چهارمین کنفرانس بین المللی
مطالعات نوین در علوم کامپیوتر و فناوری اطلاعات
4th International Conference
On New studies Of Computer And IT
FEBRUARY 24, 2018
مشهد
۰۵ اسفند ماه ۱۳۹۶

IMPROVING MONTE CARLO TREE SEARCH BY COMBINING RAVE AND QUALITY-BASED REWARDS ALGORITHMS

Masoud Masoumi Moghadam[1], Mohammad Pourmahmood Aghababa[2], Jamshid Bagherzadeh[3]

1-M.Sc Student, Urmia University of Technology

2-Associate Professor, Urmia University of Technology

3-Associate Professor, Urmia University

Abstract

Monte-Carlo Tree Search is a state-of-the-art method for building intelligent agents in games and has been focus of many researchs through past decade. By using this method, the agents are able to master the games through building a search tree based on samples gathered by randomized simulations. In most of the researchs, the reward from simulations are discrete values representing final state of the games (win, loss, draw), e.g., $r \in \{-1, 0, 1\}$. In this paper, we introduce a method which modifies reward for each playout. Then it backpropagates the reward through UCT and AMAF values. RAVE algorithm is used to evaluate the nodes more accurately in each tree breadth. We implemented the algorithm along with Last-Good-Reply, Decisive-move and Poolrave heuristics. In the end we used leaf parallelization in order to increase the samples gathered by simulations. All implementations are examined in the game of HEX in 9×9 board. We show the proposed method can improve the performance in the domain discussed.

Keywords: Monte-Carlo Tree Search, Game of HEX, Reinforcement Learning, Heuristics.

I. Introduction

Monte-Carlo-Tree-Search (MCTS) is a best-first search technique which is used for the domains which could be represented by sequential decision-making. Recently, MCTS has been really successful in the field of computer board games like GO [1], HEX [2] and Chinese Checkers [3]. Board games like GO, HEX and Checkers are classic board games which have large branching factor and it causes the automated tree search agents like alpha-beta search not to be effective due to shallow search.

In plain MCTS, The automated agent start selecting node until it reaches a leaf node. Then it runs simulation from leaf node until it reaches a terminal state which denotes the final game position. A reward signal is backpropagated through the leaf node to its ancestors. Generally the reward signal is a discrete number which is 0 for draw, 1 for win and -1 for loss. This reward signal is the only outcome used for backpropagation and any other information is not considered. In [4], a method called "score bonus" is used to distinguish between weak wins and strong wins. In this method, reward backpropagate a value in the interval $[0, \gamma]$ for a loss and $[\gamma, 1]$ for a win with the strongest win scoring 1 and the weakest win scoring γ. This method did not improve the playing strength.

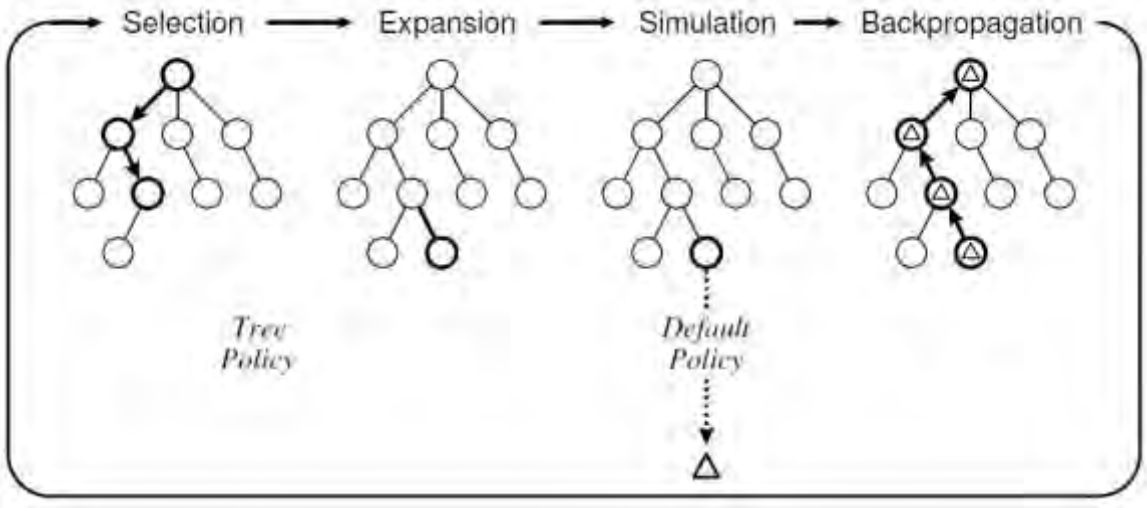
In [5], another attempt is done by Kocsis et al in which "decaying reward" method was used to weight early wins more heavily than later wins. This modification is implemented to the backpropagation process in which the reward value is multiplied by some constant $0 < \gamma \leq 1$.

