International Journal of Computer and Information Technology (ISSN: 2279 – 0764)
Volume 07 – Issue 06, November 2018

A Selection Variation for Improved Throughput and Accuracy of Monte Carlo Tree Search Algorithms

Allan Odhiambo Omondi
Faculty of Information Technology
Strathmore University
Nairobi, Kenya
Email: aomondi [AT] strathmore.edu

Ismail Ateya Lukandu
Faculty of Information Technology
Strathmore University
Nairobi, Kenya

Gregory Wabuke Wanyembi
Department of Information Technology
Mount Kenya University
Thika, Kenya

Abstract—**Reinforcement Learning (RL) is a computational approach to understanding and automating goal-directed learning and decision making. Methods such as dynamic programming, Monte Carlo, and temporal difference can all be used to implement RL. They provide room for continuous improvement. These improvements are faced with the challenge of balancing between the accuracy of the decision and the amount of time spent to make the decision. This study presents an improvement of the Monte Carlo Tree Search (MCTS) algorithm by enhancing its selection phase. The variation was implemented by applying a combination of Upper Confidence Bound applied to Trees (UCT) and "lean Last Good Reply with Forgetting (LGRF)" strategies. Consideration was made to maintain the generality of the algorithm. Two agents playing against each other in a game of Othello were used to implement the playouts while conducting the experiments. Each agent applied a different variation of the MCTS algorithm and a Chi-square test was then used to analyze the results. A win-rate, for the "lean LGRF" strategy, of 100% over 1,000 playouts against the UCT strategy was recorded. The Progressive Bias (PB) strategy had a win-rate of 45.8% against UCT. As expected, the UCT strategy had a win-rate of 49.7% (an almost 50-50 win-rate) against itself. The results show that there is a significant level of variance in the selection variation techniques and that the "lean LGRF" variation is superior to the UCT and PB variations.**

Keywords-Artificial Intelligence, Monte Carlo Tree Search; Reinformcenent Learning; decision theory, board games

I. INTRODUCTION

Estimates of the value of an unknown quantity can be made using Monte Carlo simulations. This is based on principles of inferential statistics. The use of computing power has supported numerous enhancements to the Monte Carlo simulation [1]. One of which is the combination of the generality of random simulation and the precision of a tree search to form a Monte Carlo Tree Search (MCTS); originally championed by [2] and [3].

This study reviews the four phases of the MCTS algorithm. These are the selection, expansion, simulation, and backpropagation phases. Each of these phases can be variated to improve the overall performance of the MCTS algorithm. Focusing the variation in the early phases can provide a significant effect on the performance of the algorithm. The study therefore focused on enhancing the selection phase of the MCTS algorithm by directing the search towards the most promising areas of the tree. The effect of doing this was a reduction in the amount of time taken to estimate the value of unknown quantities in a search tree and more accurate estimations.

As stated by [4], the basic MCTS algorithm can have a relatively slow convergence rate especially when not enhanced by implementing variations. However, [5] note that while enhancing the basic MCTS algorithm can indeed improve their performance in a domain specific application, it can also simultaneously reduce its generality and scalability. This remains a research challenge that requires further investigation. This study takes this into account by not adding domain-specific enhancements. The simulation phase of the MCTS algorithm is thus maintained as a random simulation.

The study submits the following hypothesis:

Null Hypothesis (H_0): There is no significant level of variance in the selection variation that is applied in a Monte Carlo Tree Search algorithm.

Alternative Hypothesis (H_a): There is a significant level of variance in the selection variation that is applied in a Monte Carlo Tree Search algorithm.

$$H_0: \sigma_s^2 = \sigma_p^2$$

$$H_a: \sigma_s^2 \neq \sigma_p^2$$

The following variables were used as the dependent and independent variables in the study:

Dependent Variable: Performance of the enhanced Reinforcement Learning (RL) agent, quantitatively measured in terms of the number of wins against the baseline RL agent

Categorical Independent Variable: Selection variation (categories in this independent variable are plain UCT, UCT with "lean LGRF", and the Progress Bias variations)

The study subsequently seeks to answer the following research question:

Is there a significant level of variance in the selection variation that is applied in an MCTS algorithm?

Section II of the paper presents a background to the study by reviewing the move from mere optimization towards Machine Learning (ML). A discussion of RL as a category of ML is also presented. The MCTS algorithm as a method of implementing RL is then presented in Section III of this paper. It details the key variations that already exist in the selection and simulation phases. The key contribution of enhancing the selection phase by combining UCT and "lean LGRF" is also specified. Section IV explains the method used to conduct the experiment and provides the results of the experiments. Section V then discusses what the experiment results mean and lastly, Section VI presents the conclusion of the study.

II. Background to the Study

A. Machine Learning

Optimization techniques such as constraint programming, mixed integer programming, and local search can be used in autonomic self-optimizing systems [6]. However, there is a need to go beyond optimization and consider ML. This is based on the expectation that an autonomic self-optimizing system should be able to automatically adapt by reacting to variable environmental conditions and runtime phenomena. ML enables the solution to generalize to unseen data thus enabling it to make a decision in an unexpected state. Decision theory thus becomes part of ML as it combines probability theory and utility theory to provide a formal and complete framework for decisions made under uncertainty. If it is a situation in which multiple actors are involved, then decision theory can be extended by game theory. The three possible ML techniques that can be applied to learn how to make a decision are RL, supervised learning, and unsupervised learning.

