Regulation of Exploration for Simple Regret Minimization in Monte-Carlo Tree Search

Yun-Ching Liu
Graduate School of Engineering
University of Tokyo
cipherman@logos.t.u-tokyo.ac.jp

Yoshimasa Tsuruoka
Graduate School of Engineering
University of Tokyo
tsuruoka@logos.t.u-tokyo.ac.jp

***Abstract*—The application of multi-armed bandit (MAB) algorithms was a critical step in the development of Monte-Carlo tree search (MCTS). One example would be the UCT algorithm, which applies the UCB bandit algorithm. Various research has been conducted on applying other bandit algorithms to MCTS. Simple regret bandit algorithms, which aim to identify the optimal arm after a number of trials, have been of great interest in various fields in recent years. However, 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, which utilize the characteristics of the confidence bounds of the improved UCB and UCB$_{\sqrt{\cdot}}$ algorithms to regulate exploration for simple regret minimization in MCTS. We will demonstrate the combined confidence bounds bandit algorithm has better empirical performance than that of the UCB algorithm on the MAB problem. We will show that the combined confidence bounds MCTS (CCB-MCTS) has better performance over plain UCT on the game of 9×9 Go, and has shown good scalability. We will also show that the performance of CCB-MCTS can be further enhanced with the application of all-moves-as-first (AMAF) heuristic.**

I. Introduction

Monte-Carlo Tree Search (MCTS) is a search algorithm which determines the best course of action by systematically performing a number of simulations. MCTS has the advantage of only needing a minimal amount of domain knowledge to achieve reasonable results, and has made a significant impact on various fields of research in AI [4]. The multi-armed bandit problem (MAB) is a decision making problem where the agent faces a number of options, and tries to solve the dilemma of whether to make the best decision based on the information at hand (*exploitation*), or to gather more information at the expense of not performing optimally (*exploration*)[1]. *Bandit algorithms* resolve this dilemma, and the UCB algorithm is one of those that can optimally solve the MAB problem [2]. The UCT algorithm is the application of the UCB algorithm to MCTS, and is also the most successful and widely used MCTS variant [3][4]. Various research on the application of other bandit algorithms, such as KL-UCB [15], Thompson sampling [16], sequential halving [14], to MCTS has also been carried out, and has shown some interesting results [6][7][8][9][10].

In recent years, the *pure exploration* MAB problem has been of interest in the research community [5]. The pure exploration MAB problem modifies the objective to identify which option would most likely be the optimal choice after a specific number of trials, and can be equivalently formulated as minimizing *simple regret*, which is the difference between the mean reward of the optimal option and the final identified choice. This objective seems to be a better fit for the task of game tree search. Therefore, investigations on applying various bandit algorithms for solving the pure exploration MAB problem, or *simple regret bandit algorithms*, to MCTS have shown some promising results. The SR+CR scheme, where the SR stands for simple regret and CR stands for cumulative regret, is an MCTS algorithm that applies the simple regret bandit algorithms to the root node, and the UCB algorithm on all other nodes [6]. The sequential halving on trees (SHOT) algorithm applies the sequential halving algorithm [14] to MCTS, and has shown to have superior performance over UCT on the game of NoGo. The Hybrid MCTS (H-MCTS) applies both the UCB and the sequential halving algorithm to MCTS, by switching from UCB to sequential halving after a predetermined number of simulations have been performed on a node [8]. However, one downside of simple regret bandit algorithms is that they do not take the cost of exploration into account. This may not be ideal in the context of game tree search, especially when time and resources are rather restricted.

The improved UCB algorithm is a modification of the UCB algorithm, and has an improved theoretical bound [11]. The $UCB_{\sqrt{\cdot}}$ algorithm is a bandit algorithm that has shown to be able to minimize simple regret [6]. In this research, we propose a method for dynamically adjusting the cost of exploration for minimizing simple regret in MCTS, by applying the iteratively updated estimation bound Δ_m in the improved UCB algorithm as an extra factor for regulating the influence of the exploration term in the confidence bounds of the $UCB_{\sqrt{\cdot}}$ algorithm. We will also develop the all-move-as-first-move (AMAF) heuristic for this *combined confidence bounds*, and finally demonstrate its empirical performance on the MAB problem and the game of 9×9 Go.

