Enhancements in Monte Carlo Tree Search Algorithms for Biased Game Trees

Takahisa Imagawa and Tomoyuki Kaneko
Graduate School of Arts and Sciences, the University of Tokyo
Email: {imagawa,kaneko}@graco.c.u-tokyo.ac.jp

***Abstract*—Monte Carlo tree search (MCTS) algorithms have been applied to various domains and achieved remarkable success. However, it is relatively unclear what game properties enhance or degrade the performance of MCTS, while the largeness of search space including pruning efficiency mainly governs the performance of classical minimax search, assuming a decent evaluation function is given. Existing research has shown that the distribution of suboptimal moves and the non-uniformity of tree shape are more important than the largeness of state space in discussing the performance of MCTS. Our study showed that another property, *bias in suboptimal moves*, is also important, and we present an enhancement to better handle such situations. We focus on a game tree in which the game-theoretical value is even, while suboptimal moves for a player tend to contain more inferior moves than those for the opponent. We conducted experiments on a standard incremental tree model with various MCTS algorithms based on UCB1, KL-UCB, or Thompson sampling. The results showed that the bias in suboptimal moves degraded the performance of all algorithms and that our enhancement alleviated the effect caused by this property.**

Keywords—Monte Carlo tree search, UCT, UCB1, KL-UCB, dynamic komi

I. Introduction

Monte Carlo tree search (MCTS) algorithms [1] have achieved remarkable success, especially in the game of Go [2]. MCTS algorithms have also been widely adopted in other domains, including general game players [3], imperfect information games [4] and real-time games [5]. However, there exist some domains where classical minimax search algorithms guided by a decent evaluation function outperform MCTS methods, e.g., in chess variants. Also, in some domains, researchers suggest combining minimax search values with MCTS [6]. Several studies have been done on synthetic game trees in order to better understand what properties in a domain govern the performance of MCTS algorithms. Finnsson showed that the distribution of suboptimal moves is a more important factor in MCTS than the largeness of state space, which is a primary factor for classical minimax search methods [3]. Ramanujan demonstrated that MCTS methods work effectively in a tree having a uniform branching factor and depth, and that the performance degrades when a tree has many traps, each of which is a terminal node where its siblings are not terminals [7]. Also, when an imperfect information game is played by searching possible worlds as a perfect information game, it is said that correlation, bias in leaf values, and a disambiguation factor are important [8].

In this paper, we define another property called "bias in suboptimal moves," and we explain its importance in perfect-information and deterministic games. We focus on a game tree in which the game-theoretical value of the root is even, while suboptimal moves for a player tend to contain more inferior moves than those for the opponent. This means that a position is fair with respect to the game-theoretical value, but somewhat unfair with respect to the amount of penalty when a player misses the optimal move. For example, if the king of a player is threatened in a long checkmate sequence in chess or shogi, the player will immediately lose when he or she missed the correct move. Also, in games designed for a human vs. an AI (artificial intelligence) player (e.g. [9]), a human player might be given difficult positions, though it is not impossible to win. In Go, a "semeai" position can have a similar property. In a position shown in [10], the majority of playouts resulted in wins for black by a large margin while the game-theoretical value is almost even and a win for white by a narrow margin if both players choose the optimal moves. Therefore, suboptimal moves for white should contain more inferior moves than those for black, in the position. It is also known that MCTS does not adapt well to handicap games in Go [11].

We show that the existence of "bias in suboptimal moves" seriously degrades the performance of MCTS algorithms. An intuitive explanation for this is that the difficulty in identifying the optimal move among suboptimal moves increases as the winning probability observed through trials for any move of the advantageous player increases. To solve this problem, we present a simple enhancement that adjusts the threshold of win/loss based on the frequency of the game scores of playouts. We conducted experiments on a standard incremental tree model with various MCTS algorithms based on KL-UCB [12] or Thompson sampling [13], in addition to standard UCT, which is based on UCB1 [14]. Incremental random trees and their variants are standard models and are widely used to evaluate the performance of game tree search algorithms, including alpha beta search and MCTS [7], [14]–[17]. The results showed that the property of bias in suboptimal moves actually degrades the performance of all algorithms, and that our enhancement alleviated the effect caused by the property. This paper extends the authors' earlier work [18] that focused only on UCB1 and limited kinds of trees having the same penalty for all suboptimal moves of each player. In this work, the importance of the property as well as the effectiveness of our enhancement are confirmed in standard models in evaluations of game tree search algorithms with the popular multi-armed bandit strategy in addition to UCB1.

II. Background and Related Work

This section briefly introduces Monte Carlo tree search methods and enhancements, as well as synthetic models that have been used in the evaluations of algorithms.

