# Quality-based Rewards for Monte-Carlo Tree Search Simulations

**Tom Pepels** and **Marc Lanctot** and **Mark H. M. Winands** [1]

**Abstract.** Monte-Carlo Tree Search is a best first search method based on sampling the state space of a given 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 in Monte-Carlo Tree Search based on their assessed quality improves results in five different two-player games. To achieve these results we propose two enhancements, the *Qualitative Bonus* and the *Relative Bonus*. The former relies on a domain-dependent assessment of the game's terminal state, whereas and the latter relies on the total number of moves made during a simulation. The proposed enhancements lead to a considerable performance increase in all five domains discussed.

## 1 Introduction

Monte-Carlo Tree Search (MCTS) is a best-first search method, based on random sampling of a given domain [7, 5]. MCTS depends on the results of numerous simulations and grows a search tree online. Each of these consist of two parts, 1) the selection step, where moves are selected and played according to the a selection policy, and 2) the play-out step, where moves are played according to a simulation strategy. 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 [4]. In this paper, two methods are proposed for determining the quality of a simulation based on the play-out sampled. The first method proposed assesses the quality of a simulation based on its length $m_{ST}$. In the second method, we propose a heuristic assessment of the terminal state $q$ to determine the simulation's quality. Moreover, we show that adjusting $r$ in a specific way using $m_{ST}$ and/or $q$ leads to increased performance in five different two-player games.

Other techniques for rewarding simulations have been proposed [15], 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 [11]. 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, using the straightforward discrete reward $r$, any information other than the win/loss/draw 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 the terminal state.

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

## 2 Monte-Carlo Tree Search

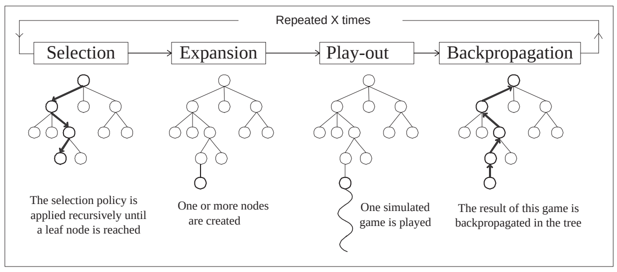
Repeated X times
Selection
Expansion
Play-out
Backpropagation
The selection policy is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

**Figure 1.** Strategic steps of Monte-Carlo Tree Search [4].

Monte-Carlo Tree Search (MCTS) is a best-first search method based on random sampling of the state space for a specified domain [7],[5]. 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 [10], Lines of Action [17], and Hex [1]. Moreover, MCTS has recently seen successes in other domains such as real-rime strategy games [3], real-time games such as Ms Pac-Man [8] and the Physical Travelling Salesman problem [9], but aslo in non-game domains such as optimization, scheduling and security [3].

In MCTS, a tree is built incrementally over time and maintains statistics at each node corresponding the rewards collected at those nodes and number of times the nodes have been visited. The root of this tree corresponds to the agent's current position. The basic version of MCTS consists of four steps, which are performed iteratively

[1] Maastricht University, Department of Knowledge Engineering, Maastricht, The Netherlands, email: {tom.pepels,marc.lanctot,m.winands}@maastrichtuniversity.nl

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 [4]:

- Selection. Starting at the root node, children are chosen according to a selection policy (described in Subsection 2.1). When a leaf node 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 strategy until a terminal state is reached.
- Backpropagation. The result of the simulated play-out is propagated immediately from the selected node back up to the root node. Statistics are updated along the tree for each node selected during the selection step and visit counts are increased.

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 rewards and visits 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) [7] 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 $C(p)$, then the policy determining which child $i$ to select:

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

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, the length of a simulation is detailed as a quality measure, second some heuristic assessment of terminal states is considered. In the next section we establish how these techniques can be used to enhance the rewards of MCTS simulations.

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

1. The number of nodes between the root node $S$ to the expanded leaf $N$, $d_{SN}$,
2. The number of moves required to reach $T$, the terminal state, from $N$ during play-out $d_{NT}$.