II. Simple and Cumulative Regret in the Multi-armed Bandit Problem

In the MAB problem, the agent is faced with K slot machines (or "one-armed bandits"). The agent can decide to play one of the slot machines (or "pull an arm") on each play, and the slot machine will produce a reward $r \in [0, 1]$. There are mainly two possible objectives in the MAB problem, and the agent needs to adopt a playing strategy, or *bandit algorithm*, accordingly.

978-1-4799-8622-4/15/$31.00 ©2015 IEEE

A. Minimizing Cumulative Regret

In the conventional MAB problem, the task of the agent is to accumulate as much reward as possible over a number of plays T [1]. The task can be stated equivalently as minimizing the *cumulative regret*, which is defined as

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

where r^* is the mean reward of the optimal arm, and r_{I_t} is the received reward when the agent decides to pull arm I_t on play t. A bandit algorithm is considered optimal if it can restrict the cumulative regret to $O(\log T)$. The *UCB* algorithm, which is used in the UCT algorithm [3], minimizes the cumulative regret, restricting its growth to $O(\frac{K\log(T)}{\Delta})$, where Δ is the difference of expected reward between a suboptimal arm and the optimal arm [2].

The *improved UCB* algorithm, shown in Algorithm 1, is a modification of the UCB algorithm, which improves the bound on cumulative regret to $O(\frac{K\log(T\Delta^2)}{\Delta})$ [11]. The improved UCB algorithm maintains a candidate set of possible optimal arms B_m, and eliminates arms that are likely to be suboptimal from B_m. A pre-determined total number of plays T is given to the algorithm, and it is further divided into $\lfloor \frac{1}{2}\log_2 \frac{T}{e} \rfloor$ rounds. In each round, the algorithm samples each arm n_m times, and proceeds to eliminate the arms whose estimated reward upper bounds are lower than the lower bound of the current best arm. The algorithm then halves the estimated difference Δ_m, and moves on to the next round. After each round, 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 from arm a_i.

In the case where the total number of plays is unknown, the improved UCB algorithm can be executed in an episodic fashion, with $T_0 = 2$ plays for the initial episode, and $T_{\ell+1} = T_\ell^2$ for subsequent episodes.

B. Minimizing Simple Regret

Another variation of the MAB problem is to identify the optimal arm after a pre-determined number of plays T [5]. The objective can be stated 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 objective is to identify the optimal arm, the amount of accumulated reward is irrelevant in this variation. It has been shown that there is a trade-off between minimizing the cumulative regret CR_T and the simple regret SR_T, i.e., decreasing CR_T will increase SR_T, and vice versa [5]. Therefore, a different class of bandit algorithms is needed for minimizing the simple regret.

The $UCB_{\sqrt{\cdot}}$ algorithm, shown in Algorithm 2, is a bandit algorithm that bounds the simple regret to $O((\Delta\exp(-\sqrt{T}))^K)$ [6]. The $UCB_{\sqrt{\cdot}}$ algorithm is essentially

Algorithm 1 The Improved UCB Algorithm [11]

Input: A set of arms A, total number of trials 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 the arm a_i and update its average reward w_i
end for
end for

(2) Arm Elimination:
$a_{max} \leftarrow$ 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 a_i from B_m
end if
end for

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

Algorithm 2 The $UCB_{\sqrt{\cdot}}$ algorithm [6]

Initialization: Play each machine once.
for $t = 1, 2, 3, \cdots$ **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

the same as the UCB algorithm, and only differs in the 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.

III. Regulating Exploration in Simple Regret Minimization

The main objective of game tree search is to identify the best course of action to take, and thus we mainly care about the quality of the decision after the search has been performed rather than the process itself. Therefore, simple regret bandit algorithms seem to be more suited for MCTS.

However, since simple regret bandit algorithms try to identify the optimal arm, and do not need to take the cost of exploration, i.e.; in the context of game tree search, time is the main measure of cost, into account, they tend to devote much time and resources in verifying whether an arm is suboptimal or not, rather than exploiting possible optimal arms. Because time and resources are rather limited when we are performing game tree search, this characteristic may not be desirable, since

the algorithm may end up spending most of its time exploring parts of the tree that are irrelevant before it can search deeper into more promising subtrees if the size of the game tree is large. Therefore, it would be ideal to regulate the cost of exploration in simple regret bandit algorithms by incorporating some characteristics of cumulative regret bandit algorithms.

