Quality-based Rewards for Monte-Carlo Tree Search Simulations

Tom Pepels and **Mandy J .W. Tak** and **Marc Lanctot** and **Mark H. M. Winands**[1]

Abstract. Monte-Carlo Tree Search is a best-first search technique based on simulations to sample the state space of a decision-making problem. In games, positions are evaluated based on estimates obtained from rewards of numerous randomized play-outs. Generally, rewards from play-outs are discrete values representing the outcome of the game (loss, draw, or win), *e.g.,* $r \in \{-1, 0, 1\}$, which are backpropagated from expanded leaf nodes to the root node. However, a play-out may provide additional information. In this paper, we introduce new measures for assessing the a posteriori quality of a simulation. We show that altering the rewards of play-outs based on their assessed quality improves results in six distinct two-player games and in the General Game Playing agent CADIAPLAYER. We propose two specific enhancements, the *Relative Bonus* and *Qualitative Bonus*. Both are used as control variates, a variance reduction method for statistical simulation. Relative Bonus is based on the number of moves made during a simulation and Qualitative Bonus relies on a domain-dependent assessment of the game's terminal state. We show that the proposed enhancements, both separate and combined, lead to significant performance increases in the domains discussed.

1 INTRODUCTION

Monte-Carlo Tree Search (MCTS) [7, 12] is a simulation-based best-first search technique for decision-making problems. Recently, MCTS has shown to improve performance in various domains, such as the two-player games Go [17], Lines of Action [25], and Hex [1]. Moreover, MCTS has seen successes in other domains such as real-time strategy games [5], arcade games such as Ms Pac-Man [14] and the Physical Traveling Salesman [15], but also in real-life domains such as optimization, scheduling, and security [5].

Standard MCTS runs simulated play-outs until a terminal state is reached. The returned reward signal represents the outcome of the game's final position (win, draw, or loss), but any other information about the play-out is disregarded. Several techniques for determining the quality of simulations have been previously proposed. Early cut-offs terminates a play-out and returns a heuristic value of the state [24]. Evaluating the final *score* of a game, *e.g.,* using the number of points achieved, and combining it with the outcome has shown to improve results in points-based games [19]. However, for some domains, a heuristic evaluation may not be available or too time-consuming. Additionally, many games do not have point-based scores and only have the distinct possible outcomes, win, draw, or loss.

[1] Games and AI Group, Department of Knowledge Engineering (DKE), Maastricht University, Maastricht, The Netherlands, email: {tom.pepels,mandy.tak,marc.lanctot,m.winands}@maastrichtuniversity.nl

In this paper, two techniques are proposed for determining the quality of a simulation based on properties of play-outs. The first, Relative Bonus, assesses the quality of a simulation based on its length. The second, Qualitative Bonus, formulates a quality assessment of the terminal state. Adjusting results in a specific way using these values leads to increased performance in six distinct two-player games. Moreover, we determine the advantage of using the Relative Bonus in the General Game Playing agent CADIAPLAYER [4], winner of 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. This is followed by pseudo-code outlining the proposed algorithm in Section 5. 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

Monte-Carlo Tree Search (MCTS) is a simulation-based search method [7, 12]. MCTS grows a search tree incrementally over time, by *expanding* a leaf node of the tree every simulation. Values of the rewards stored at nodes, when averaged over the results of numerous simulations, represent an estimate of the win probability of simulations that pass through the node. 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.

In its basic form, MCTS does not require an evaluation function. Nonetheless, in most domains it is beneficial to add some domain knowledge to the play-out policy. MCTS can be terminated at any time, for instance when some computational limit is reached, to select a move to return. The move to make is selected by choosing either the child of the root with the highest number of visits, the highest average reward, or a combination [6].

During the selection step, a policy is required to explore the tree to decide on promising options. The UCT [12] is derived from the UCB1 policy [2] for maximizing the rewards of a multi-armed bandit. In UCT, each node is treated as a bandit problem whose arms are the moves that lead to different child nodes. UCT balances the exploitation of rewarding nodes whilst allowing exploration of less often visited nodes. Consider a node p with children $I(p)$, then the policy determining which child i to select is defined as:

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

where v_i is the score of the child i based on the average result of simulations that visited it, n_p and n_i are the visit counts of the current node and its child, respectively. C is the exploration constant to tune.

