions performed from S to T, the score is based on an assessment of T given the progression $\mathsf{S} \dots \mathsf{N} \dots \mathsf{T}$ (see Figure 2).

4 QUALITY-BASED SIMULATION REWARDS

Based on the classification of quality measures in the previous section, we propose two reward alterations for MCTS: *Relative Bonus (RB)* and the *Qualitative Bonus (QB)*, relating to the length of simulations and the quality assessment of terminal states, respectively.

In the proposed framework, MCTS simulations return a tuple of four reward values, $\langle r, \tau, q, d_{NT} \rangle$ representing the outcome $r \in \{-1, 0, 1\}$, the winning player τ, the quality assessment of the terminal state $q \in (0, 1)$, and the distance from the expanded node N to the terminal state T, d_{NT}, respectively. $d \in (0, m)$ is then computed as shown in Equation 2, which is bounded above by the theoretical maximum duration of the game m. Apart from q, these values are available without requiring extra computational effort.

This section discusses the mathematical basis for altering MCTS rewards. In Subsection 4.1, control variates are discussed as a means of variance reduction and how they can be used to improve MCTS' performance in games. In Subsection 4.2 the Relative Bonus is defined, based on the value of d. Subsection 4.3 details the Qualitative Bonus, which is similar to RB aside from being based on the quality measure q. To conclude, we introduce a method for determining an appropriate value for a, a constant used in the propsed methods in Subsection 4.4.

4.1 Control Variates

Variance reduction methods in mathematical simulation are used to improve estimates by reducing the variance in a simulation's output [10]. Recently, using variance reduction techniques for MCTS has been proposed by Veness et al. [21]. They applied, among others, control variates to UCT in different stochastic games to improve results by the reduction of variance in the reward signal. Say how our approach differs from theirs.

Control variates take advantage of a possible correlation between two random variables X and Y, to improve the estimate $\mathrm{E}\left(X\right)$ given that the mean $v = \mathrm{E}\left(Y\right)$ is known. This is achieved by adding the deviation of Y from its mean, scaled by a constant a, to X. Which results in a new, controlled estimator Z

$$Z = X + a\left(Y - v\right) \tag{3}$$

For a, one can derive a constant $a^* = -\mathrm{Cov}\left(X, Y\right) / \mathrm{Var}\left(Y\right)$ such that the reduction in variance is optimal.

If we define X as the simulation output, i.e. $X_i = r$, and define Y as as one of the quality measures discussed in Section 3, $Y_i = d$ or $Y_i = q$. Then assuming that X and Y are correlated, i.e. $\mathrm{Corr}\left(X, Y\right) \neq 0$, we can find an optimal a^* such that variance in the reward is reduced. In common practical domains, no fixed values for v, $\mathrm{Cov}\left(X, Y\right)$, or $\mathrm{Var}\left(Y\right)$ are known and appropriate estimators for these quantities are required

Although using the quality measures as a control variates is appropriate for MCTS, it is not necessarily the case that optimal variance reduction results in performance increase. In conclusion, although we expect that reducing the variance in the reward signal of MCTS benefits overall performance, it is not a guarantee. Moreover, it is possible that a larger performance increase is gained by using a non-optimal value for a, as the quality measures may provide more advantage than variance reduction alone. Therefore, althoug we propose to define quality-based rewards as control variates, in this paper we are not concerned with the actual reduction in variance, but rather the improvement in performance.

4.2 Relative Bonus

In this subsection the Relative Bonus (RB) is introduced as an enhancement for the rewards generated by MCTS simulations. The enhancements is based on the simulation duration discussed in Subsection 3.1 and used as a control variate as defined in the previous subsection.

First, note that d depends on both the domain and the progress of the game. By itself, the variable is neither normalized, nor relative to a central tendency over time. As such, using it as a control variate as is, leads to a biased distribution of the value over time, where, at the beginning of a game, d takes on higher values than when the game nears its end. Moreover, considering that the length of a game cannot be determined beforehand, we have no accurate way of normalizing the observed values absolutely, based on the expected total length of the game. Therefore, d is standardized as a t-statistic. A sample mean can be approximated online, by maintaining an average $\bar{D}^\tau$ for each player (indexed by τ), over the distribution of observed d values D^τ. After each simulation, $\bar{D}^\tau$ is updated with the observed d, then $\hat{\sigma}^\tau_D$ is the sample standard deviation of the distribution D^τ. Using these statistics, we can define a standardized value λ_r as follows:

$$\lambda_r = \frac{\bar{D}^\tau - d}{\hat{\sigma}^\tau_D} \tag{4}$$

λ_r is both normalized with the sample standard deviation, and is relative to $\bar{D}^\tau$. It is both independent of the progress of the game, and normalized with respect to the current variance in the length of simulations. Since $\mathrm{E}\left(\lambda_r\right) = 0$ due to standardization, λ_r can be added to r as a control variate with $v = 0$. Note that, values of λ_r are higher for shorter simulations.

