博　　士　　論　　文

Application of Simple Regret Bandit Algorithms on Monte-Carlo Tree Search

(単純リグレットバンディットアルゴリズムを利用したモンテカルロ木探索)

平成 28 年 6 月 1 日提出

指導教員　鶴岡 慶雅　准教授

東京大学大学院工学系研究科
電気系工学専攻

37-137071

劉 雲青

Abstract

Monte-Carlo Tree Search (MCTS) has made a significant impact on various fields in AI, especially on the field of computer Go. The key factor to the success of MCTS lies in its combination with bandit algorithms, which solves the multi-armed bandit problem (MAB). The MAB problem is a problem where the agent needs to decide whether it should act optimally based on current available information, or gather more information at the risk of suffering losses incurred by performing suboptimal actions. One of the most widely used MCTS variants is the UCT algorithm, which simply applies the UCB algorithm to every node in the tree.

The pure exploration MAB problem seeks to identify the optimal best arm, rather than gathering as much reward as possible. The pure exploration MAB problem can also be formally stated as the minimization of simple regret, which is defined as the difference between the expected reward of the optimal bandit and the bandit that has been identified as the optimal bandit. Since the main objective of game tree search is to identify the best action to take, it has been considered that bandit algorithms that solve the pure exploration MAB problem would be a better match for application in MCTS. However, the application of simple regret bandit algorithms to MCTS is far from trivial.

The simple regret bandit algorithm has the tendency to spend more time on sampling suboptimal arms, which may be a problem in the context of game tree search. In this research, we will propose combined confidence bounds MCTS (CCB-MCTS) algorithm, which utilize the characteristics of the confidence bounds of the improved UCB and $\text{UCB}_{\sqrt{\cdot}}$ algorithms to regulate exploration for simple regret minimization in MCTS.

Another possible approach is based on the observation that max nodes and min nodes in game trees have different concerns regarding their value estimation, and different bandit algorithms should be applied accordingly. We develop the Asymmetric-MCTS algorithm, which is an MCTS variant that applies a simple regret algorithm on max nodes, and the UCB algorithm on min nodes.

Both the performance of the CCB-MCTS and Asymmetric-MCTS algorithm has shown good performance on the games of 9×9 Go and 9×9 NoGo. The empirical performance of the Asymmetric-MCTS algorithm also revealed the effectiveness of the applying simple regret bandit algorithm seems to be related to the structure and distribution of the values at the leaf nodes of the game tree.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

“Playing games” has always been an important research topic since the dawn of artificial intelligence (AI) [36]. Apart from being a display of intelligent behaviour, games have a well-defined objective of “winning”, and also they have a robust way of measuring performance, which makes it an ideal vehicle for developing new methods in AI. Therefore, traditional strategic board games, such as Chess, have been deemed as “the drosophila of AI” [24].

1.1 Conventional Methods: $\alpha\beta$ Pruning

The most widely used conventional method for constructing a game-playing program was proposed by Shannon in 1950 [29]. Games can essentially be represented by **game trees**, which is a tree structure that systematically enumerates all possible sequence of each player's move from a given position in the game. Ideally, if we are able to expand the game tree from a given position to every possible final outcome of the game, we will be able to determine the optimal action or strategy for a player to achieve the best possible result.

However, this would not be practical due the the huge size of the game tree. For example, the size of the game tree of Chess is roughly in the order of 10^{120} [29], which is more than the number of atoms in the observable universe. Therefore, we can only approximate this process by constructing the game tree to a certain depth and perform an estimate evaluation of how good or bad are the positions at the leaf nodes.

Conventional game-playing programs consist of two main components: the *evaluation function* and the *search algorithm*.

The **evaluation function** is the component that estimates how good or bad a position is at the leaf node. The evaluation is performed by an overall assessment of a number of criteria and features in the game. For example, commonly used features in Chess are the number of materials, the safety of the king, pawn structure, and so on. The evaluation function typically provides a numeric value indicating its assessment of the situation, and usually the larger the value, the more advantageous is the position to the player in question.

The **search algorithm** dictates how the game tree is expanded and there are two possible strategies:

- **Type A**: (*brute force expansion*) expand every possible sequence.
- **Type B**: (*selective expansion*) only expand the sequences that seems to be promising.

Most conventional game-playing programs adopts the *type A* expansion strategy and the underlying method for searching the optimal move or game-play strategy is the **minimax** algorithm. The minimax algorithm propagates the values of the leaf nodes, which are given by the evaluation function, up to the root node by taking the purpose of each player into account. The player who is applying the minimax algorithm is trying to maximize her advantage, and hence will always take the action that has the highest value. On the other hand, her opponent will try to restrict and minimize the player's advantage and choose the action that leads to the lowest value. The minimax algorithm essentially tries to find the **principal variation**, which is the move sequence if all players behave optimally. There are various algorithms and heuristics that can be applied to reduce the size of the game tree that is necessary to be expanded.

$\alpha\beta$ **pruning** maintains a lower bound α and upper bound β of the evaluation values as the algorithm expands and traverse the game tree. Parts of the tree can be safely discarded or pruned if the value of those parts are sure to be outside of the interval $[\alpha, \beta]$ [20]. The optimal performance of $\alpha\beta$ pruning can be achieved if the best moves are always searched first and then the number of leaf node that is examined would be in the order of $O(b^{d/2})$, where b is the branching factor of the tree, and d is the depth of the expanded tree [20]. Other methods and heuristics, such a killer heuristics [1], quiescence search [18], null-window search [17], can be applied with $\alpha\beta$ pruning to further improve the performance [26].

The victory of IBM's DeepBlue over the World Chess Champion Garry Kasparov in 1997 marks a milestone achievement of the conventional methods in computer game-play [9].

1.2 Monte-Carlo Tree Search (MCTS)

Despite the success of conventional methods in traditional strategic board games, the ancient game of Go seems to be still out of reach. The size of the game tree of Go is in the order of 10^{360}, which is much larger than that of Chess or any other game that has been conquered by conventional methods [35]. Therefore, *type B* strategy with pattern matching and pattern recognition algorithms was the mainstream method in computer Go until the early 2000s [25].

Apart from deterministic methods, stochastic approaches have also been investigated. One of the first known attempts in applying Monte-Carlo methods to computer Go was by Bruegmann in 1993 [7]. The program "Gobble", which was developed by Bruegmann, performs a number of random self-plays and then decides upon its move according to the statistics it has gathered from these self-plays. The idea originated from *simulated annealing*, in which the randomness is controlled by a temperature parameter. However, the performance was relatively poor compared to conventional methods, and hence did not attract much attention.

The idea of *Monte-Carlo tree search* (MCTS) was revisited in the early 2000s, with key revisions such as the selective expansion and sampling of the game tree [5][14]. MCTS has made a significant impact on various fields in AI, especially on the field of computer Go [6].

The key ingredient to the success of MCTS lies in its combination with bandit algorithms, which solves the **multi-armed bandit** (MAB) problem [21]. The MAB problem is a problem where the agent needs to decide whether it should act optimally based on current available information (*exploitation*), or gather more information at the risk of suffering losses incurred by performing suboptimal actions (*exploration*) [22]. By viewing each node in the game tree as a single independent instance of the MAB problem, the MCTS algorithm will essentially be a best-first search algorithm that is guided by bandit algorithms.

The **upper confidence bound on trees** (UCT) algorithm is an MCTS algorithm that applies the **upper confidence bound** (UCB) bandit algorithm on every node in its expanded game tree [21]. The UCT algorithm is currently the most successful MCTS variant and it is used in a wide range of applications [6]. Various heuristics were also developed to further boost the performance of the MCTS algorithm and they are mainly along two major directions: the *incorporation of offline knowledge* and the *utilization of the online knowledge.*

1.2.1 Incorporation of Offline Knowledge

The integration of offline knowledge was mainly focused on using logistic models to improve the quality of the simulations [13][31][19]. These models are also used to determine prior values of newly expanded leaf nodes [19].

Deep neural networks have received a lot of attention in recent years and the application of deep neural networks to computer Go has being of interest in the research community. The initial efforts are mainly in the direction of move prediction by convolutional neural networks [11][23]. Convolutional neural networks are mainly used in the field of computer vision, and they can be applied to the game of Go by viewing the board as a 19×19 image that has three possible values, namely Black, White, and Empty [11].

As the prediction accuracy increases, the next logical step would be to combine them with MCTS [33][30]. Because the convolutional neural networks takes a significant amount of time to perform a prediction, most efforts are focused on applying them in the initialization of prior values of the leaf nodes, rather than in the simulations [33].

A major breakthrough was made by the program *AlphaGo*, which essentially combines convolutional neural networks with the PUCB bandit algorithm [28] on MCTS, and has beaten an elite professional player Lee Sedol in a five-game challenge match [30].

1.2.2 Utilization of Online Knowledge

On the other hand, various investigations in increasing the efficiency of the utilization of online knowledge have also been carried out. One of them is exploring the possibility of using bandit algorithms other than the UCB algorithm with MCTS. **Simple regret bandit algorithms**, which are a class of bandit algorithms that solves the **pure exploration** MAB problem, have attracted some attention in recent years [8].

The pure exploration MAB problem seeks to identify the optimal best arm, rather than gathering as much reward as possible. The pure exploration MAB problem can also be formally stated as the minimization of *simple regret*, which is defined as the difference between the expected reward of the optimal bandit and the bandit that has been identified as the optimal bandit. Because the main objective of game tree search is to determine the best possible action, it has been considered that simple regret bandit algorithms that solve would possibly be a better match for application in MCTS. Therefore, several attempts in applying simple regret bandit algorithms to MCTS have been carried out.

The SR+CR scheme is an MCTS algorithm that applies a simple regret bandit algorithm on the root node, and the UCB algorithm on all other nodes [34]. The SR+CR scheme is based on the argument that edges of the root node represents the decision that the agent needs to make, and hence simple regret bandit algorithm should be applied on the root node while other nodes can remain the same as in the UCT algorithm. The sequential halving on trees (SHOT) algorithm applies the sequential halving algorithm, which minimizes the simple regret by systematically eliminate suboptimal actions from considerations, on every node throughout the tree [10]. The Hybrid MCTS (H-MCTS) algorithm merges the UCB algorithm and the sequential halving algorithm into a single MCTS algorithm by switching the UCB algorithm to the sequential halving algorithm when the number of playouts performed reach a certain threshold on each individual node [27]. Various analytical work on the relation between minimzing simple regret and MCTS has also been carried out, revealing some finer points in their relationship [15].

1.3 Motivation

In this dissertation, we will follow the direction of improving the efficiency of utilizing online knowledge, and try to address the question of *whether minimizing simple regret does lead to better performance in MCTS*. We will also explore *ways to apply simple regret bandit algorithms to MCTS effectively*. Our line of investigate will be along the following questions:

1. What characteristics of a bandit algorithm are desirable for the application to MCTS? How can we modify a bandit algorithm to make it more suitable for application to MCTS?
2. How can we regulate the excessive exploration performed in simple regret bandit algorithms for it to be more suitable for application in MCTS? What is the effect of such a regulation on the performance of the simple regret bandit algorithms?
3. The application of bandit algorithms to MCTS has been symmetric so far, that is the same bandit algorithm are applied on every node. How will an asymmetric application of bandit algorithm perform in MCTS?

We will first introduce some algorithms and concepts on which our proposed method are built on in Chapter 2. We will then proceed to investigate and identify the various characteristics of the improved UCB algorithm which are not desirable for direct application to MCTS, and propose modifications to overcome these deficiencies in Chapter 3. In Chapter 4, we will introduce the *combined confidence bound (CCB)* bandit algorithm, which utilizes *exploration regulating factor* to further regulate the amount of exploration that is performed by the UCB$_{\sqrt{\cdot}}$ bandit algorithm. We will also discuss the implications of the CCB bandit algorithm and its combination with the MCTS algorithm. The asymmetric application of bandit algorithms to MCTS will be discussed in Chapter 5, and finally a conclusion will be given in Chapter 6.

Chapter 2

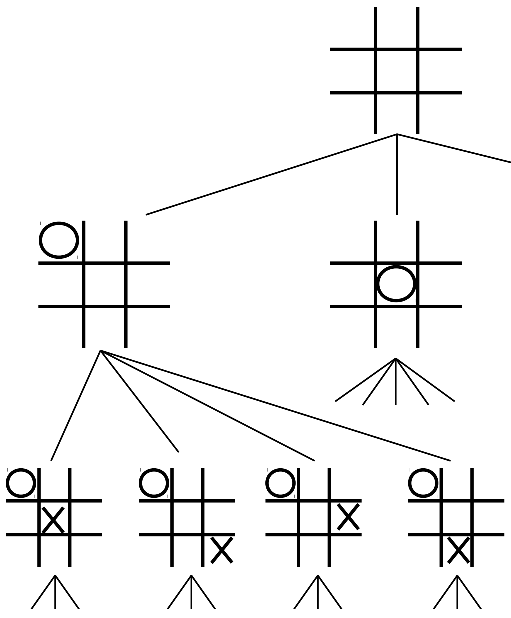
Preliminaries

In this chapter, we will introduce foundational concepts and algorithms on which our proposed methods will be based on. We will begin by introducing the concept of game trees and principal variation. Next, we will introduce the four steps in each iteration of the Monte-Carlo tree search (MCTS). Finally, we will examine the two main settings of the multi-armed bandit (MAB) problem, with the introduction of the UCB algorithm, the improved UCB algorithm, and the $\text{UCB}_{\sqrt{\cdot}}$ algorithm.

2.1 Game Tree Search

A game can essentially be represented by a **game tree**, which is a tree structure that systematically enumerates all possible sequences of each player's move. Figure 2.1 shows the game tree of the game tic-tac-toe. The player with the circle is the first to move, and she has three possible first moves if we take symmetries into consideration. For the move on the top left corner, the game tree can be further expanded by considering the possible response by the opponent, who plays with the cross. By counting this process of expanding all possible moves of each player

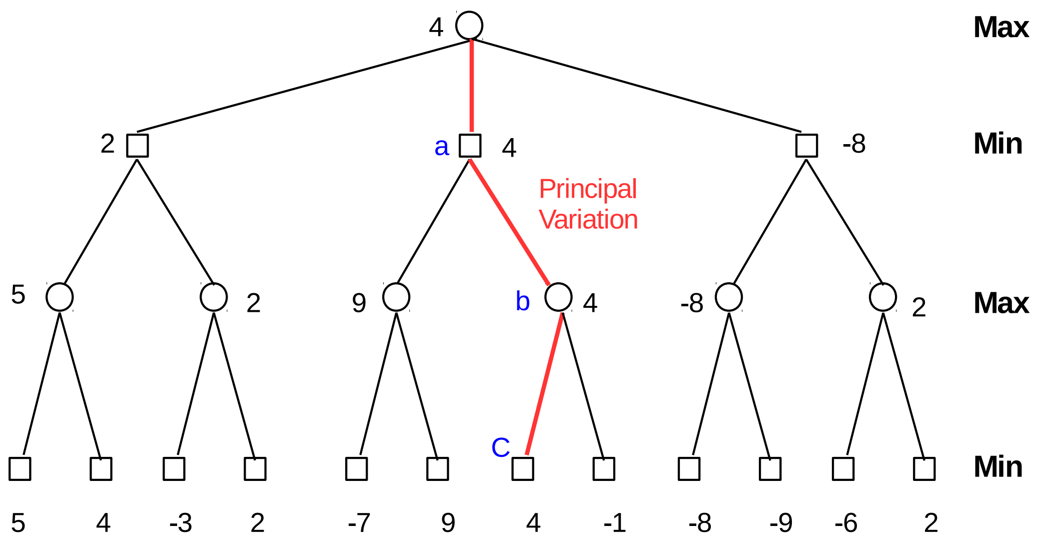
Figure 2.1. Game tree of Tic-Tac-Toe.



at each level until all paths from the root node reach a decisive outcome of the game on all leaf nodes, we will be able to enumerate all possible sequence of plays by the two players. Although this could be achieved for tic-tac-toe, it would not be practical for more complex games, such as Chess, Shogi, and Go, as the size of the game tree would increase at an exponential rate. In practice, we would only expand the tree to a certain depth, and apply an **evaluation function**, which gives an assessment of the outcome of a given position, to every leaf node to approximate this process.