A. Monte Carlo Tree Search

Monte Carlo tree search (MCTS) [1] is a kind of best-first algorithm that iteratively expands and evaluates a game tree by using a random simulation (often referred to as a *playout*). Iteration of MCTS consists of four steps: (1) selection, (2) expansion, (3) simulation, and (4) backpropagation. First, in the selection step, the algorithm descends from the root to a leaf by recursively selecting the most urgent child at each node. Several strategies in *multi-armed bandit problems* have been employed to determine the urgency in this step. For example, UCT [14] adopts UCB1 here. If the number of times the leaf was visited reaches a threshold (usually two), the leaf is expanded, and one of its children is randomly selected. In the simulation step, the rest of the game is randomly played starting at the position corresponding to the leaf. The game typically consists of uniformly random moves. However, some studies suggest that the incorporation of domain knowledge improves the performance. Finally, the outcome of the game (e.g., win, draw, or loss with respect to the player who is to move next) is propagated from the leaf to the root in the backpropagation step.

When given computational resources (e.g., time) run out, the most visited child at the root is returned as the best move.

B. Multi-armed Bandit Problem and Strategies

The multi-armed bandit problem is a well-known problem in which a player needs to repeatedly choose an action called an *arm*. When arm i is pulled, the reward is stochastically returned following the fixed but unknown distribution with expectation μ_i. The goal of the player is to maximize the summation of the rewards of n trials. Several strategies to solve this problem have been presented. Although many of them do not assume a specific distribution, here, we assume a Bernoulli distribution in the rewards for simplicity of discussion. In typical applications involving MCTS, pulling an arm means to descend to the corresponding move in the selection step, and a reward would be 1 for a win or 0 for a loss.

UCB1 [19] is the most popular strategy and it selects arm $i \in \mathbb{K}$ having the highest upper confidence bound:

$$\overline{X}_{i,T_i(t)} + \sqrt{\frac{2\ln t}{T_i(t)}}, \tag{1}$$

where $\mathbb{K}$ is the set of available arms, $T_i(n)$ is the number of times arm i was pulled up to now, $\overline{X}_{i,T_i(n)}$ is the average reward of arm i over $T_i(n)$ trials. Variable t represents the total number of trials so far in the original problem, but in MCTS, it represents the number of times that the source node of move i was visited. The left term is used for exploitation to pull the best arm in observed trials, and the right term is used for exploration to find a possible good arm that has been tried relatively few times so far.

KL-UCB [12] selects arm i having the highest expected estimation:

$$\max q_i \in [0,1] \text{ s.t. } d(\overline{X}_{i,T_i(t)}, q_i) \le \frac{\ln t + c\ln(\ln t)}{T_i(t)}, \tag{2}$$

where c is a constant and $d(p,q)$ represents Kullback-Leibler divergence between two Bernoulli distributions with expectation p and q; $d(p,q) := p\log\frac{p}{q} + (1-p)\log\frac{1-p}{1-q}$. KL-UCB is a relatively recent algorithm and has a better theoretical upper bound than that of UCB1, with respect to regrets, which are the accumulated penalties of selecting suboptimal arms. Also, it was empirically shown that KL-UCB performs better than UCB1 in numerical experiments [12].

Thompson sampling [20] is a kind of randomized algorithm that selects the arm with the highest sample drawn from Beta distribution $B_e(\alpha_i + 1, \beta_i + 1)$, where α_i or β_i is the number of times that the reward of arm i is 1 or 0, respectively:

$$\alpha_i = \overline{X}_{i,T_i(t)} T_i(t),\ \beta_i = (1 - \overline{X}_{i,T_i(t)}) T_i(t). \tag{3}$$

Therefore, the probability of an arm being selected becomes higher when more wins are observed with the arm. This strategy was proposed more than twenty years ago, but it was theoretically analyzed only recently [13], [21].

We compared MCTS with UCB1, KL-UCB, and Thompson sampling in our experiments. Similar comparisons were conducted in a simultaneous game of Tron [5]. However, we discusse in this paper the first results of comparison with deterministic combinatorial games.

C. Artificial Game Trees and Empirical Analysis of MCTS

The performance of MCTS, in addition to being theoretically analyzed, has been empirically evaluated through experiments. An incremental random tree [15] is a synthetic model of a game tree, and it provides a domain-dependent and therefore preferable way to analyze the efficiency of game tree search algorithms [15]–[17]. An incremental random tree usually has a uniform branching factor and depth, where a random value is assigned to each edge in the tree. The game score at a terminal node is the summation of those values in the path from the root to the terminal node. It is designed so that sibling nodes tend to have similar scores. The game score of each internal node is identified by integrating the scores of descendant nodes via minimax search. A score that is positive, negative, or zero respectively means win, loss, or draw for the maximizing player.