In this section, we will first introduce the *combined confidence bounds*, which utilizes the characteristics of the improved UCB algorithm to regulate the influence of the exploration term in the $UCB_{\sqrt{\cdot}}$ algorithm. We will then describe the bandit algorithm that uses the combined confidence bounds, and finally show how it is extended to MCTS.

A. Combined Confidence Bounds

In the previous section, we have observed that the improved UCB algorithm effectively estimates the expected reward of arm a_i as $(w_i \pm \frac{\Delta_m}{2})$ after each round. We will utilize this characteristic for regulating the cost of exploration in the $UCB_{\sqrt{\cdot}}$ algorithm by modifying the confidence bounds in the improved UCB algorithm to

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

where $r_i = \frac{\sqrt{T}}{t_i}$ and c_Δ is a constant. By adding the factor r_i, the expected reward of arm a_i will effectively be estimated as

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

after each round. Since $r_i = \frac{\sqrt{T}}{t_i}$, we have $w_i \pm c_\Delta \cdot \frac{\Delta_m}{2}\sqrt{r_i} = w_i \pm c_\Delta \cdot \frac{\Delta_m}{2}\sqrt{\frac{\sqrt{T}}{t_i}}$, which differs from the confidence bound of the $UCB_{\sqrt{\cdot}}(c)$ algorithm only by the extra factor $\frac{\Delta_m}{2}$. Therefore, the cost of exploration will be dynamically regulated by the restriction imposed on the exploration term by the $\frac{\Delta_m}{2}$ factor. Another perspective is that the $\frac{\Delta_m}{2}$ factor dynamically tunes the constant c_Δ in the confidence bound of the the $UCB_{\sqrt{\cdot}}$ algorithm.

B. Combined Confidence Bounds Bandit Algorithm

The bandit algorithm that uses the proposed combined confidence bounds is shown in Algorithm 3. The algorithm consists of three steps in each play, which is similar to the improved UCB algorithm.

In the first step, we only sample the current best arm. By sampling only the current best arm, the update of the simple regret part of the combined confidence bounds will still be in keeping with the $UCB_{\sqrt{\cdot}}$ algorithm. As for the update of Δ_m, the current best arm will be sampled at least n_m times, and hence the guarantee for the bound on the current best arm to hold will be stronger than in the improved UCB algorithm, although the guarantee for other candidates will be weaker. In the context of game tree search, since it would be more desirable for the search algorithm to explore the most promising parts of the game tree, being "more certain" about the current best arm would be more advantageous; hence this trade-off is acceptable.

Algorithm 3 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 Δ_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 Δ_k when Deadline T_{Δ_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

The second step is to update the count of arms that are still candidates for being the optimal arm, i.e.; the arms whose upper bound is still higher than that of the lower bound of the current best arm. In the improved UCB algorithm, Δ_m is halved after each round, and each round consists of $(|B_m| \times n_m)$ plays, where $|B_m|$ is the number of arms that are still in the candidate set. Since we are only sampling the current best arm, we do not need to maintain a candidate set, but we still need to maintain the count of the arms that are still under consideration for updating Δ_m.

The third and final step is halving Δ_m if the deadline has been reached. After halving Δ_m, the new value of n_m and the next deadline $T_{\Delta_{k+1}}$ will also be updated accordingly.

C. Combined Confidence Bounds MCTS (CCB-MCTS)

The application of the proposed bandit algorithm to MCTS is the same way as the UCB algorithm applied in the UCT algorithm. The details of the Combined Confidence Bounds MCTS (CCB-MCTS) are shown in Algorithm 4.

Because it is difficult to decide beforehand the appropriate total number of plays that is needed for each node, the bandit algorithm is run in an episodic manner, with $N.T_0 = 2$ for the initial episode, and $N.T_{\ell+1} = N.T_\ell^2$ for the subsequent episodes. When a new episode begins, which is when $N.t \geq N.T$, the estimated regret $N.\Delta$ and the count of candidate arms

$N.armCount$ will be re-initialized, and the new deadline for halving $N.\Delta$ will also be updated accordingly.