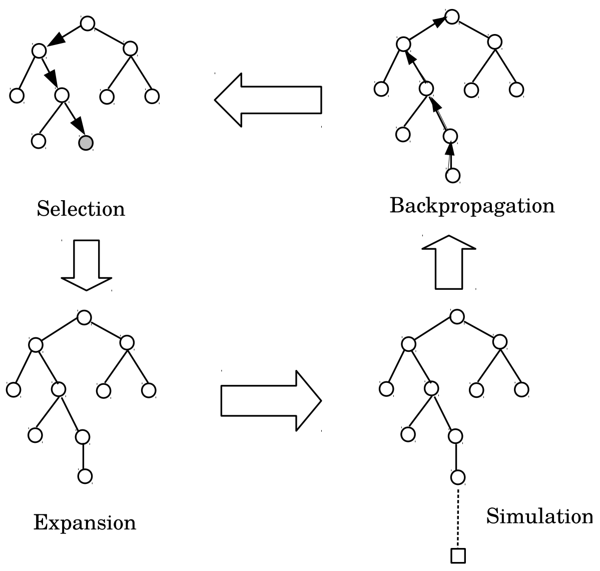
The main objective of game tree search is to decide upon the best possible action to take or the optimal strategy to adopt. This task is essentially finding the **principal variation** within the game tree. The principal variation is a path from the root node to a leaf node in the game tree, which is also the move sequence of the strongest possible moves made by every player on their turn. The principal variation can be identified by the **minimax** algorithm. Figure 2.2 shows an exam-

Figure 2.2. An example of the minimax algorithm



ple of the minimax algorithm. The game tree is expanded to the depth of 3, and an evaluation is performed on the leaf nodes. The evaluation values are given in the point of view of the agent. The higher the value, the more advantageous the agent is in that position. The circle nodes represent the positions when it is the player's turn to move and square nodes represent when it is the opponent's turn to move. When it is the agent's turn to move, she will always try to maximize her advantage by playing the move that leads to the position that has the highest value. On the other hand, the opponent will try to do the opposite by choosing the moves that leads to the position that has the lowest value. Therefore, the circle nodes are **max nodes**, as it will have the largest value of its child nodes, and the square nodes are **min nodes**, as it will have the smallest value of its child nodes. After assigning a value to each node by the minimax algorithm, we will be able to identify the principal variation, which is the path that has the same value on every node. Therefore, the optimal move sequence in Figure 2.2 is the path $\{a, b, c\}$.

Figure 2.3. Monte-Carlo tree search (MCTS)



2.2 Monte-Carlo Tree Search

Monte-Carlo tree search (MCTS) is a sample-based tree search algorithm [6]. MCTS mainly consists of number of iterations, and in each iteration there are four major steps: *selection*, *expansion*, *simulation*, and *backpropagation.*

In the **selection** phase, the MCTS algorithm starts from the root node and ascends down the tree by selecting a child node according to some criteria until it reaches a leaf node. When the MCTS algorithm has reached a leaf node, it will then proceed to the **expansion** phase by expanding a child of the selected leaf node. The MCTS algorithm will then proceed to perform random self-play, or **simulation** until the end of the game or a has a definitive outcome. The MCTS algorithm will finally propagate the result back up to the root node, while updating relevant information along the path in the **backpropagation** phase, and then proceeds to the next iteration. The total number of iterations can be pre-determined or can be run until time or resource runs out.

The key to the success of the upper confidence bound of trees (UCT) algorithm lies in its application of the UCB bandit algorithm as the selection criteria in the *selection* phase.

2.3 The Multi-armed Bandit Problem

The *multi-armed bandit problem* (MAB) is a problem in which the agent is given K slot machines (or "one-armed bandits") and a total number of T plays. The agent can choose to pull the arm of one machine at each play, and the machine will produce a reward $r \in [0, 1]$. There are two main possible objectives that the agent can aim for, and the playing strategy that the agent adopts to achieve these goals is called a *bandit algorithm*.

2.3.1 Cumulative Regret Minimization

The conventional objective in the MAB problem is to accumulate as much reward as possible over the total number T of plays. This task can equivalently be formulated as minimizing the **cumulative regret**, which is defined as

$$CR_T = \textstyle\sum_{t=1}^{T}(r^* - r_{I_t}),$$

where r^* is the mean reward of the optimal arm and r_{I_t} is the reward received from choosing to pull arm I_t on play t. A bandit algorithm is considered to be optimal if it can restrict the cumulative regret to the order of $O(\log T)$ [22].

Algorithm 1 *UCB* algorithm

```
**Initialization**: Play each machine once.
for $t = 1, 2, 3, \cdots T$ do
    play arm $a_i = \arg\max_{i \in K} w_i + c\sqrt{\frac{\log t}{t_i}}$,
    where $w_i$ is the current average reward, $t_i$ is the number of times arm $a_i$ has been sampled.
end for
```

The Upper Confidence Bound (UCB) Algorithm

The upper confidence bound (UCB) algorithm, shown in Algorithm 1, maintains an estimated confidence bound of the average reward of an arm. After playing every arm once in the beginning, the algorithm will always play the arm that has the hight upper confidence bound.

The UCB algorithm has an upper bound of $O(\frac{K \log T}{\Delta})$ on the cumulative regret, where Δ is the difference of the expected reward between a suboptimal arm and the optimal arm [2]. Therefore, it is considered to optimally solve the conventional MAB problem.

The Improved UCB Algorithm

The improved UCB algorithm, shown in Algorithm 2, constitutes an improvement to the UCB algorithm that further restricts the cumulative regret to the order of $O(\frac{K \log T \Delta^2}{\Delta})$ [3].

The improved UCB algorithm essentially maintains a candidate set B_m of potential optimal arms, and proceeds to eliminate the arms that are most likely to be suboptimal from the set B_m. The predetermined number of total plays T is divided into $\lfloor \frac{1}{2} \log_2(\frac{T}{e}) \rfloor$ rounds. In each round, the algorithm samples each arm

Algorithm 2 Improved UCB Algorithm

```
**Input:** A set of arms $A$, total number of plays $T$
**Initialization**: Expected regret $\Delta_0 \leftarrow 1$, a set of candidates arms $B_0 \leftarrow A$
for rounds $m = 0, 1, \cdots, \lfloor \frac{1}{2} \log_2 \frac{T}{e} \rfloor$ do

    (1) Arm Selection:
    for all arms $a_i \in B_m$ do
        for $n_m = \lceil \frac{2 \log(T\Delta_m^2)}{\Delta^2} \rceil$ times do
            sample arm $a_i$ and update its average reward $w_i$
        end for
    end for

    (2) Arm Elimination:
    $a_{max} \leftarrow \textsc{MaximumRewardArm}(B_m)$
    for all arms $a_i \in B_m$ do
        if $(w_i + \sqrt{\frac{\log(T\Delta^2)}{2n_m}}) < (w_{max} - \sqrt{\frac{\log(T\Delta^2)}{2n_m}})$ then
            remove arm $a_i$ from $B_m$
        end if
    end for

    (3) Update $\Delta_m$:
    $\Delta_{m+1} = \frac{\Delta_m}{2}$
end for
```

that is still in the candidate set $n_m = \lceil \frac{2\log(T\Delta_m^2)}{\Delta_m^2} \rceil$ times. The algorithm then proceeds to eliminate those arms whose upper bounds for estimated rewards are lower than the lower bound of the current best arm. Finally, the algorithm halves the estimated difference Δ_m, and proceeds to the next round.

Therefore, the mean reward for arm a_i, where $i \in K$, is estimated to be within

$$w_i \pm \sqrt{\frac{\log(T\Delta_m^2)}{2n_m}} = w_i \pm \sqrt{\frac{log(T\Delta_m^2)\cdot\Delta_m^2}{4\log(T\Delta_m^2)}} = w_i \pm \frac{\Delta_m}{2},$$

where w_i is the current average reward received by sampling arm a_i in round m.

If the total number of plays T is not given beforehand, then the improved UCB can be executed in an episodic fashion, with $T_0 = 2$ for the initial episode and $T_{\ell+1} = T_\ell^2$ for subsequent episodes, at the expense of a looser bound on the cumulative regret.

2.3.2 Simple Regret Minimization

Another possible objective of the MAB problem is to identify the optimal arm after a total number of T plays, rather than maximizing the total accumulated reward. This objective can be formulated as minimizing the **simple regret**, which is defined as

$$SR_T = r^* - r_T,$$

where r^* is the mean reward of the optimal arm, and r_T is the mean reward of the arm that the agent identifies as the optimal arm after T plays.

Since the task is to identify which arm is the optimal arm, the agent needs to gather as much information on each arm as possible, and thus the amount reward accumulated during this process is irrelevant. It has been shown that a trade-off exists between the minimization of the cumulative regret CR_T and the simple regret SR_T, i.e., if CR_T decreases, then SR_T increases, and vice versa [8]. Therefore, a different class of bandit algorithm is needed for this variation of the MAB problem.

Algorithm 3 $UCB_{\sqrt{\cdot}}$ algorithm

```
Initialization: Play each machine once.
for $t = 1, 2, 3, \cdots T$ do
    play arm $a_i = \arg\max_{i \in K} w_i + c\sqrt{\frac{\sqrt{t}}{t_i}}$,
    where $w_i$ is the current average reward, $t_i$ is the number of times arm $a_i$ has been sampled.
end for
Recommend arm $a_i = \arg\max_{i \in K} t_i$
```

The UCB$_{\sqrt{\cdot}}$ Algorithm

The UCB$_{\sqrt{\cdot}}$ algorithm, shown in Algorithm 3, is a bandit algorithm that is able to restrict the simple regret to the order of $O(K \cdot \Delta \exp(-\sqrt{T}))$ [34].

The algorithmic aspect of the UCB$_{\sqrt{\cdot}}$ algorithm is essentially the same as the UCB algorithm in that both algorithms selects the arm that has the current highest upper confidence bound to sample at each play. The two algorithms only differ in their definition of the exploration term of the confidence bound, i.e., the exploration term for the UCB algorithm is $c \cdot \sqrt{\frac{\log T}{t_i}}$ and the exploration term for the UCB$_{\sqrt{\cdot}}$ algorithm is $c \cdot \sqrt{\frac{\sqrt{T}}{t_i}}$, where c is a constant, and t_i is the number of times that arm a_i has been sampled. The UCB$_{\sqrt{\cdot}}$ algorithm recommends the arm that it has been sampled the most as the optimal arm after the total number of T plays.

Chapter 3

Adaptation of the Improved UCB Algorithm

In this chapter, we will first discuss the issues concerning whether a bandit algorithm is suited for application in MCTS. We will then proceed to adapt the improved UCB algorithm to address these issues, and demonstrate the impact of each modification on the bandit algorithm.

3.1 Issues Regarding the Application of Bandit Algorithms to MCTS

The main reason for the success of the UCT algorithm lies is the combination of the UCB bandit algorithm with MCTS. The UCB algorithm essentially controls which part of the game tree one should examine more closely by viewing each node as an independent instance of the MAB problem. However, not every bandit algorithm seems to be suitable for application to MCTS.

Various characteristics of the improved UCB algorithm highlight some properties of bandit algorithms that may be problematic when applied to MCTS:

- **Early explorations**. The improved UCB algorithm seeks to identify the optimal arm through *the process of elimination.* Therefore, it initially tends to devote more plays to sampling suboptimal arms, in order to eliminate them from consideration as early as possible. This property may cause MCTS to focus more on irrelevant parts of the game tree in the early stages, while trying to verify whether those parts can be discarded. Because time and resources are rather restricted when performing a game tree search, the search may still be focusing on these irrelevant areas when time or resources run out, causing the search to miss the more important parts, and hence make erroneous decisions.

- **Not an anytime algorithm**. The improved UCB algorithm requires the total number of plays to be determined beforehand, and hence its various properties may not hold if it is stopped prematurely. Because each node in MCTS is considered as a single independent instance of the MAB problem, the algorithm is likely to be stopped prematurely at nodes that are deeper in the tree or closer to the leaf nodes. The "temporal" solutions provided by these nodes might be erroneous, and these errors may be magnified as they propagate upward toward the root node. Although it is possible to ensure that the required conditions are met at each node, this would be prohibitively expensive, as the necessary number of playouts increases exponentially as more nodes are expanded in MCTS.

Therefore, we propose some modifications to the improved UCB algorithm to address these issues.

Algorithm 4 Combined Confidence Bounds Bandit Algorithm

```
Input: A set of arms $A$, total number of trials $T$
Initialization: Expected regret $\Delta_0 \leftarrow 1$, arm count $N_m \leftarrow |A|$, plays till $\Delta_k$ update $T_{\Delta_0} \leftarrow n_0 \cdot N_m$, where $n_0 \leftarrow \lceil \frac{2\log(T\Delta_0^2)}{\Delta_0^2} \rceil$, number of times arm $a_i \in A$ has been sampled $t_i \leftarrow 0$.
for rounds $m = 0, 1, \cdots T$ do
    (1) Sample Best Arm:
    $a_{max} \leftarrow \arg\max_{i \in |A|}(w_i + \sqrt{\frac{\log(T\Delta_k^2) \cdot r_i}{2n_k}})$, where $r_i = \frac{\sqrt{T}}{t_i}$
    $w_{max} \leftarrow$ CurrentMaxAverageReward($A$)
    $t_i \leftarrow t_i + 1$

    (2) Arm Count Update:
    for all arms $a_i$ do
        if $(w_i + \sqrt{\frac{\log(T\Delta_k^2)}{2n_k}}) < (w_{max} - \sqrt{\frac{\log(T\Delta_k^2)}{2n_k}})$ then
            $N_m \leftarrow N_m - 1$
        end if
    end for

    (3) Update $\Delta_k$ when Deadline $T_{\Delta_k}$ is Reached
    if $m \geq T_{\Delta_k}$ then
        $\Delta_{k+1} = \frac{\Delta_k}{2}$
        $n_{k+1} \leftarrow \lceil \frac{2\log(T\Delta_{k+1}^2)}{\Delta_{k+1}^2} \rceil$
        $T_{\Delta_{k+1}} \leftarrow m + (n_{k+1} \cdot N_m)$
        $k \leftarrow k + 1$
    end if
end for
```

3.2 Combined Confidence Bounds Bandit Algorithm

The combined confidence bounds (CCB) bandit algorithm is a modification of the improved UCB algorithm, and is shown in Algorithm 4. The modifications attempts to address the issues mentioned above while retaining the main characteristics of the improved UCB algorithm as far as possible.

3.2.1 Algorithmic Modifications

We have performed two major algorithmic modifications to the improved UCB algorithm in the CCB bandit algorithm:

- **Greedy optimistic sampling**. We only sample the arm that currently has the highest upper confidence bound, instead of sampling every possible optimal arm n_m times.
- **Maintain candidate arm count**. We only maintain a count of the number of arms that could potentially be the optimal arm for keeping track of when to halve the estimated difference Δ_m, rather than maintaining a candidate set.

Because we only sample the current best arm, we effectively perform an extremely aggressive arm elimination. Arms that are considered suboptimal are not sampled, and hence there is no need to maintain a candidate set. The arms that are initially considered to be suboptimal can possibly still be sampled if their upper confidence bound eventually overtakes the arm that was previously considered to be optimal, while in the improved UCB algorithm these arms are entirely eliminated from consideration. As a result, the guarantee for the confidence bounds of the current best arm will be stronger than in the improved UCB algorithm, because it will be sampled at least n_m times. However, it should be noted that the guarantee for the bounds of other candidates will be weaker. This modification effectively changes the priority of the bandit algorithm from eliminating suboptimal arms to finding the most promising arm as soon as possible. Therefore, if the algorithm is stopped prematurely, then the returned result will more likely be optimal, rather than an arm that the algorithm perceives as suboptimal and is still trying to eliminate.

Despite the fact that it is no longer necessary to maintain a candidate set, we still need to maintain a count of the number of arms that should still be in the candidate set. The reason is that the confidence bounds in the improved UCB algorithm for arm a_i are defined as $w_i \pm \sqrt{\frac{\log(T\Delta_m^2)}{2n_m}}$, and the updates of Δ_m and n_m are both dictated by the number of plays in each round, which is determined by $(|B_m| \cdot n_m)$, i.e., the total number of plays needed to sample each arm in the candidate set B_m a number of n_m times. Therefore, the count of potential optimal arms $|B_m|$ is still required.

3.2.2 Confidence Bound Modifications

Because we have applied the algorithmic modification of performing greedy optimistic sampling, the confidence bounds for the current best arm should be tighter than for other arms, and thus adjustments are also required in the definition of the confidence bound.

The confidence bound in the CCB bandit algorithm for arm a_i in round m is defined as

$$w_i \pm \sqrt{\frac{\log(T\Delta_m^2)\cdot r_i}{2n_m}},$$

where we have added an exploration regulating factor r_i to reflect the fact the current best arm is sampled more than other arms. The most straight forward definition of the exploration regulating factor would be $r_i = \frac{T}{t_i}$. However, a number of other possible definitions exist for r_i, which we will discuss later in further detail. Because $n_m = \lceil \frac{2\log(T\Delta_m^2)}{\Delta_m^2} \rceil$, the expected reward for arm a_i in the CCB bandit algorithm can effectively be estimated as

$$w_i \pm \sqrt{\frac{\log(T\Delta_m^2)\cdot r_i}{2n_m}} = w_i \pm \sqrt{\frac{log(T\Delta_m^2)\cdot\Delta_m^2\cdot r_i}{4\log(T\Delta_m^2)}} = w_i \pm \frac{\Delta_m}{2}\sqrt{r_i}$$

after round m.

3.3 Application of Combined Confidence Bounds to Monte-Carlo Tree Search

The CCB-MCTS algorithm is a variant of the MCTS algorithm in which the CCB bandit algorithm is employed. The details of the CCB-MCTS algorithm are shown in Algorithm 5.

The CCB-MCTS algorithm adopts the same game tree expansion paradigm as the UCT algorithm. The game tree is expanded over a number of iterations, and each iteration consists of four steps: *selection*, *expansion*, *simulation*, and *backpropagation* [21]. The difference is that the tree policy is replaced by the CCB-MCTS algorithm. The CCB bandit algorithm is run on each node in an episodic manner, with a total of $T_0 = 2$ plays in the algorithm in the initial episode, and $T_{\ell+1} = T_\ell^2$ plays in subsequent episodes.