3 ASSESSING SIMULATION QUALITY

In this section, we discuss two quality assessments of the terminal state of a simulation. First, in Subsection 3.1 the length 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 Length

The first assessment of a simulation's quality is the length of the simulated game played. Consider a single MCTS simulation as depicted in Figure 1, then we can define two separate distances:

1. The number of moves made from the root S to the 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 defined as the sum of these distances:

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

i.e., the total number of moves made before reaching the terminal state of the simulation T from S. A play-out policy chooses moves at each state uniformly random, is rule-based, reactive, or combined with a source of expert or heuristic information such as an ϵ-greedy policy [20, 21]. Alternative methods have been proposed, such as using low-level $\alpha\beta$ searches [24], and methods that learn a strategy online, such as the Last-Good-Reply policy [3], Move-average Sampling Technique (MAST) [10], or N-Grams [22]. However, any play-out policy balances computational effort required to select high-quality moves, with the number of simulations performed in the allowed search time. Therefore, moves sampled from the play-out policy are far from optimal. Consequently, each move made in the play-out ultimately increases the uncertainty of the result obtained. Hence, the length of the simulation can be regarded as an indicator of the certainty in the accuracy of its result. The depth of the leaf is included in d for two reasons, 1) to prevent biasing play-out results based on d, as it ensures the search is not merely biased to deeper subtrees, instead preferring the most robust decision regardless of search depth, and 2) as shown in Section 4, a single mean can be used to relate the observed values of d to their central tendency.

The main benefit of using simulation length as a quality measure is that it is domain independent. Unless the number of moves in the game is fixed, the length of a simulation can be informative in determining its quality. In addition, simulation length has also previously been used to enhance the performance of MCTS [8, 11, 18]. Also, the simulation length was used as a way to terminate wasteful play-outs in general Game Playing [9].

3.2 Terminal State Quality

The second measure of a simulation's quality is a quality assessment of its specific terminal state reached. Rather than evaluating intermediary states, we are interested in determining the quality of terminal states directly in order to augment the information used by MCTS. In some domains MCTS' performance is improved by using either a static, or an early cut-off of the simulations and evaluating this position. However, a heuristic evaluation may not be available or too time-consuming. In this paper, we consider any available information that can help describe the quality of a terminal state, such as how convincing of a win it was based on in-game scores.

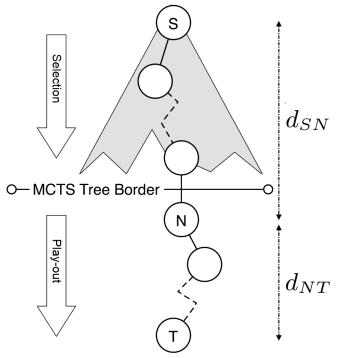

Figure 1. A single MCTS simulation [9].

As before, consider a single MCTS simulation as depicted in Figure 1. 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 S ... N ... T (see Figure 1).

Score-based bonuses have previously been shown to enhance the performance of MCTS in Go [8, 26], Mancala [16], and Blokus-Duo [19]. As detailed in the next section, our approach differs from previous work because the bonuses are normalized based on their deviation from the mean, whereas previous works add an absolute bonus to the results of play-outs. Control variates are introduced to show how these bonuses are related to variance reduction in MCTS. Previous work focused on reducing variance introduced by chance events in the games Pig, Can't Stop, and Dominion [23]. Here, each bonus is a control variate with respect to the reward value returned by the play-out policy.

4 QUALITY-BASED SIMULATION REWARDS

This section discusses the foundation for altering MCTS simulation rewards. 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. The distance $d \in (0, m)$, bounded above by the theoretical maximum duration of the game m, is then computed as shown in Equation 2. Apart from q, these values are available with minimal extra computational effort.

In Subsection 4.1, we describe control variates and explain how they are used as a basis of the proposed quality measures discussed in the previous section. In Subsections 4.2 and 4.3, *Relative Bonus (RB)*

and *Qualitative Bonus (QB)* are defined, respectively. In Subsection 4.4, a method for determining an approximate value for a, a constant used in the proposed methods, is introduced.