IV. Applying All-Moves-As-First Heuristics to Combined Confidence Bounds

The all-move-as-first (AMAF) heuristic [4][12] 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. 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) [12] 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* [12]. 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:

1) 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 [12], by only applying RAVE on the win rate

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

2) 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

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

Algorithm 4 Combined Confidence Bound MCTS Algorithm

function CombConfBound-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
      $r_i \leftarrow \frac{\sqrt{N.t}}{n_i.t}$
      $n_i.ucb \leftarrow n.w + \sqrt{\frac{\log(N.T\times N.\Delta^2)\times r_i}{2N.k}}$
    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 $n_{best}.times = 0$ **then**
    $result \leftarrow$RandomSimulation((n_{best}))
  else
    if n_{best} is not yet expanded **then** NodeExpansion((n_{best}))
    $result \leftarrow$ CombConfBound-MCTS((n_{best}))
  end if

  $N.w \leftarrow (N.w \times N.t + result)/(N.t+1)$
  $N.t \leftarrow N.t + 1$

  if $N.t \geq N.T$ **then**
    $N.\Delta \leftarrow 1$
    $N.T \leftarrow N.t + N.T \times N.T$
    $N.armCount \leftarrow$ Total number of child nodes
    $N.k \leftarrow \lceil \frac{2\log(N.T\times N.\Delta^2)}{N.\Delta^2} \rceil$
    $N.deltaUpdate \leftarrow N.t + N.k \times N.armCount$
  end if

  if $N.t \geq N.deltaUpdate$ **then**
    for all child nodes n_i of N **do**
      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**
        $N.armCount \leftarrow N.armCount - 1$
      end if
    end for

    $N.\Delta \leftarrow \frac{N.\Delta}{2}$
    $N.k \leftarrow \lceil \frac{2\log(N.T\times N.\Delta^2)}{N.\Delta^2} \rceil$
    $N.deltaUpdate \leftarrow N.t + N.k \times N.armCount$
  end if
  return $result$
end function

function NodeExpansion(Node N)
  $N.\Delta \leftarrow 1$
  $N.T \leftarrow 2$
  $N.armCount \leftarrow$ Total number of child nodes
  $N.k \leftarrow \lceil \frac{2\log(N.t\times N.\Delta^2)}{N.\Delta^2} \rceil$
  $N.deltaUpdate \leftarrow N.k \times N.armCount$
end function

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

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

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 4 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.

V. Experimental Results

In this section, we will demonstrate the performance of the combined confidence bounds on the MAB problem, and the game of 9×9 Go.

A. Performance in the Multi-armed Bandit Problem

The settings of the MAB problem follow the multi-armed bandit testbed specified in Sutton et al. [1]. The results are the average of 2000 randomly generated K-armed bandit problems. A total of $20,000$ plays were given. The rewards of each bandit were generated from a normal (Gaussian) distribution with the mean w_i, $i \in K$, and variance 1. The mean w_i of the bandits in each K-armed bandit problem instance were randomly selected from a normal distribution with mean 0 and variance 1.

We have compared the performance of various bandit algorithms:

- **UCB**: the UCB algorithm
- **UCBsqrt**: the $UCB_{\sqrt{\cdot}}$ algorithm
- **Improved UCB**: the improved UCB algorithm
- **Combined Confidence Bound**: the episodic combined confidence bound bandit algorithm. Since the combined confidence bound is executed in an episodic fashion when we applied it to MCTS, the behaviour of the episodic version is of more interest.

The performance of the bandit algorithms on MAB problem with $K = 60$ and $K = 300$ is shown in Fig. 1 and Fig. 2, respectively.

It can be observed that the *Combined Confidence Bound* bandit algorithm provides the best restriction on the growth of cumulative regret, and the highest optimal percentage in both cases. The "slack" of the combined confidence bound bandit algorithm is due to the re-initialization when a new episode starts. The cumulative regret of the $UCB_{\sqrt{\cdot}}$ algorithm increases almost linearly, which confirms the trade-off between minimizing cumulative regret and simple regret. Although the $UCB_{\sqrt{\cdot}}$ algorithm did not perform as well as expected in restricting simple regret. It can be observed in Fig. 1c and Fig. 2c that the combined confidence bound and the UCB algorithm both outperformed the $UCB_{\sqrt{\cdot}}$ algorithm. It is interesting to observe that the cumulative regret of the improved UCB algorithm is much higher than that of the UCB algorithm, despite the fact that the improved UCB algorithm has a tighter bound on the cumulative regret. In both Fig. 1b and Fig. 2b, it can be observed that the rate of growth in the optimal percentage of the improved UCB algorithm is greater than that of the UCB algorithm. This may suggest that the improved UCB algorithm tries to identify the optimal arm by the process of elimination, which tries to verify and eliminate suboptimal arms as early as possible. Therefore, it may have the tendency to distribute the "necessary" number of plays on the suboptimal in the early stage. In contrast, the UCB algorithm exploits the possible optimal arm as early as possible, which effectively distributes the "necessary" suboptimal plays evenly throughout the whole process.