The CCB-MCTS algorithm keeps track of when to update $N.\Delta$ and of the starting point of a new episode by using the variables $N.deltaUpdate$ and $N.T$, respectively. When the number of playouts $N.t$ of the node N reaches the updating deadline $N.deltaUpdate$, the algorithm halves the current estimated regret $N.\Delta$, and calculates the next deadline for halving $N.\Delta$. The variable $N.T$ marks the starting point of a new episode. Hence, when $N.t$ reaches $N.T$, the related variables $N.\Delta$ and $N.armCount$ are re-initialized, and the starting point $N.T$ of the next episode is calculated along with the new $N.deltaUpdate$.

Algorithm 5 Combined Confidence Bound MCTS (CCB-MCTS) Algorithm

```
1: function CombConfBound-MCTS(Node $N$)
2:     $best_{ucb} \leftarrow -\infty$
3:     for all child nodes $n_i$ of $N$ do
4:         if $n_i.t = 0$ then
5:             $n_i.ucb \leftarrow \infty$
6:         else
7:             $n_i.ucb \leftarrow n.w + \sqrt{\frac{\log(N.T \times N.\Delta^2) \times r_i}{2N.k}}$
8:         end if
9:         if $best_{ucb} < n_i.ucb$ then
10:            $best_{ucb} \leftarrow n_i.ucb$
11:            $n_{best} \leftarrow n_i$
12:        end if
13:    end for
14:
15:    if $n_{best}.t = 0$ then
16:        $result \leftarrow$RandomSimulation(($n_{best}$))
17:    else
18:        if $n_{best}$ is not yet expanded then NodeExpansion(($n_{best}$))
19:        $result \leftarrow$ CombConfBound-MCTS(($n_{best}$))
20:    end if
21:
22:    $N.w \leftarrow (N.w \times N.t + result)/(N.t + 1)$
23:    $N.t \leftarrow N.t + 1$
24:
25:    if $N.t \geq N.T$ then
26:        $N.\Delta \leftarrow 1$
27:        $N.T \leftarrow N.t + N.T \times N.T$
28:        $N.armCount \leftarrow$ Total number of child nodes
29:        $N.k \leftarrow \lceil \frac{2 \log(N.T \times N.\Delta^2)}{N.\Delta^2} \rceil$
30:        $N.deltaUpdate \leftarrow N.t + N.k \times N.armCount$
31:    end if
32:
33:    if $N.t \geq N.deltaUpdate$ then
34:        for all child nodes $n_i$ of $N$ do
35:            if $(n_i.w + \sqrt{\frac{\log(N.T \times N.\Delta^2)}{2n.k}}) < (N.w - \sqrt{\frac{\log(N.T \times N.\Delta^2)}{2n.k}})$ then
36:                $N.armCount \leftarrow N.armCount - 1$
37:            end if
38:        end for
39:
40:        $N.\Delta \leftarrow \frac{N.\Delta}{2}$
41:        $N.k \leftarrow \lceil \frac{2 \log(N.T \times N.\Delta^2)}{N.\Delta^2} \rceil$
42:        $N.deltaUpdate \leftarrow N.t + N.k \times N.armCount$
43:    end if
44:    return $result$
45: end function
46:
47: function NodeExpansion(Node $N$)
48:    $N.\Delta \leftarrow 1$
49:    $N.T \leftarrow 2$
50:    $N.armCount \leftarrow$ Total number of child nodes
51:    $N.k \leftarrow \lceil \frac{2 \log(N.t \times N.\Delta^2)}{N.\Delta^2} \rceil$
52:    $N.deltaUpdate \leftarrow N.k \times N.armCount$
53: end function
```

3.4 Experimental Results

First, we will observe the effects of various modifications to the improved UCB algorithm in the MAB problem. We will then proceed to demonstrate the performance of the CCB-MCTS algorithm ($r_i = \frac{T}{t_i}$) on the game of 9×9 Go and 9×9 Nogo.

3.4.1 Performance of Various Modifications on the Improved UCB Algorithm

The experimental settings are in accordance with the multi-armed bandit testbed specified in Sutton et. al [32]. The results are averaged over 2000 randomly generated K-armed bandit tasks. The reward distribution of each bandit is a normal (Gaussian) distribution with mean w_i, $i \in K$, and variance 1. The mean w_i of each bandit for every generated K-armed bandit task is randomly selected according to a normal distribution with mean 0 and variance 1. We have set $K = 60$ in order to more closely simulate the conditions faced by bandit algorithms face when they are employed in MCTS for games with a middle-high branching factor.

The results are illustrated in Figure 3.1. The various modifications correspond to the following algorithms:

- **UCB**: the UCB algorithm.
- **I-UCB**: the improved UCB algorithm.
- **I-UCB (episodic)**: the improved UCB algorithm run episodically.

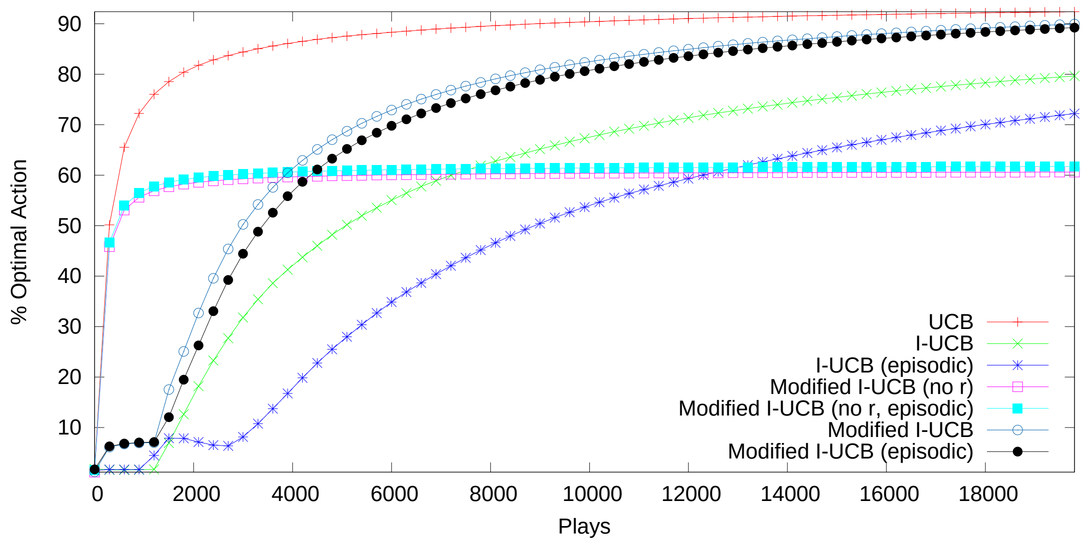
- **Modified I-UCB (no r)**: only algorithmic modifications on the improved UCB algorithm.
- **Modified I-UCB (no r, episodic)**: only algorithmic modifications on the improved UCB algorithm run episodically.
- **Modified I-UCB**: both algorithmic and confidence bound modifications on the improved UCB algorithm.
- **Modified I-UCB (episodic)**: both algorithmic and confidence bound modifications on the improved UCB algorithm run episodically.

The modified I-UCB and modified I-UCB (episodic) algorithms are essentially the CCB bandit algorithm with the regulating factor defined as $r_i = \frac{T}{t_i}$.

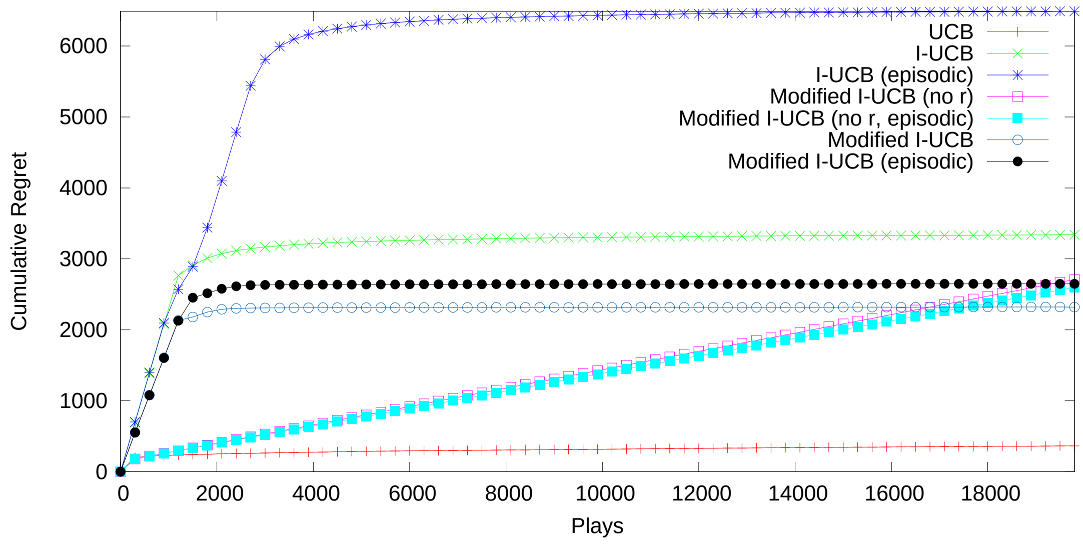
It is surprising to observe that the original improved UCB, i.e., both I-UCB and I-UCB (episodic), produced the worst cumulative regret, which is not consistent with known theoretical results. However, their optimal action percentages increase very rapidly, and are likely to overtake the UCB algorithm if more plays are introduced. This suggests that the improved UCB algorithm does indeed devote more plays to exploration in the early stages.

It can be observed that by making only algorithmic modifications, the bandit algorithm persists on a suboptimal arm, and adding the exploration regulating factor r_i to the confidence bounds solves this problem.

The "slack" in the curves of the algorithms that were run episodically are the points at which a new episode begins. Because the confidence bounds are essentially re-initialized after every episode, extra explorations are effectively performed. Therefore, there are resulting penalties on the performance, which can be observed in both the optimal percentage and the cumulative regret.



(a) Percentage of selecting the optimal arm



(b) Cumulative regret

Figure 3.1. Performance of various modifications on the improved UCB algorithm

Table 3.1. Win rate of CCB-MCTS algorithm ($r_i = \frac{T}{t_i}$) against plain UCT on 9×9 Go

constant C	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1000 playouts	57.1%	55.2%	57.5%	52.2%	58.6%	58.4%	55.8%	55.3%	54.5%
3000 playouts	50.8%	50.9%	50.3%	52.2%	52.2%	54.4%	56.5%	56.0%	54.1%
5000 playouts	54.3%	54.2%	52.4%	51.0%	52.4%	57.5%	54.9%	56.1%	55.3%

3.4.2 Performance of CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm against Plain UCT on 9×9 Go

We will demonstrate the performance of the CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm against the plain UCT algorithm on the game of Go played on a 9×9 board.

For an effective comparison of the two algorithms, no performance enhancing heuristics were applied. The simulations are all pure random simulations without any patterns or simulation policies. A total of 1000 games were played for each constant C setting of the UCT algorithm, each taking turns to play Black. The total number of playouts was fixed to 1000, 3000, and 5000 for both algorithms.

The results are shown in Table 3.1. It can be observed that the performance of the CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm is quite stable against various constant C settings of the plain UCT algorithm, and is roughly on the same level. The CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm seems to have better performance when only 1000 playouts are given, but slightly deteriorates when more playouts are available.

Table 3.2. Win rate of CCB-MCTS algorithm ($r_i = \frac{T}{t_i}$) against plain UCT on 9×9 NoGo

constant C	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1000 playouts	58.5%	56.1%	61.4%	56.7%	57.4%	58.4%	59.6%	56.9%	57.8%
3000 playouts	50.3%	51.4%	53.1%	51.0%	49.6%	54.4%	56.0%	54.2%	53.9%
5000 playouts	45.8%	48.8%	48.5%	49.6%	55.1%	51.3%	51.3%	55.0%	52.7%

3.4.3 Performance of CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm against Plain UCT on 9×9 NoGo

We will demonstrate the performance of the CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm against the plain UCT algorithm on the game of NoGo played on a 9×9 board. NoGo is a misere version of the game of Go, in which the first player that has no legal moves other than capturing the opponent's stone loses.

All the simulations are all pure random simulations, and no extra heuristics or simulation policies were applied. A total of 1000 games were played for each constant C setting of the UCT algorithm, each taking turns to play Black. The total number of playouts was fixed to 1000, 3000, and 5000 for both algorithms.

The results are shown in Table 3.2. We can observe that the CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm significantly dominates the plain UCT algorithm when only 1000 playouts were given, and the performance deteriorates rapidly when more playouts are available, although it is still roughly on the same level as the plain UCT algorithm.

3.5 Discussion

The success of Monte-Carlo tree search (MCTS) is mainly due to its combination with bandit algorithms. However, some characteristics of various bandit algorithms may not be suitable for application in MCTS. The improved UCB algorithm has a better regret upper bound than the UCB algorithm, but because of characteristics such as performing early explorations and not an anytime algorithm, a direct application to MCTS may not be practical. We have proposed some possible modifications to the improved UCB bandit algorithm, making it more suitable for application in MCTS.

The combined confidence bounds (CCB) bandit algorithm is a result of such modifications to the improved UCB algorithm. We have chosen the exploration regulating factor to be $r_i = \frac{T_i}{t_i}$, and has demonstrated the effects of various modifications we have made to the improved UCB algorithm. The CCB-MCTS algorithm is an MCTS algorithm that combines the CCB bandit algorithm with MCTS. We have demonstrated the performance of the CCB-MCTS algorithm on the games of 9×9 Go and 9×9 NoGo.

The results on both 9×9 Go and 9×9 NoGo suggest that the performance of the CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm is comparable to that of the plain UCT algorithm, but scalability seems to be poor. Since the proposed CCB bandit algorithm essentially estimates the expected reward of each bandit by $w_i + \frac{\Delta_m}{2}\sqrt{r_i}$, where $r_i = \sqrt{\frac{T}{t_i}}$, the exploration term converges slower than the of the UCB algorithm, and hence more exploration might be needed for the combined confidence bounds to converge to a "good-enough" estimation value; this might be the reason why CCB-MCTS ($r_i = \frac{T}{t_i}$) algorithm has poor scalability. Therefore, we might able to overcome this problem by trying other definitions for the exploration regulating factor r_i.

Chapter 4

Regulation of Exploration in Simple Regret Minimization

This chapter mainly consists of two major parts. First, we will investigate possible definitions for the exploration regulating factor r_i in the CCB bandit algorithm and its implications. By choosing the definition of $r_i = \frac{T}{t_i}$, we are able to transform the CCB bandit algorithm to the UCB$_{\sqrt{\cdot}}$ algorithm with regulations on the amount of exploration that it performs. We will then examine and analyse the impact of such an regulation on the minimization of simple regret. For the second part, we will demonstrate how commonly used heuristics in MCTS can be applied with the CCB-MCTS ($r = \frac{\sqrt{T}}{t_i}$)algorithm. We have chosen the rapid action value estimation (RAVE) heuristics which increases the efficiency of the utilization of online knowledge, and hence is also one of the most domain independent heuristics used in MCTS.

4.1 Selection of Exploration Regulating Factor

The difference between the definitions of the confidence bounds in the CCB bandit algorithm and the improved UCB algorithm lies in the inclusion of the *exploration regulating factor* r_i. That is, the confidence interval of the improved UCB algorithm for arm a_i is $w_i \pm \frac{\Delta_m}{2}$, and for the CCB bandit algorithm it is $w_i \pm \frac{\Delta_m}{2}\sqrt{r_i}$, after round m.

The exploration regulation factor fulfills two main roles:

1. Tightening the confidence bound for the current best arm, because the CCB bandit algorithm only samples the current best arm.

2. Carrying information across episodes. Because we apply the CCB bandit algorithm to MCTS in an episodic fashion, all of the terms in the confidence bounds will be re-initialized before entering a new episode, with the exception of the average reward w_i. This re-initialization essentially throws away most of the exploration information gained from previous episodes. Therefore, in order to carry exploration information through episodes, the exploration regulating factor will not be re-initialized when entering a new episode.

An obvious choice for the exploration regulating factor is $r_i = \frac{T}{t_i}$, where T is the total number of plays and t_i is the number of times arm a_i has been sampled so far. The more times arm a_i is sampled, the smaller r_i becomes, hence providing a tighter bound for arm a_i. Therefore, the confidence bound effectively becomes

$$w_i \pm \sqrt{\frac{\log(T\Delta_m^2)\cdot r_i}{2n_m}} = w_i \pm \frac{\Delta_m}{2}\sqrt{r_i} = w_i \pm \frac{\Delta_m}{2}\sqrt{\frac{T}{t_i}}.$$

Another possible choice is $r_i = \frac{\log T}{t_i}$. This choice will effectively transform the confidence bounds in the CCB bandit algorithm to

$$w_i \pm \sqrt{\tfrac{\log(T\Delta_m^2)\cdot r_i}{2n_m}} = w_i \pm \tfrac{\Delta_m}{2}\sqrt{r_i} = w_i \pm \tfrac{\Delta_m}{2}\sqrt{\tfrac{\log T}{t_i}},$$

which only differs from the definition of the confidence bound in the UCB algorithm by the extra factor $\frac{\Delta_m}{2}$ in the exploration term. Because we always sample the current best arm in the CCB bandit algorithm, the algorithmic aspect is still consistent with the UCB algorithm, and hence the only difference is the factor $\frac{\Delta_m}{2}$, which dynamically regulates the influence of the exploration term.