RL maps environmental conditions and runtime phenomena to appropriate actions. This is done with the aim of maximizing a predefined reward. The reward in this case is the maximization of a utility function. RL is therefore a computational approach to understanding and automating goal-directed learning and decision-making. It makes use of Markov Decision Processes to define the interaction between a learning agent and its environment. This definition is presented in terms of states, actions, and rewards. However, there are cases when the state of the environment is not fully observable. Such a case can apply a Partially Observable Markov Decision Process (POMDP) which is similar to an MDP but adds an observation model, $O(s, o)$, to specify the probability of perceiving observation o in state s.

Supervised learning on the other hand involves learning (in the process of generalizing to unseen data) from a training set of labeled examples provided by a knowledgeable external supervisor. Each labeled example is a description of a situation (environmental condition or runtime phenomenon) and a specification of the correct action the system should take in that situation. It is expected that there is a reliance on the knowledge provided by a domain expert. This can cause a problem in cases where the knowledge is not easily accessible or only part of it is available. Unsupervised learning on the other hand involves finding structure hidden in collections of unlabeled data. Unlike unsupervised learning, RL goes a step further to maximize a reward signal (defined by a utility function). For these reasons, the study considered RL as the most appropriate way to apply ML in the solution.

B. Reinforcement Learning

Elements of RL include a policy, a reward signal, a value function, and an optional model of the environment. Policies are stochastic in general and involve a simple function or extensive computation to determine the behavior of an RL agent. It therefore forms the core of the RL agent as it is a mapping from states to probabilities of selecting each possible action. Rewards are stochastic functions of the state of the environment and the action taken. They specify what should be achieved rather than how to achieve it. An agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but cumulative reward in the long run. A value function on the other hand, specifies the total amount of reward an agent can expect to accumulate over the long run starting from the present state it is in. This makes a value a prediction of a reward. It is important to be able to efficiently estimate values. Lastly, a model of the environment mimics the behavior of the environment. It allows inferences to be made about how the environment will behave. It is possible to solve RL problems using either model-based methods or model-free (trial-and-error) methods. This study focused on a model-based method.

Unlike supervised learning and unsupervised learning which use training information to instruct the agent on what action to perform (instructive feedback), RL uses training information to evaluate the actions taken (evaluative feedback). However, there is still a need to balance between exploring new solutions while exploiting already known good solutions.

A Markov Decision Process (MDP) which includes Bellman equation and value function concepts, defines the general, complete problem formulation process [7]. Solutions to a Markov Decision Process or a Partially Observable Markov Decision Process can be categorized into three main classes: dynamic programming methods, Monte Carlo methods, and temporal difference methods. Dynamic programming methods are well developed mathematically but require a complete and accurate model of the environment. Monte Carlo methods on the other hand do not require a complete and accurate model of the environment. However, they are not suited for step-by-step incremental computation in their default state. And lastly, temporal difference methods which also require no accurate model of the environment but are complex to analyze. This study therefore focuses on variating of Monte Carlo methods in a model-based environment.

III. MONTE CARLO TREE SEARCH

Monte Carlo methods are founded in statistical physics where they have been applied in obtaining approximations of intractable integrals. Monte Carlo simulation is thus a method of estimating the value of an unknown quantity using the principles of inferential statistics. It is possible for the unknown quantity to be the value of an agent's action, as in game theory. This approximation is made possible using simulations and is then used to adjust the agent's policy towards its most optimum state. The accuracy of the approximated value of an action is directly proportional to the number of simulations performed. This is however limited by a computational budget such as computational time or memory or by an iteration constraint.

Traditional MCTS is a best-first, heuristic-driven algorithm that combines the generality of random simulation with the precision of a tree-search [8]. In doing so, it enables an agent to plan to reach a pre-defined goal or to avoid a failure. One of the advantages of MCTS is that it can function without complex domain-dependent rules written by a human expert. It will determine on its own how to reach a win. It can therefore determine how to "master" an open-ended task without any kind of direct guidance or supervision. It works based on the ML principle of RL. The basic MCTS algorithm is as follows:

Algorithm 1: The Basic Monte Carlo Tree Search Algorithm

```
Input  : The state, $s_0$, of the current root node, $n_0$
Output: The action, $a$, that leads to the most optimum child
        node that has a desirable state
1 Function $MCTS(s_0)$
2     create root node $n_0$ with state $s_0$;
3     while within computational budget do
          /* The $SELECTEXPAND$ function is used to
             select a child node according to a predefined
             utility function.  This selected child node
             $n_i$ is then added to the search tree in the
             process of expansion.                          */
4         $n_i \leftarrow SELECTEXPAND(n_0)$ ;
          /* The $SIMULATEBACKPROPAGATE$ function
             is used to simulate possible scenarios from
             the newly expanded node $n_i$.  A reward value
             is calculated by executing each of the
             possible actions available in the newly
             expanded node.  This is then stored in $\Delta$ and
             backpropagated up the tree to update the node
             values.                                        */
5         $\Delta \leftarrow SIMULATEBACKPROPAGATE(s(n_i))$ ;
      /* The result of the overall search,
         $a(SELECT(n_0))$, is the action, $a$, that leads to
         the best child of the root node, $n_0$, where the
         exact definition of ''best'' is defined by the
         implementation.                                    */
6     return $a(SELECT(n_0))$ ;
```