B. Comparison with Plain UCT on 9×9 Go

We will demonstrate the performance of combined confidence bounds applied to the MCTS on the game of 9×9 Go, with the komi of 6.5. The baseline for all experiments is the plain UCT algorithm. For a more direct and effective comparison, all MCTS algorithms used pure random simulations, and no extra performance enhancing heuristics.

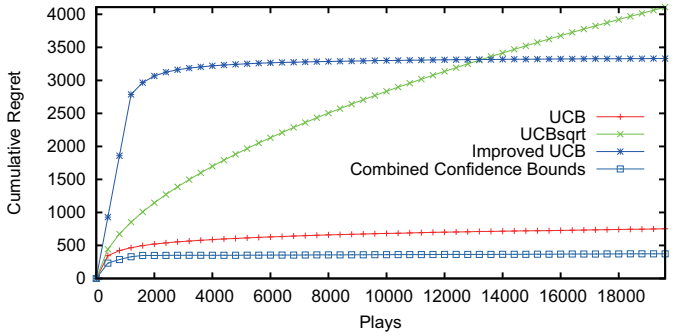
1) Performance of $UCB_{\sqrt{\cdot}}$ MCTS and SR+CR scheme: First, we will demonstrate the performance of the confidence bound defined in the $UCB_{\sqrt{\cdot}}$ algorithm in 9×9 Go. Table I shows the win rate of various constant $c_{\sqrt{\cdot}}$ settings of the $UCB_{\sqrt{\cdot}}$ MCTS and the SR+CR scheme against plain UCT. The $UCB_{\sqrt{\cdot}}$ MCTS applies the $UCB_{\sqrt{\cdot}}$ bandit algorithm on every node, and the SR+CR scheme applies the $UCB_{\sqrt{\cdot}}$ bandit algorithm on the root node and the UCB algorithm for other nodes, with constant c of the UCB algorithm set to 0.4 [6]. The constant of the plain UCT algorithm was also set to $c = 0.4$. The results are the average of 2000 games, with 5000 playouts for each move. Both algorithms took turns in playing with Black and White.

It can be observed that the best win rate that $UCB_{\sqrt{\cdot}}$ MCTS and SR+CR scheme can achieve with the best setting is around 50% to 51%, which is only nearly the same as the plain UCT algorithm.

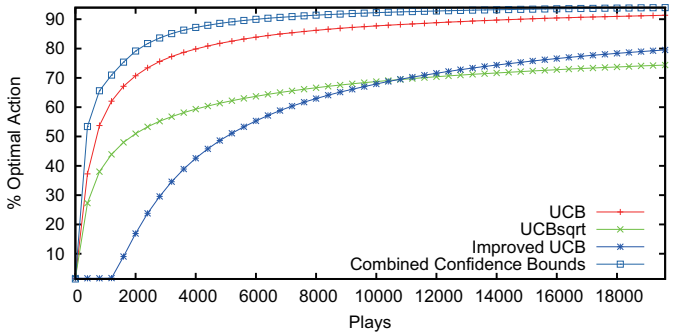
2) Tuning the C constants: We will proceed to find the best settings for the constant c_Δ in combined confidence bounds, and c for the plain UCT algorithm. We have found the optimal setting for c_Δ is 0.47, and Table II shows its performance against various constant c settings for plain UCT. All the 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.

It can be observed that the best setting for c in the UCT algorithm is 0.37, against which the CCB-MCTS have achieved a win rate of 53.83%. This result not only indicates that the CCB-MCTS is slightly better than the UCT algorithm, but also demonstrated that regulating the exploration term in the confidence bound $UCB_{\sqrt{\cdot}}$ algorithm is effective.

