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 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 based on their assessed quality improves results in five distinct two-player games. To achieve these results we propose two enhancements, the *Relative Bonus* and *Qualitative Bonus*. Both are used as control variates, and based on variance reduction methods. 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 all five domains discussed.

1 INTRODUCTION

Monte-Carlo Tree Search (MCTS) is a search method based on random sampling of a domain [6, 10]. 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 according to the a selection policy, and 2) the play-out step, where moves are played according to a simulation policy. 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 [5]. In this paper, two techniques are proposed for determining the quality of a simulation based on the play-out sampled. The first enhancement proposed, assesses the quality of a simulation based on its length d. The second enhancement considers a quality assessment of the terminal state q. Moreover, we show that adjusting r in a specific way using d or q leads to increased performance in five distinct two-player games.

Other techniques for rewarding simulations have been proposed [20], 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 [15]. 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 in Section 2. Next, in Section 3, two different techniques for assessing the quality of play-outs are detailed. 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 enhancements is determined in Section 6, accompanied by a discussion and conclusion.

2 MONTE-CARLO TREE SEARCH

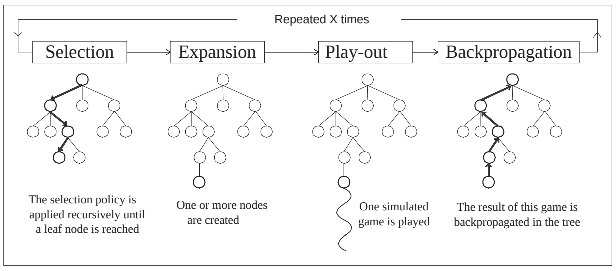
Repeated X times
Selection
Expansion
Play-out
Backpropagation
The selection policy is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 1. Strategic steps of Monte-Carlo Tree Search [5].

Monte-Carlo Tree Search (MCTS) is a best-first search technique based on random sampling of the state space for a specified domain [6, 10]. 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 [13], Lines of Action [21], and Hex [1]. Moreover, MCTS has recently seen successes in other domains such as real-time strategy games [4], arcade games such as Ms Pac-Man [11] and the Physical Travelling Salesman problem [12], but also in real-life domains such as optimization, scheduling and security [4].

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 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 [5]:

- **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 policy 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) [10] 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:

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

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, a 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 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:

$$d = d_{SN} + d_{NT} \tag{2}$$

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.

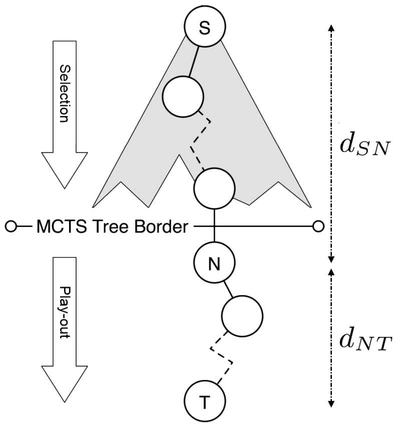

Figure 2. A single MCTS simulation [8].

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 ϵ-greedy selection [16, 17]. Various alternative methods have been proposed, such as using low-level $\alpha\beta$ searches [20], and methods that learn a strategy online, such as N-Grams and the Last-Good-Reply policy [18], or the Move-average Sampling Technique (MAST) [7]. However, each of these methods and strategies have some random element in common. Moreover, they select moves that are far from optimal, 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. Moreover, in certain games such as Chinese Checkers, simulation length has been considered part of the evaluation function [14].

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 states 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 states. This leaves potentially less informative features to be evaluated, yet provides a direct application to MCTS.

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 $q \in (0, 1)$ 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 $\mathsf{S} \ldots \mathsf{N} \ldots \mathsf{T}$ (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, and the distance from the expanded node N to the terminal state T, d_{NT}, respectively. d is then computed as shown in Equation 2. Except for q, these values are readily available without requiring extra computational effort.

This section discusses the mathematical basis for altering MCTS rewards. Control variates as a means of variance reduction are discussed and how they can be used to improve MCTS' performance in games. In the second subsection the Relative Bonus is defined, based on the value of d. And the following subsection 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 a, a constant used in the propsed methods.

4.1 Control Variates

Variance reduction methods in mathematical simulation are used to improve estimates by reducing the variance in a simulation's output [9]. Recently, using variance reduction techniques for MCTS has been proposed by Veness et. al. [19]. 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 $\mathrm{E}(X)$ given that the mean $v = \mathrm{E}(Y)$ 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 $Z := X + a(Y - v)$. For a, one can derive an optimal constant $a^* = -\mathrm{Cov}(X, Y) \,/\, \mathrm{Var}(Y)$ such that the reduction in variance is optimal.