4.1 Control Variates

Variance reduction methods in mathematical simulation are used to improve estimates by reducing the variance in a simulation's output [13]. Recently, variance reduction techniques have been proposed for MCTS by Veness et al. [23]. They applied, among others, control variates to UCT in different stochastic games to improve results by the reducing variance of the estimators. However, their applications were focused on reducing the variance that occurred from the stochastic events in the domain. Furthermore, their control variates are recursively defined over sequences of states and actions. In this paper, we focus on a simpler application of reducing the variance in the reward signal due to randomized play-outs.

Control variates take advantage of a correlation between two random variables X and Y, to improve estimators of X given that the mean $\mathbb{E}(Y)$ is known. This is achieved by adding the deviation of Y from its mean, scaled by a constant a, to X. We define a new random variable,

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

Y is called a *control variate* because its deviation from $\mathbb{E}(Y)$ is used to control the observed value X. Given that a non-zero correlation between X and Y exists, one can show that there is a value $a^* = -\mathrm{Cov}(X,Y) \,/\, \mathrm{Var}(Y)$ that minimizes $\mathrm{Var}(Z)$.

We define X as the simulation outcome, *i.e.,* $X_i = r$, and define Y 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.,* $\mathrm{Corr}(X,Y) \neq 0$, we can compute an estimate of a^* from observations such that variance in the reward is reduced. In common practical domains, no fixed values for $\mathbb{E}(Y)$, $\mathrm{Cov}(X,Y)$, or $\mathrm{Var}(Y)$ are known and appropriate estimators for these quantities are required.

4.2 Relative Bonus

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

Note that d depends on the domain, the progress of the game, and the play-out policy. As such, the range of d varies accordingly. Therefore, d is standardized by defining it as a *t*-statistic, as an offset from its central tendency. 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 $\hat{\sigma}_D^\tau$ is the sample standard deviation of the distribution D^τ. Using these statistics, we define a standardized value λ_r as follows:

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

Note that λ_r is a function of d but to simplify the notation we omit the dependency. Also, 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. When the number of samples is large, $\mathbb{E}(\lambda_r) \approx 0$ due to standardization and so λ_r can be added to r as a control variate with $\mathbb{E}(Y) = 0$ in Equation 3. 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 direction of shorter games. Although there is no immediate solution to this problem, we propose to reset $\bar{D}^\tau$ and $\hat{\sigma}_D^\tau$ between moves. Moreover, rewards of the first 5% of the expected number of simulations are not altered during search, and $\bar{D}^\tau$ and $\hat{\sigma}_D^\tau$ 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 heavy tails on either side. Moreover, its relationship with the desired reward is not necessarily linear. As such, in order to both bound and shape the values of the bonus it is passed to a sigmoid function centered around 0 on both axes whose range is $[-1, 1]$,

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

Here, k is a constant to be determined by experimentation, it both slopes and bounds the bonus to be added to r. This type of function is commonly used to smooth reward values of evaluation functions. In [19], r was replaced by a sigmoid representing the final score.

Finally, the modified reward with the relative bonus is,

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

This value is backpropagated from the expanded leaf to the root node. The range of r_b is $[-1-a, 1+a]$, *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

The Qualitative Bonus (QB) follows the same procedure as RB. Similar to RB, the average $\bar{Q}^\tau$ and standard deviation $\hat{\sigma}_Q^\tau$ 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:

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

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

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

4.4 Estimating a

In gameplay, X is a nominal variable, *i.e.,* loss, draw, or win, and in our case, Y is a discrete scalar. Therefore the method of determining a^* is not straightforward. Moreover, since the quantities required to compute a^*, either online during search, or offline, are unknown for complex domains, a^* can be an approximation at best. Efforts to determine a value for a^* based on the intuitive definition of X and Y, shown in Subsection 4.1 did not result in practical values. Among others, determining the biserial covariance between X and Y was tried. However, due to the small covariance measured, the resulting range of a^* was 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 running MCTS. Now separate Y in another distinct random variable Y^w such that

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

Using this definition we can determine the sample covariance, $\widehat{\text{Cov}}\left(Y^w, Y\right)$ in terms of Y values only. This ensures there are no numerical differences between the quantities. Next, we can compute

$$a' = \left| \widehat{\text{Cov}}\left(Y^w, Y\right) / \widehat{\text{Var}}\left(Y\right) \right|, \quad (10)$$

and use it, as the value for a in Equations 6 and 8.