Because the task of a game tree search is to identify the best move to make, the identification of the optimal arm in the MAB problem, which is also the minimization of the simple regret, appears to be more compatible with the objective of MCTS. By choosing $r_i = \frac{\sqrt{T}}{t_i}$, the confidence bound in the CCB bandit algorithm will be transformed to

$$w_i \pm \sqrt{\tfrac{\log(T\Delta_m^2)\cdot r_i}{2n_m}} = w_i \pm \tfrac{\Delta_m}{2}\sqrt{r_i} = w_i \pm \tfrac{\Delta_m}{2}\sqrt{\tfrac{\sqrt{T}}{t_i}},$$

which is similar to the previous case considered above. It only differs from the confidence bound in the UCB$_{\sqrt{\cdot}}$ algorithm by the extra factor $\frac{\Delta_m}{2}$ in the exploration term, and the algorithmic aspect is still in keeping with the UCB$_{\sqrt{\cdot}}$ algorithm. The bandit algorithm that minimizes the simple regret attempts to verify that suboptimal arms are indeed suboptimal, and so more plays will be spent on those suboptimal arms. Therefore, the search will end up devoting most of its time to verifying whether some part of the game tree is indeed inferior, and may end before it reaches the relevant part of the tree. The extra $\frac{\Delta_m}{2}$ factor in the CCB bandit algorithm is able to regulate the performance of excessive exploration, but comes at the cost of achieving a looser bound on the simple regret.

4.2 Simple Regret of the CCB Bandit Algorithm

As described above, by defining $r_i = \frac{\sqrt{T}}{t_i}$, the CCB bandit and UCB$_{\sqrt{\cdot}}$ algorithms differ only in the extra $\frac{\Delta_k}{2}$ factor in the exploration term. Because Δ_k is halved after every round, at round k the CCB bandit algorithm is effectively the UCB$_{\sqrt{\cdot}}$ algorithm with the constant in the exploration term set to c_k, and thus the upper confidence bound for arm a_i is

$$ucb_k(a_i) = w_i + c_k \cdot \sqrt{\tfrac{\sqrt{T}}{t_i}},$$

where $c_k = \frac{\Delta_{k-1}}{2} \cdot c_{k-1}$. Let the initial constant be set to $c = c_0$. Then, because $\Delta_0 = 1$, at round k the constant is given by $c_k = \frac{\Delta_{k-1}}{2} \cdot c_{k-1} = \frac{\Delta_0}{2^k} \cdot c_0 = \frac{c}{2^k}$. Therefore, the entire CCB bandit algorithm can be viewed as the progression of the UCB$_{\sqrt{\cdot}}$ algorithm with the constant set to c_k in round k, with $c_k = c/2^k$.

Now, we will introduce two theoretical results concerning the UCB$_{\sqrt{\cdot}}$ and improved UCB algorithms, on which our arguments will be based.

Fact 1. *(Tolpin et. al [34]) For every* $0 < \eta < 1$ *and* $\gamma > 1$*, there exists* τ *such that for every* $T > \tau$ *the probability of a suboptimal arm* a_i *being sampled is bounded by*

$$P_k \leq 2\gamma \exp(-\tfrac{c_k\sqrt{T}}{2}).$$

Fact 2. *(Auer et. al [3]) In the improved UCB algorithm, the probability that a suboptimal arm* a_i *is not eliminated in round* k *(or before) is bounded by*

$$P_e \leq \tfrac{2}{T\Delta_k^2}.$$

First, We will examine a more general algorithm, the CCB$_\delta$ bandit algorithm, which represents the progression of constant settings $\{c_0, c_1, \cdots, c_k\}$ in the UCB$_{\sqrt{\cdot}}$ algorithm, but with the number of plays scheduled in each round defined arbitrarily.

Theorem 1. *For every $0 < \eta < 1$ and $\gamma > 1$, there exists τ such that for any number of samples $T > \tau$ the CCB$_\delta$ bandit algorithm divides T into M rounds, and there are t_k plays in round k. The simple regret of the CCB$_\delta$ bandit algorithm is bounded from above as*

$$SR_\delta \leq 2\gamma|A|\Delta M\delta \exp(-\frac{c\sqrt{T}}{2^{M+1}}),$$

where $\Delta = \max_i \Delta_i$ and $\delta = \max_k(n_k/T)$, with the probability at least $1-\eta$.

Proof. According to Fact 1, for every $0 < \eta < 1$ and $\gamma > 1$ there exists τ such that for any number of samples $T > \tau$, the probability of a suboptimal arm a_i being sampled in round k is $P_k \leq 2\gamma \exp(-\frac{c_k\sqrt{T}}{2})$. Let $\delta_k = (t_k/T)$ be the proportion of the number of plays in round k compared to the total plays T. Then, the probability of suboptimal arm a_i being sampled by the CCB$_\delta$ bandit algorithm over T plays is bounded by

$$P_i = \sum_{k=0}^{M} \delta_k \cdot P_k \leq 2\gamma \sum_{k=0}^{M} \delta_k \exp(-\frac{c_k\sqrt{T}}{2}).$$

Suppose that the difference of the expected reward of suboptimal arm a_i and the optimal arm is Δ_i. Then, the simple regret of the CCB$_\delta$ bandit algorithm is bounded from above as

$$SR_\delta \leq \sum_{i=1}^{|A|} \Delta_i \cdot P_i \leq \sum_{i=1}^{|A|} \Delta_i \cdot (2\gamma \sum_{k=0}^{M} \delta_k \exp(-\frac{c_k\sqrt{T}}{2})).$$

That is,

$$SR_\delta \leq 2\gamma \sum_{i=1}^{|A|} \Delta_i \cdot (\sum_{k=0}^{M} \delta_k \exp(-\frac{c\sqrt{T}}{2^{k+1}})).$$

Let $\Delta = \max_i \Delta_i$ and $\delta = \max_k(t_k/T)$. Then,

$$SR_\delta \leq 2\gamma|A|\Delta M\delta \exp(-\frac{c\sqrt{T}}{2^{M+1}}).$$

□

It can be observed that gradually reducing the influence of the exploration term by halving the constant c incurs the penalty of decreasing the simple regret. The more rounds there are, the weaker the restriction on the simple regret becomes. This is to be expected because when the influence of the exploration term is reduced, the bandit algorithm performs less exploration, which impacts the quality of the final recommendation. Therefore, in order to reduce the impact of reducing the simple regret, a higher proportion of rounds should be earlier than later. That is, $\delta_0 \geq \delta_1 \geq \cdots \geq \delta_M$. However, if $\frac{\sqrt{T}}{2^{M+1}} \leq 1$, then the most dominant term in the bound $\exp(-\frac{c\sqrt{T}}{2^{M+1}})$ will have the effect of increasing rather than decreasing, and hence the total number of rounds should be $M \leq \frac{1}{2}\log T - 1$.

Now, we will proceed to examine the simple regret of the CCB bandit algorithm with $r_i = \frac{\sqrt{T}}{t_i}$.

Theorem 2. *For every $0 < \eta < 1$ and $\gamma > 1$, there exists τ such that for any number of samples $T > \tau$ the simple regret of the CCB bandit algorithm is bounded from above as*

$$SR_{ccb} \leq 4\gamma \exp(2 - \frac{c\sqrt{e}}{4})|A| \sum_{i=1}^{|A|} \Delta_i \cdot \log_2(\frac{T}{e})\frac{\log T}{T^4},$$

with the probability at least $1 - \eta$.

Proof. We begin by deriving the upper bound on the number of plays δ_k in round k divided by the total plays T. By Fact 2, the probability that a suboptimal arm a_i is still not eliminated in round k is bounded by $P_e \leq \frac{2}{T\Delta_k^2}$. Therefore, the arm count in round k can be bounded by $|B_k| = P_e \cdot |A| \leq \frac{2|A|}{T\Delta_k^2}$. Because the number of times that each arm is sampled in the improved UCB algorithm is $n_k = \frac{2\log(T\Delta_k^2)}{\Delta_k^2}$, the proportion of plays in round k compared to the total number of plays T is bounded from above as

$$\delta_k = \frac{(|B_k| \cdot n_k)}{T} \leq \frac{1}{T} \cdot \frac{2|A|}{T\Delta_k^2} \cdot \frac{2\log(T\Delta_k^2)}{\Delta_k^2}.$$

That is,

$$\delta_k \leq \frac{4|A|\log(T\Delta_k^2)}{T^2\Delta_k^4}.$$

Because $\Delta_0 = 1$, it follows that $\Delta_k = \frac{1}{2^k}$, δ_k can be further bounded as

$$\delta_k \leq \frac{4|A|\log(T\Delta_k^2)}{T^2\Delta_k^4} \leq \frac{4|A|\log(T)}{T^2\Delta_M^4},$$

where M is the total number of rounds. As $M = \frac{1}{2}\log_2 \frac{T}{e}$, it follows that $\frac{1}{\Delta_M^4} = \frac{e^2}{T^2}$ and

$$\delta_k \leq \frac{4|A|\log T}{T^2} \cdot \frac{e^2}{T^2}.$$

That is,

$$\delta_k \leq \frac{4e^2|A|\log T}{T^4}.$$

Therefore, by applying the bound from Fact 1, with the fact that $c_k = \frac{c}{2^k}$, the probability of the suboptimal arm a_i being sampled in the CCB bandit algorithm is bounded from above as

$$P_i = \sum_{k=0}^{M} \delta_k \cdot P_k \le \sum_{k=0}^{M} \frac{4e^2|A|\log T}{T^4} \cdot 2\gamma \exp(-\frac{c_k\sqrt{T}}{4}) \le 2\gamma \cdot M \cdot \frac{4e^2|A|\log T}{T^4} \cdot \exp(-\frac{c\sqrt{T}}{4\cdot 2^M}).$$

Considering that $M = \frac{1}{2}\log_2 \frac{T}{e}$, the bound can be further simplified as

$$P_i \le 4\gamma \exp(2 - \frac{c\sqrt{e}}{4})|A| \log_2(\frac{T}{e})\frac{\log T}{T^4}.$$

Therefore, the simple regret of the CCB bandit algorithm is bounded from above as

$$SR_{ccb} = \sum_{i=1}^{|A|} \Delta_i \cdot P_i \le \sum_{i=1}^{|A|} \Delta_i \cdot 4\gamma \exp(2 - \frac{c\sqrt{e}}{4})|A| \log_2(\frac{T}{e})\frac{\log T}{T^4}.$$

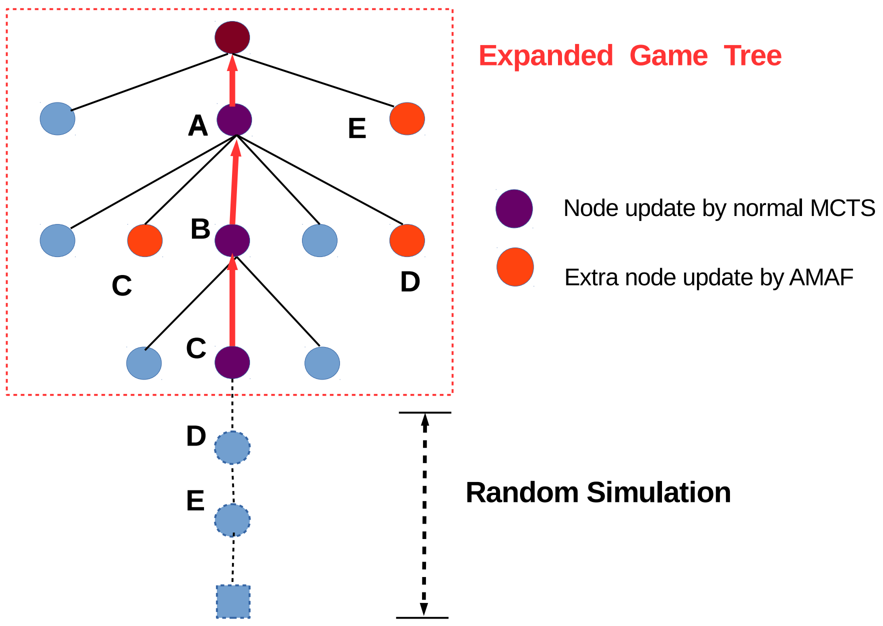
That is,

$$SR_{ccb} \le 4\gamma \exp(2 - \frac{c\sqrt{e}}{4})|A| \sum_{i=1}^{|A|} \Delta_i \cdot \log_2(\frac{T}{e})\frac{\log T}{T^4}.$$

$\square$

We can observe from Theorem 2 that the simple regret of the CCB bandit algorithm decreases at a rate of $O(\frac{(\log T)^2}{T^4})$, which is slightly inferior to that of the UCB$_{\sqrt{\cdot}}$ algorithm. The most dominant exponential term in the more general Theorem 1 reduces to a constant term $\exp(2-\frac{c\sqrt{e}}{4})$ in Theorem 2, hence the penalty of performing multiple rounds rather than using a single constant setting through the whole process is more dominant.

Figure 4.1. Example of all-moves-as-first (AMAF) heuristics



4.3 Applying All-Moves-As-First Heuristics with CCB-MCTS

The CCB-MCTS algorithm with $r_i = \frac{\sqrt{T}}{t_i}$ seems to have good performance in its purest form. So we will now investigate the possibility of applying commonly used heuristics in MCTS to the CCB-MCTS algorithm.

The all-move-as-first (AMAF) heuristic [6][16] is a widely used performance enhancement technique in MCTS, which exploits the fact that in some games, such as Go, the value of a move is often unaffected by moves played elsewhere or when it is played. More specifically, in the *backpropagation* stage of MCTS, instead of only updating the values of the nodes on the path which we descended in the *selection* stage, we also update the values, i.e., the win rate and the simulation time counts, of the sibling nodes if their move also occurred in the deeper depth of the path or in the *simulation* stage according to the result of the playout.

An example is shown in Figure 4.1. The normal MCTS only updates the nodes that are on the path that we descended from in the expanded game tree, that is the A, B, and C nodes. However, when we update upwards and reach node B in our path, we found that moves C and D occurred in our sibling nodes and they both also occurred on the deeper part of our descending path. Hence according the the AMAF heuristic, we will update those sibling nodes as well. The same is also done for the node E in the expanded tree.

Although AMAF allows the information from the playouts to be shared across the related positions or moves in the game tree, it also introduces bias to its value. Therefore, a common way of applying AMAF in MCTS is to use the AMAF value to speed up the convergence rate in the initial stages of a node, and gradually decrease the influence of the AMAF value as the number of playouts on a node exceed a certain point.

Rapid action value estimation (RAVE) [16] is currently the most widely used method for combining AMAF values and the original MCTS values. RAVE combines the win rate of a function by

$$w_{RAVE} = (1 - \beta) \cdot w_{MCTS} + \beta \cdot w_{AMAF},$$

where β is a variable that diminishes as the number of playouts increases. There are various scheduling schemes for β, and one of which is the **minimum MSE schedule** [16]. The minimum MSE schedule, which tries to minimize the mean squared error in the combined estimate, defines the value of β as

$$\beta = \frac{N_{AMAF}}{N_{AMAF}+N_{MCTS}+(N_{AMAF} \cdot N_{MCTS})/D},$$

where N_{MCTS} and N_{AMAF} are the number of playouts performed on the node in MCTS and AMAF sense respectively, and D is the bias between the AMAF value and the MCTS value. The bias D can be viewed as a parameter, which can either be tuned manually or automatically with machine learning methods.

There are two possible ways of applying RAVE to the combined confidence bounds:

Apply RAVE to Win Rate

RAVE can be applied to the combined confidence bound in the same way as it is applied in the UCT-RAVE algorithm [16], by only applying RAVE on the win rate

$$w_{i_{RAVE}} \pm c_\Delta \sqrt{\tfrac{\log(T\Delta_m^2)\cdot r_i}{2n_m}}.$$

Apply RAVE to both Win Rate and Halving Δ_m

RAVE can be further applied to the update of Δ_m by modifying the playout counts for a node to

$$n_{RAVE} = \lfloor (1-\beta)\cdot n_{MCTS} + \beta \cdot n_{AMAF} \rfloor,$$

but the deadline for halving Δ_m remains the same, i.e., the calculation of the deadline will only use MCTS values and no AMAF values. Therefore, RAVE can be applied in two places in the combined confidence bounds

$$w_{i_{RAVE}} \pm c_\Delta \sqrt{\tfrac{\log(T\Delta_{m_{RAVE}}^2)\cdot r_i}{2n_m}}.$$

Note that the scheduling for $w_{i_{RAVE}}$ and $\Delta_{m_{RAVE}}$ should be different, because AMAF values may have different biases in these two terms. The playout count

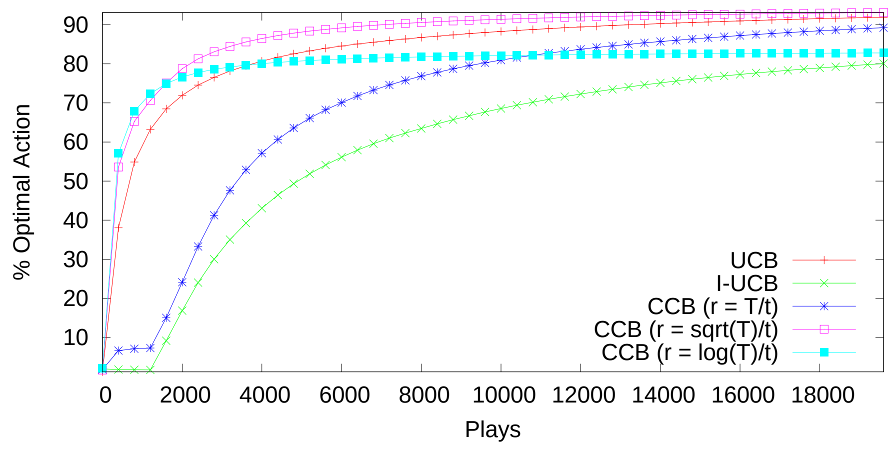
n_{RAVE} is also used for speeding up the episode iteration as well, i.e., the conditions in Algorithm 5 are modified to $N.t_{RAVE} \geq N.deltaUpdate$ and $N.t_{RAVE} \geq N.T$, where $N.t_{RAVE}$ is the RAVE simulation count.