Using an estimated mean may cause the search to be biased, i.e. moving into the dierction of shorter games. Although there is no immediate sollution to this problem, we propose to reset $\bar{D}^\tau$ and $\hat{\sigma}^\tau_D$ between moves. Moreover, rewards of the first 5% of the expected number of simulations are not altered during search, and $\bar{D}^\tau$ and $\hat{\sigma}^\tau_D$ are updated during this time without altered selection.

Since the distribution of D^τ is not known, λ_r can still take on unrestricted values, particularly if the distribution of D^τ is skewed, or has long tails on either side. Moreover, the relation with the desired reward is not neccesarily linear. As such, in order to both bound, and shape the values of the bonus $b(\lambda_r)$ it is passed to a sigmoid function centered around 0 on both axes, with range $b(\lambda) \in [-1, 1]$.

$$b(\lambda) = \left(-1 + \frac{2}{1+e^{-k\lambda}}\right) \tag{5}$$

k is a constant to be determined by experimentation, it both slopes and bounds the bonus to be added to r. Higher values of k determine both the steepness, and the start and end of the horizontal asymptotes of the sigmoid function. This type of function is commonly used to smooth reward values of evaluation functions. Moreover in [16] r was replaced by a sigmoid representing the final score in Go.

Finally, the reward r returned by the original simulation is given by $b(\lambda_r)$ as follows:

$$r_b = r + \text{sgn}(r) \times a \times b(\lambda_r) \tag{6}$$

This value is backpropagated from the expanded leaf to the root node. The range of r_b is now $[-1-a, 1+a]$, i.e. the bonus r_b is centered around the possible values of r. a is either an empirically determined value, or computed off or on-line as described in Subsection 4.4.

4.3 Qualitative Bonus

Calculation of the Qualitative Bonus follows the same procedure as the Relative Bonus. Similar to RB, the average $\bar{Q}^\tau$ and standard deviation $\hat{\sigma}^\tau_Q$ of observed q values is maintained for each player τ. The value of q is determined by an assessment of the quality of the match's terminal state. Assuming that higher values of q represent a higher quality terminal state for the winning player τ, λ_q is defined as:

$$\lambda_q = \frac{q - \bar{Q}^\tau}{\hat{\sigma}^\tau_Q} \tag{7}$$

Finally the bonus $b(\lambda_q)$ is computed using the sigmoid function in Equation 5 with an optimized k constant, and summed with the result of the simulation r.

$$r_q = r + \text{sgn}(r) \times a \times b(\lambda_q) \tag{8}$$

4.4 Estimating a

In gameplay, X is a nominal variable, i.e. loss, draw, or win, and Y is a discrete scalar. Therefore the method of approximating a^* by determining $-\text{Cov}\,(X,Y)\ /\ \text{Var}\,(Y)$ is not straightforward, which may cause numerical issues. Also note that computing a^* online based on the result of simulations depends heavily on the accuracy of the results of these simulations, and may cause the value of a^* to be incorrect. Furthermore, determining a^* offline, fixes the value over the duration of the game, which is once again suboptimal because its value can be different over the course of the game, e.g. lower values at the start due to lower quality results.

The range of a for the defined control variates in Subsections 4.2, and 4.3 is dependent on the range of r, the simulation's result. Computing a^*, optimal for variance reduction, can thus be achieved by defining X as $X_i = r$, if r is in respect of one player, and $Y_i = d$ or $Y_i = q$. We can compute $a^* = -\widehat{\text{Cov}}\,(X,Y)\ /\ \widehat{\text{Var}}\,(Y)$, i.e. using the sample covariance and variance, online during search, or offline. Effort to determine a value for a^* based on the intuitive definition of X did not result in practical values. Due to the small covariance measured, the resuling range of a^* is too small to make any impact on performance.