Because the choice of Y^w is arbitrary, since every game is won by either one of the players, the actual value of a' is treated as a magnitude, and its sign is not used. This works because the assumption is made that 1) shorter games are preferred over long ones (Equation 4) and, 2) higher q values indicate better play-out quality (Equation 7).

5 PSEUDO-CODE

Enhancing MCTS with RB and/or QB is explained in Algorithm 1, which summarizes a single iteration of MCTS. Note that negamax backups are used in this set-up, and therefore r is relative to the player to move at the start of the play-out. Although we use the MCTS-Solver [25], details of its implementation are omitted in the pseudo-code. Whenever *update* is used in the algorithm, it refers to updating the average reward for a node, or the sample mean and standard deviation for $\bar{D}^\tau$ and $\bar{Q}^\tau$.

During selection, starting from the root, the depth of the current node is updated on line 4. 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 6 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 3.1. On line 9 the relative bonus is applied to r, using the difference with the winning player's current mean $\bar{D}^\tau - d$. After which the current mean and standard deviation are updated on line 10. QB is applied similarly on line 12 using the assessed quality of the play-out q. Positive deviations of q from its mean imply better results. The BONUS function on line 17, computes the normalized λ (line 20) and, successively the bonus b (line 21) using the sigmoid function, as defined in Subsections 4.2 and 4.3. The constant a on lines 9 and 12 can be either fixed, or computed online as shown in Subsection 4.4.

```
1  MCTS(node p, node depth d_Sp):
2      if isLeaf(p) then Expand(p)
3      Select a child i according to Eq. 1
4      d_Si ← d_Sp + 1
5      if n_i = 0 then
6          ⟨r, τ, q, d_iT⟩ ← PLAYOUT(i)
7          d ← d_Si + d_iT
8          if enabled(b_r) and σ̂²_D > 0 then
9              r ← r + sgn(r) · a · BONUS(D̄^τ − d, σ̂²_D)
10             update D̄^τ and σ̂²_D with d
11         if enabled(b_q) and σ̂²_Q > 0 then
12             r ← r + sgn(r) · a · BONUS(q − Q̄^τ, σ̂²_Q)
13             update Q̄^τ and σ̂²_Q with q
14         update node i with r
15     else
16         r ← -MCTS(i, d_Si)
17     update node p with r
18 return r
19 BONUS(offset from mean δ, sample std. dev. σ̂):
20     λ ← δ/σ̂
21     b ← −1 + 2/(1+e^{−kλ})
22 return b
```

Algorithm 1: Pseudo-code of quality-based MCTS (Section 4).

6 EXPERIMENTS

To determine the performance impact of RB and QB, experiments were run on six distinct two-player games. Moreover, RB is evaluated in the General Game Playing (GGP) agent CADIAPLAYER [4], winner of the International GGP competition in 2007, 2008, and 2012.

6.1 Experimental Setup

The proposed enhancements are validated in six distinct two-player games, which are implemented to use a single MCTS engine.

- *Amazons* is played on a 10×10 chessboard. Players each have four Amazons that move as queens in chess. Moves consist of two parts, movement, and shooting an arrow to block a square on the board. The last player to move wins the game.
- *Breakthrough* is played on an 8×8 board. Players start with 16 pawns. The goal is to move one of them to the opponent's side.
- *Cannon* is a chess-like game, where the goal is to checkmate your opponent's immobile town. Players each have one town that is placed at the start of the game, and 15 soldiers. Soldiers can move, and capture single squares forward or retreat two squares if next to an opponent's soldier. Three soldiers in a row form a cannon that can shoot up to three squares in the direction of the line formed.
- *Checkers (English)* is played on an 8×8 board. The player without pieces remaining, or who cannot move, loses the game.
- *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.

The following quality assessments are used for each game. *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 game a fixed normalizer brings the values of q within the $[0, 1]$ range.