Kocsis and Szepesvári showed that UCT performs better than alpha-beta search in those trees [14]. Ramanujan showed that the shape of a tree having a uniform branching factor is important for MCTS, and that the performance is degraded when a tree has many traps, each of which is a terminal node where its siblings are not terminals [7]. Finnsson presented a simplified tree model where a set of optimal and suboptimal moves is identical in all nodes; i.e., randomness is removed from edge values [3]. It was shown that the distribution of suboptimal moves is more important in MCTS than the largeness of state space, which is a primary factor in classical minimax search methods. In this paper, two kinds of trees are adopted to discuss the property of bias in suboptimal moves: a *random tree* and a *constant tree*. The former tree is an extension of a standard incremental random tree, and the latter is an extension of Finnsson's model.

In this paper, we assume that a discrete value is returned for the reward in each playout, i.e., 1 for a win, 0 for a loss, or 0.5 for a draw. Alternatively, we can assign continuous values in a

Algorithm 1 Score Situational (simplified by the authors)

```
Require: $\overline{\text{score}}$ is the average score of all playouts
  phase $\leftarrow$ BoardOccupiedRatio $+s$
  rate $\leftarrow 1/(1+\exp(c \cdot \text{phase}))$
  threshold $\leftarrow$ threshold $+$ rate $\cdot\overline{\text{score}}$
  return threshold
```

Algorithm 2 Value Situational (simplified by the authors)

```
Require: $\overline{X}$ is the observed winning rate.
  if $\overline{X} < X_{\text{lo}}$ then
    if threshold $> 0$ then
      Ratchet $\leftarrow$ threshold
    end if
    threshold $\leftarrow$ threshold $-1$
  else
    if $\overline{X} > X_{\text{hi}}$ and threshold $<$ Ratchet then
      threshold $\leftarrow$ threshold $+1$
    end if
  end if
  return threshold
```

range $[0, 1]$. For some games, scores are available in addition to wins or losses, e.g., the difference in the number of discs in Othello, or area scoring in Go. Historically, many Go programs adopted linearly transformed scores as rewards to distinguish better wins and losses. However, the playing strength has greatly improved by adopting discrete values, which give clear differences between wins and losses. Nowadays, many Go programs adopt this configuration, e.g., [22]. Further research on the quality of wins has been proposed in the literature [23]. In this paper, we also assume that a multi-valued game score is possible, and we convert a score to a win or loss when the score is greater than or less than the *threshold*, respectively. The threshold is usually zero but can be adjusted to any value.

It is known that MCTS is not very effective in handicap games in Go where a player and the opponent are facing an extreme advantage and disadvantage, respectively. Dynamic komi [11] is an enhancement to alleviate the problem by adjusting the threshold for converting a score into a win or loss. In a published study [11], three variations of dynamic komi were introduced: Linear Handicap, Score Situational, and Value Situational. Because the first one uses domain-dependent statistics, we briefly explain the second and third ones that can be applied to our model. Score Situational, shown in Algorithm 1, adjusts the threshold so that the average score of playouts becomes zero, with parameters c and s. Although BoardOccupiedRatio represents the progress of a game in Go, we adopted the number of moves played for this value. Value Situational, shown in Algorithm 2, adjusts the threshold so that the winning rate of the root player $\overline{X}$ is in the range $[X_{\text{lo}}, X_{\text{hi}}]$. To prevent fluctuation of the threshold, the variable Ratchet is introduced, whose value is ∞ at the beginning.

III. Bias in Suboptimal Moves in Synthetic Model

This section introduces several instances of synthetic models of game trees to represent the property of bias in suboptimal moves, following the standard model of incremental game trees [15]. We assume that a tree has a uniform branching factor and depth where the depth is an even number. An integer value is assigned to each move (edge) in the tree, and that one move corresponding to the optimal move has a value of zero, and the remaining moves have negative (positive) scores in each internal node of the maximizing (minimizing) player. The game score at a terminal node is the summation of those values in the path from the root to the terminal node. If the score is greater than, less than, or equal to the threshold (usually zero), it means a win, loss, or draw for the maximizing player, respectively. Therefore, there exists exactly one optimal move in each node, and the game result of the root is draw when both players play optimally, as in [16]. Also, for simplicity, we assume that the maximizing player moves first at the root position.

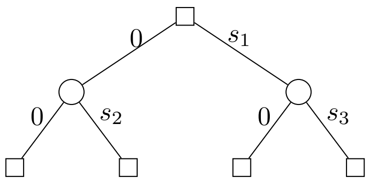

Fig. 1. An example of our game tree model, where the branching factor and depth are two. A rectangle and circle represent a node of the maximizing player and that of the minimizing player, respectively. Each edge represents a move and has a score, where the value of the optimal move is zero and that of a suboptimal move depends on the model. For Constant(a, b) tree, $s_1 = -a$ and $s_2, s_3 = b$. For Random(p, q) tree, $s_1 \sim \mathcal{U}([-p, -1])$ and $s_2, s_3 \sim \mathcal{U}([2^q, 2^q + p - 1]$, where $\mathcal{U}(\cdot)$ is uniform distribution.