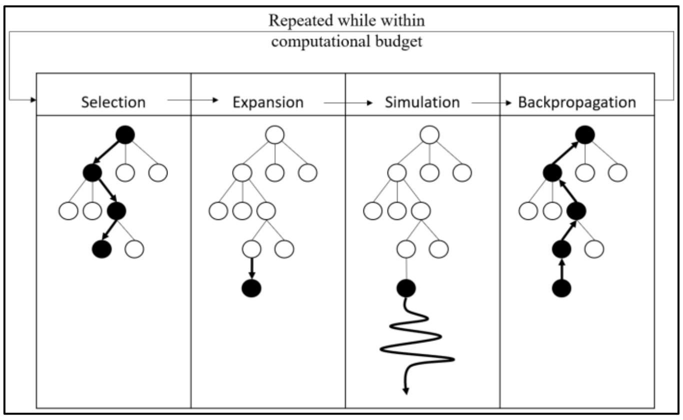

Figure 1. Number of Experiments against Average Number of Wins
Adapted from: [3]

As explained in a study by [5], the MCTS algorithm is made up of a tree policy and a default policy. The tree policy is used to select an existing node according to a utility function. The selection process continues through the search tree until either a terminal node is reached or until a node that has not been fully expanded to allow its children to be explored is encountered. The node that has not been fully expanded does not have to be a leaf node in the search tree. It can be an inner node with a combination of explored and unexplored child nodes. If it encounters a state that has not been fully expanded, then the tree policy expands the tree by creating a new node.

On the other hand, the default policy is used to simulate possible scenarios. The results of these simulation are used to calculate the value of nodes. A simulation starts from a newly expanded node until either a terminal node is reached or until a predefined computational budget (computational time or memory or an iteration constrained) is reached. As explained in Section III. C., this simulation can either be pseudo random based strategies such as N-Grams and LGRF or it can be random. This research applied a random simulation to reduce the level of bias that can be introduced by a human-designed algorithm. The calculated reward is then used to update the reward value of each of the nodes traversed. This is done using the backup function.

The tree policy and the default policy can be broken down into 4 phases of the MCTS algorithm as shown in Fig. 1. Section III. A explains the selection phase which involves recursively selecting optimal child nodes until a leaf node is encountered. It reviews the UCT, UCT-ULM, and PB variations to the selection phase. It also explains a unique approach to the selection phase which is the main contribution of this paper. Section III. B explains the expansion phase which involves adding unexplored child nodes to the search tree. Section III. C explains the simulation phase which involves executing a play out until either a terminal state or a pre-defined computational budget is reached. It also reviews two common simulation variations which are the use of N-Grams and the LGRF strategy. Lastly, Section III. D explains the backpropagation phase which updates the calculated values of each node that has been traversed.

A. SELECTION

Selection is the first phase of the MCTS algorithm. The output of this phase is the identification of the most promising child node to traverse. The accuracy of this identification can be improved by selecting nodes based on their average reward value. This implies the need to keep track of the reward values of the nodes that have already been traversed. However, as shown in a study by [9], this also causes the challenge of focusing too much on the most promising nodes while neglecting nodes whose immediate reward is inadequate but may turn out to lead to a superior reward in the long-run. Thus, the exploration-exploitation balance dilemma emerges because an optimal node can be missed because of temporarily promising reward values from sub-optimal nodes.

1) Upper Confidence Bound

The most common way to address this dilemma in the selection stage of the MCTS algorithm is to apply the Upper Confidence Bound (UCB) technique when performing a selection of the next node [10]. When applied to MCTS, UCB uses a logarithm-based formula on collected reward values correlated to the nodes on the search path. This combines MCTS and UCB to form "MCTS with UCB applied to trees (UCT)" as shown in (1).

$$\underset{i \in childrenOfTheExpandedNode}{\operatorname{argmax}} \left(\frac{w_i}{x(n_i)} + \sqrt{c \times \frac{\ln t}{n_i}} \right) \quad (1)$$

Where:

$\boldsymbol{w_i}$ is the number of simulated wins that have occurred when the node i was selected

$\boldsymbol{x(n_i)}$ stands for the total number of simulations that have occurred for the node i ($\frac{\boldsymbol{w_i}}{\boldsymbol{x(n_i)}}$ is therefore the value of the node and forms the exploitation parameter of the UCB formula)

$\boldsymbol{c}$ stands for the exploration parameter. An increase in the value of $\boldsymbol{c}$ results in more exploration. The study used a value of $\boldsymbol{c = 2}$ as the baseline

$\boldsymbol{t}$ stands for the total number of simulations that have occurred for the parent of node i

There is a difference between a pure Monte Carlo algorithm and an MCTS algorithm. A pure Monte Carlo algorithm runs a multitude of randomly simulated game states. An MCTS algorithm also runs a multitude of randomly simulated game states but goes a step further to keep reward value statistics of each node. The MCTS algorithm can then apply the Upper Confidence Bound (UCB) algorithm to the node's statistics to address the exploration-exploitation dilemma. UCB determines a confidence interval for win ratios (good rewards) of each node and returns the highest value in the confidence interval.