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

$$m_{ST} = d_{SN} + d_{NT} \tag{2}$$

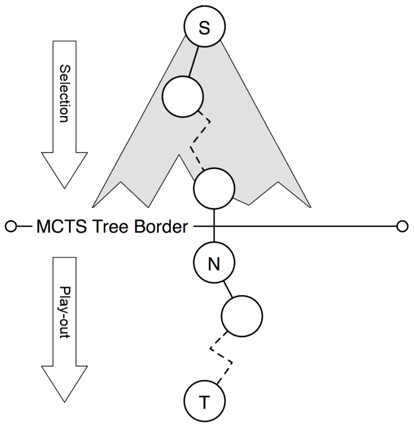

**Figure 2.** A single MCTS simulation [6].

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 selected during play-out are generated 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 $\epsilon$-greedy selection [13, 12]. Various alternative methods have been proposed, such as using low-level $\alpha\beta$ searches [16], and methods that learn a strategy online, such as N-Grams and the Last-Good-Reply policy [14], or the Move-average Sampling Technique (MAST) [6]. However, each of these methods and strategies have some random element in common. Moreover, they select moves based on some imperfect technique, because they have to make quick decisions to allow for numerous simulations to be made during the allowed time. As such, each move played ultimately increases uncertainty with respect to the accuracy of the final result, by some degree. Hence, the length of the simulation may be regarded as an indicator of the accuracy of its result.

The main benefit of using simulation length as a quality measure is that it is domain independent. Unless the game's length is fixed, the variance of the length of a play-out in particular can be informative in determining its quality.

The second measure of a simulation's quality is based on a heuristic assessment of the game's terminal state. Although evaluation functions can be designed for most games, they're used to evaluate non-terminal positions and assign them a specific value. However, MCTS generally performs a play-out until a terminal state is reached. Therefore, we are interested in assessing the terminal state of a game rather than intermediary positions. This limits the number of features that can be evaluated, yet provides a direct application to MCTS.

As before, consider a single MCTS simulation as depicted in Figure 2. When the terminal state $T$ is reached, an quality assessment function is called to evaluate the position with respect to the winning player. This measure; $q \in (0, 1)$ should reflect the quality of the 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 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 game progress from $S$ to $T$ the score is based on an evaluation of $T$ given the progression $S \ldots N \ldots T$.

## 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* and the *Qualitative Bonus*, relating to the length of the simulation and the heuristic assessment of the terminal state, respectively.

In our proposed framework, rewards of MCTS simulations return a tuple of three reward values, $\langle r, w, q, d_{NT} \rangle$ representing the reward $r \in \{-1, 0, 1\}$, the winning player $w$, the heuristic assessment of the terminal state $q$, and the distance from the expanded node $N$ to the terminal state $T$: $d_{NT}$, respectively. $m_{ST}$ is then computed as shown in Equation 2. First we will define the Relative Bonus (RB) based on $m_{ST}$ information, after which, the Qualitative Bonus (QB) is described which is similar, except for being based on $q$.

### 4.1 Relative Bonus

First, note that $m_{ST}$ depends considerably on both the domain in question and the progress of the game. By itself, the variable is neither normalized, nor relative to a central tendency. As such, adding the value to $r$ as is, leads to a biased distribution of the value over time, where, at the beginning of a game, $m_{ST}$ takes on higher values than when the game approaches its terminal state. 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, two normalizers need to be approximated: 1) a central measure, to relate the desired variable to the current progress, and 2) a normalizing distance to the central measure, since the variance of observed $m_{ST}$ values may also differ over time.

A central measure can be approximated online, by maintaining an average $\bar{M}^w$, for each player (indexed by $w$), of the observed $m_{ST}$ values so far. After each simulation ends, $m_{ST}$ is compared to $\bar{M}^w$, and $\bar{M}^w$ is updated with the observed $m_{ST}$.

$$\delta_m = \bar{M}^w - m_{ST} \tag{3}$$

This ensures $\delta_m$ is relative to the current progress of the game, and independent of the domain's tendency regarding the length of games. Moreover we achieve the desired effect of valuating simulations shorter than average with a positive value and those longer with a negative value. However, $\delta_m$ still depends on the progress of the game. In most domains, the start of any game will have a high variance with respect to the length of simulations. Insufficient prior moves have been made, and MCTS can not yet accurately predict the progression of the game. As a consequence, play-outs will last longer at the start of the game than when it reaches the terminal state. As such the variance of the observed $m_{ST}$ values over time may differ substantially. Which leads to the requirement of a normalizer.