Nonetheless, a usable value for a, a' can be computed and used online by using an alternative definition of a^*. As before, let Y be either one of the proposed quality measured, i.e. $Y_i = d$ or $Y_i = q$, and let ρ be the search player, i.e. the player running MCTS. Now separate Y in another distinct random variable Y^w such that

$$Y_i^w = \begin{cases} Y_i & \text{if } \rho \text{ wins the play-out,} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Using this definition we can define $\widehat{\text{Cov}}\,(Y^w, Y)$, which is in terms of Y only. Y^w is then used to compute $a' = \widehat{\text{Cov}}\,(Y^w, Y) / \widehat{\text{Var}}\,(Y)$.

5 PSEUDO-CODE

Algorithm 1 summarizes a single iteration of MCTS enhanced with RB and QB. Note that negamax backups are used in this setup, and therefore r is relative to the player to move at the node that initiates the play-out. The basic MCTS algorithm used in this paper is the MCTS Solver [23], although the details of its implementation are omitted in the psuedo-code. Whenever *update* is used in the algoritm, it refers to updating the average reward for a node, or the sample mean and standard deviation for $\bar{D}^\tau$ and $\bar{Q}^\tau$.

```
1  MCTS(node $p$, node depth $d_{Sp}$):
2      if isLeaf(p) then
3          Expand($p$)
4      Select a child $i$ according to Eq. 1
5      $d_{Si} \leftarrow d_{Sp} + 1$
6      if $n_i = 0$ then
7          $\langle r, \tau, q, d_{iT} \rangle \leftarrow$ Playout($i$)
8          $d \leftarrow d_{Si} + d_{iT}$
9          if enabled($b_r$) and $\hat{\sigma}^\tau_D > 0$ then
10             $r \leftarrow r + \text{sgn}(r) \times a \times$ BONUS($\bar{D}^\tau - d, \hat{\sigma}^\tau_D$)
11             update $\bar{D}^\tau$ and $\hat{\sigma}^\tau_D$ with $d$
12         else if enabled($b_q$) and $\hat{\sigma}^\tau_Q > 0$ then
13             $r \leftarrow r + \text{sgn}(r) \times a \times$ BONUS($q - \bar{Q}^\tau, \hat{\sigma}^\tau_Q$)
14             update $\bar{Q}^\tau$ and $\hat{\sigma}^\tau_Q$ with $q$
15         update node $i$ with $r$
16     else
17         $r \leftarrow$ -MCTS($i$, $d_{Si}$))
18     update node $p$ with $r$
19 return $r$
20
21 BONUS(offset from mean $\delta$, sample std. dev. $\hat{\sigma}$):
22     $\lambda \leftarrow \delta/\hat{\sigma}$
23     $b \leftarrow -1 + 2/1+e^{-k\lambda}$
24 return $b$
```

Algorithm 1: Pseudo-code of the MCTS and BONUS functions (Section 4

During selection, starting from the root, the depth of the current node is updated on line 5. Whenever an expandable node is reached, its children are added to the tree and a play-out is initiated from one of them. A play-out returns a tuple of results, on line 7 four different values are returned: 1) the result of the play-out $r \in \{-1, 0, 1\}$, 2) the winning player τ, 3) the assessed quality of the play-out's terminal state $q \in (0, 1)$, and 4) the number of moves made during play-out d_{iT} defined in Subsection 4. Using these values r is altered. On line 5 the relative bonus is applied to r, using the difference with the winning player's current mean $\bar{D}^{\tau} - d$, i.e. lower values of d give a higher reward. After which the current mean and standard deviation are updated on line 11. QB is applied on line 5 using the assessed quality of the play-out q. Note that the offset from the mean is defined as $q - \bar{Q}^{\tau}$, because in contrast to RB, positive deviation of q from its mean imply better results. The BONUS function on line 20, computes the normalized λ (line 22) and, successively the bonus b (line 23) using the sigmoid function, as defined in Subsections 4.2 and 4.3. The constant a on lines and can be either fixed, or computed online as shown in Subsection 4.4.

6 EXPERIMENTS

To determine the impact on performance of RB and QB, experiments were run on six different two player games. Moreover, the performance of RB is evaluated in the General Gameplaying agent CADIAPLAYER [4], which won the International GGP competition in 2007, 2008, and 2012.