UCB gives nodes with more simulations a narrower confidence interval while enlarging the confidence intervals of nodes with fewer simulations. This implies that the higher the number of traversals made on a node, the more confident the MCTS algorithm can be of the calculated average reward value of that node. MCTS then selects the node that has the highest UCB value. Nodes that have high UCB values will therefore be selected multiple times at first.

According to the UCB algorithm, the more these nodes are selected, the narrower their confidence interval becomes. As the interval becomes narrower, their upper confidence bound value loses its position as the highest value. MCTS is therefore forced to choose another action which has the highest upper confidence bound value but a wide confidence interval and in doing so, it addresses the exploration-exploitation dilemma. Using this technique in the selection stage of the MCTS algorithm ensures no potential node is starved of selections, and at the same time, favorable nodes are selected more often than their counterparts. The following algorithm shows the MCTS with UCB applied to trees (UCT) algorithm which this study used as the baseline:

Algorithm 2: Selection and Expansion Phases Based on the UCT Monte Carlo Tree Search Algorithm

```
Input  : The root node, $n_0$
Output: The selected child node $n_i$ that has the highest value
1  Function SELECTEXPAND($n_0$)
2      while $n_0$ is non-terminal do
3          if $n_0$ has been fully expanded then
4              return $\Delta \leftarrow SELECT(n_0)$ ;
5          else
6              return $EXPAND(n_0)$ ;

7  Function SELECT($n_0$)
       /* Determines the best child, Δ, to select out of
          all the other possible children of node $n_0$.
          This is used as the baseline in this research
          and it is based on the value derived by the
          Upper Confidence Bound applied to Trees (UCT)
          formula below.                                */
8      return
       $\underset{i \in childrenOfTheNewlyExpandedNode}{\arg\max} \left( \frac{w_i}{x(n_i)} + \sqrt{c \times \frac{\ln t}{x(n_i)}} \right)$ ;

9  Function EXPAND($n_0$)
10     perform $a \in$ untried actions from $A(s(n_0))$ ;
11     add new child $n_i$ to $n_{i-1}$
         with $a(n_i) \leftarrow a$
         and $s(n_i) \leftarrow f(s(n_{i-1}), a)$ ;
12     return $n_i$
```

Attempts such as [9] and [11] to enhance the selection stage of the MCTS algorithm are evidence that sophisticated selection strategies can greatly enhance the MCTS algorithm. Sections III.A.1 presents a unique enhancement to the Upper Confidence Bound applied to Trees – Unvisited Legal Moves (UCT-ULM) variation. Section III.A.2 then reviews the Progress Bias variation to the MCTS selection phase.

2) Upper Confidence Bound applied to Trees – Unvisited Legal Moves (UCT-ULM)

The idea behind the UCT-ULM selection variation, is to bias the focus of the search to regions of the tree which contain states that have a history of higher rewards. This is as opposed to wasting valuable time resource in searching regions of the tree that have states that have a history of low rewards.

The UCT-ULM selection variation works by checking each node in the tree region to see whether it has child nodes that have been visited before. If all the children have been visited before, then UCT-ULM applies regular UCT to those visited children to find the node with the highest reward.

If the UCT-ULM selection variation finds child nodes that have not been visited before in the tree region, then UCT will select the Unvisited Legal Move (ULM) child node that has the highest heuristic reward value. This heuristic reward value is not determined using UCT. As demonstrated by [12], it is instead determined using an N-Gram strategy. This is similar to the N-Gram simulation strategy explained in Section III.C.2 of this paper. A 2-Gram or 3-Gram strategy is preferred based on their low computational budget. The 2-Gram or 3-Gram strategy is then applied to the node sequence to determine each node sequence's average heuristic reward value. This results in a case where ULM child nodes contained in a sequence which has a low average heuristic reward value may never be selected.

This study experimented on the application of a Last Good Reply with Forgetting (LGRF) strategy instead of the common use of N-Grams in the ULM step. However, the LGRF strategy made use of only LGR-1 tables as explained in Section III.C.2 of this paper. Section IV documents the results of the experiment.

3) *Progressive Bias (PB)*

Similar to the UCT-ULM strategy, the PB strategy determines the average heuristic value of child nodes in a sequence by applying N-Grams. However, unlike UCT-ULM, PB combines the N-Gram technique with UCT as one selection strategy. PB applies the following shown in (2).

$$\underset{i}{\operatorname{argmax}}\left(\frac{w_i}{n_i} + \sqrt{c \times \frac{\ln t}{n_i}} + \frac{W \times H_i}{l_i + 1}\right) \tag{2}$$

Where:

$\boldsymbol{w_i}$ is the number of simulated wins that have occurred for the node i

$\boldsymbol{n_i}$ stands for the total number of simulations that have occurred for the node i ($\frac{\boldsymbol{w_i}}{\boldsymbol{n_i}}$ is therefore the value of the node and forms the exploitation parameter of the UCT formula)

$\boldsymbol{c}$ stands for the exploration parameter. An increase in the value of $\boldsymbol{c}$ results in more exploration. The study used a value of $\boldsymbol{c} = \boldsymbol{2}$ as the baseline.