Assuming that the distribution of $m_{ST}$ values over time is $M \sim \mathcal{N}(\bar{M}, \hat{\sigma}_m)$, then $\hat{\sigma}^w_m$ is the sample standard deviation which can be used to normalize $\delta_m$ with respect to the current variance of the length of simulated games. Now we can define a normalized variable $\lambda$ as follows:

$$\lambda = \frac{\delta_m}{\hat{\sigma}^w_m} \tag{4}$$

$\lambda$ is both normalized with respect to the sample standard deviation of $M$, and relative to $\bar{M}^w$. This means $\lambda$ wil be both independent of the progress of the game, and normalized with respect to the current variance in the length of simulations.

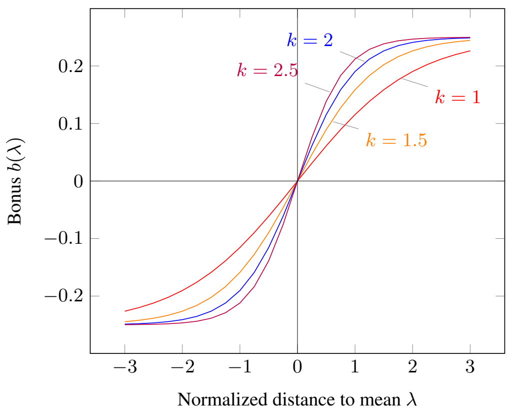

**Figure 3.** Bonusses resulting from different $k$ constants in Equation 5.

Since the distribution of $M$ is not known, $\lambda$ can still take on unrestricted values, particularly if $M \sim \mathcal{N}(\bar{M}, \hat{\sigma}_m)$ is close to uniform. Moreover, the relation with the desired reward is not neccesarily linear. As such, in order to both bound, and shape the values of the relative bonus $b_r$ we propose to use a sigmoid function centered around 0 on both axes, and with a range of $[-0.25, 0.25]$.

$$b(\lambda) = -0.25 + \frac{0.5}{1 + e^{-k\lambda}} \tag{5}$$

Figure 3 shows the graphs for three different values of $k$ in Equation 5. $k$ is a constant to be determined empirically, which shapes the bonus to be added to $r$. Higher values of $k$ determine both the slope, and the start of the horizontal asymptotes of the sigmoid function.

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

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

This value is backpropagated from the expanded leaf to the root node. The range of $r_b$ is now $[-1.25, 1.25]$, i.e. the bonus $b(\lambda)$ is centered around the possible values of $r$.

### 4.2 Qualitative Bonus

Calculation of the Qualitative Bonus follows the same procedure as the Relative Bonus. Similar to RB, a distribution over observed $q$ values is maintained $Q \sim \mathcal{N}(\bar{Q}, \hat{\sigma}_q)$ for each player $w$. The value of $q$ is determined by a domain dependent assessment of quality of the terminal state. Assuming that higher values of $q$ represent a higher quality terminal state for the winning player $w$, then $\delta_q$ becomes:

$$\delta_q = q - \bar{Q}^w \tag{7}$$

Where $\bar{Q}^w$ is updated with $q$ after every simulation. Consequently, $\lambda$ is redefined as:

$$\lambda = \frac{\delta_q}{\hat{\sigma}^w_q} \tag{8}$$

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

## 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, ap-

```
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          $\langle r, w, q, d_{iT} \rangle \leftarrow$ Playout($i$)
8          $m_{ST} \leftarrow d_{Si} + d_{iT}$
9          if enabled($b_r$) and $\hat{\sigma}^w_m > 0$ then
10             $r \leftarrow r + \text{sgn}(r) \times$ BONUS($\bar{M}^w - m_{ST}, \hat{\sigma}^w_m$)
11             update $\bar{M}^w$ and $\hat{\sigma}^w_m$ with $m_{ST}$
12         if enabled($b_q$) and $\hat{\sigma}^w_q > 0$ then
13             $r \leftarrow r + \text{sgn}(r) \times$ BONUS($q - \bar{Q}^w, \hat{\sigma}^w_q$)
14             update $\bar{Q}^w$ and $\hat{\sigma}^w_q$ with $q$
15         update node $i$ with $r$
16     else
17         $r$ = -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     $b \leftarrow {}^{0.5}/_{(1+\exp(-K\lambda))} - 0.25$
24 return $b$
```