Appropriate play-out policies are used to select moves to make during play-out for all games except Checkers, in which moves are selected uniformly random. These policies are implemented to ensure that neither obvious mistakes nor flawed play is observed. All results are reported with these play-out policies enabled. The policies select over a non-uniform move distribution based on the properties of the moves. For Breakthrough, Cannon, and Chinese Checkers decisive moves are always played when available. Moreover, for

Amazons, Breakthrough, and Pentalath MAST [10] with ϵ-greedy selection is applied. MCTS with these play-out policies enabled wins between 57%-99% of games against MCTS with random play-outs.

GGP experiments are performed using the CADIAPLAYER code base. In GGP, no domain knowledge is available in advance, and rules of the games are interpreted online resulting in a low number of simulations. Moreover, play-out policies are learned online, using N-Grams [22]. The Relative Bonus enhancement is tested in the following two-player sequential games: Zhadu, TTCC4, Skirmish, SheepWolf, Quad, Merrils, Knightthrough, Connect5, Checkers, Breakthrough, 3DTicTacToe, and Chinese Checkers. We show results for two simultaneous move games: Battle and Chinook.

Experiments were run on a 2.2Ghz AMD Opteron CPU running 64-bit Linux 2.6.18. For each game, the constant k used by the sigmoid function was determined by experimenting with values between 0 and 10, with varying increments. The C constant, used by UCT (Equation 1) was optimized for each game without any enhancements enabled and was not re-optimized for the experiments.

6.2 Results

For each result, the winning percentage is reported for the player with the enhancement enabled, along with a 95% confidence interval. For each experiment, the players' seats were swapped such that 50% of the games are played as the first player, and 50% as the second, to ensure no first-player or second-player bias.

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** $\pm$ 1.4	**55.7** $\pm$ 1.4	**54.8** $\pm$ 1.4	**54.7** $\pm$ 1.4
Breakthrough	8.0	50.0 $\pm$ 1.4	51.0 $\pm$ 1.4	47.6 $\pm$ 1.4	**51.6** $\pm$ 1.4
Cannon	3.0	**62.8** $\pm$ 1.3	**60.6** $\pm$ 1.3	**58.8** $\pm$ 1.4	**58.1** $\pm$ 1.4
Checkers	2.8	**52.1** $\pm$ 0.8	**52.7** $\pm$ 0.8	48.9 $\pm$ 0.7	**50.7** $\pm$ 0.6
Chin. Checkers	1.2	**56.8** $\pm$ 1.4	**53.2** $\pm$ 1.4	**54.9** $\pm$ 1.4	**52.5** $\pm$ 1.4
Pentalath	1.0	**51.4** $\pm$ 1.4	50.3 $\pm$ 1.4	49.3 $\pm$ 1.4	49.5 $\pm$ 1.4

Results for relative bonus are shown in Table 1. Depending on the search time, significant increases in performance are shown for five of the six games. The value of k used for each game is reported in the second column. We used a fixed value of $a = 0.25$ in addition to the online definition of a' from Equation 10. We see that the online a' leads to increased performance over a fixed value for five games. Interestingly, a fixed value tends to lead to more consistent results over the five games. In Breakthrough, because the play-out policy does not contain heuristics for defensive positioning, the play-outs favor quick wins and excludes defensive moves, leading to bad approximations of the actual game's length. Chinese Checkers, Cannon, and Amazons achieve the most increase in performance using RB. The difference between a simulation's length, and the length of the actual matches played differ the most in these games. As such RB improves the estimates of the simulations' lengths over the course of the match by favoring the shorter ones. These gains are significant at both one second and five seconds of search time. Pentalath is a game with a restricted length. As such, the additional information provided by the length of games is limited.

For GGP, results are presented in Table 2, for all data points at least 950 games were played. A single value for a was used for GGP because a significant number of simulations are required to compute an accurate a' online. Moreover, since values for k cannot 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 approximation can be used as is made clear by the results in Table 2. Note that the results for Chinese Checkers, Checkers, and Breakthrough are similar to those in Table 1, demonstrating the robustness of the enhancement across implementations. In Quad, the variance in the game length is small, which could mean that the overhead imposed by RB is detrimental.