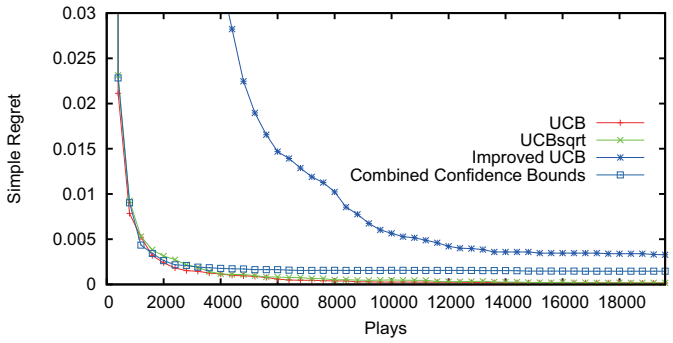
3) Scalability of CCB-MCTS: We will now proceed to investigate the scalability of the CCB-MCTS as the total number of playout increases. The result is shown in Table III. All the results are the average of 2300 games, with both algorithms taking turns in playing with Black and White. The



(a) Cumulative Regret

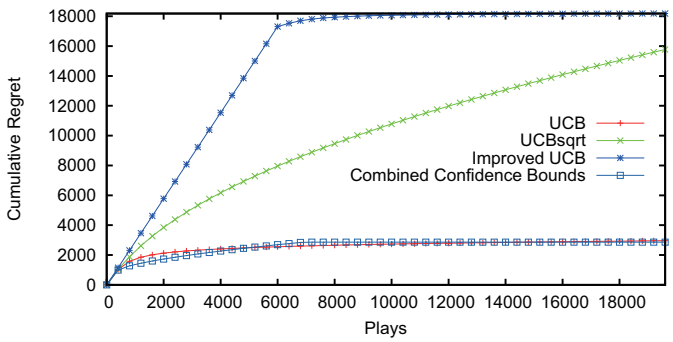


(b) Optimal Percentage

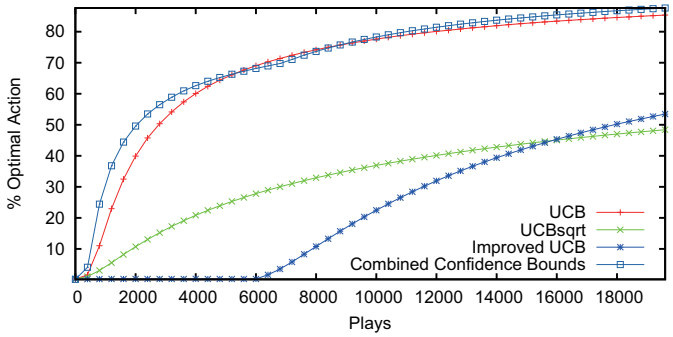


(c) Simple Regret

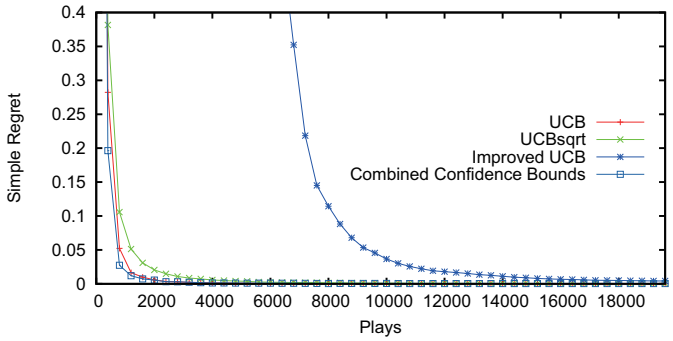
Fig. 1: Performance of Various Bandit Algorithms ($K = 60$)



(a) Cumulative Regret



(b) Optimal Percentage



(c) Simple Regret

Fig. 2: Performance of Various Bandit Algorithms ($K = 300$)

settings are $c_\Delta = 0.47$ for the CCB-MCTS, and $c = 0.37$ for the plain UCT algorithm, and both algorithms have the same number of total playouts for each move.

Since the difference between CCB-MCTS and the UCT algorithm is mainly 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.

It can be observed that the CCB-MCTS has a small edge when the number of playouts is less or equal to 7000, and it has superior performance when given more than 9000 playouts. This result suggests that the CCB-MCTS has increased performance when more playouts are given.

