Quality-based rewards for Monte-Carlo Tree Search simulations

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

Abstract. In this paper methods for assessing the a posteriori quality of Monte-Carlo simulations are introduced. 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 2-player boardgames. To achieve these results we propose two novel enhancements, the *Qualitative Bonus* and the *Relative Bonus*. The former relies on an uncomplicated heuristic evaluation of the game's terminal state, whereas and the latter relies on the total number of moves made during any simulated play-out. The proposed enhancements lead to a considerable performance increase in all five domains discussed.

1 Introduction

2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a best-first search method based on random sampling of the state space for a specified domain [6],[4]. 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 board games games such as Go [7] and Hex [1].

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 [3]:

- **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 heuristic 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 heuristic state evaluation, nonetheless, in most cases it is beneficial to add some domain knowledge for selecting moves to play during play-out.

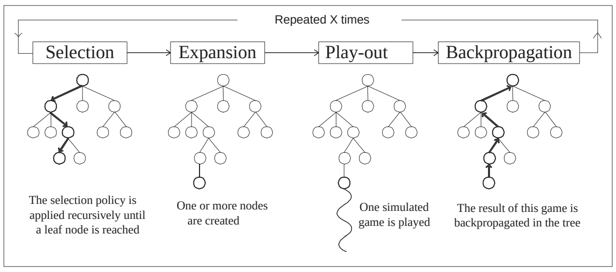
Repeated X times
Selection
Expansion
Play-out
Backpropagation
The selection policy is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 1. Strategic steps of Monte-Carlo Tree Search [3].

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) [6] 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. The policy that determines which child to select given the current node is the one that maximizes the following equation:

$$X_i = v_i + C\sqrt{\frac{\ln n_p}{n_i}} \tag{1}$$

v_i is the score of the current child based on the average result of simulations that visited it. In the second term, 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 play-out quality

In the MCTS framework, play-outs are run by finishing a single game according to a simulation strategy until a natural terminal state is reached. The result r is expressed numerically in some discrete range, e.g. $r \in [-1, 0, 1]$ a loss, draw or win, respectively, and back-propagated along the tree from the expanded leaf to the root node. Other techniques have been proposed [8], 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 score

[1] Maastricht University, The Netherlands, email: tom.pepels@maastrichtuniversity.nl

based games. However, for some domains a strong 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, all information regarding the quality of the play-out's final position is lost. For these reasons, we propose assessing the rewards of play-outs based on information other than the loss, draw, or win state of the final position.

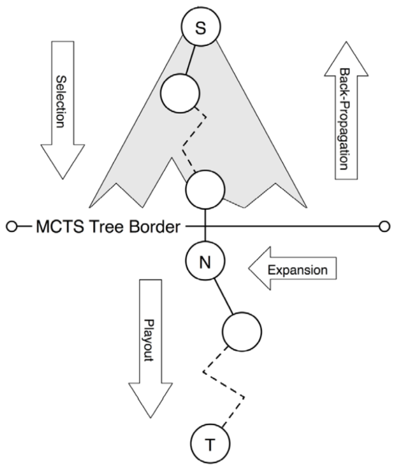

Figure 2. A single MCTS simulation [5].

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

1. the number of moves from the root 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 p_T.

The length of the simulation is defined as the sum of these distances $m_{ST} = d_{SN} + p_{NT}$, i.e. the total number of moves made by both players to reach the terminal state of the game from the current gamestate.

Considering that, in Monte-Carlo methods a random element is present in simulations, each move played increases the uncertainty of the final result. The most important benefit of using this information is that it can be obtained independent of the domain. Unless the game has a fixed-length, the variance of the length of a play-out can be informative in determining its quality.

```
MCTS(node $p$, cumulative node depth $d_{S,p}$):
    if isLeaf(p) then
        Expand($p$)
    Select a child $c_i$ that maximizes $X_i$ from Eq. 1
    if $n_i = 0$ then
        $\{q_t, w_t, m_t, r_t\} \leftarrow$ Playout($c_i$)
        if enabled($rb$) then
            $r_t \leftarrow r_t + \operatorname{sgn} r * BONUS(\bar{M} - m_t, s_m)$
            Update $\bar{M}$ and $s_m$ with $m_t$
        if enabled($qb$) then
            $r_t \leftarrow r_t + \operatorname{sgn} r * BONUS(\bar{Q} - q_t, s_q)$
            Update $\bar{Q}$ and $s_q$ with $q_t$
    else
        $r_t$ = -MCTS($c_i$, $d_{S,c_i}$))
    return $r_t$

BONUS(distance to mean $\delta$, sample std.dev $s$):
    $\lambda \leftarrow \frac{\delta}{s_m}$
    $b \leftarrow \frac{0.5}{1+\exp -K\lambda} - 0.25$
    return $b$
```

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

4 Quality-based play-out rewards

4.1 Relative Bonus

4.2 Qualitative Bonus

5 Algorithm

6 Experiments

6.1 Experimental setup

6.2 Results

7 Conclusion

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] 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).

[4] 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.

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

[6] 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).

[7] 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).

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