If we define X as the simulation output, i.e. $X_n = r$, and define Y as as one of the quality measures discussed in Section 3, $Y_n = d$ or $Y_n = q$. Then assuming that X and Y are correlated, i.e. $\mathrm{Corr}(X, Y) \neq 0$, we can find an optimal a^* such that variance in the reward will be reduced. In common practical domains, no fixed values for v, $\mathrm{Cov}(X, Y)$, or $\mathrm{Var}(Y)$ 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 optimal 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. It is possible that a larger performance increase is gained by using a non-optimal value for a, as our quality measures may provide more advantage than variance reduction alone. Therefore, althoug we propose to define quality-based rewards as control variates, we will not be concerned with the actual reduction in variance, but rather the improvement in performance.

4.2 Relative Bonus

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. 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. Using these statistics, we can define a standardized value λ_r as follows:

$$\lambda_r = \frac{\bar{D}^{\tau} - d}{\hat{\sigma}_D^{\tau}} \tag{3}$$

Now, 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. The expectation of λ_r values will be 0 due to standardization, hence λ_r can be added to r to be treated as a control variate. 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 faster games. Although there is no immediate sollution to this problem, we propose to reset $\bar{D}^{\tau}$ and $\hat{\sigma}_D^{\tau}$ in between moves. Moreover, rewards of the first 5% of the expected number of simulations are not altered during search, yet $\bar{D}^{\tau}$ and $\hat{\sigma}_D^{\tau}$ are updated during this time without introducing bias.

Since the distribution of D is not known, λ_r can still take on unrestricted values, particularly if the distribution of D 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. The range of the sigmoid is $[-a, a]$. Where a can be an empirically optimized value, or computed off or on-line.

$$b(\lambda) = a\left(-1 + \frac{2}{1 + e^{-k\lambda}}\right) \tag{4}$$

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.

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

$$r_b = r + \mathrm{sgn}(r) \times b(\lambda_r) \tag{5}$$

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 $b(\lambda)$ is centered around the possible values of r.

4.3 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 for each player τ. The value of q is determined by a domain dependent assessment of the quality of the terminal state. Assuming that higher values of q represent a higher quality terminal state for the winning player τ, then δ_q is: Where $\bar{Q}^{\tau}$ is updated with q after every simulation. Consequently, λ is redefined as:

$$\lambda_q = \frac{q - \bar{Q}^\tau}{\hat{\sigma}^\tau_Q} \tag{6}$$

Finally the bonus $b(\lambda_q)$ is computed using the sigmoid function in equation 4 with an optimized k constant, and summed with the result of the simulation r.

$$r_q = r + \operatorname{sgn}(r) \times b(\lambda_q) \tag{7}$$

4.4 Estimating a

In gameplay, X is a nominal variable, i.e. win, draw or loss, and Y is a discrete scalar. Therefore the method of approximating a^* by determining $-\operatorname{Cov}(X, Y) / \operatorname{Var}(Y)$ is not straightforward and 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 wrong or suboptimal. 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. higher values at the start due to more random results.

The scaling constant for the defined control variates 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 $X_n = r$, if r is in respect of one player, and $Y_n = r$ or $Y_n = q$. We can compute $a^* = -\widehat{\operatorname{Cov}(X, Y)} \,/\, \widehat{\operatorname{Var}(Y)}$, i.e. using the sample covariance and variance, online during search, or offline. Regrettably, any effort to determine a value for a based on the preceding method has been fruitless. Due to the small covariance measured the resuling range of a is too small to make an impact on performance.

Nonetheless, a value for a can be computed and used online by altering the definition of X. First, let Y be one of the proposed quality measured, i.e. $Y_n = r$ or $Y_n = q$. Let ρ be the search player, i.e. the player running MCTS. Now, define X' as:

$$X'_n = \begin{cases} Y_n & \text{if } \rho \text{ wins the play-out,} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

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 start of the play-out. 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 $\bar{D}^\tau$ and $\bar{Q}^\tau$. 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 $q \in (0, 1)$ 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 $\bar{D}^\tau - d$, 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 13 using the assessed quality of the play-out q. Note that the distance to the mean is defined as $q - \bar{Q}^\tau$,

```
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             $r \leftarrow r + \operatorname{sgn}(r) \times$ 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             $r \leftarrow r + \operatorname{sgn}(r) \times$ 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     $b \leftarrow a\left(-1 + 2/1+e^{-k\lambda}\right)$
24 return $b$
```

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

because in contrast to RB, higher values of q 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 can be either fixed, or computed online.

6 EXPERIMENTS

To determine the impact on performance of RB and QB, experiments were run on five different two player games. Moreover, the performance of RB is evaluated in the General Gameplaying agent CADIAPLAYER [3], which won the International GGP competition in 2007 and 2008.

6.1 Experimental setup

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

- *Amazons* is played on a 10 x 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 x 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 x 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 hexagonical board. The goal is to place 5 pieces in a row. Pieces can be captured by fully surrounding an opponents 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.

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.

ACKNOWLEDGEMENTS

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] 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).

[4] 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).

[5] 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).

[6] 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.

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

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

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

[10] 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).

[11] 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).

[12] 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).

[13] 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).

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

[15] 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).

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

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

[18] 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).

[19] 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).

[20] 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).

[21] 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).