TABLE I: Win Rate of $UCB_{\sqrt{\cdot}}$ MCTS and SR+CR scheme [6] against plain UCT algorithm in 9×9 Go. The $UCB_{\sqrt{\cdot}}$ MCTS has a best win rate of 49.10% win with $c_{\sqrt{\cdot}} = 0.2$. The SR+CR scheme has a best win rate of 52.30% when $c_{\sqrt{\cdot}} = 0.9$ and $c = 0.4$.

$c_{\sqrt{\cdot}}$	$UCB_{\sqrt{\cdot}}$ MCTS	SR+CR Scheme
0.1	45.30% ± 3.09%	41.95% ± 2.16%
0.2	**49.10% ± 3.10%**	48.70% ± 2.19%
0.3	42.65% ± 2.17%	50.55% ± 2.19%
0.4	38.05% ± 2.18%	50.45% ± 2.19%
0.5	41.35% ± 2.16%	51.75% ± 2.19%
0.6	43.95% ± 2.18%	49.85% ± 2.19%
0.7	46.55% ± 2.19%	51.95% ± 2.19%
0.8	48.75% ± 2.19%	51.55% ± 2.19%
0.9	47.05% ± 2.19%	**52.30% ± 2.19%**

TABLE II: The win rate of the combined confidence bound MCTS with $c_\Delta = 0.47$ against plain UCT algorithm with various constant c settings on 9×9 Go. The optimal setting for plain UCT algorithm is $c = 0.37$, achieving a 53.82% win rate.

c	Win Rate	c	Win Rate	c	Win Rate
0.1	62.57% ± 1.98%	0.31	62.70% ± 1.98%	0.41	57.70% ± 2.02%
0.2	59.91% ± 2.00%	0.32	61.04% ± 1.99%	0.42	58.65% ± 2.01%
0.3	62.91% ± 1.98%	0.33	60.17% ± 2.00%	0.43	60.78% ± 2.00%
0.4	54.04% ± 2.04%	0.34	57.83% ± 2.02%	0.44	61.78% ± 1.99%
0.5	63.87% ± 1.96%	0.35	54.65% ± 2.03%	0.45	61.09% ± 1.99%
0.6	62.87% ± 1.97%	0.36	55.26% ± 2.03%	0.46	62.17% ± 1.98%
0.7	63.13% ± 1.97%	0.37	**53.82% ± 2.03%**	0.47	63.87% ± 1.96%
0.8	61.70% ± 1.99%	0.38	54.65% ± 2.03%	0.48	62.87% ± 1.97%
0.9	59.26% ± 2.01%	0.39	55.60% ± 2.03%	0.49	64.57% ± 1.95%

TABLE III: Scalability of the CCB-MCTS on 9×9 Go. The win rate of the CCB-MCTS against plain UCT algorithm gradually increases when more playouts are given.

Playouts	Win Rate
1000	53.52% ± 2.04%
3000	54.35% ± 2.04%
5000	53.82% ± 2.03%
7000	54.17% ± 2.04%
9000	58.70% ± 2.01%
11000	57.35% ± 2.02%
13000	55.39% ± 2.03%
15000	55.22% ± 2.03%
17000	55.43% ± 2.03%
19000	56.52% ± 2.03%
21000	57.15% ± 2.02%
23000	55.61% ± 2.03%
25000	56.48% ± 2.03%

4) CCB-MCTS with AMAF Heuristics: Finally, we will investigate the effectiveness of applying AMAF heuristics to the CCB-MCTS.

The performance of the UCT-RAVE algorithm [12], in which only the AMAF heuristic is applied to the win rate of the UCB confidence bound, is shown in Table IV. The results of the CCB-MCTS with AMAF heuristics are shown in Table V.

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

We can observe in Table IV that the UCT-RAVE algorithm can achieve a win rate of 62.70% against plain UCT, and Table V shows 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.

TABLE IV: The win rate of the UCT-RAVE algorithm against plain UCT algorithm in 9×9 Go. The UCT-RAVE algorithm achieved the best result of winning 62.70% of the games with the setting of $D_{rate} = 6000$.