**Algorithm 1:** Pseudo-code of the MCTS and BONUS functions

propriate for two player games. Whenever and *update* is used in the algoritm, it refers to updating the average reward for a node, or the sample mean and standard deviation for $\bar{M}^w$ and $\bar{Q}^w$. 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 4 different values are returned: 1) the result of the play-out $r \in \{-1, 0, 1\}$, 2) the winning player $w$, 3) the assessed quality of the play-out's terminal state $q \in (0, 1)$ defined in Subsection 4, and 4) the number of moves made during play-out $d_{iT}$ defined in Subsection 4. Using these values $r$ is altered. On line 10 the relative bonus is applied to $r$, using the difference with the winning player's current mean $\bar{M}^w - m_{ST}$, i.e. lower values of $m_{ST}$ give a higher reward. After which the current mean and standard deviation are updated on line 11. QB is applied on line 13 using the assessed quality of the play-out $q$. Note that the distance to the mean is defined as $q - \bar{Q}^w$, because in contrast to RB, higher values of $q$ imply better results. The BONUS function on line 20, computes the normalized $\lambda$ (line **??**) and, successively the bonus $b$ (line 23) using the sigmoid function, as defined in Subsections 4.1 and 4.2.

## 6 Experiments

To determine the impact on performance of RB and QB, experiments were run on 5 different two player games.

### 6.1 Experimental setup

The proposed enhancements were tested in several two player games.

- **Amazons**
- **Breakthrough**
- **Cannon**
- **Chinese Checkers**
- **Pentalath**

Each game uses a play-out strategy including a re

### 6.2 Results

- Results UCT vs R - Results UCT vs QB - Results UCT vs RB + RB - Results RB vs QB - Graph of K's / UCT C's

## 7 Conclusion

Relative bonus - interesting because requires no domain knowledge, works best in games with long play-outs. QB works in all domains, but requires domain knowledge, nonetheless, even using simple evaluation of the terminal state improved results considerably.

RB especially interesting for General Game Playing (GGP), where knowledge of games is sparse. RB improves results without domain-dependent knowledge.

Would be interesting to determine if RB/QB could improve results in non-game domains.

## Acknowledgments

This work is partially funded by Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938.

## 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] 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).

[4] 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 Mathematics and Natural Computation*, **4**(3), 343–357, (2008).

[5] R. Coulom, 'Efficient selectivity and backup operators in Monte-Carlo tree search', in *Computers and Games (CG 2006)*, 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.

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

[7] L. Kocsis and C. Szepesvári, 'Bandit Based Monte-Carlo Planning', in *Machine Learning: ECML 2006*, eds., J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, volume 4212 of *Lecture Notes in Artificial Intelligence*, 282–293, (2006).

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

[9] 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 *Comput. Intelligence and Games (CIG)*, pp. 234–241. IEEE, (2012).

[10] Arpad Rimmel, Olivier Teytaud, Chang-Shing Lee, Shi-Jim Yen, Mei-Hui Wang, and Shang-Rong Tsai, 'Current frontiers in computer Go', *IEEE Trans. Comput. Intell. AI in Games*, **2**(4), 229–238, (2010).

[11] K Shibahara and Y Kotani, 'Combining final score with winning percentage by sigmoid function in monte-carlo simulations', in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pp. 183–190. IEEE, (2008).

[12] N. R Sturtevant, 'An analysis of UCT in multi-player games', in *Computers and Games*, eds., H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, volume 5131 of *LNCS*, 37–49, Springer, (2008).

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

[14] Mandy J. W. Tak, Mark H. M. Winands, and Yngvi 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).

[15] M H. M. Winands and Y Björnsson, 'Evaluation Function Based Monte-Carlo LOA', in *Proc. Adv. Comput. Games, LNCS 6048*, pp. 33–44, Pamplona, Spain, (2010).

[16] 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).

[17] 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).