Previous studies show lack of performance in distinguishing between weak wins and strong wins through modifying reward. In this paper we propose a method which considers playout length as a quality measure. Then we use this quality measure to modify the rewards and bias the search tree to more promising nodes for better performance. Then we combine this strategy with reinforcing playout policies namely as Rapid Action Value Estimation [6], Last-Good-Reply [7], Decisive move strategy [8] and Poolrave [9].

II. Background

1. Monte Carlo Tree Search (MCTS)

MCTS is a method which builds a tree search over samples gathered by simulations from the observed states of environment. Generally it consists of 4 steps. At first agent uses a *tree policy* to find the most urgent node in each tree breadth from root node until it reaches a leaf node (selection step in Fig 1). In second step, agent expanses the tree by adding the node pointing to the state after the leaf (Expansion step in Fig 1). In third step, To Estimate value for the newly expanded node, It runs simulation from expanded node to a terminal state (Simulation step in Fig 1). Terminal state



is a state in which the game ends in draw, win or loss. The result would be backpropagated from expanded node to its ancestors in the 4th step (Backpropagation part in Fig 1).
In selection step, A tree policy is required to search effeciently between the nodes and lead the tree to more promising nodes. In this way, it needs to balance the exploration (areas where no statistics are availbale yet) and exploitation (areas which look promising according to statistics available).

Figure 1. The whole process of MCTS

UCT algorithm [5] which is derived from UCB1 algorithm [10] in the field of multi-armed bandits, is the basic algorithm which can find the best possible answer each time. It evaluates each node with this:

$$UCT = \bar{X}_j + 2C_p\sqrt{\frac{2\,ln\,n}{n_j}} \qquad (1)$$

In this formula, n is the number of times that current (parent) node is visited and n_j is the number of times that child node is visited. $\bar{X}_j$ is the division of cumulated rewards into n_j and it is understood to be within [0,1]. C_p is the "exploration constant" and is considered within [0,1]. Bigger values of C_p encourages the exploration and as it gets less, it would encourage the exploitation. In this formula X_j is the exploitation term and $\sqrt{\frac{2\,ln\,n}{n_j}}$ is the exploration term.

2. Last-Good-Reply Strategy (LGR)

In this strategy each move is considered a reply against opponent move. So after each playout, the opponent moves are collected and for each opponent move, its reply in playout is considered. In the next playouts, If the move selected by opponent exists in the list, then its assigned reply is used against in the playout. If the playout ended in loss, then the reply is removed from the list. The replies which are theoritically good replies will last longer in the playouts and get more rewards [11].

3. Rapid Action Value Estimation (RAVE)

The RAVE selection strategy has been proposed in order to speed up the learning process inside the MCTS tree [6]. The UCT algorithm bases the selection of a move in a node on the estimated value obtained by sampling this move in the node multiple times. However, especially when the state space is large, the algorithm needs many simulations before it can sample all the moves in a node and more simulations before it can accumulate enough samples for the moves to reduce the variance of their estimated scores. To have this deficiencies obviated, RAVE keeps track of other statistics, also known as All Moves As First (AMAF) values [13]. In each node, two values are memorized:

- The average result Q(s, a), obtained from all the simulations in which move a is performed in state s (the same value used in Formula (1)).
- The average result AMAF(s, a), obtained from all the simulations in which move a is performed further down the path that passes by node s.