4.4 Experimental Results

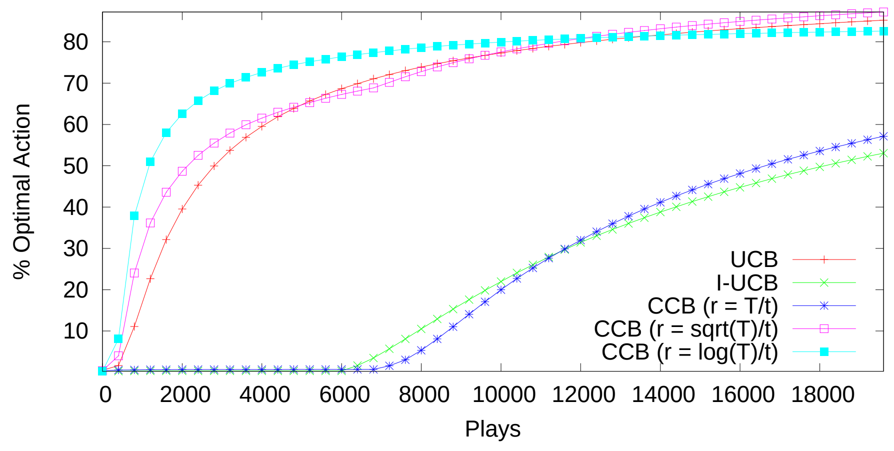
We will first demonstrate the performance of the CCB-bandit algorithm given various definitions of the exploration regulating factor r_i on the MAB problem. Next, we will examine the performance of the CCB-MCTS algorithm when different definitions are given to the exploration regulating factor r_i with and without the application of the RAVE heuristic on 9×9 Go and 9×9 NoGo.

4.4.1 Performance of Various Exploration Regulating Factor

The experimental settings follow the multi-armed bandit testbed that is specified in Sutton et. al [32]. The results are averaged over 2000 randomly generated K-armed bandit tasks. The reward distribution of each bandit is a normal (Gaussian) distribution with the mean w_i, $i \in K$, and variance 1. The mean w_i of each bandit of every generated K-armed bandit task was randomly selected according to a normal distribution with mean 0 and variance 1. Observations were made on the cases of $K = 60$ and $K = 300$.



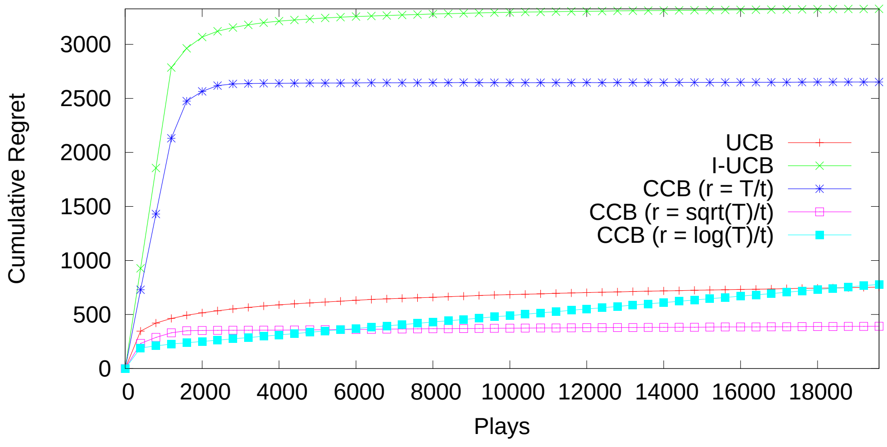
(a) $K = 60$



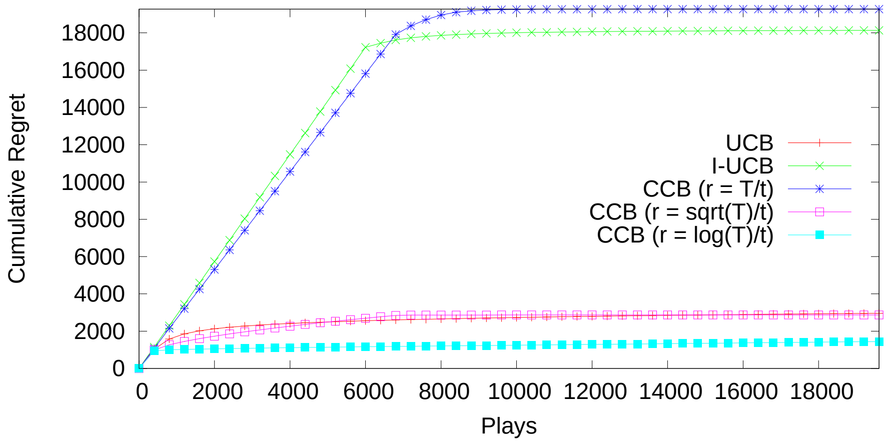
(b) $K = 300$

Figure 4.2. Optimal percentage of various choice of exploration regulating factor in the CCB Bandit Algorithms

The optimal arm selection percentage of each setting of r_i are shown in Figure 4.2. It be observed that the CCB bandit algorithm with $r_i = \frac{\sqrt{T}}{t_i}$ initially has a low optimal action percentage, but gradually overtakes the others after 12000 plays, ending up with the highest percentage of selecting the optimal arm in both $K = 60$ and $K = 300$. It For $r_i = \frac{\log T}{t_i}$, it can be observed that the percentage rapidly decreases despite it having a similar confidence bound with the UCB algorithm. This may be due to the fact that the exploration regulating factor will cause the exploration term to diminish much more rapidly than the UCB algorithm, and hence not enough exploration has been performed. The opposite can be stated for the case of $r_i = \frac{T}{t_i}$, in which the exploration regulating factor may encourage more excessive exploration.

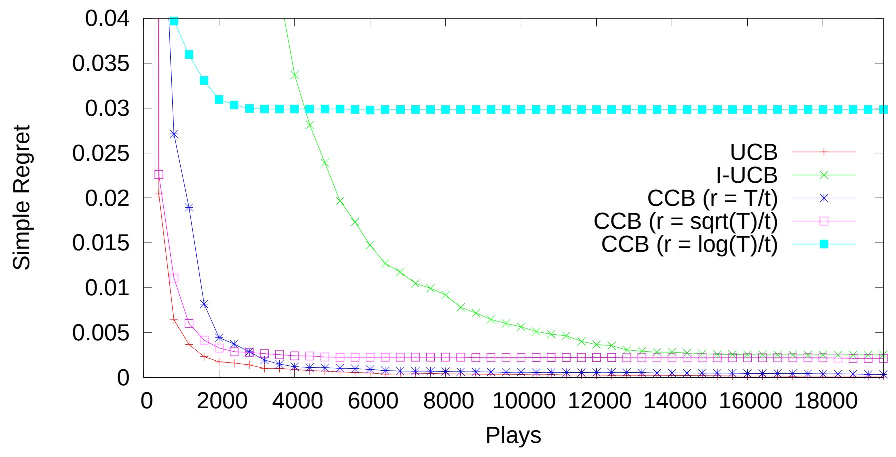


(a) $K = 60$

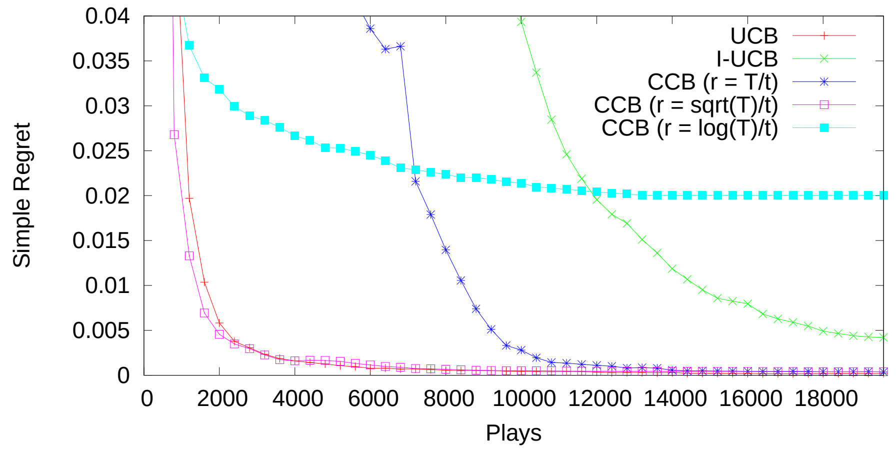


(b) $K = 300$

Figure 4.3. Cumulative regret of various choice of exploration regulating factor in the CCB Bandit Algorithms



(a) $K = 60$



(b) $K = 300$

Figure 4.4. Simple regret of various choice of exploration regulating factor in the CCB Bandit Algorithms

The consequence of the optimal arm selection percentage of each r_i settings are reflected in their cumulative regret and the simple regret. We can observe that the cumulative regret of the CCB bandit algorithm with $r_i = \frac{\sqrt{T}}{t_i}$ is smaller than the other definitions of r_i in $K = 60$, and is the second lowest in $K = 300$, as shown in Figure 4.3. The reason that $r_i = \frac{\sqrt{T}}{t_i}$ has the second lowest cumulative regret in $K = 300$ is due to the fact that it performs more exploration in the early stage than $r_i = \frac{\log T}{t_i}$ as observed in the optimal arm selection percentage. The CCB bandit algorithm with $r_i = \frac{\sqrt{T}}{t_i}$ achieves the best simple regret minimization in $K = 300$, and the third best in $K = 60$, as shown in Figure 4.4.

4.4.2 Performance of Various Exploration Regulating Factor in 9×9 Go and 9×9 NoGo

We will demonstrate the performance of CCB-MCTS algorithm on the game of 9×9 Go and 9×9 NoGo. The komi of 9×9 Go is set to 6.5, that is Black needs to score 6.5 more points than White to win the game. The game of 9×9 NoGo is a misère game of Go, which is roughly an "opposite game of Go", where the basic rules are the same as in Go, but the first player to capture a stone or runs out of legal move loses. In order to make a direct and effective comparison of the impact of bandit algorithms on MCTS, all MCTS algorithms used pure random simulation, without any performance enhancing heuristics.

Since the difference between the CCB-MCTS and the UCT algorithm is only in the extra computational efforts needed for the maintenance and reinitialization of various variables such as expected regret Δ_k, arm count N_m, and deadline T_{Δ_k}, the computation time of the two algorithms are roughly equal to each other when given the same amount of playouts.

We will first investigate the impact of different choice of exploration regulating factor r_i. Table 4.1 shows the win rate of the various settings of the CCB-MCTS algorithm against the plain UCT algorithm. The fourth column is the optimal constant settings of the two algorithms, where c_{ccb} is the constant of the CCB-MCTS algorithm, and c_{uct} is the constant of the plain UCT algorithm. All results

Game	Exploration factor r_i	Win Rate against plain UCT	c_{ccb}	c_{uct}
9×9 Go	$\frac{\log T}{t_i}$	51.39%±2.04%	0.90	0.40
	$\frac{\sqrt{T}}{t_i}$	53.82%±2.03%	0.47	0.37
	$\frac{T}{t_i}$	49.60%±2.04%	1.0	0.40
9×9 NoGo	$\frac{\log T}{t_i}$	52.83%±2.04%	0.10	0.40
	$\frac{\sqrt{T}}{t_i}$	53.26%±2.04%	0.84	0.80
	$\frac{T}{t_i}$	51.78%±2.04%	0.30	0.41

Table 4.1. Win rate of CCB-MCTS with various exploration regulating factor against plain UCT on 9×9 Go and 9×9 NoGo

are the average of 2300 games, with both algorithms taking turns in playing White and Black. A total of 5000 playouts are given to both algorithms for each move.

It can be observed that for $r = \frac{\sqrt{T}}{t_i}$, the CCB-MCTS algorithm achieves a win rate around 53%, showing that there is a slight improvement over the plain UCT algorithm in both games. For both $r_i = \frac{T}{t_i}$ and $r_i = \frac{\log T}{t_i}$, the CCB-MCTS algorithm achieves around 49% to 51% win rate in both games, which is only roughly on the same level of the plain UCT algorithm. Therefore, defining the exploration regulation factor as $r = \frac{\sqrt{T}}{t_i}$ in the CCB-MCTS algorithm seems to produce the best empirical performance.

We will next inspect the scalability of the CCB-MCTS algorithm with $r = \frac{\sqrt{T}}{t_i}$ as the total number of playout increases. The results are shown in Table 4.2. All results are the average of 2300 games, with both algorithms taking turns in playing White and Black.

It can be observed that the CCB-MCTS algorithm has a slight edge over plain UCT when less than or equal to 7000 playouts are given to both algorithms in the game of 9×9 Go, and less than or equal to 9000 playouts in 9×9 NoGo. As

Playouts	Win Rate in 9×9 Go	Win Rate in 9×9 NoGo
1000	$53.52\% \pm 2.04\%$	$52.43\% \pm 2.04\%$
3000	$54.35\% \pm 2.04\%$	$54.13\% \pm 2.04\%$
5000	$53.82\% \pm 2.03\%$	$53.26\% \pm 2.04\%$
7000	$54.17\% \pm 2.04\%$	$53.08\% \pm 2.04\%$
9000	$58.70\% \pm 2.01\%$	$53.47\% \pm 2.04\%$
11000	$57.35\% \pm 2.02\%$	$56.08\% \pm 2.03\%$
13000	$55.39\% \pm 2.03\%$	$55.30\% \pm 2.03\%$

Table 4.2. Scalability of the CCB-MCTS ($r_i = \frac{\sqrt{T}}{t_i}$)on 9×9 Go.

for more than 9000 playouts in 9×9 Go, and 11000 playouts in 9×9 NoGo, the CCB-MCTS algorithm is shown to be superior to the plain UCT algorithm.

4.4.3 CCB-MCTS with AMAF Heuristics on 9×9 Go

Finally, we will investigate the effectiveness of applying AMAF heuristics to the CCB-MCTS algorithm. The exploration regulating factor is set to the definition of $r_i = \frac{\sqrt{T}}{t_i}$.

The performance of the UCT-RAVE algorithm [16], in which only the AMAF heuristic is applied to the win rate of the UCB confidence bound, is shown in Table 4.3. The results of the CCB-MCTS with AMAF heuristics are shown in Table 4.4.

D_{rate} and D_{Δ} are the parameters for RAVE in win rate and Δ_m update, respectively. All the results are the average of 2300 games, with both algorithms taking turns in playing with Black and White. A total of 5000 playouts are given to both algorithms for each move. The settings are $c_{\Delta} = 0.47$ for the CCB-MCTS, and $c = 0.37$ for both the plain UCT and the UCT-RAVE algorithm. The AMAF

Table 4.3. Win rate of the UCT-RAVE algorithm against plain UCT algorithm in 9×9 Go.

D_{rate}	Win Rate
500	55.48% ± 2.03%
1000	58.78% ± 2.01%
2000	59.09% ± 2.01%
4000	61.87% ± 1.99%
6000	**62.70% ± 1.98%**

heuristics are only applied on the CCB-MCTS and UCT-RAVE algorithm, and not on the plain UCT algorithm.

We can observe in Table 4.3 that the UCT-RAVE algorithm can achieve a win rate of 62.70% against plain UCT, and Table 4.4 showns that by applying RAVE only to the win rate estimation term in the CCB-MCTS, the win rate can be significantly improved from 53.82% to 66.61% against plain UCT, an increase of around 13%. If RAVE is also applied to Δ_m update, a further improvement of about 2% may be expected. Observing from the rate of increase of win rate, the CCB-MCTS seems to benefit more from AMAF heuristics.

We have to note that these are just sample settings to show the effectiveness of applying AMAF, and not the optimal settings; therefore there might be still room for further enhancement. It can also be observed that D_{rate} and D_Δ should have different values, where D_{rate} is may be a few hundred times larger than D_Δ.

Table 4.4. Win rate of the CCB-MCTS ($r_i = \frac{\sqrt{T}}{t_i}$) with AMAF heuristics against plain UCT algorithm in 9×9 Go.

D_{rate}	D_{Δ}	Win Rate
No RAVE	No RAVE	53.82% ± 2.03%
500	No RAVE	55.52% ± 2.03%
1000	No RAVE	57.82% ± 2.02%
2000	No RAVE	60.70% ± 2.00%
4000	No RAVE	64.39% ± 1.96%
6000	No RAVE	**66.61% ± 1.93%**
2000	1000	61.30% ± 1.99%
2000	800	60.83% ± 1.99%
2000	400	62.70% ± 1.98%
2000	200	63.13% ± 1.97%
2000	100	62.43% ± 1.98%
2000	50	63.96% ± 1.96%
3000	50	65.13% ± 1.95%
4000	50	67.13% ± 1.92%
5000	50	65.70% ± 1.91%
6000	50	65.57% ± 1.94%
7000	50	**67.26% ± 1.92%**
8000	50	66.30% ± 1.93%

4.5 Discussion

There are many possible definition for the exploration regulating factor r_i in the CCB bandit algorithm. We have made a comprehensive comparison of different definitions and observed their performance on the MAB problem. We have also shown that by choosing the definition $r_i = \frac{\sqrt{T}}{t_i}$, the CCB bandit algorithm is essentially the progression of the UCB$_{\sqrt{\cdot}}$ algorithm with different constant settings, and has the upper bound of $O(\frac{(\log T)^2}{T^4})$ on the simple regret.