$\boldsymbol{t}$ stands for the total number of simulations that have occurred for the parent of node i. Therefore, $\sqrt{\boldsymbol{c} \times \frac{\ln \boldsymbol{t}}{\boldsymbol{n_i}}}$ forms the exploration parameter of the UCT formula and $\frac{\boldsymbol{w_i}}{\boldsymbol{n_i}} + \sqrt{\boldsymbol{c} \times \frac{\ln \boldsymbol{t}}{\boldsymbol{n_i}}}$ forms the UCT parameter of the PB formula.

$\boldsymbol{W}$ is a constant set to 10

$\boldsymbol{H_i}$, is the N-Gram based heuristic value for child i. If a 3-Gram is used, this value will be based on the average reward value of the move sequence consisting of the child node i, its parent p and the parent of node p

$\boldsymbol{l_i}$ is the number of losses of encountered during the simulation of node $\boldsymbol{i}$ That is, $\boldsymbol{n_i} - \boldsymbol{w_i}$

As explained by [12], the division of the heuristic value by the number of losses assures that nodes that do not turn out well are not biased for too long. This means that when a node has been visited only a few times, then the heuristic knowledge will have a major influence on the final decision of which node to select. The same heuristic value becomes less significant when the move does not perform too well.

PB has one main advantage over other selection strategies. That is, through the combination of UCT and ULM as one technique, PB can apply heuristic evaluation to perform selection when the UCT parameters have not yet had enough iterations to settle.

B. *EXPANSION*

The expansion phase of the MCTS algorithm checks the previously selected action node to determine if it has any unexplored child nodes. If an unexplored child node is found, it is added to the search tree. However, if the previously selected child node is a terminal node then an expansion is not necessary because the end of the game or the scenario is implied at that terminal node.

Differences in the expansion stage are based on a choice of implementation technique and not on the variation of the algorithm. A possible implementation technique is to add multiple child nodes from the selected node at the same time. This can be done in a multi-threaded architecture. As stated by [5], the implementation choice of either single node expansion or multiple node expansion depends on the domain and the computational budget. Other than this, the expansion phase of the MCTS algorithm has no significant variations given its straight-forward function.

C. *SIMULATION*

The simulation stage of the MCTS algorithm automatically plays out possible scenarios from the newly expanded leaf node in a replica environment. This play-out is repeated until either a pre-defined computational budget or a terminal node is reached. As explained in a study by [13], simulation is dependent on predefined game rules and it is performed in a replica environment. Performing the actions in a replica environment avoids making trial changes to the actual environment during the simulations.

The actions performed during the simulation can be chosen either randomly or quasi-randomly. The algorithm below shows the implementation of a random simulation.

Algorithm 3: Simulation and Backpropagation Phases of the Monte Carlo Tree Search Algorithm

```
Input  : The state of the current node $s(n_i)$
Output: The reward value of node $n_i$
Function SIMULATEBACKPROPAGATE($s(n_i)$)
    while $A(s(n_i))$ is not empty do
        perform $a \in A(s(n_i))$ at random ;
        $s(n_i) \leftarrow f(s(n_{i-1}), a)$ ;
        /* $x(n_i)$ is the number of times node $n_i$ has been
           traversed                                          */
        $x(n_i) \leftarrow x(n_i) + 1$ ;
        if result is a win then
            /* The number of simulated wins that have
               occurred when node $n_i$ was traversed; where
               a ``win'' is defined by the
               implementation.                                */
            $w_i \leftarrow w_i + 1$ ;
    BACKPROPAGATE($n_i$) ;

Function BACKPROPAGATE($n_i$)
    while $n_i$ is not null do
        update $rewardValueOfNode\ n_i$ ;
        /* $w_i$ = number of wins of node $n_i$              */
        update $w_i$ ;
        /* $x_i$ = number of traversals(visits) made on
           node $n_i$                                         */
        update $x_i$ ;
        update $rewardValueOfNode\ n_i = \frac{w_i}{x(n_i)}$ ;
        $n_i \leftarrow$ parent of $n_i$ ;
```

Research by [9] suggests that using a quasi-random simulation strategy can improve the performance of MCTS significantly. A quasi-random simulation in this case involves the use of a knowledgeable, heuristic-based simulation strategy. Two of the most successful MCTS simulation strategies are N-Grams and LGRF. Sections III.C.1 and III.C.2 of the paper review these two strategies respectively.

1) N-Grams

Even though the foundation of N-Grams is statistical Natural Language Processing (NLP), they can also be used for predicting the next move of players in game theory. [14] and [15] demonstrated the use of N-Grams to turn free text into analyzable numerical values [9] built on this statistical NLP foundation to demonstrate the use of N-Grams in selecting the best move for an autonomic game player. This was based on the formulation of knowledge of a sequence of possible future actions.