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	**54.3** $\pm$ 1.9	**53.3** $\pm$ 1.9
TTCC4	**55.3** $\pm$ 2.0	**53.3** $\pm$ 2.0
Skirmish	51.9 $\pm$ 2.2	50.7 $\pm$ 2.2
SheepWolf	51.7 $\pm$ 1.8	**52.3** $\pm$ 1.9
Quad	44.7 $\pm$ 1.7	44.7 $\pm$ 1.7
Merrills	51.9 $\pm$ 2.6	48.9 $\pm$ 2.6
Knightthrough	49.9 $\pm$ 2.1	49.2 $\pm$ 2.1
Connect5	**54.0** $\pm$ 1.8	**54.4** $\pm$ 1.8
Checkers	**54.4** $\pm$ 3.0	52.1 $\pm$ 3.2
Breakthrough	51.3 $\pm$ 2.9	51.0 $\pm$ 2.9
3DTicTacToe	**55.0** $\pm$ 1.6	**54.5** $\pm$ 1.6
Chin. Checkers	**56.3** $\pm$ 1.8	**56.0** $\pm$ 1.8
Battle	50.0 $\pm$ 2.0	49.2 $\pm$ 2.0
Chinook	48.5 $\pm$ 2.0	49.0 $\pm$ 2.0

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** $\pm$ 1.3	**58.0** $\pm$ 1.4	**63.0** $\pm$ 1.3	**57.7** $\pm$ 1.4
Breakthrough	2.0	**74.8** $\pm$ 1.2	**71.9** $\pm$ 1.2	**76.3** $\pm$ 1.2	**78.6** $\pm$ 1.1
Cannon	4.0	**65.9** $\pm$ 1.3	**63.0** $\pm$ 1.3	**54.7** $\pm$ 1.4	**57.4** $\pm$ 1.4
Checkers	2.0	**53.8** $\pm$ 0.8	**52.7** $\pm$ 0.7	**51.9** $\pm$ 0.6	**52.3** $\pm$ 0.6
Chin. Checkers	2.8	**65.7** $\pm$ 1.3	**60.1** $\pm$ 1.4	**61.4** $\pm$ 1.3	**58.9** $\pm$ 1.4
Pentalath	1.6	46.6 $\pm$ 1.4	50.5 $\pm$ 1.4	48.7 $\pm$ 1.4	50.1 $\pm$ 1.4

Results for QB are shown in Table 3. A significant increase in performance is presented for five of the six games. The quality assessment in Pentalath is relatively expensive, and the benefit of QB may be diminished by the extra computational effort required. All other games use simple assessments of their terminal states, which do not require much computational effort. Breakthrough and Cannon show the highest overall performance increase, the piece difference is a valuable indicator for the overall quality of the play-outs in these games. As is the case with RB, the results are invariant with respect to the allowed search time.

Note that, while the gains are higher for QB, RB does not require any domain knowledge. An immediate question is whether the two methods are complementary. Results with both methods enabled are shown in Table 4. Comparing to Tables 1 and 3, in 14 of the 24 cases the gain for the combination is statistically significantly higher than each bonus on its own.

Finally, we verified whether replacing the actual reward r by q completely would also have led to a performance increase. The last column of Table 4 reveals a significant performance decrease in five of the six games.

Table 4. Qualitative Bonus and Relative Bonus combined using different search times, 5000 games. Reward = terminal quality, 500 games

Search time	**1 second**		**5 seconds**		**1 second**
Game	a'	$a = 0.25$	a'	$a = 0.25$	$r = q$
Amazons	**65.9** $\pm$ 1.3	**61.9** $\pm$ 1.4	**62.0** $\pm$ 1.4	**60.9** $\pm$ 1.4	23.7 $\pm$ 3.7
Breakthrough	**77.9** $\pm$ 1.2	**72.9** $\pm$ 1.2	**78.6** $\pm$ 1.1	**78.6** $\pm$ 1.1	35.4 $\pm$ 4.2
Cannon	**72.5** $\pm$ 1.2	**69.3** $\pm$ 1.3	**61.1** $\pm$ 1.4	**62.3** $\pm$ 1.3	45.4 $\pm$ 4.4
Checkers	**53.6** $\pm$ 0.8	**54.1** $\pm$ 0.8	**51.7** $\pm$ 0.7	**53.5** $\pm$ 0.7	29.4 $\pm$ 4.0
Chin. Checkers	**69.9** $\pm$ 1.3	**63.1** $\pm$ 1.3	**64.6** $\pm$ 1.3	**70.0** $\pm$ 1.4	48.5 $\pm$ 4.4
Pentalath	50.7 $\pm$ 1.4	51.2 $\pm$ 1.4	**51.9** $\pm$ 1.4	**51.4** $\pm$ 1.4	27.0 $\pm$ 3.9