We present the bias in three tree models: Constant, Constant', and Random. For parameters $0 < a \leq b$, Constant(a, b) represents an instance of constant trees where the value of each suboptimal move of the maximizing (minimizing) player is $-a$ (b). Constant'(a, b) is a variation where the value of each suboptimal move of the minimizing player is also a except for the last move, which has the value b. The difference between a and b represents the bias in suboptimal moves. In special cases when a equals b, the bias vanishes, and trees are equivalent to Finnsson's model [3]. For parameters $0 < p$ and $0 \leq q$, Random(p, q) represents an instance of random trees where the value of each suboptimal move of the maximizing (minimizing) player has a uniformly random value within the range $[-p, -1]$ $([2^q, 2^q + p - 1])$. Parameter q represents bias in suboptimal moves. In the special cases when q equals 0, the bias vanishes, and trees are equivalent to standard incremental game tree [15]. Fig. 1 shows an example of a tree. As found in later experiments and shown in Figs. 3(a) and 7(a), the bias in these models drastically degrades the performance of MCTS algorithms, with respect to the failure rate of identifying the optimal move at the root.

IV. Maximization of Reward Difference

We present an enhancement of MCTS based on the maximization of the average difference in the rewards of the optimal move and that of other moves. Let $\mathbb{K}$ be the set of legal moves at a root, μ_* the expected reward of the optimal move, and μ_i that of any move i where $i \in \mathbb{K}$. Then, we define the expected difference in rewards between the optimal move and the runner up:

$$\Delta := \mu_* - \max_{i \in \mathbb{K} \setminus \{*\}} \mu_i.$$

Intuitively, it is easier to identify the optimal move in positions with a large positive Δ. For example, in UCB1, it is proven that

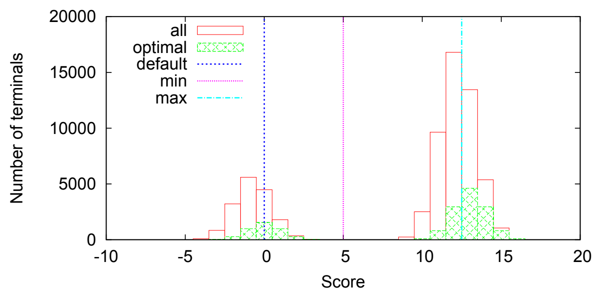

Fig. 2. Histogram of the game scores at terminal nodes of Constant'(1,13).

the expected number of visits to suboptimal moves is bounded by the following expression, which is inversely proportional the square of Δ [19]:

$$\mathbb{E}[T_i(n)] \leq \frac{8\ln(n)}{\Delta_i^2} + 1 + \frac{\pi^2}{3} \tag{4}$$

where $\Delta_i := \mu_* - \mu_i$. We can expect a similar effect in tree searches because of the similarities between the tree search of depth one and multi-armed bandit problems.

A. Analysis of Constant Models in One-Ply Search

By analyzing the Constant and Constant' models assuming one-ply search, we show the difficulty introduced by the bias in suboptimal moves. When we fix the depth of a game tree of MCTS to one and ignore the expansion step described in Sect. II-A, MCTS is equivalent to a multi-armed bandit problem at the root. As an intuitive example of the importance of Δ, Fig. 2 shows the histogram of the game scores at all terminal nodes in a game tree of Constant'(1,13), where the branching factor and depth are four and eight, respectively. In the figure, "all" includes all terminal nodes, while "optimal" includes the terminal nodes in the subtree corresponding to the optimal move at the root. In this configuration, the majority of terminal nodes have a positive score, and therefore, they are the win for the maximizing player with threshold zero that is a default value, as shown by "default" in the figure. When the bias in suboptimal moves increases, the win rate becomes much more uneven. Because a terminal node that is reached is uniformly random in each playout, the expected winning rate for all moves is 0.875, which almost equals that of the optimal move, 0.798. This means Δ is 0.077, almost zero. If the threshold is set around 4 to 8, as indicated by "min" in the figure, the winning rate of both all moves and the optimal move is 0.75, making Δ zero. This small value of Δ causes the difficulty in identifying the optimal move by MCTS.

However, when we set the threshold between 12 and 13 as indicated by "max" in the figure, then the winning rate for the optimal move becomes 0.515, while that of all moves is 0.234, where Δ realizes its maximum value of 0.28. In general, a good threshold can make Δ large and contribute to identifying the optimal move. Note that adjustments yielded by dynamic komi are not suitable in this situation. For example, because Score Situational tries to keep the average score zero, the threshold would be around 10 to 11, which is the weighted average of the red bars.