The evaluation formula for each node is:

$$\big(1-\beta(s)\big)\times Q(s.a)+\beta(s)\times AMAF(s.a) \qquad (6)$$

And the term $\beta(s)$ is calculated as follows:

$$\beta(s) = \sqrt{\frac{K}{3\times N(s)+K}} \qquad (7)$$

where N(s) is the number of times node s has been visited and K is the equivalence parameter, that indicates for how many simulations the two scores are weighted equal.

4. Decisive-Move strategy

In [8], Teytaud et al have proposed three algorithms for boosting the MCTS in the game of Havannah. Decisive moves are the moves which the player would win the game after choosing those moves. Anti-Decisive moves are the moves which has to be played before the opponent choose those moves because those are opponent decisve moves. The decisive moves is proved that increases the perfomance.

5. Poolrave strategy

Hook et al [9], proposed an enhancement to the RAVE algorithm which would be as follows:

- Build a pool of *k* best moves according to RAVE values
- Choose one move *m* from the pool.
- With probability *p* play move m, and with probability *(1-p)* play random move.

The good thing about the poolrave is that it's independent of domain.

III. Proposed method

The proposed method is based on combining RAVE and quality-based rewards. In this way, we explain these two strategy in the folloing.

1. Quality-based rewards

In [12], It is recommended to use continuous rewarding system instead of using discrete rewards like {-1,0,1} for loss,draw and win in MCTS. In this paper, Two quality measures are used to modify the rewards after each playout which are based on terminal state quality and simulation length. If we consider a single MCTS simulation as Fig 1, then two separate distances are defined:

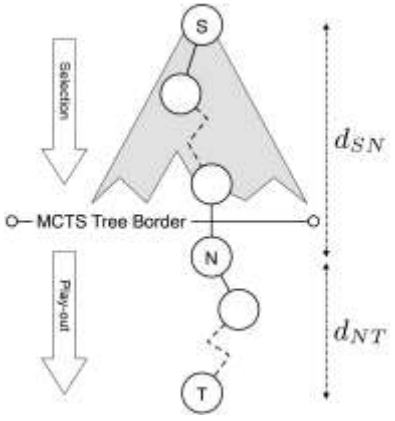


Figure 2. single MCTS simulation [11]

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 simulation is defined as the sum of these distances:

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

The resulted d is stored in a list and after each simulation the list is updated accordingly. In the backpropagation part of MCTS, we calculate λ as below:

$$\lambda = \frac{\bar{D}^{\tau} - d}{\sigma_D^{\tau}} \quad (3)$$

In which $\bar{D}^{\tau}$ is the sample mean over the distribution of observed d for player τ, σ_D^{τ} is sample standard deviation of distribution $\bar{D}^{\tau}$. 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],

$$b(\lambda) = -1 + \frac{2}{1 + e^{-k\lambda}} \quad (4)$$

Here, k is a constant to be determined by experimentation, it both slopes and bounds the bonus to be added to r.
Finally, the modified reward with the relative bonus is,

$$r_b = r + sgn(r) \cdot b \cdot b(\lambda) \quad (5)$$

This value is backpropagated from the expanded leaf to the root node. The range of r_b is [-1+a, 1+a], the bonus r_b is centered around the possible values of r.

3. QB-RAVE method

This method is the combination of RAVE and Quality-based rewards. The modified rewards calculated by formula 5 are used in backpropagation part. The nodes which wins could be achieved with less possible moves through them are more favored if this modification be used. This reward modification is used for both UCT and AMAF rewards. Then in the selection part, RAVE method is used to evaluate the nodes more accurately.

4. Leaf parallelization

One way to increase the precision in evaluating nodes is to augment the simulations. Leaf parallelization method starts MCTS process as general MCTS does until it reaches to the simulation phase. Then it starts multiple simulations in parallel until all of them finish in terminal state. Then it backpropagates all the rewards to the nodes in selection path. You can see the leaf parallelization in Fig 3.