7 CONCLUSION AND FUTURE RESEARCH

In this paper, we show that the performance of MCTS is improved by treating the rewards of simulations as a combination of the win/loss state, with a quality measure. The enhancements are implemented using control variates, a well-known variance reduction technique. We show performance can be improved when there is a non-zero correlation between the reward-signal and the quality measure. This was true for both bonuses in most of the game domains we used.

The Relative Bonus (RB) treats the length of a simulation as a measure of its quality. The benefit of this method is that it is domain-independent. It seems to perform best in games where there is high variance in play-out lengths, favoring the shorter ones. In General Game Playing, RB improved empirical performance in 7 of the 12 sequential games, and only significantly decreased performance in one of the cases. The Qualitative Bonus (QB) improved results in all (non-GGP) domains, though its implementation requires additional knowledge. Nonetheless, even simple quality assessments of terminal states, such as a piece count, improve results considerably. This type of knowledge could be generated online in a GGP context.

For some domains this is not feasible or practical for play-outs to reach a natural terminal state. Therefore, we propose combining early and static cut-offs of play-outs as future research. Although a static cut-off may not be compatible with RB, we expect both to improve performance in combination with QB. Moreover, combining RB and QB with the reward signal may be improved by computing the a' from the covariance matrix as is standard when combining control variates. Finally, we believe that the proposed enhancements could improve estimates of online learning methods for play-out policies such as N-Grams or MAST.

ACKNOWLEDGEMENTS

This work is partially funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938, and in the framework of the project GoGeneral, with grant number 612.001.121.

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 *LNCS*, pp. 72–83. Springer-Verlag, (2007).

[8] M. Enzenberger, M. Müller, B. Arneson, and R. Segal, 'Fuego: An open-source framework for board games and Go engine based on Monte Carlo tree search', *IEEE Trans. Comput. Intell. AI in Games*, **2**(4), 259–270, (2010).

[9] H. Finnsson, 'Generalized Monte-Carlo tree search extensions for general game playing', in *AAAI*, pp. 1550–1556, (2012).

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

[11] T. Keller and M. Helmert, 'Trial-based heuristic tree search for finite horizon MDPs', in *Int. Conf. on Autom. Plan. and Sched. (ICAPS)*, (2013).

[12] 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 *LNAI*, 282–293, (2006).

[13] A. M. Law and D. W. Kelton, *Simulation modeling and analysis*, McGraw Hill Boston, MA, 2000.

[14] 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).

[15] 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).

[16] R. Ramanujan and B. Selman, 'Trade-Offs in Sampling-Based Adversarial Planning', in *Proc. 21st Int. Conf. Automat. Plan. Sched.*, pp. 202–209, Freiburg, Germany, (2011).

[17] 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).

[18] M. Roschke and N. R. Sturtevant, 'UCT enhancements in Chinese Checkers using an endgame database', in *Computer Games (CGW 2013)*, eds., T. Cazenave, M. H. M. Winands, and H. Iida, volume 408 of *CCIS*, pp. 57–70, (2014).

[19] 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, (2008).

[20] 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).

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

[22] 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).

[23] 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).

[24] 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, (2011).

[25] M. H. M. Winands, Y. Björnsson, and J-T. Saito, 'Monte Carlo Tree Search in Lines of Action', *IEEE Trans. Comp. Intell. AI Games*, **2**(4), 239–250, (2010).

[26] F. Xie and Z. Liu, 'Backpropagation modification in monte-carlo game tree search', in *Proc. Int. Symp. Intell. Inform. Tech. Applicat.*, volume 2, pp. 125–128, (2009).