Although it is not available for players, we can identify the threshold that maximizes Δ in each tree by using the scores

Algorithm 3 Maximum Frequency method

```
Histogram[score] += 1
**return** $\arg\max_t$ Histogram[t]
```

at all terminal nodes. We employed that in our experiments to estimate the best cases of threshold adjustments. For models Constant(a, b) and Constant'(a, b), Δ with threshold t is identified by the following simple equation [18]:

$$\Delta(t) = \sum_{t<s<t+a} P[s] + 0.5P(t) + 0.5P(t+a), \tag{5}$$

where s is an integer, 0.5 stands for the reward of draw, $P[x]$ is the probability that the game score of a playout starting at the optimal move is x, and $P(x)$ is either $P[x]$ or zero if x is an integral value or not, respectively. Note that $P[x+a]$ represents the probability of the score of a playout starting at a suboptimal move because subtrees for the children of the root are equivalent except for the edge value from the parent. Therefore, the equation is derived from the definition $\Delta(t) = \mu_* - \mu = \left(\sum_{s>t} P[s] + 0.5P(t)\right) - \left(\sum_{s>t} P[s+a] + 0.5P(t+a)\right)$, by using the equivalence $\sum_{t<s\leq t+a} P[s] = \sum_{t<s<t+a} P[s] + P(t+a)$.

B. Maximum Frequency Threshold and Exploration Term

To increase Δ in practical situations in game playing, we present the Maximum Frequency method that sets the threshold as the maximum observed frequency, as listed in Algorithm 3. At each playout, the algorithm maintains the frequency of each score in a histogram and returns the best threshold based on it. The background is that the best threshold t that maximizes $\Delta(t)$ is given by $t = s - 0.5$, where s yields the maximum $P[s]$, as shown in (5) when assuming one-ply search and $a = 1$.[1] Because we do not know the optimal move $*$ nor the expected reward of each move μ_i in a tree search, we approximate $P[s]$, which is the probability that a playout starting at the optimal move will yield score s, by the observed frequency of all playouts yielding score s. Because the number of playouts for suboptimal moves is bounded (for example, by $\ln(n)$ in UCB1 as shown in (4)), it is expected that the error in this approximation becomes smaller when the number of playouts increases. For random trees, we do not yet have a similar analysis. Thus, we conducted numerical experiments to evaluate the effectiveness of the Maximum Frequency method. Note that the method does not have any effect in situations where the score with the maximum frequency is zero.

We also present an adjustment in the exploration term at each time when the threshold is changed by the method. The background is that, when the threshold changes, the winning ratio with the new threshold may be very different from that with the previous threshold. Let t the current time, t_c the last time the threshold was changed, t' the elapsed time from t_c, i.e., $t' = t - t_c$, and $T'_i(t)$ the number of visits to move i after t_c, i.e., $T'_i(t) = T_i(t) - T_i(t_c)$. We use time t' instead of t for the exploration terms of UCB1, KL-UCB, and Thompson sampling, from (1), (2), and (3) to (6), (7), and (8),

[1]Note that Constant(a, b) is almost equivalent to Constant$(1, b/a)$ except for granularity of scores.

respectively:

$$\overline{X}_{i,T_i(t)} + \sqrt{\frac{2\ln t'}{T'_i(t)}}, \tag{6}$$

$$\max q_i \in [0,1] : d(\overline{X}_{i,T_i(t)}, q_i) \le \frac{\ln t' + c\ln(\ln t')}{T'_i(t)}, \tag{7}$$

$$\alpha_i = \overline{X}_{i,T_i(t)} T'_i(t),\ \beta_i = (1-\overline{X}_{i,T_i(t)})T'_i(t). \tag{8}$$

When $t_c = 0$, i.e., the threshold has not been changed, the equations are equivalent to the original ones. Finally, to stabilize behavior at the beginning of a search, the threshold is frozen unless a certain number of playouts (e.g., ten for our experiments) has been conducted with the same threshold.

V. Experiments

In this section, we show through the results of experiments in which the bias in suboptimal moves strongly degrades the effectiveness of MCTS algorithms, and we explain how adjusting the threshold solves the problem. In this experiment, we adopt not only UCB1 but also KL-UCB and Thompson sampling as the base algorithm in MCTS to examine the effect not only in a specific case but in general. Moreover, we evaluated various enhancements for the algorithms; "Score" or "Value" means the Score Situational or Value Situational method of dynamic komi, respectively, as described in Sect. II-C. "MaxFrequency" means the Maximum Frequency method as presented in Sect. IV-B. "Plain" stands for MCTS without any adjustment, and "Static" uses a fixed threshold that maximizes Δ defined in (5), calculated for each tree by using the scores of all terminal positions that are not available for a player.