The CCB-MCTS algorithm has been shown to have better performance than the plain UCT algorithm on the game of 9×9 Go and 9×9 NoGo when the exploration regulating factor is set to $r_i = \frac{\sqrt{T}}{t_i}$, and also has good scalability in both games. This result seems to suggest the minimization of simple regret in MCTS is effective, however the level of exploration needs to be regulated.

We have also demonstrated two possible ways of applying AMAF heuristics to the combined confidence bounds. The empirical performance of the combined confidence bounds bandit algorithm outperforms the UCB algorithm in the MAB problem. The Combined Confidence Bounds MCTS (CCB-MCTS) has shown to have better performance over the plain UCT algorithm, and also seems to have good scalability. The application of AMAF heuristics greatly enhances the performance of the CCB-MCTS, increasing the win rate over plain UCT by around 15%.

Exploring the possibility of applying the various modifications we have made on the improved UCB algorithm to other bandit algorithms having similar characteristics would be of interest. It would also be interesting to investigate the possibility of other choices of the exploration regulating factor, and how they will perform in different situations.

Chapter 5

Asymmetric Move Selection Strategies

The paradigm of applying bandit algorithms in all MCTS variants is still essentially the same: viewing every node in the game tree as an independent instance of the MAB problem, and applying the same bandit algorithm and heuristics on every node. Although this approach allows MCTS to be applied in general domains other than game-play, it leaves certain properties of the game tree unexploited.

The adversarial game tree consists of two types of nodes: min nodes and max nodes. Max nodes and min nodes generally represent the decision of different players in the game tree, and it is conventional knowledge in various games that which strategy to adopt should be based on which player he or she is. For example, in the game of Go, a komi of 6.5 is given to the Black player, that is the Black player needs to obtain at least 6.5 points more than the White player to win the game. Therefore, the Black player needs to adopt a more aggressive strategy, while the White player can play more conservatively or defensively. The same can also be observed in the game of Chess, where White is generally considered to have the initiative from the start, and hence needs to play more actively, while Black needs to solve its passivity first. Therefore, max nodes and min nodes are intrinsically different from this high-level point of view, and it would be natural to treat them differently, rather than symmetrically.

Some methods have been proposed to reflect the min-max property of game trees in MCTS, but still essentially treat max nodes and min nodes symmetrically, and apply the same heuristic on every node [4]. The SR+CR scheme differs only the root node from other nodes, rather than between max nodes and min nodes [34].

In this chapter, we will investigate the possibility of treating max nodes and min nodes differently by applying different bandit algorithms for each node type in MCTS. We will develop the *Asymmetric-MCTS* algorithm, which applies the UCB$_{\sqrt{\cdot}}$ algorithm on max nodes and the UCB algorithm on min nodes. We will demonstrate its performance on the game of 9×9 Go, 9×9 NoGo, and Othello.

5.1 Asymmetric Move Selection Policy in Monte-Carlo Tree Search

MCTS consists of four major steps: *selection*, *expansion*, *simulation*, and *backpropagation*. Bandit algorithms are mainly applied in the *selection* phase by viewing each node as an independent instance of the MAB problem, where each child node is a single candidate arm. Currently, the most popular variant of MCTS is the UCT algorithm, in which the UCB algorithm is the applied bandit algorithm.

Although this general MCTS paradigm allows it to be applied in a wide range of domains, it leaves a number of properties of the game tree unexploited.

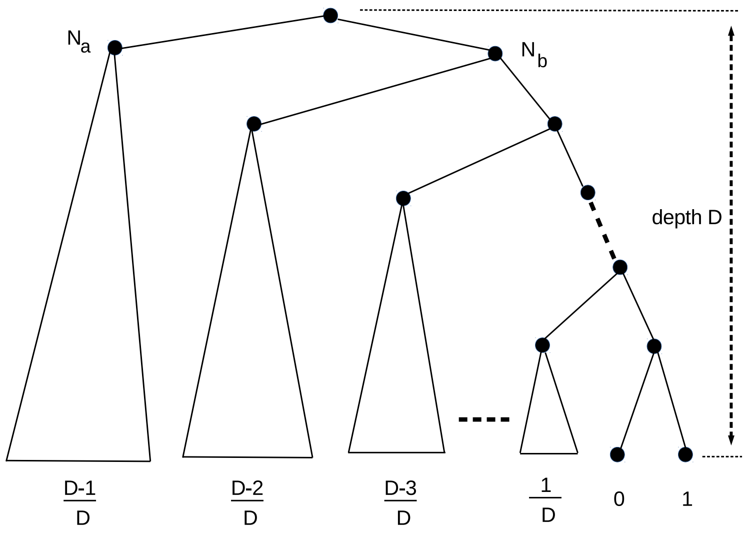
5.1.1 Concerns on Value Estimation of Different Node Types

The role of the bandit algorithm on every node of MCTS is to estimate the value of the node and perform selection according to the estimated value. As the search progresses, the estimation value of the nodes also converges. Although the general goal is to obtain a good estimation as fast as possible, it can be observed that different node types in the game tree have different requirements to their estimated values:

- **Max node**: since the max nodes represent the point of view of the agent, and hence we need to be more certain about the estimated value of each possible decision. Estimations should also be more cautious, and not overly optimistic.
- **Min node**: since the min nodes represent the reaction of the opponent, it is not necessary to determine the best possible reaction of the opponent. Just a *good enough* reaction that is sufficient to refute a decision made by the decision maker will do.

Due to the selection and expansion performed in MCTS, the reward of the MAB problem at each node is non-stationary [12]. For example, consider the binary tree used for constructing a lower bound of the convergence rate of the UCT algorithm [12], shown in Fig. 5.1. The binary tree has the depth of D, and the rightmost path, which is from the root node to the rightmost leaf node, is the optimal path. For a node N at depth $d < D$ on the optimal path, if the left action is chosen, then a reward of $\frac{D-d}{D}$ is received. In other words, all the leaf nodes of the subtree rooted at N have the value of $\frac{D-d}{D}$. If the right action is chosen, the agent can proceed to expand further down the optimal path. At depth $D-1$ of the optimal path, the left action will give the reward 0, and the right action will

Figure 5.1. An example tree for which the UCT algorithm has very poor performance [12].



give the reward 1. Therefore, MCTS will most likely spend the majority of its time expanding the subtrees of the left action along the optimal path, as it seems to be better. Consider the MAB problem at the root node, which has two arms node N_a and node N_b. Since the leaf nodes of the subtree rooted at N_a all have the value of $\frac{D-1}{D}$, the reward produced by N_a will most likely be fixed around $\frac{D-1}{D}$. However, as the search gradually expand down the optimal path, the reward produced by N_b will most likely be along the sequence

$$\{\tfrac{D-2}{D}, \cdots, \tfrac{D-2}{D}, \tfrac{D-3}{D}, \cdots, \tfrac{D-3}{D}, \cdots, \tfrac{1}{D}, \cdots, \tfrac{1}{D}, 1\},$$

instead of being more evened out. Therefore, although the distribution of the reward of the MAB problem on each node is fixed and determined by the values of the leaf node, due to the selection and expansion performed in MCTS, the reward of the MAB problem on each node is biased, and hence affect the estimation made by the bandit algorithms.

Consider the case where a sequence of rewards $(r_1, r_2, r_3, \cdots, r_n)$ are drawn from distribution $\mathcal{N}$, but due to some sampling bias, the sequence is ordered in a non-decreasing order, that is $r_i \leq r_j$ if $i < j$. Therefore, the estimated mean reward will be higher than the true mean reward in the early period of the sequence, and hence causing the agent to be too optimistic. Similarly, if the sequence is in a non-increasing order, that is $r_i \geq r_j$ if $i > j$, then the agent tends to be underestimate the mean in the early stages.

Therefore, one should choose a bandit algorithm that is most likely to resist over optimistic estimations caused by biased reward to deploy on max nodes, and a bandit algorithm that can adapt itself rapidly to provide a "good enough" estimation on min nodes.

5.1.2 Different Bandit Algorithms for Different Node Types

As simple regret and cumulative regret bandit algorithms have different properties, they can be deployed to different node types accordingly to fulfill the requirements on the estimation value of each node type:

- **Max node**: *simple regret* bandit algorithms, which determine the optimal arm, have a higher level of confidence in its estimation value of each arm. Moreover, in order to provide a better estimation of the value of each arm, simple regret bandit algorithms tend to perform more exploration, and spread its sampling more evenly across the candidates, which effectively make it less likely to be too optimistic.
- **Min node**: *cumulative regret* bandit algorithms, which try to accumulate as much reward as possible, tend to focus on the current optimal arm, and adapt rapidly if the current optimal arm changes. Therefore, cumulative

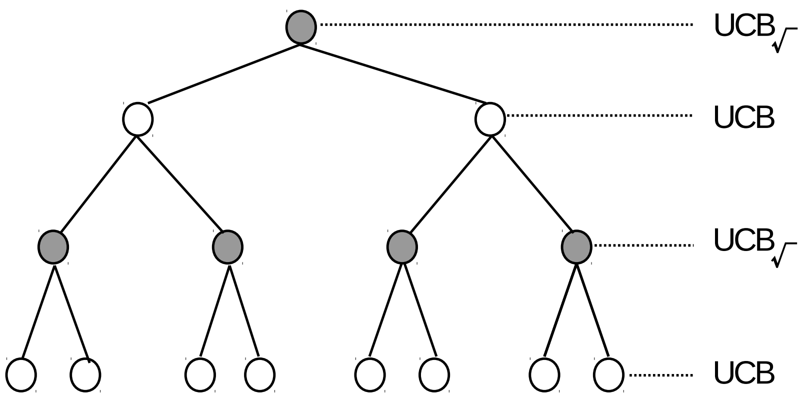
Algorithm 6 Asymmetric-MCTS Algorithm

```
function ASYMMETRIC-MCTS(Node $N$)
    $best_{ucb} \leftarrow -\infty$
    for all child nodes $n_i$ of $N$ do
        if $n_i.t = 0$ then
            $n_i.ucb \leftarrow \infty$
        else
            if $N.type$ is $MAX$ then
                $n_i.ucb \leftarrow n.w + c_s \cdot \sqrt{\frac{\sqrt{N.t}}{n_i.t}}$
            else
                $n_i.ucb \leftarrow n.w + c_r \cdot \sqrt{\frac{\log N.t}{n_i.t}}$
            end if
        end if
        if $best_{ucb} \leq n_i.ucb$ then
            $best_{ucb} \leftarrow n_i.ucb$
            $n_{best} \leftarrow n_i$
        end if
    end for

    if $n_{best}.t = 0$ then
        $result \leftarrow$RANDOMSIMULATION($n_{best}$)
    else
        if $n_{best}$ is not expanded then EXPAND($n_{best}$)
        $result \leftarrow$ ASYMMETRIC-MCTS($n_{best}$)
    end if

    $n_{best}.w \leftarrow (n_{best}.w \times n_{best}.t + result)/(n_{best}.t + 1)$
    $n_{best}.t \leftarrow n_{best}.t + 1$
    $N.t \leftarrow N.t + 1$
    return $result$
end function
```

Figure 5.2. Asymmetric-MCTS algorithm. The gray nodes are max nodes, which the UCB$_{\sqrt{\cdot}}$ bandit algorithm are applied, and the white nodes are min nodes, which the UCB bandit algorithm applied.



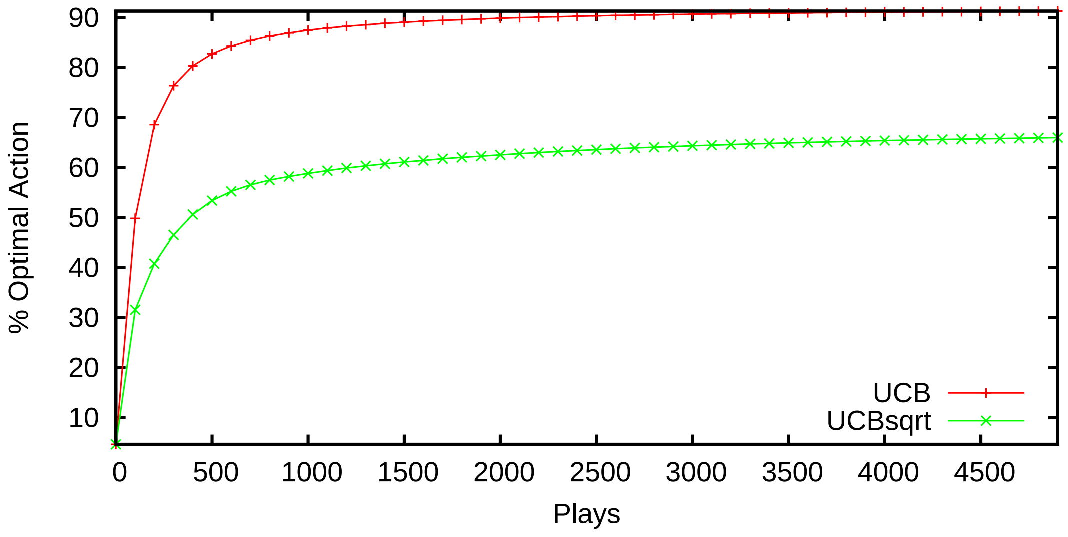
regret bandit algorithms seem to fit the requirement of finding a *good enough* reaction to refute a candidate decision.

The *Asymmetric-MCTS* algorithm, which is shown in Algorithm 6, still retains the four steps in conventional MCTS, namely *selection*, *expansion*, *simulation*, and *backpropagation*. The main characteristic of the Asymmetric-MCTS is that it applies the UCB$_{\sqrt{\cdot}}$ algorithm, which is a simple regret bandit algorithm, on max nodes, and the UCB algorithm, which is a cumulative regret bandit algorithm, on min nodes, as shown in Figure 5.2.

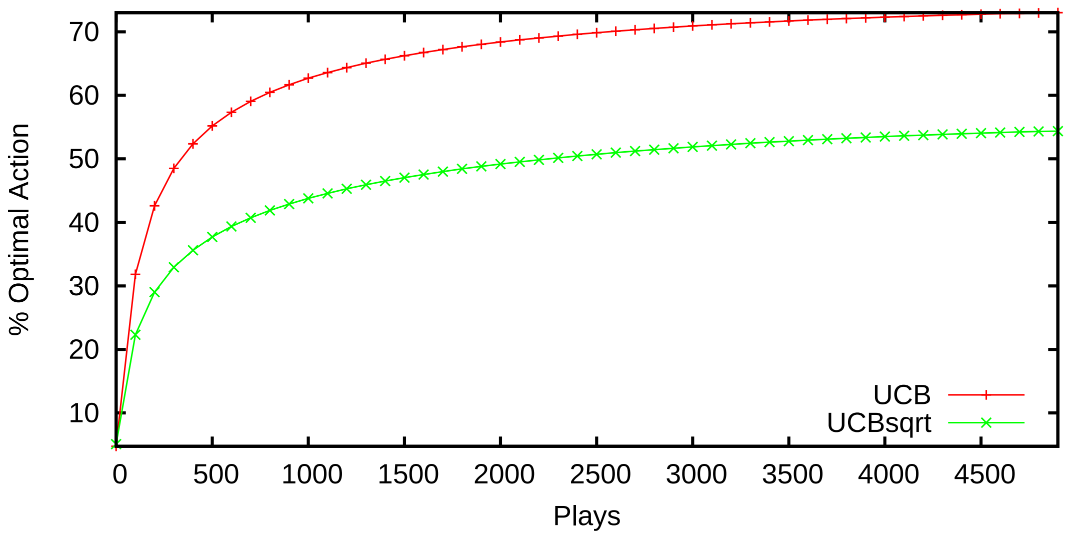
5.2 Experimental Results

In this section, we will first demonstrate the effect of biased reward on the UCB and UCB$_{\sqrt{\cdot}}$ algorithm. We will then proceed to demonstrate the performance of the Asymmetric-MCTS algorithm on the game of 9×9 Go, 9×9 Nogo, and