The sequence of actions performed during the simulation stage of an MCTS provide input used to build an N-Gram. N denotes the number of actions in a sequence. If a sequence of actions does not exist in an N-Gram, it is added to the N-Gram and assigned an initial reward value, $R(S)$. If the sequence of actions exists in the N-Gram, then the initial reward value is

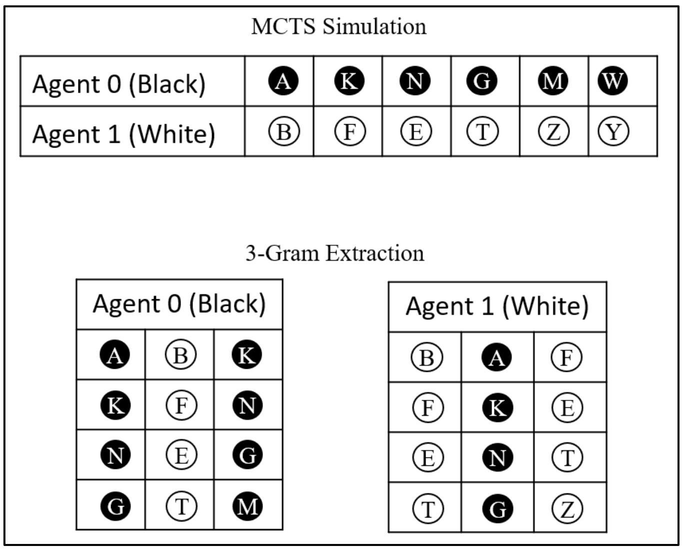

Figure 2. Example of a 3-Gram sequence of actions extracted from a simulation

updated with the average reward value obtained from past simulations. The sequence of actions with the highest $R(S)$ value is then selected based on a greedy algorithm. Fig. 2 provides an example of 3-Gram sequences of actions extracted from an MCTS simulation.

A study by [13] provided evidence that the use of N-Grams in the simulation strategy could increase the win-rate of an agent to 70%. However, caution needs to be taken when applying N-Grams with a large N due to their memory-intensive nature. The win-rate may increase but it will take the agent more time to make a decision.

2) Last Good Reply with Forgetting

LGRF can be used during the MCTS simulation phase to determine which action to perform next based on the previous actions that have already been performed before. LGRF works by keeping track of the reactions/replies that resulted in a high reward. This includes the environment or opposing agent's action followed by the agent's reaction/reply. However, as shown in Fig. 3 and further explained in a study by [11] and by [16], LGRF records only one reaction (the best) to a sequence of actions. For this reason, LGRF has a much smaller memory footprint compared to the N-Gram technique.

LGRF uses two tables. The LGR-1 table records the best reply to the opponent's action while the LGR-2 table records the best reply to a sequence containing the agent's action followed by the opponent's action. During an MCTS simulation, the LGR-2 table is consulted first to know which action to perform. If it does not have an answer, then the LGR-1 table is consulted. In the event that both the LGR-2 and LGR-1 tables do not have answers, then the default policy is applied. Whereby the default policy is to simulate the execution of a random action from the set of legal actions.

Unlike the N-Gram MCTS simulation strategy, the LGRF MCTS simulation strategy does not calculate an average reward

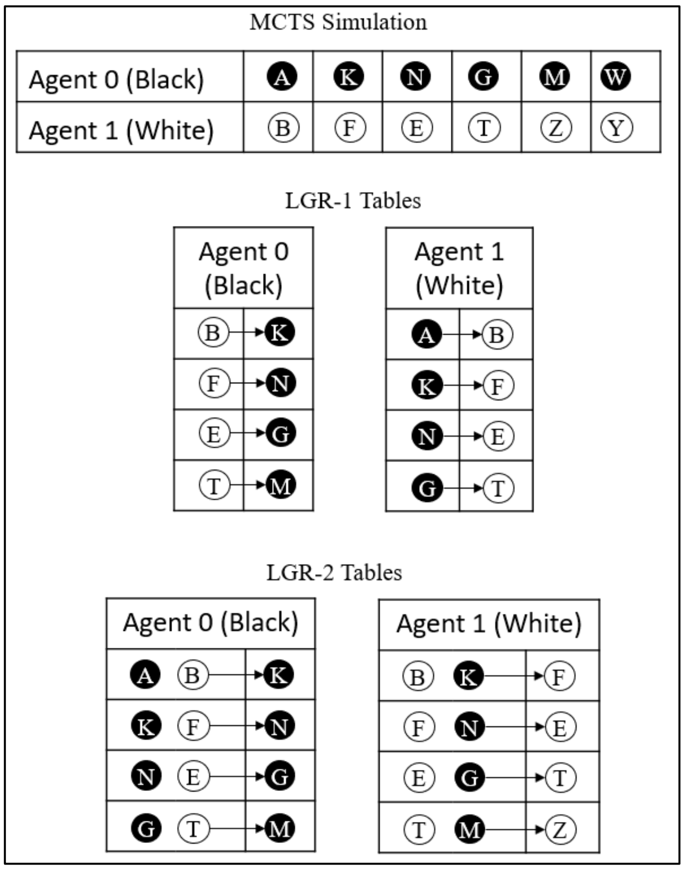

Figure 3. Example of extraction of actions based on an LGRF strategy from a simulation

value. If an agent's reply results in a higher reward for an already recorded action, then the action's reward value is overwritten with the value of the reply move that received the higher reward. However, if the agent's reply results in a lower reward for an already recorded action, then the recorded move's reward value is deleted (forgotten) from the LGR-1 and LGR-2 tables. This is known as the LGRF MCTS simulation strategy.