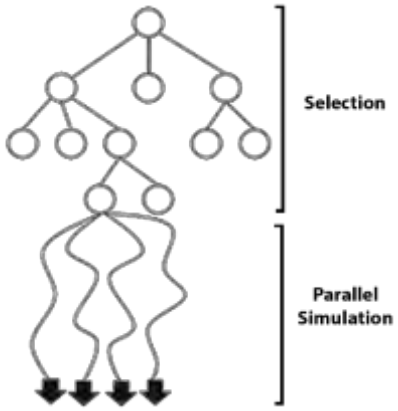


Figure 3. The Leaf parallelization process

VI. Experiments

1. Experimental setup

The proposed method is implemented on the game of HEX. Hex has simple rules. black and white alternate turns, and on each turn a player places a single stone of their color on any unoccupied cell. The winner is the player who forms a chain of their stones connecting their two opposing board sides. See Fig 4.

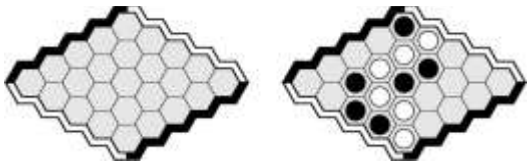


Figure 4. An empty 5×5 Hex board and a game won by white

The whole framework is implemented in python version 3.6 and all experiments were run in system with core i7 cpu, each core 2.20 Ghz and 8 GB of RAM which runs Linux.
For strengthening simulation, we used Last Good Reply (LGR), Decisive move strategy (DM) and POOLRAVE.

2. Results

Tournaments are considered to evaluate the agents' performance against each other. Each tournament consists of 100 games in which half of them starts with player one acting the first move and the others start with player two acting the first move.
In first place we used quality-based rewards on UCT and UCB1-TUNED to investigate its performance against UCT. The results are obtained from 5 tournaments on 11×11 game board and each agent plays 2000 playouts for each action and shown in Fig 5.

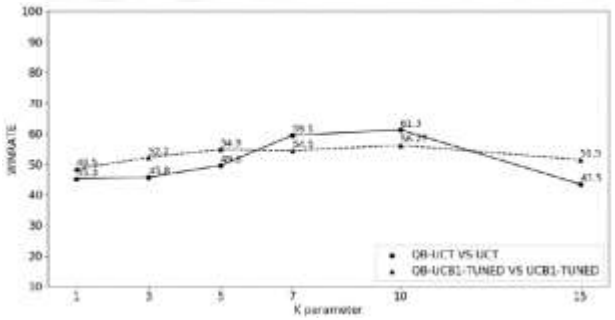


Figure 5. The impact of k parameter in winnig percentage in UCT and UCB1-TUNED

As you can see, the best performance is obtained by setting *k* parameter to 10. As parameter *k* increases, it slopes the rewarding sigmoid function in formula 4 and the more it slopes, its performance becomes more similar to discrete rewarding.
We also investigated the performance of Quality-based rewards when different simulation numbers are used for each move in decision process and the results are shown in Fig 6.

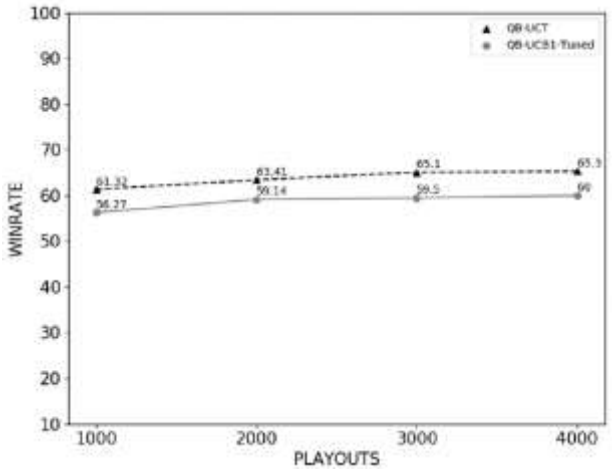


Figure 6. The performance of QB-UCT and QB-UCB1-TUNED for different simulations.

The results shown in Fig 6 indicate that performance tend to increase while the simulations are increasing. This is because QB-UCT and QB-UCB1-TUNED rewards are modified to distinguish between weak wins and strong wins. So the nodes with more strong wins are more promising for better better performance.
In this point we combine the RAVE algorithm with Quality-based rewards to see if any improvement is achieved. The QB-RAVE algorithm is compared with RAVE in 1000, 2000, 3000, 4000 number of simulations. Results are plotted in Fig 7.