6.1 Experimental Setup

The proposed enhancements were tested in six distinct two player games.

- *Amazons* is played on a 10×10 chessboard. Each player has four amazons that move (and shoot) as queens in chess. However, each move consist of two parts, first the amazon moves, after which she must fire an arrow on an empty position in range, and this square on the board is blocked. The last player to move wins the game.
- *Breakthrough* is played on an 8×8 board. Each player starts with 16 pawns on one side of the board and the aim is to move one of them to the opposite side.
- *Cannon* is a chess-like game where the goal is to checkmate your opponents immobile town. Each player has one town he must place at the start of the game, and 15 soldiers. Soldiers can move or capture forward or may retreat if next to an opponent's soldier. Moreover, three soldiers in a row form a cannon that can move and shoot across the board.
- *Checkers* is played on an 8×8 board, and the goal is to capture all opponent's pieces.
- *Chinese Checkers* is played on a star shaped board. Each player starts with six pieces placed in one of the star's points, and the aim is to move all six pieces to the opposite side of the board. This is a variation of the original Chinese Checkers which is played on a larger board with 10 pieces per player.
- *Pentalath* is a connection game played on a hexagonal board. The goal is to place 5 pieces in a row. Pieces can be captured by fully surrounding an opponent's set of pieces.

For the value of q the following quality measures are used: *Amazons*: the combined number of moves available for the winning player. *Breakthrough* and *Cannon*: the total piece difference between the winning and losing player. *Checkers*: the total number of pieces in play for the winning player. *Chinese Checkers*: the inverse number of the losing player's pieces that reached the home-base. *Pentalath*: the inverse of the longest row of the losing player, given that this row can be extended to a length of 5. For each quality measure an appropriate fixed, normalizer was used to bring the measure within the $[0, 1]$ range.

For each game, an appropriate simulation strategy is used to select moves to make during play-out. Although they were validated to improve performance in all games, the strategies used are not on the level of award-winning programs. Rather, they are implemented to ensure that no obvious mistakes or faulty play is observed in any of the games. All results are reported with these simulation strategies enabled. The results presented for CADIAPLAYER use the n-grams to learn a simulation strategy online [20], the statistics for the n-grams were updated with the original result r.

All experiments were run on 2.2Ghz AMD Opteron CPU, on a Linux operating system. For each game, the constant k used by the sigmoid function was empirically determined by experimenting with values between 0 and 10, with varying increments.

6.2 Results

For each result, the winning percentage is reported for the player with the enhancement enabled, alongside the 95% confidence interval for the result. For each experiment, the players' seats are swapped such that 50% of the games are played as the first player, and 50% as the second.

Table 1. Relative Bonus enabled using different search times, 5000 games

Search time		**1 second**		**5 seconds**	
Game	k	a'	$a = 0.25$	a'	$a = 0.25$
Amazons	2.2	**54.7**(±1.38)	**55.7**(±1.38)		**54.7**(±1.38)
Breakthrough	8.0	50.0(±1.39)	51.0(±1.39)		**51.6**(±1.39)
Cannon	3.0	**62.8**(±1.34)	**60.6**(±1.35)		**58.1**(±1.37)
Checkers	2.8	**52.1**(±0.79)	**52.7**(±0.79)		**50.7**(±0.64)
Chin. Checkers	1.2	**56.8**(±1.37)	**53.2**(±1.38)		**52.5**(±1.38)
Pentalath	1.0	**51.4**(±1.39)	50.3(±1.39)		49.5(±1.39)

For the relative bonus, results for the implemetned games are shown in Table 1. A significant increase in performance is shown for five of the six games, and no adverse results in the other. The value of k was optimized empirically once for each game, and all experiments use the reported value in the second column. Furthermore, we show that using the online definition of a' leads to increased performance over a fixed value for five games. In Breakthrough, defense is equally important as offense, and since the implemented simulation strategy does not contain heuristics for complicated defensive positioning, the play-outs' lengths are biased to quick wins and exclude defensive moves. Chinese Checkers, Cannon and Amazons achieve the most increase in performance using RB. These games improve the estimates of their length over time, and as such, penalizing long games at the beginning of the match ensures better estimations, since the length of the actual match is much shorter overall. Pentalath is a game with a limited lenght, when the board is nearly filled the game is sure to end. As such, the additional information provided by the length of games is limited. For the GGP domain, results are presented in Table 2. A single value for a was used for GGP because a significant number of simulations is required to compute an accurate a'. Moreover, since values for k can not be optimized beforehand, we present the results for two different k values. Although k has an influence on the performance of RB, it is robust with respect to suboptimal values, and an approximate can be used as is made clear by