Figure 5.3. Optimal percentage of biased reward MAB problem

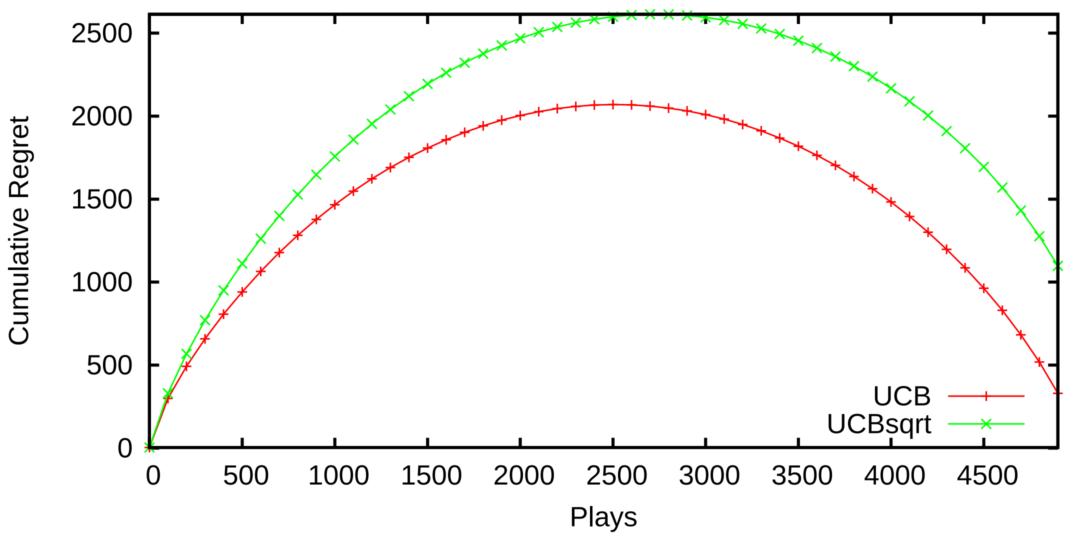


(a) ascending reward

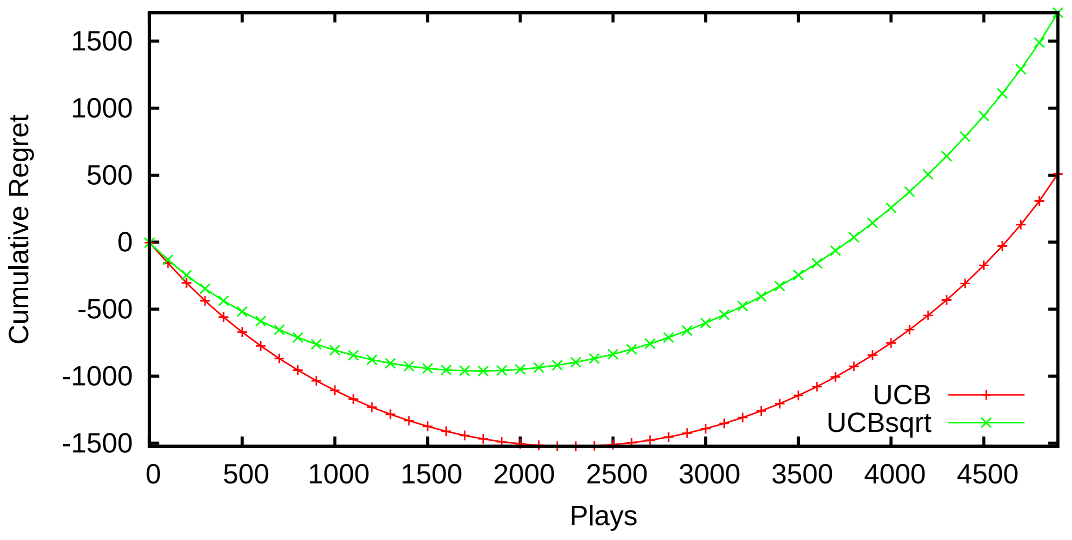


(b) descending reward

Figure 5.4. Cumulative regret of biased reward MAB problem.

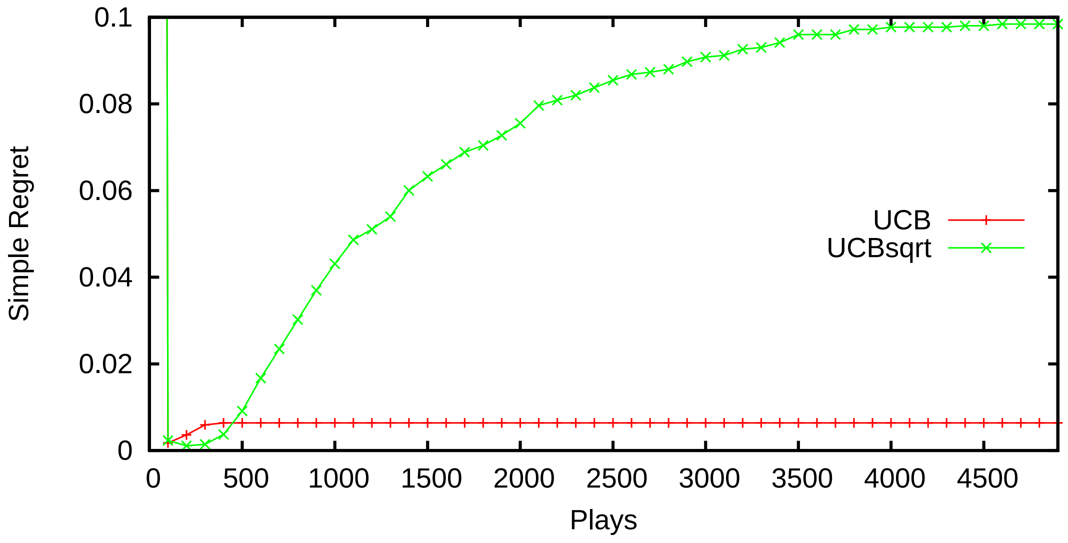


(a) ascending reward

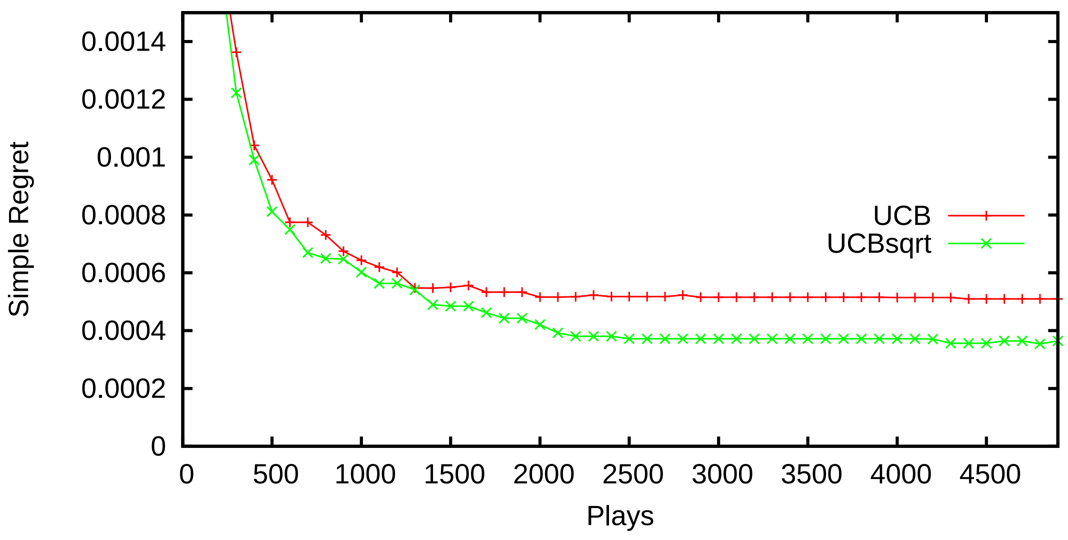


(b) descending reward

Figure 5.5. Simple regret of biased reward MAB problem.



(a) ascending reward



(b) descending reward

Othello. The baseline for all experiments is the plain UCT algorithm. For a direct comparison of the effect of the bandit algorithms, all MCTS algorithms used pure random simulations, and no performance enhancement heuristics were applied. Every experimental result is the average of 2300 games, and each algorithm took turns in playing with Black and White.

5.2.1 Effect of Biased Reward in the MAB problem

We will first demonstrate how bias in the reward affects the performance of the UCB and $\text{UCB}_{\sqrt{\cdot}}$ algorithm. In order to enhance the effect of the biased reward, we will examine two extreme cases: the rewards are biased to be produced in ascending order, and descending order.

The MAB problem testbed mainly follows the settings specified in Sutton et al. [32]. The results are the average of 2000 randomly generated K-armed bandit problems, with $K = 20$. A total of 5000 plays were given to each problem. The rewards of each bandit are first generated from a normal (Gaussian) distribution with the mean w_i, $i \in K$, and variance of 1. The mean w_i of the bandits were randomly selected from a normal distribution with mean 0 and variance 1. To simulate biased rewards, the rewards are then sorted in ascending order and descending order.

It can be observed from Figure 5.3a and Figure 5.3b that regardless of the order in which the rewards are biased, the UCB algorithm has a higher percentage of pulling the optimal arm than the $\text{UCB}_{\sqrt{\cdot}}$ algorithm, and hence suggesting the $\text{UCB}_{\sqrt{\cdot}}$ algorithm tends to distribute its plays more evenly across the candidates. As a result, The UCB algorithm also has lower cumulative regret than the $\text{UCB}_{\sqrt{\cdot}}$ algorithm in both cases, as shown in Figure 5.4a and Figure 5.4b.

However, the UCB$_{\sqrt{\cdot}}$ algorithm has a lower simple regret than the UCB algorithm when the rewards are produced in descending order, as shown in Figure 5.5b. As the UCB$_{\sqrt{\cdot}}$ algorithm performs more exploration, it is able to obtain a better estimation of the mean reward of each candidate, and thus can make more informed recommendations, achieving lower simple regret. On the other hand, the extensive explorations performed by the UCB$_{\sqrt{\cdot}}$ algorithm cause its estimations to be too conservative and pessimistic, and hence lower the quality of the recommendations, as shown in Figure 5.5a.

Therefore, it can be observed that the UCB$_{\sqrt{\cdot}}$ algorithm is more conservative in its estimations, and more resistant to situations where it is more likely to make overly optimistic estimations. One the other hand, the UCB algorithm follows closely the change in the reward with high efficiency.

5.2.2 Performance of the Asymmetric-MCTS on 9×9 Go

We will first investigate the performance of the Asymmetric-MCTS on the game of Go played on the 9×9 board, with the komi of 6.5.

Performance of SR+CR scheme

For comparison, we demonstrate the performance of SR+CR scheme on the game of 9×9 Go. The SR+CR scheme applies the UCB$_{\sqrt{\cdot}}$ bandit algorithm only on the root node, and the UCB bandit algorithm on all other nodes [34]. Table 5.1 shows the win rate of various settings for the constant c_s in the UCB$_{\sqrt{\cdot}}$ algorithm in the SR+CR scheme algorithms. The best constant setting for the UCB algorithm is $c_r = 0.4$ in the SR+CR scheme and the plain UCT algorithm is $c = 0.4$. A total of 5000 playouts are given to both algorithms for each move.

Table 5.1. Win rate of SR+CR scheme against plain UCT algorithm in 9×9 Go.

c_s	SR+CR Scheme
0.1	50.00% ± 2.04%
0.2	51.29% ± 2.04%
0.3	51.80% ± 2.04%
0.4	53.91% ± 2.04%
0.5	53.50% ± 2.04%
0.6	51.19% ± 2.04%
0.7	52.23% ± 2.04%
0.8	51.80% ± 2.04%
0.9	**54.83% ± 2.04%**

It can be observed that the SR+CR scheme achieves around 54% with its best constant setting, which is slightly better than the plain UCT algorithm.

Tuning the C constants

We now proceed to find the best settings for the constant c_r in the UCB algorithm applied on min nodes, and the constant c_s in the the UCB$_{\sqrt{\cdot}}$ algorithm applied on the max nodes, in the Asymmetric-MCTS algorithm. We have found the optimal setting as $c_r = 0.5$ and $c_s = 0.4$, and Table 5.2 shows the win rate of the Asymmetric-MCTS against various constant settings for the plain UCT algorithm. A total of 5000 playouts are given to both algorithms for each move.

It can be observed that even against the best setting $c = 0.3$ of the plain UCT algorithm, the Asymmetric-MCTS still manages to achieve a win rate of around 57.70%. In comparison to the performance of SR+CR scheme, this result suggests that applying the UCB$_{\sqrt{\cdot}}$ algorithm on the max nodes throughout the game tree,

Table 5.2. Win rate of the Asymmetric-MCTS with $c_r = 0.5, c_s = 0.4$ against plain UCT algorithm with various constant c settings on 9×9 Go.

c	Win Rate
0.1	58.96% ± 2.01%
0.2	59.52% ± 2.01%
0.3	**57.70% ± 2.02%**
0.4	58.61% ± 2.01%
0.5	58.30% ± 2.01%
0.6	60.74% ± 2.00%
0.7	62.30% ± 1.98%
0.8	61.91% ± 1.99%
0.9	61.61% ± 1.99%

instead of only on the root node, can make a difference.

Scalability of Asymmetric-MCTS

We now investigate the scalability of the Asymmetric-MCTS as the total number of playouts increases. The result is shown in Table 5.3. The settings for Asymmetric-MCTS is $c_r = 0.5$ and $c_s = 0.4$, and that for the plain UCT algorithm is set to $c = 0.3$.

We can observe that the Asymmetric-MCTS achieves a very good win rate of around 65% over the plain UCT algorithm when 1000 playouts are given, and keeps the win rate to around 60% as more playouts are given to both algorithms. The results suggest that the Asymmetric-MCTS algorithm has very steady performance on the game of 9×9 Go.

Table 5.3. Scalability of the Asymmetric-MCTS on 9×9 Go.

Playouts	Win Rate
1000	65.22% ± 1.95%
3000	60.00% ± 2.00%
5000	57.70% ± 2.02%
7000	59.39% ± 2.01%
9000	59.57% ± 2.01%
11000	62.61% ± 1.98%

5.2.3 Performance of the Asymmetric-MCTS on 9×9 NoGo

We now demonstrate the performance of the Asymmetric-MCTS on the game of Nogo. Nogo is a misere variation of the game of Go, in which the first player who has no legal moves other than capturing the stones of the opponent loses.

Performance of SR+CR scheme

As in 9×9 Go, we first demonstrate the performance of the SR+CR scheme on the game of 9×9 NoGo for comparison. Table 5.4 shows the win rate of various settings for the constant c_s in the SR+CR scheme. The constant setting for the plain UCT algorithm is $c = 0.3$. A total of 5000 playouts are given to both algorithms for each move.

We can observe that SR+CR scheme did extremely well against the plain UCT algorithm, achieving a near 68% win rate against the plain UCT algorithm.

Table 5.4. Win rate of $UCB_{\sqrt{\cdot}}$ MCTS and SR+CR scheme against plain UCT algorithm in 9×9 NoGo.

c_s	SR+CR Scheme
0.1	50.23% ± 2.04%
0.2	51.80% ± 2.04%
0.3	52.70% ± 2.04%
0.4	59.60% ± 2.01%
0.5	64.35% ± 1.96%
0.6	65.63% ± 1.94%
0.7	65.28% ± 1.95%
0.8	66.53% ± 1.93%
0.9	**67.21% ± 1.92%**

Table 5.5. Win rate of the Asymmetric-MCTS with $c_r = 0.5, c_s = 0.4$ against plain UCT algorithm with various constant c settings on 9×9 NoGo.

c	Win Rate
0.1	64.74% ± 1.95%
0.2	66.48% ± 1.93%
0.3	66.17% ± 1.93%
0.4	**62.43% ± 1.98%**
0.5	65.65% ± 1.94%
0.6	67.00% ± 1.92%
0.7	67.65% ± 1.91%
0.8	67.83% ± 1.91%
0.9	69.83% ± 1.88%

Table 5.6. Scalability of the Asymmetric-MCTS on 9×9 NoGo.

Playouts	Win Rate
1000	57.57% ± 2.02%
3000	59.48% ± 2.01%
5000	62.43% ± 1.98%
7000	65.65% ± 1.94%
9000	64.96% ± 1.95%
11000	65.96% ± 1.94%

Tuning the C constants

We now proceed to find the best settings for the constants c_r and c_s the $\text{UCB}_{\sqrt{\cdot}}$ in the Asymmetric-MCTS algorithm. The optimal setting for the Asymmetric-MCTS algorithm is $c_r = 0.5$ and $c_s = 0.4$. Table 5.5 shows the win rate of the Asymmetric-MCTS against various constant settings for the plain UCT algorithm. A total of 5000 playouts are given to both algorithms for each move.

It can be observed that the Asymmetric-MCTS algorithm achieves at least a win rate of 62.43% against the plain UCT algorithm. This result suggests that differentiating max nodes and min nodes also produces very good performance, although the SR+CR scheme might be a better choice on the game of 9×9 NoGo.

Scalability of Asymmetric-MCTS

We now investigate the scalability of the Asymmetric-MCTS as the total number of playouts increases when applied on 9×9 Nogo. The results are shown in Table 5.6. The settings for Asymmetric-MCTS is $c_r = 0.5$ and $c_s = 0.4$, and the constant for the plain UCT algorithm is set to $c = 0.4$.

Table 5.7. Win Rate of the SR+CR scheme against plain UCT algorithm on Othello.

c_s	SR+CR Scheme
0.1	37.74% ± 1.98%
0.2	51.34% ± 2.04%
0.3	53.26% ± 2.04%
0.4	**53.87% ± 2.04%**
0.5	52.35% ± 2.04%
0.6	50.74% ± 2.04%
0.7	50.30% ± 2.04%
0.8	49.43% ± 2.04%
0.9	49.78% ± 2.04%

It can be observed that the Asymmetric-MCTS algorithm dominates the plain UCT algorithm from a total of 1000 playouts to 11000 playouts, and the win rate gradually increases to near 66% when 11000 playouts are given to both algorithms. This result suggests that the effect of differentiating max nodes and min nodes will gradually increase with the number of total playouts.

5.2.4 Performance of the Asymmetric-MCTS on Othello

Finally, we proceed to demonstrate the performance of the Asymmetric-MCTS algorithm on the game of Othello.