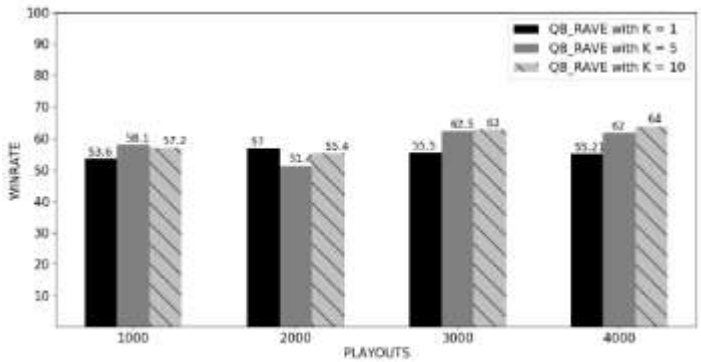


Figure 7. The winrate of QB-RAVE against RAVE in different simulation numbers

As seen in Fig 7, the QB-RAVE is more stable in winning RAVE when *k* parameter is set for 10. The results also show that the more simulation number becomes, the more winrate it gets. This shows the precision of QB-RAVE algorithm that it searchs the environment more accurately.

In the end, to check the proposed algorithm competetiveness against LGR, DM and POOLRAVE which are combined with RAVE algorithm, we make them to compete with each other. The results are shown in Fig 8.

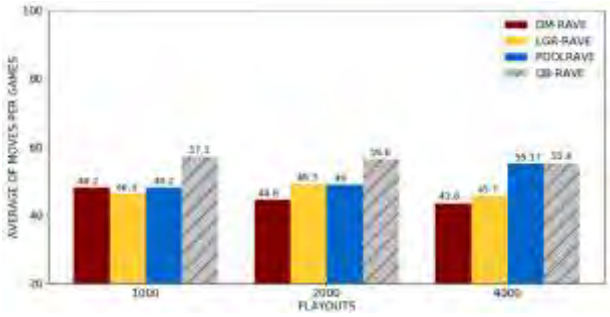


Figure 8. Tournament between DM, LGR, POOLRAVE and QB_RAVE algorithms

The results show that QB_RAVE is stronger in building search tree against DM, LGR and POOLRAVE.
Combination of quality-based rewards with leaf parallelization is used to improve the performance. The results in Fig 9 show the winrate of paralleled QB-UCT against QB-UCT.

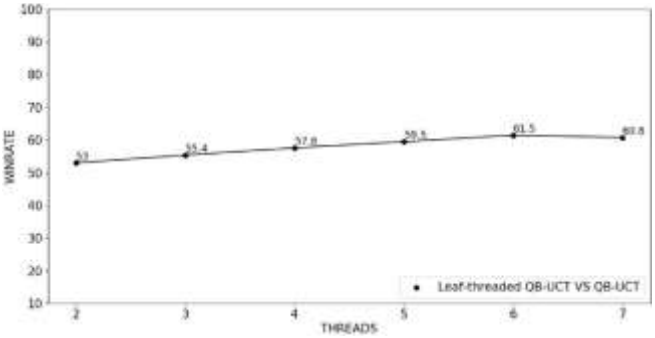


Figure 9. Winrate of leaf-threaded QB-UCT against QB-UCT in 1 second for each move.

The simulation number will be augmented by adding threads. Fig 10 shows the growth of simulations in accordance to threads increment.

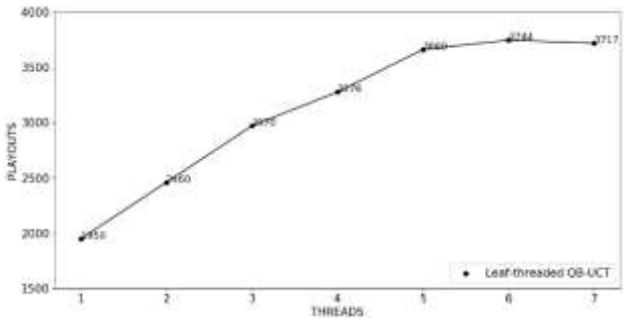


Figure 10. The growth of simulation numbers in comparison to number of threads

As you can see in Fig 10, as the number of threads increasing, the number of simulations decreases if we consider the average simulations per each thread. Fig 11 shows the average number of simulations per number of thread thread.