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%**

TABLE V: The win rate of the combined confidence bounds MCTS with AMAF heuristics against plain UCT algorithm in 9×9 Go. The combined confidence bounds MCTS won 66.61% of the games by applying RAVE to win rate with the setting of $D_{rate} = 6000$, and a 67.26% win rate when RAVE are applied to both win rate and halving Δ_m, with the setting of $D_{rate} = 7000$ and $D_\Delta = 50$ respectively.

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%

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.

VI. Conclusion

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.

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%.

Exploring the possibility of applying other performance enhancement heuristics and techniques, such as more intelligent playouts [4][13], and application to other games would naturally be the next step. Applying exploration regulation on other more refined simple regret bandit algorithms, such as the lil'UCB algorithm [17], would also be of interest. Another possible extension would be to incorporate contextual information to the combined confidence bound [18]. Finally, since the combined confidence bound does not retain the entire original properties of the improved UCB and $UCB_{\sqrt{\cdot}}$ algorithm, an investigation to its theoretical properties would be of interest, and may provide further insights to the inner workings of MCTS.

References

[1] R. S. Sutton, and A. G. Barto, "Reinforcement learning: an introduction," *MIT Press*, Cambridge, MA, 1998.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, issue 2-3, pp. 235 - 256, 2002.

[3] L. Kocsis, and C. Szepesvári, "Bandit based monte-carlo planning," *Proceedings of the 17th European Conference on Machine Learning (ECML'06)*, pp. 282–293, 2006.

[4] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte-carlo tree search methods," *IEEE Transactions on Computational Intelligence and Al in Games*, 4(1):1-43, 2012.

[5] S. Bubeck, R. Munos, and G. Stoltz, "Pure exploration in multi-armed bandits problems", *Proceedings of the 20th International Conference on Algorithmic Learning Theory (ALT 2009)*, 2009.

[6] D. Tolpin, and S.E. Shimony, "MCTS based on simple regret," *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pp. 570-576, 2012.

[7] T. Cazenave, " Sequential halving applied to trees," *IEEE Transactions on Computational Intelligence and Al in Games*, 7(1): 102-105, 2015.

[8] T. Pepels, T. Cazenave, M.H.M. Winands, M.H.M., and M. Lanctot, "Minimizing simple and cumulative regret in monte-carlo tree Search," *Proceedings of Computer Games Workshop at the 21st European Conference on Artificial Intelligence*, 2014.

[9] P. Perrick, D.L. St-Pierre, F. Maes, and D. Ernst, "Comparison of different selection strategies in monte-carlo tree search for the game of tron," *Proceedings of the Conference on Computational Intelligence and Games (CIG 2012)*, 2012.

[10] T. Imagawa and T. Kaneko, " Applying multi-armed bandit algorithms to MCTS and those analysis," *Proceedings of the 19th Game Programming Workshop (GPW-14)*, pp. 145-150, 2014.

[11] P. Auer, and R. Ortner, "UCB revisited: improved regret bounds for the stochastic multi-armed bandit problem," *Periodica Mathematica Hungarica* , vol. 61, pp. 1-2, 2010.

[12] S. Gelly, and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artificial Intelligence*, vol. 175, Issue 11, pp. 1856-1875, 20011.

[13] R. Coulom, "Computing "elo ratings" of move patterns in the game of go," *ICGA Journal*, vol. 30(4), pp. 198-208, 2007.

[14] Z. Karnin, T.Koren, S. Oren, "Almost optimal exploration in multi-armed bandits," *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pp. 1238-1246, 2013.

[15] A. Garivier, and A. Cappe, "The KL-UCB algorithm for bounded stochastic bandits and beyond," *Proceedings of 24th Annual Conference on Learning Theory (COLT '11)*, pp. 359-376, 2011.

[16] E. Kaufmann, N. Korda, R. Munos, "Thompson sampling: an asymptotically optimal finite-time analysis," *Proceedings of 23rd Algorithmic Learning Theory (ALT'12)*, pp. 199-213, 2012.

[17] L. Jamieson, M. Malloy, S. Bubeck, and R. Nowak, "lil' UCB: an optimal exploration algorithm for multi-armed bandits," *Proceedings of the 27th annual conference on Computational Learning Theory (COLT' 14)*, 2014.

[18] C. Rosin, "Multi-armed bandits with episode context," *Annals of Mathematics and Artificial Intelligence*, vol. 61, issue 3, pp. 203-230, 2011.