Table 2. Relative Bonus in GGP, CadiaPlayer, $a = 0.25$
30 sec. startclock, 15 sec. playclock

	$k = 2$	$k = 1.4$
Game	$a = 0.25$	$a = 0.25$
Zhadu	**54.7**(±3.00)	**53.3**(±1.86)
TTCC4	**54.8**(±3.19)	**53.3**(±2.02)
Skirmish	49.7(±3.52)	50.7(±2.20)
SheepWolf	50.9(±2.95)	**52.3**(±1.85)
Quad	43.4(±2.78)	44.7(±1.75)
Merrills	51.6(±4.10)	48.9(±2.56)
Knightthrough	48.9(±3.36)	49.2(±2.11)
Connect5	**53.9**(±2.88)	**54.4**(±1.81)
Chinook	49.3(±3.21)	49.0(±2.00)
Checkers	**58.2**(±5.00)	52.1(±3.16)
Breakthrough	47.1(±4.59)	51.0(±2.88)
Battle	50.1(±3.21)	49.2(±2.01)
3DTicTacToe	**55.7**(±2.56)	**54.5**(±1.62)
Chinese Checkers	**56.9**(±2.86)	**56.0**(±1.79)

the results in Table 2. Each game that benefits from RB does so for either both values, or it is not disadvantageous for either value.

Table 3. Qualitative Bonus using different search times, 5000 games

Search time		**1 second**		**5 seconds**	
Game	k	a'	$a = 0.25$	a'	$a = 0.25$
Amazons	1.6	**64.5**(±1.33)	**58.0**(±1.37)		
Breakthrough	2.0	**74.8**(±1.20)	**71.9**(±1.25)		
Cannon	4.0	**65.9**(±1.31)	**63.0**(±1.34)		
Checkers	2.0	**53.8**(±0.76)	**52.7**(±0.75)		
Chin. Checkers	2.8	**65.7**(±1.32)	**60.1**(±1.36)		
Pentalath	1.6	46.6(±1.38)	50.5(±1.39)		

Results for QB are shown in Table 3. A significant increase in performance is achieved for five of the six games. For Pentalath, the quality assessment is expensive and not very informative as the longest row of the opponent is not likely to make a difference in winning the game. Notably, all other games use simple assessments of their terminal states, which required little to no added computational effort. And in the case of Breakthrough and Cannon, which show the highest overall performance increase, the assessment were not directly linked to winning the game, i.e. the piece count. The results for both RB and QB show that using an additional informative statistic as a control variate in MCTS results can improve performance.

7 CONCLUSION

Monte-Carlo Tree Search (MCTS) bases decisions on sampling a domain and collecting rewards. So far, not much work has been done to improve the values of the reward signal. In this paper, we show that the performance of MCTS is improved by treating the rewards of simulations as a combination of the reward and a quality measure. The combination is performed by treating the quality measure as a control variate, a variance reduction technique. We show that, given that there is a non-zero correlation between the reward-signal and the quality measure, results can be improved in two player games. First, the Relative Bonus (RB) treats the length of a simulation as a measure of its quality. A benefit of this method is that it requires no domain knowledge. Although it works best in games with long play-outs, favoring the shorter ones. When the length of simulations is close to the lenght of the match played, RB provides less added information, and therefore only minor performance enhancements. RB is especially interesting for General Game Playing (GGP), where knowledge of the games played is sparse. The Quality Bonus (QB) improved results in all domains, but requires a some additional domain knowledge. Nonetheless, even using simple quality assessment of the terminal state, such as the piece difference between players, improves results considerably.

REFERENCES

[1] B. Arneson, R. B. Hayward, and P. Henderson, 'Monte-Carlo tree search in Hex', *IEEE Trans. Comput. Intell. AI in Games*, **2**(4), 251–258, (2010).

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer, 'Finite-time analysis of the multiarmed bandit problem', *Machine Learning*, **47**(2-3), 235–256, (2002).

[3] H. Baier and P. D. Drake, 'The power of forgetting: Improving the last-good-reply policy in monte carlo go', *IEEE Trans. on Comput. Intell. AI in Games*, **2**(4), 303–309, (2010).