A. Configurations

We measured the effectiveness of each algorithm through the failure rate, following previously reported experiments [14]. The failure rate is the rate in which the algorithm fails to choose the optimal move at the root. For each tree instance, each algorithm, and each time $t = 2^n$, whether the algorithm fails or not is counted by whether the most visited move in the root so far is the optimal one. By averaging them over tree instances, we obtained the failure rate of each algorithm at time t. Because of space limitations, we show the results for trees where the branching factor is four and the depth is eight. Similar results were observed in some preliminary experiments with different tree sizes [18]. All MCTS algorithms expand a leaf node when it visits the leaf the second time. Parameters in dynamic komi methods were adjusted in advance in order to minimize the failure rate at time 625 in preliminary experiments. For Score, s and c were respectively set to 0.2 and 24, where they gave the best results when s was tested from 0 up to 1 in units of 0.1, and c was tested from 0 up to 30 in units of 1. For Value, X_{lo} and X_{hi} were set to 0.05 and 0.9, respectively.

Because a game tree governed by MCTS grows as the number of playouts increases in the expansion step explained in Sect. II-A, a leaf reaches a terminal state at some point. For each terminal node, a game-theoretical value of the win, loss, or draw is recognized by the algorithm. The threshold is always set to zero for terminal nodes even when either

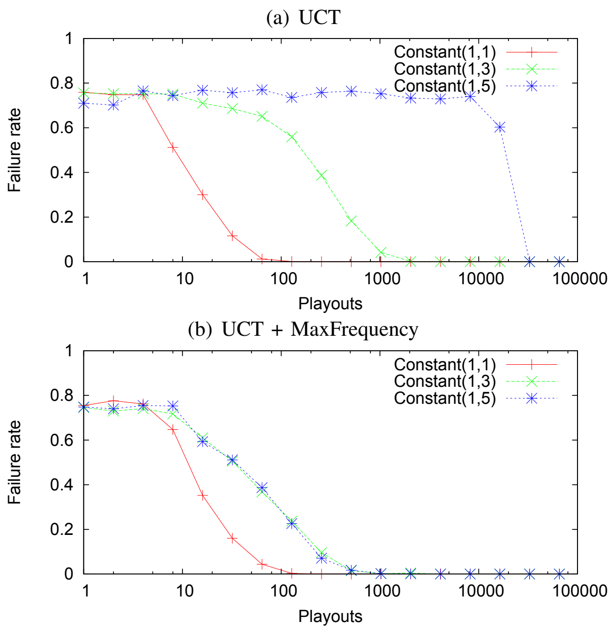

Fig. 3. Comparison between (a) original UCT (Plain) and (b) UCT enhanced by MaxFrequency, in Constant trees.

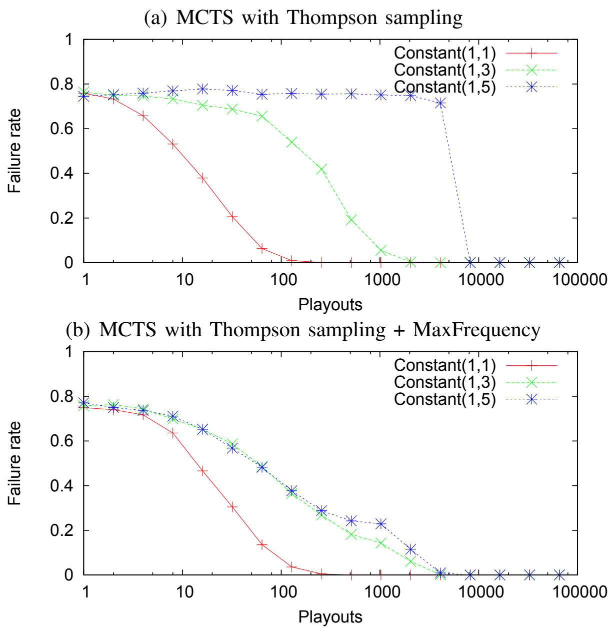

Fig. 4. Comparison between (a) MCTS with Thompson sampling (Plain) and (b) that enhanced by MaxFrequency, in Constant trees.

the Maximum Frequency or dynamic komi methods is employed in order to help converge the game-theoretical values. Also, game-theoretical values are backed up as in and-or tree search [24]. When the game-theoretical values of some moves are already identified, the winning (losing) move is always chosen (ignored) at the selection step. The losing moves are also ignored at the root when choosing the best move to count the failure rate.

B. Constant Trees

For experiments with constant trees, the failure rate is defined as the average over 1 000 runs with different random seeds. We measured how the performance is degraded by the bias in suboptimal moves with trees Constant(a, b) and by changing bias parameter b.