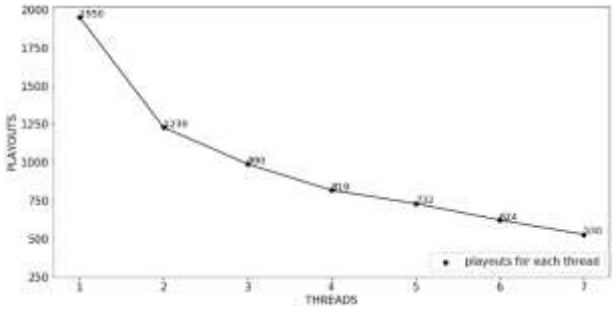


Figure 11. average number of simulations per each thread.

The reason that why the number of simulation decreases is because the more simulations run in parallel, the more threads with shorter simulation paths have to wait for the threads with longer simulation paths to return their results to sum up the total results and backpropagate it to the nodes met in selection path.

VII. Conclusion

In this paper, we proposed a method which is a combination of Quality-based rewards and RAVE algorithm. The resulted algorithm (QB-RAVE) uses the length of the simulation as a quality measure and modifies reward in the way that earlier wins get more reward and later wins get less reward. This priviledge lets the algorithm to consider more precision when backpropagating the reward to the nodes in the simulation path for both UCT and AMAF rewards.
On the other hand, The RAVE algorithm evaluates the nodes more accurately and takes adventage of both UCT and AMAF rewards stored in nodes. This priviledge will improve the selection of nodes in the selection part of MCTS algorithm.
The results obtained from this research shows that RAVE and Quality-based rewards are well-combined with each other and have a high win percentage against LGR, DM and POOLRAVE algorithms which are well-known algorithms in the field of monte-carlo tree search. Its also

observed that the winrate increses as the simulation number increses. This shows better performance of QB-RAVE in higher simulation number.
We also made use of leaf parallelization to increase the simulations and improve the performance. The results show that utilizing leaf parallelization with QB-UCT would result in improvement in winrate. It also shows that as the threads are increasing, the simulations per each thread is diminishing because the program has to wait for the longest simulated game to end.

References

[1] D. Silver *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[2] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Comput. Intell. Mag.*, vol. 2, no. 4, pp. 251–258, 2011.
[3] C. F. Sironi and M. H. M. Winands, "Comparison of rapid action value estimation variants for general game playing," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2017.
[4] D. Tom and M. Müller, "A study of UCT and its enhancements in an artificial game," in *Lecture Notes in Computer Science*, 2010, vol. 6048 LNCS, pp. 55–64.
[5] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo Search," 2006, no. 1, p. 22.
[6] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artif. Intell.*, vol. 175, no. 11, pp. 1856–1876, 2011.
[7] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-Grams and the Last-Good-Reply Policy Applied in General Game Playing," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 73–83, 2012.
[8] F. Teytaud and O. Teytaud, "On the huge benefit of decisive moves in Monte-Carlo Tree Search algorithms," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG2010*, 2010, pp. 359–364.
[9] J. B. Hoock, A. Rimmel, F. Teytaud, O. Teytaud, C. S. Lee, and M. H. Wang, "Intelligent agents for the game of go," *IEEE Comput. Intell. Mag.*, vol. 5, no. 4, pp. 28–42, 2010.
[10] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, May 2002.
[11] P. Drake, "The Last-Good-Reply Policy For Monte-Carlo Go," *ICGA J.*, vol. 32, no. 4, pp. 221–227, 2010.
[12] T. Pepels, M. J. W. Tak, M. Lanctot, and M. H. M. Winands, "Quality-based Rewards for Monte-Carlo Tree Search Simulations," in *21st European Conference on Artificial Intelligence*, 2016.
[13] D. P. Helmbold and A. Parker-wood, "All-Moves-As-First Heuristics in Monte-Carlo Go," in *In Proceedings of the 2009 International Conference on Artificial Intelligence*, 2009, pp. 605–610.