D. BACKPROPAGATION

This is the last phase of the MCTS algorithm. It involves returning the reward value of each action node, obtained through simulation, back up the search tree (backpropagation). The returned reward value is used to update the previous value of the action node. The reward value can be a discrete reward value, or a vector of reward values relative to each agent for more complex multi-agent domains. Backpropagation also involves incrementing each node's visit count.

IV. EXPERIMENTS

A. Experiment Test Bed

The experiment's test bed hardware was made up of an 8-core Intel® Core™ i7-4790 CPU with a clock rate of 3.60GHz and a RAM of 11.5GB. The software in the experiment test bed consisted of a 64-bit Linux-based Operating System (Ubuntu 18.04.1 LTS) running on an x64 based processor. An Apache HTTP Server was also used to support the running of the algorithm's implementation as a web-based system.

B. Othello Strategy Board Game

Reversi, also known as Othello, a strategy board game played by two players on an 8x8 uncheckered board was used to apply the MCTS algorithm. It was first created in 1971 by a Japanese salesman called Goro Hasegawa. 64 disks with a top side and a bottom side, such that each side has a different color (usually one side black and the other side white) are used to play the game.

It is a sequential game whereby players take turns to place disks on the board. A player places a disk on the board with their assigned color facing up. When the opposing player's disk, say a white disk, is surrounded by the other player's disk, say a black disk, then the white disk is flipped and becomes a black disk. A player can flip any number of opposing disks so long as they have a disk on either side of consecutive opposing disks. If a player cannot place a disk so that it is on either side of the opposing player's disk, then that player loses their turn. The game continues until either all spaces are occupied, or no player can make a move. The rules are simple, but strategy is what enables a player to win.

C. Experiment Procedure

The following steps outline the procedure that was followed to conduct the experiments:

TABLE I. EXPERIMENT PROCEDURE

Step	**Description**
I	Define the rules of the game (Othello)
II	Derive the algorithm to be used in the game
III	Establish a hypothesis (induction)
IV	Use the hypothesis to make predictions (deduction)
V	Conduct experiments to test the predictions
If the experiment results in **Step V** are not satisfactory, then:	
VI	Manipulate the algorithm and go back to **Step III**
If the experiment results in **Step V** are satisfactory, then:	
VI	Conclude the experiment

D. Experiment Results

The study used the chi-square test during data analysis to compare the observed level of variance to the expected, theoretical level of variance. This level of variance was based on the categorical independent variable, that is, three selection variations. The three selection variations were UCT, UCT-ULM with LGRF, and PB.

TABLE II. CONTINGENCY TABLE OF SELECTION VARIATIONS VERSUS PERFORMANCE

	Selection Variations			
Performance	**UCT**	**UCT with "lean LGRF"**	**PB**	**Total**
Win	497	1000	458	1,955
Loss	503	0	542	1,045
Total	1,000	1,000	1,000	3,000

TABLE III. CHI-SQUARE CALCULATIONS FOR COMPARING THE LEVEL OF ASSOCIATION IN THE SELECTION VARIATIONS

Selection Variation	**Observed frequency** O_i	**Expected frequency** E_i	$(O_i - E_i)$	$(O_i - E_i)^2$	$\frac{(O_i - E_i)^2}{E_i}$
UCT-ULM with LGRF	1,000	729	271	73,441	100.7421
PB	458	729	-271	73,441	100.7421
				$\sum \frac{(O_i - E_i)^2}{E_i}$	201.4842