Fig. 3(a) plots the failure rates of UCT, which is MCTS based on UCB1. In the figures in this section, the vertical axis is the failure rate, and the horizontal axis is the number of playouts. We observed that the failure rate was initially about 0.75.[2] Then, it basically decreased, as the number of playouts increased, and converged to zero when all terminal nodes were expanded. However, we can see that the speed of convergence heavily depends on the kind of tree. For Constant$(1, 1)$ where the tree is not biased, about 100 playouts are enough for convergence, while the failure rate is still high even with $10\,000$ playouts for tree Constant$(1, 5)$. Note that Constant$(1, 5)$ is in an extremely difficult situation because the playout is almost always a win for the maximizing player. With our configuration of tree depth eight, tree Constant$(1, 5)$ is the most extreme case; when the minimizing player chooses suboptimal move of value $+5$ even once, the maximizing player wins because there is no chance for the maximizing player to choose a suboptimal move of value -1 more than four times. Therefore, we can see that the performance of UCT degrades with respect to the failure rate when a tree becomes more biased. Fig. 3(b) plots the failure rates of UCT with MaxFrequency (the Maximum Frequency enhancement presented in Sect. IV-B). We can see that MaxFrequency performed better than Plain in Constant$(1, 3)$ and in Constant$(1, 5)$, and that the number of playouts needed for the convergence was eight times smaller. However, Plain performs slightly better in Constant$(1, 1)$. Therefore, there is a trade-off between the efficiency in the unbiased situation and that in the biased situation.[3] We used two-tailed Welch's t-test with significance level 0.05 and found that the differences in failure rates are statistically significant in many points. For example, the failure rates of MaxFrequency were significantly lower in Constant$(1, 3)$ and in Constant$(1, 5)$, except for the point when the number of playouts is less than 16 in both cases and when $2\,048$ in Constant$(1, 3)$.

Fig. 4 plots the results of the same experiments when MCTS adopted Thompson sampling. Similar to the results with UCT, the performance of MCTS using Thompson sampling degrades when a tree becomes more biased, and the MaxFrequency enhancement improves the convergence speed in biased trees. The results of KL-UCB were similar to those observed with UCT or with Thompson sampling, and were therefore omitted due to space limitations.

To compare the failure rates of various enhancements, we show the results with two kinds of constant trees: Constant$(1, 5)$ and Constant'$(1, 13)$. Fig. 5(a) plots the failure rates of UCT with or without enhancements for tree Constant$(1, 5)$. While Plain maintains a high failure rate until $10\,000$ playouts, all enhancements effectively reduced the failure rates earlier. We can see that MaxFrequency as well as Static are most effective. Fig. 5(b) and 5(c) show the same

[2]Note that the expected failure rate is 0.75 if moves are randomly chosen with uniform probability.

[3]The inefficiency of MaxFrequency in the unbiased situation can be alleviated by making the threshold more stable. Also, even with the trade-off, MaxFrequency can be an effective way to make an agent for a majority voting system [25], [26]. It is because the diversity in agents improves the playing strength [26].

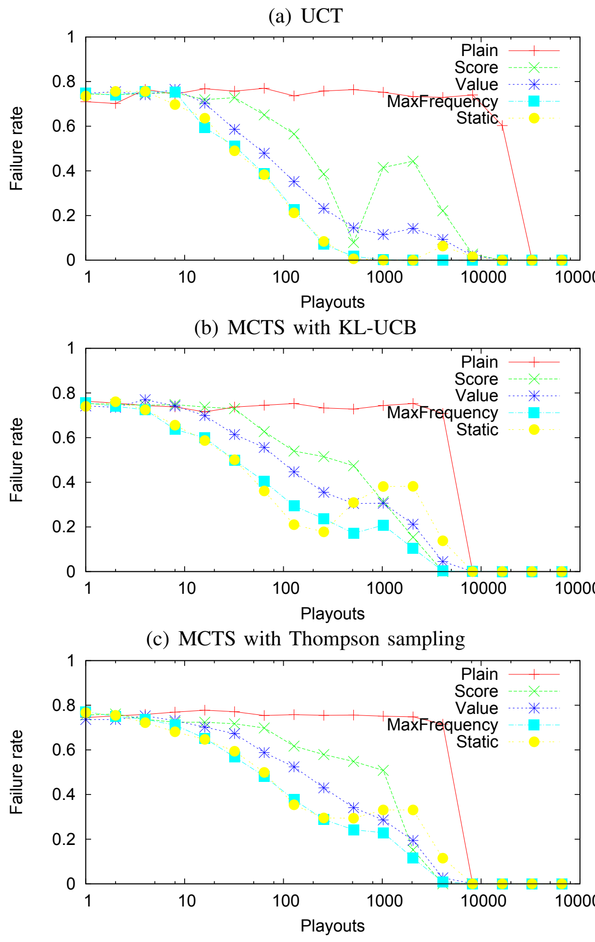

Fig. 5. Comparison of failure rates with or without enhancements, for tree Constant(1,5).