Performance of SR+CR scheme

We will first investigate the performance of the SR+CR scheme on Othello for comparison. Table 5.7 shows the win rate of various settings for the constant

Table 5.8. Win rate of the Asymmetric-MCTS with $c_r = 0.7, c_s = 0.4$ against plain UCT algorithm with various constant c settings on Othello.

c	Win Rate
0.1	88.86% ± 1.29%
0.2	81.74% ± 1.58%
0.3	70.48% ± 1.86%
0.4	57.61% ± 2.02%
0.5	53.87% ± 2.04%
0.6	**50.47% ± 2.04%**
0.7	53.39% ± 2.04%
0.8	52.13% ± 2.04%
0.9	53.22% ± 2.04%

c_s in the UCB$_{\sqrt{\cdot}}$ algorithm of the SR+CR scheme. The constant setting for the UCB algorithm in the SR+CR scheme is $c_r = 0.6$ and the plain UCT algorithm is $c = 0.6$. A total of 5000 playouts are given to both algorithms for each move.

It can be observed that the SR+CR scheme can produce a best win rate of around 53%, which is slightly better but still around the same level of the plain UCT algorithm.

Tuning the C constants

We will now proceed to find the best settings for the constants c_r and c_s the UCB$_{\sqrt{\cdot}}$ in the Asymmetric-MCTS algorithm. The optimal setting for the Asymmetric-MCTS algorithm is $c_r = 0.7$ and $c_s = 0.4$. Table 5.5 shows the win rate of Asymmetric-MCTS against various constant settings for the plain UCT algorithm. A total of 5000 playouts are given to both algorithms for each move.

Table 5.9. Scalability of the Asymmetric-MCTS on Othello.

Playouts	Win Rate
1000	52.37% ± 2.04%
3000	52.04% ± 2.04%
5000	53.43% ± 2.04%
7000	50.87% ± 2.04%
9000	51.22% ± 2.04%
11000	53.43% ± 2.04%

It can be observed that the Asymmetric-MCTS algorithm can only achieve a win rate of around 50% against the plain UCT algorithm. This result suggests that differentiating max nodes and min nodes is not effective on the game of Othello, and is around the same level of performance as the plain UCT algorithm.

Scalability of Asymmetric-MCTS

We will now investigate the scalability of the Asymmetric-MCTS as the total number of playouts increases when applied on Othello. The results are shown in Table 5.9. The settings for Asymmetric-MCTS is $c_r = 0.7$ and $c_s = 0.4$, and the plain UCT algorithm is set to $c = 0.4$.

It can be observed that the performance of the Asymmetric-MCTS algorithm does not change with the increase of the number of playouts. The win rate of Asymmetric-MCTS algorithm holds steady around 50%, which is around the same performance level as the plain UCT algorithm.

5.3 Discussion

MCTS has made quite an impact on various fields, and the key to its success lies in the application of bandit algorithms, which solve the MAB problem. In most MCTS variants, the same bandit algorithm and heuristics are applied to every node in the game tree by viewing each node as an independent instance of the MAB problem. The current most dominate variant of MCTS is the UCT algorithm, which applies the UCB bandit algorithm on every node. Although this paradigm has the advantage of allowing MCTS to be applied in a wide spectrum of fields, it leaves a number of properties of the game tree unexploited.

In this chapter, we have proposed that max nodes and min nodes should be treated differently by applying different bandit algorithms according to its intrinsic nature, rather than using the same bandit algorithm throughout the whole tree. We have observed that different node types have different concerns in their estimation value, and the simple regret bandit algorithms seem to fit the requirements of max nodes, and cumulative regret bandit algorithms seem to fulfill the requirement of min nodes.

The Asymmetric-MCTS algorithm, which applies the $\mathrm{UCB}_{\sqrt{\cdot}}$ algorithm on max nodes, and the UCB algorithm on min nodes is proposed based on this observation. The experimental results show that the Asymmetric-MCTS algorithm has a really good performance and scalability on the games of 9×9 Go. The Asymmetric-MCTS also did well on the game of 9×9 NoGo, but the SR+CR scheme seems to be a better choice. However, the Asymmetric-MCTS performed only on par with the UCT algorithm on the game of Othello.

As the main difference between the Asymmetric-MCTS algorithm and the UCT algorithm lies in the application of the $\mathrm{UCB}_{\sqrt{\cdot}}$ algorithm on max nodes, and hence the effectiveness of the Asymmetric-MCTS algorithm seems to depend on whether the UCB algorithm is more likely to be too optimistic in its estimations on max nodes. Therefore, it can be suggested from the experimental results that the UCB algorithm may make too optimistic estimations on max nodes in the game of 9×9 Go, and on the root node in the game of 9×9 Nogo. On the other hand, situations where the UCB algorithm is likely to be too optimistic rarely occurs in Othello.

Applying bandit algorithms other than the UCB and the $\mathrm{UCB}_{\sqrt{\cdot}}$ algorithm would be a natural direction for further investigation. Apart from bandit algorithms, most performance enhancement methods and heuristics in MCTS, also treats each node in the game tree as equal [31][13][30]. Therefore, it would be interesting to further investigate the possibility of developing enhancement heuristics according to node types as well.

Chapter 6

Conclusion

Monte-Carlo Tree Search (MCTS) has made a significant impact on various fields in AI, especially on the field of computer Go [6]. The development of MCTS in recent years can be broadly classified into two main directions: one is the integration of knowledge learnt offline, and the other is increasing the effectiveness of the knowledge accumulated online.

In the direction of increasing the effectiveness of online knowledge the use of various bandit algorithms with MCTS, especially the bandit algorithms that solve the *pure exploration* MAB problem has received much attention in recent years.

Simple regret bandit algorithms aim to identify the optimal arm in a given time constraint, and hence seem to be promising candidates for application in MCTS. However, the cost of exploration is ignored in simple regret bandit algorithms, which may not be desirable in the context of game tree search, and thus an effective application of simple regret bandit algorithms to MCTS is far from trivial.

We have proposed the combined confidence bounds, which utilize the Δ_m term in the confidence bounds of the improved UCB algorithm to dynamically adjust the influence of the exploration term of confidence bounds of the $UCB_{\sqrt{\cdot}}$ algorithm, hence regulating the cost of exploration. We have also demonstrated two possible ways of applying AMAF heuristics to the combined confidence bounds. The empirical performance of the combined confidence bounds bandit algorithm outperforms the UCB algorithm in the MAB problem. The Combined Confidence Bounds MCTS (CCB-MCTS) has shown to have better performance over the plain UCT algorithm, and also seems to have good scalability. The application of AMAF heuristics greatly enhances the performance of the CCB-MCTS, increasing the win rate over plain UCT by around 15%.

Another possible approach is based on the observation that max nodes and min nodes should be treated differently by applying different bandit algorithms according to its intrinsic nature, rather than using the same bandit algorithm throughout the whole tree. We have observed that different node types have different concerns in their estimation value, and the simple regret bandit algorithms seem to fit the requirements of max nodes, and cumulative regret bandit algorithms seem to fulfill the requirement of min nodes.

The Asymmetric-MCTS algorithm, which applies the UCB$_{\sqrt{\cdot}}$ algorithm on max nodes, and the UCB algorithm on min nodes is proposed based on this observation. The experimental results show that the Asymmetric-MCTS algorithm has a really good performance and scalability on the games of 9×9 Go. The Asymmetric-MCTS also did well on the game of 9×9 NoGo, but the SR+CR scheme seems to be a better choice. However, the Asymmetric-MCTS performed only on par with the UCT algorithm on the game of Othello.

The recent breakthrough by AlphaGo seems to suggest a good prior value can greatly enhance the performance of MCTS [30]. Although, the power consumption used for training and using the convolutional neural networks is quite large, making not at generally applicable as it could be. Exploration into the possibilities of further exploitation of online knowledge may lead to a simpler model for prior value estimation would be interesting. Also, the construct of the convolutional neural networks are also still symmetric. Therefore, it would be interesting to investigate the possibility of training different neural networks for different types of nodes.

Bibliography

[1] S. G. Akl and M. M. Newborn, *The principal continuation and the killer heuristic*, in Proceedings of the 1977 Annual Conference, ACM '77, 1977, pp. 466–473.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer, *Finite-time analysis of the multiarmed bandit problem*, Maching Learning, 47 (2002), pp. 235–256.

[3] P. Auer and R. Ortner, *UCB revisited: improved regret bounds for the stochastic multi-armed bandit problem*, Periodica Mathematica Hungarica, 61 (2010), pp. 55–65.

[4] H. Baier and M. H. M. Winands, *Monte-carlo tree search and minimax hybrids with heuristic evaluation functions*, in Proceedings of 3rd Workshop on Computer Games, Held in Conjunction with the 21st European Conference on Artificial Intelligence, CGW 2014, 2014, pp. 45–63.

[5] B. Bouzy, *Associating shallow and selective global tree search with Monte Carlo for 9x9 Go*, in Proceedings of the 4th International Conference on Computers and Games, CG 2004, 2004, pp. 67–80.

[6] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and

S. Colton, *A survey of Monte Carlo tree search methods*, IEEE Transactions on Computational Intelligence and AI in Games, 4 (2012), pp. 1–43.

[7] B. Bruegmann, *Monte Carlo Go*, unpublished manuscript, (1993).

[8] S. Bubeck, R. Munos, and G. Stoltz, *Pure exploration in multi-armed bandits problems*, in The Proceedings of the 20th International Conference on Algorithmic Learning Theory, 2009, pp. 23–37.

[9] M. Campbell, A. Hoane, and F. hsiung Hsu, *Deep Blue*, Artificial Intelligence, 134 (2002), pp. 57 – 83.

[10] T. Cazenave, *Sequential halving applied to trees*, IEEE Transactions on Computational Intelligence and AI in Games, 7 (2015), pp. 102–105.

[11] C. Clark and A. Storkey, *Training deep convolutional neural networks to play Go*, in Proceedings of The 32nd International Conference on Machine Learning, ICML15, 2015.

[12] P.-A. Coquelin and R. R. Munos, *Bandit algorithms for tree search*, in Proceedings of 2007 Uncertainty in Artificial Intelligence, UAI07, 2007.

[13] R. Coulom, *Computing Elo ratings of move patterns in the game of Go*, ICGA Journal, 30 (2007), pp. 198 – 208.

[14] R. Coulom, *Efficient selectivity and backup operators in Monte-Carlo tree search*, in Proceedings of the 5th International Conference on Computers and Games, CG 2006, 2007, pp. 72–83.

[15] Z. Feldman and C. Domshlak, *Simple regret optimization in online planning for Markov decision processes*, Journal of Artificial Intelligence Research, 51 (2014), pp. 165–205.

[16] S. Gelly and D. Silver, *Monte-carlo tree search and rapid action value estimation in computer Go*, Artificial Intelligence, 175 (2011), pp. 1856 – 1875.

[17] G. Goetsch and M. S. Campbell, *Experiments with the null-move heuristic*, Computers, Chess, and Cognition, (1990), pp. 159–168.

[18] L. Harris, *The heuristic search and the game of Chess. a study of quiescence, sacrifices, and plan oriented play*, Computer Chess Compendium, (1988), pp. 136–142.

[19] S.-C. Huang, R. Coulom, and S.-S. Lin, *Monte-Carlo simulation balancing in practice*, in Proceedings of the 26th 7th International Conference on Computers and Games, CG 2010, 2010.

[20] D. E. Knuth and R. W. Moore, *An analysis of alpha-beta pruning*, Artificial Intelligence, 6 (1975), pp. 293 – 326.

[21] L. Kocsis and C. Szepesvári, *Bandit based Monte-Carlo planning*, in Proceedings of the 17th European Conference on Machine Learning, ECML'06, 2006, pp. 282–293.

[22] T. Lai and H. Robbins, *Asymptotically efficient adaptive allocation rules*, Advances in Applied Mathematics, 6 (1985), pp. 4 –22.

[23] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, *Move evaluation in Go using deep convolutional neural networks*, in Proceedings of 2015 International Conference on Learning Representations, ICLR15, 2015.

[24] J. McCarthy, *Chess as the drosophila of AI*, Computers, Chess, and Cognition, (1990), pp. 227–237.

[25] M. Müller, *Computer Go*, Artificial Intelligence, 134 (2002), pp. 145 – 179.

[26] M. Newborn, *Computer Chess*, Academic Press, 1975.

[27] T. Pepels, T. Cazenave, M. H. M. Winands, and M. Lanctot, *Minimizingsimpleandcumulativeregret inMonte-Carlotreesearch*, in Proceedings of 3rd Workshop on Computer Games, Held in Conjunction with the 21st European Conference on Artificial Intelligence, CGW 2014, 2014, pp. 1–15.

[28] C. D. Rosin, *Multi-armed bandits with episode context*, Annals of Mathematics and Artificial Intelligence, 61 (2011), pp. 203–230.

[29] C. E. Shannon, *Programming a computer for playing Chess*, Philosophical Magazine, (1950), pp. 256–275.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of Go with deep neural networks and tree search*, Nature, 529 (2016), pp. 484–489.

[31] D. Silver and G. Tesauro, *Monte-Carlo simulation balancing*, in Proceedings of the 26th International Conference on Machine Learning, ICML09, 2009.

[32] R. Sutton and A. G. Barto, *Introduction to reinforcement learning*, MIT Press, 1st ed., 1998.

[33] Y. Tian and Y. Zhu, *Better computer Go player with neural network and long-term prediction*, in Proceedings of 2016 International Conference on Learning Representations, ICLR16, 2016.

[34] D. Tolpin and S. Shimony, *MCTS based on simple regret*, in Proceedings of the 26th AAAI Conference on Artificial Intelligence, AAAI12, 2012.

[35] J. Tromp and G. Farnebäck, *Combinatorics of Go*, Springer Berlin Heidelberg, 2007, pp. 84–99.

[36] A. M. Turing, *Computing machinary and intelligence*, Mind, 59 (1950), pp. 433–460.

Publications

I. Related Publications

Journal

- Y.-C. Liu and Y. Tsuruoka, "Modification of Improved Upper Confidence Bounds for Regulating Exploration in Monte-Carlo Tree Search", *Theoretical Computer Science*, 2016.

International Conference

- Y.-C. Liu and Y. Tsuruoka, "Regulation of Exploration for Simple Regret Minimization in Monte-Carlo Tree Search", *Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games (CIG-15)*, 2015.
- Y.-C. Liu and Y. Tsuruoka, "Adapting Improved Upper Confidence Bounds for Monte-Carlo Tree Search", *Proceedings of the 14th International Conference on Advances in Computer Games (ACG-15)*, 2015.

Manuscript

- Y.-C. Liu and Y. Tsuruoka, "Asymmetric Move Selection Strategies in Monte-Carlo Tree Search: Minimizing the Simple Regret at Max Nodes", *arXiv:1605.02321*, 2016.

II. Other Publications

International Conference

- Y.-T. Chiang, T.-S. Hsu, Y.-C. Liu, C.-H. Shen, D.-W. Wang, C.-J. Liau, and J. Zhan, "An Information-Theoretic Approach for Secure Protocol Composition", *Proceedings of the 10th International Conference on Security and Privacy in Communication Networks (SecureComm'14)*, 2014.

Domestic Conference

- Y.-C. Liu and Y. Tsuruoka, "Prior Estimation in UCT based on the Sequential Halving Algorithm", *Proceedings of the 18th Game Programming Workshop 2014 (GPW-14)*, 2014.
- Y.-C. Liu, M. Miwa, Y. Tsuruoka, T.-S. Hsu. and T. Chikayama, "Trend Oriented Opening Book Construction", *Proceedings of the 18th Game Programming Workshop 2013 (GPW-13)*, 2013.

Book Chapter

- Y.-C. Liu and T.-S. Hsu, "Monte-Carlo Tree Search",*Theory of Computer Games*, 2016.

Report

- Y.-C. Liu, K.-Y. Wu, and S.-J. Yen, "Developments of Computer 55Shogi in Taiwan", *TCGA Computer Games Workshop 2014.*

Acknowledgement

I would like to express my sincere gratitude to my supervisor Prof. Tsuruoka for the continuous support of my Doctoral study, for his patience, motivation, and guidance. I have received countless advice and suggestions from Prof. Tsuruoka, which inspired me to strive for higher standard in research and life in general. I could not have imagined having a better mentor for my Doctoral study. I am very grateful and honoured that he would have me as his student.

I would also sincerely thank the rest of my dissertation committee: Prof. Aida, Prof. Sato, Prof. Taura, and Prof. Hasegawa, for their invaluable suggestions, and also for their insightful questions which allowed me to view my work from various perspectives.

I would like to give a special thanks to Prof. Chikayama for his guidance and help during my initial year of my Doctoral study. He introduced me to Prof. Tsuruoka, and also provided me with a lot of opportunities to widen my scope and experience.

My sincere thank also goes to the members of Tsuruoka laboratory: Prof. Miwa, Dr. Pontus, Wan-san, Kameko-san, Mizukami-san, Hashimoto-san, Eriguchi-san, Murakami-san for the countless discussions and the great times we had together. I would also like to thank Prof. Shun-Chin Hsu, Prof. Tsan-Sheng Hsu, Dr. Yi-Ting Chiang, and Dr. Hung-Jui Chang for their support and advice.

Finally, I would like to express my sincerest gratitude to my father Ru-Shi Liu, my mother Shu-Fen Hu, and my brother Yun-Ping Liu. Their unconditional love and support allowed me to pursue my goals and dreams.