χ^2_{calc} (Calculated chi-square) value = 201.4842

d.f. (Degrees of freedom) = $(c-1) \times (r-1)$ =

(2-1) × (2-1) = 1

α (Level of significance) = 5%

χ^2_{tab} (Table chi-square) value = 3.841

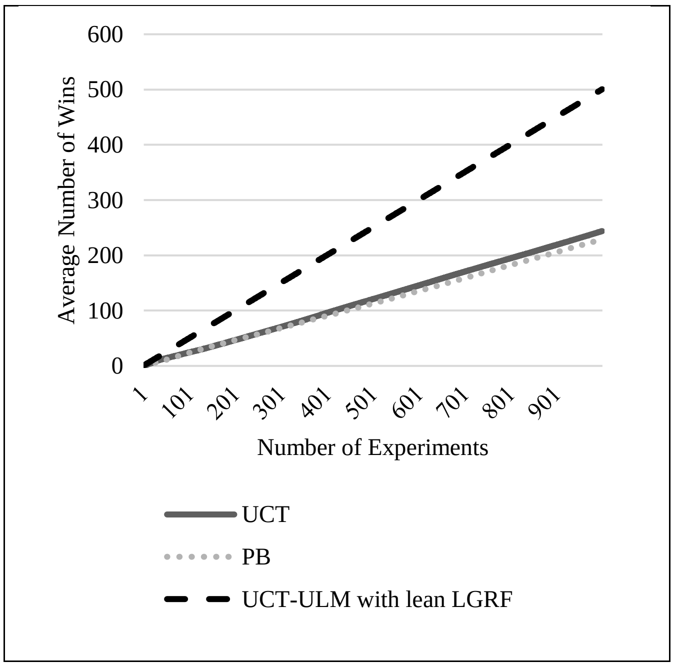

Figure 4. Number of Experiments against Average Number of Wins

V. DISCUSSION

The UCT selection variation won 49.7% of the time while the PB selection variation won 45.8% of the time. As indicated in a study by [12], the PB selection variation combines UCT and heuristic values into one technique. This enables it to make use of heuristic evaluations when the UCT parameters have not yet settled. It was expected that the PB selection variation would have the best performance because it combines the best of two techniques. However, the computational budget of time meant that it had to present a result based on the point it had reached when the time allocated had expired. The time allocated to decide which move to perform was 4 seconds. This was maintained as a constant across all selection variations.

As indicated by [9], using a quasi-random simulation strategy can improve the performance of MCTS significantly. However, this research applied a random simulation strategy in order to focus on enhancing the selection strategy. Unlike the PB strategy which combines two techniques, the strategy applied in the research selected one of two techniques based on a condition. The condition was such that if all the children of a node have been visited, then the UCT selection strategy was applied. However, if not all the children had been visited before, then a variation of the LGRF strategy was used to select a child node. By doing this, the search was focused on regions of the tree that contain good move sequences.

The results are evidence that the calculated value (201.4842) of χ^2 is much higher than its table value (3.841). The calculated value did not occur by chance; it is significant. We can therefore fail to reject the alternative hypothesis such that we can infer that there is a significant level of variance in the selection variation that is applied in an MCTS algorithm. Because the two variations are not similar, then it follows that one must be superior than the other. The UCT with "lean LGRF" strategy proposed in this study won 100% of the games played against the baseline UCT strategy.

VI. CONCLUSION

This study submits a new MCTS selection variation that applied UCT and "lean LGRF" strategies. Numerous opportunities to extend the work further exist. One of which is the generalization of the results by applying the algorithm in different fields other than the Othello board game. One such context is the field of automation that involves theoretical concepts such as optimization theory, control theory, decision theory, and game theory.

VII. References

[1] D. Hartmann, "Let's Catch the Train to Monte-Carlo," *International Computer Games Association Journal,* vol. 39, no. 1, pp. 44-46, 2017.

[2] S. Gelly, Y. Wang, R. Munos and O. Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," Institut National de Recherche en Informatique et en Automatique, 2006.

[3] R. . Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," *Computers & Graphics,* vol. , no. , pp. 72-83, 2006.

[4] T. Vodopivec, "Monte Carlo Tree Search Strategies," University of Ljubljana, 2018.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen and S. Colton, "A survey of monte carlo tree search methods," in *IEEE Transactions on Computational Intelligence and AI in games*, 2012.

[6] A. De Filippo, M. Lombardi and M. Milano, "Methods for off-line/on-line optimization under uncertainty," in *International Joint Conferences on Artificial Intelligence*, 2018.

[7] G. Su, T. Chen, Y. Feng, D. S. Rosenblum and P. S. Thiagarajan, "An iterative decision-making scheme for markov decision processes and its application to self-adaptive systems," in *International Conference on Fundamental Approaches to Software Engineering*, 2016.

[8] K. Rocki and R. Suda, "Parallel monte carlo tree search scalability discussion," in *Australasian Joint Conference on Artificial Intelligence*, Berlin, Heidelberg., 2011.

[9] D. J. Soemers, C. F. Sironi, T. Schuster and M. H. Winands, "Enhancements for real-time Monte-Carlo Tree Search in General Video Game Playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016.

[10] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine*, Berlin, Heidelberg., 2006.

[11] J. A. Stankiewicz, M. H. M. Winands and J. W. H. M. Uiterwijk, "Monte-Carlo Tree Search Enhancements for Havannah," in *Advances in Computer Games,* 2011.

[12] N. G. Den Teuling and M. H. Winands, "Monte-Carlo Tree Search for the simultaneous move game Tron," University of Maastricht, Netherlands, Tech. Rep., 2011.

[13] M. J. Tak, M. H. Winands and Y. & Bjornsson, "N-grams and the last-good-reply policy applied in general game playing," in *IEEE Transactions on Computational Intelligence and AI in games*, 2012.

[14] V. Ranganathan, A. Raja, A. Ravichandran and N. Viswanathan, "Text Highlighting: A Machine Learning Approach.," *International Research Journal of Engineering and Technology,* vol. 5, no. 8, pp. 1603-1606, 2018.

[15] M. Schonlau, N. Guenther and I. Sucholutsky, "Text mining with n-gram variables," *Stata Journal,* vol. 17, no. 4, pp. 866-881, 2017.

[16] H. Baier and P. Drake, "The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go," *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 2, no. 4, pp. 303-309, 2010.