[4] Y. Björnsson and H. Finnsson, 'Cadiaplayer: A simulation-based general game player.', *IEEE Trans. on Comput. Intell. AI in Games*, **1**(1), 4–15, (2009).

[5] C. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, 'A survey of Monte-Carlo tree search methods', *IEEE Trans. on Comput. Intell. AI in Games*, **4**(1), 1–43, (2012).

[6] G. M. J-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, 'Progressive strategies for Monte-Carlo tree search', *New Math. Nat. Comput.*, **4**(3), 343–357, (2008).

[7] R. Coulom, 'Efficient selectivity and backup operators in Monte-Carlo tree search', in *Proc. 5th Int. Conf. Comput. and Games*, eds., H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, volume 4630 of *Lecture Notes in Computer Science (LNCS)*, pp. 72–83, Berlin Heidelberg, Germany, (2007). Springer-Verlag.

[8] H. Finnsson and Y. Björnsson, 'Simulation-Based Approach to General Game Playing', in *Proc. Assoc. Adv. Artif. Intell.*, volume 8, pp. 259–264, (2008).

[9] H. Finnsson and Y. Björnsson, 'Learning simulation control in general game-playing agents.', in *AAAI*, volume 10, pp. 954–959, (2010).

[10] W David Kelton and Averill M Law, *Simulation modeling and analysis*, McGraw Hill Boston, MA, 2000.

[11] L. Kocsis and C. Szepesvári, 'Bandit Based Monte-Carlo Planning', in *Euro. Conf. Mach. Learn.*, eds., J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, volume 4212 of *Lecture Notes in Artificial Intelligence*, 282–293, (2006).

[12] T. Pepels and M. H. M. Winands, 'Enhancements for Monte-Carlo tree search in Ms Pac-Man', in *IEEE Conf. Comput. Intell. Games*, pp. 265–272, (2012).

[13] E. J. Powley, D. Whitehouse, and P. I. Cowling, 'Monte Carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem', in *IEEE Conf. Comput. Intell. Games*, pp. 234–241. IEEE, (2012).

[14] A. Rimmel, O. Teytaud, C. Lee, S. Yen, M. Wang, and S. Tsai, 'Current frontiers in computer Go', *IEEE Trans. Comput. Intell. AI in Games*, **2**(4), 229–238, (2010).

[15] M. Roschke and N. Sturtevant, 'Uct enhancements in chinese checkers using an endgame database', *IJCAI Workshop on Computer Games*, (2013).

[16] K. Shibahara and Y. Kotani, 'Combining Final Score with Winning Percentage by Sigmoid Function in Monte-Carlo Simulations', in *Proc. IEEE Conf. Comput. Intell. Games*, pp. 183–190, Perth, Australia, (2008).

[17] N. R. Sturtevant, ‘An analysis of UCT in multi-player games’, in *Proc. Comput. and Games*, eds., H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, volume 5131 of *LNCS*, 37–49, Springer, (2008).

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT Press, 1998.

[19] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, ‘N-Grams and the Last-Good-Reply Policy Applied in General Game Playing’, *IEEE Trans. Comp. Intell. AI Games*, **4**(2), 73–83, (2012).

[20] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, ‘N-Grams and the last-good-reply policy applied in general game playing’, *IEEE Trans. Comput. Intell. AI in Games*, **4**(2), 73–83, (2012).

[21] J. Veness, M. Lanctot, and M. Bowling, ‘Variance reduction in monte-carlo tree search’, in *Adv. Neural Inf. Process. Syst.*, eds., J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, volume 24, pp. 1836–1844, (2011).

[22] M. H. M. Winands and Y. Björnsson, ‘$\alpha\beta$-based Play-outs in Monte-Carlo Tree Search’, in *IEEE Conf. Comput. Intell. Games*, pp. 110–117, Seoul, South Korea, (2011).

[23] M. H. M. Winands, Y. Björnsson, and J. Saito, ‘Monte-Carlo Tree Search Solver’, in *Proc. Comput. and Games, LNCS 5131*, volume 5131 of *LNCS*, pp. 25–36, Beijing, China, (2008).

[24] M. H. M. Winands, Y. Björnsson, and J. Saito, ‘Monte Carlo Tree Search in Lines of Action’, *IEEE Trans. Comp. Intell. AI Games*, **2**(4), 239–250, (2010).