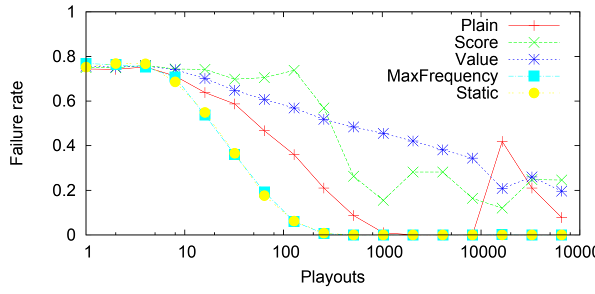

Fig. 6. Failure rates of enhancements (UCT, Constant'(1,13))

experimental results for MCTS using KL-UCB or Thompson sampling. The results were similar to those observed with UCT, though the performance without enhancement was better than that with UCT. Some fluctuations were observed before convergence for Score in Fig. 5(a) and for Static in Fig. 5(b) and 5(c). It is expected that the difference in the threshold (zero for terminal nodes and non-zero values for other leaves) caused the fluctuations. A similar comparison is shown in Fig. 6 for tree Constant'$(1, 13)$. While MaxFrequency and Static were still effective here, Score and Value had higher failure rates than that of Plain at almost all plot points. This is because they tend to set the threshold where $\Delta = 0$, as depicted in

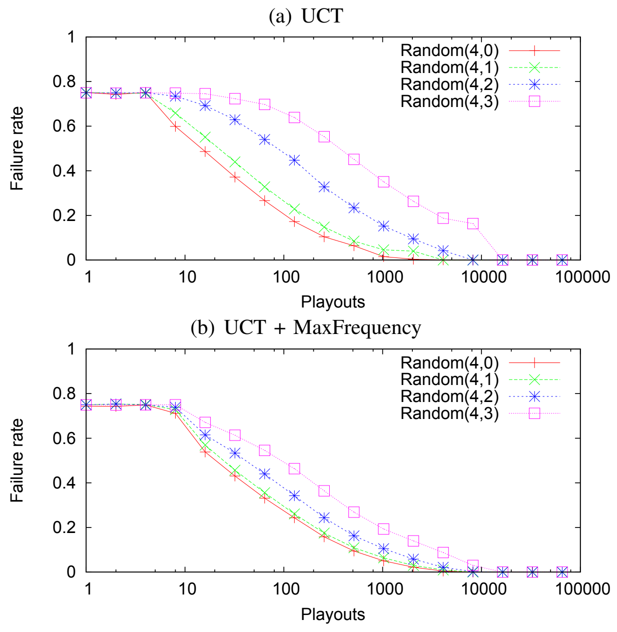

Fig. 7. Comparison between (a) original UCT and (b) that enhanced by MaxFrequency, in Random trees.

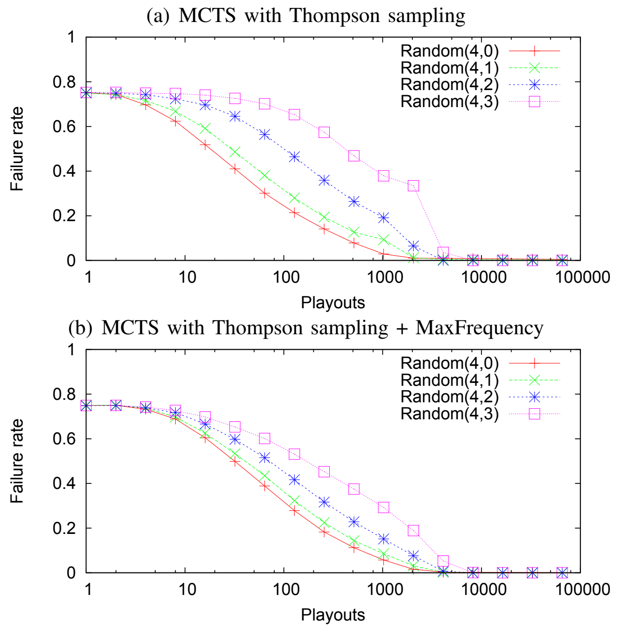

Fig. 8. Comparison between (a) MCTS with Thompson sampling and (b) that enhanced by MaxFrequency, in Random trees.

the histogram in Fig. 2. The failure rate of Plain rises before convergence. This is because Plain needs to investigate the suboptimal moves and understand that they are not winning position, during the time the winning rate of the optimal move is converging to a draw. Note that the empirical winning rate of suboptimal moves tends to be promising when no threshold adjustment is employed.

C. Random Trees

For experiments with incremental random game trees, 200 trees were randomly constructed where we ran each algorithm 200 times with different random seeds for each tree. Therefore, the failure rate is the average over $40\,000$ runs, following the experiments in the literature [14].

Figs. 7 and 8 show how the bias in suboptimal moves degrades the performance of UCT and MCTS with Thompson sampling for various random trees $\mathrm{Random}(4, q)$, where q represents the bias. Similar to the results in constant trees, the speed of convergence heavily depends on the kind of tree, especially when our enhancement is not employed, as shown in Figs. 7(a) and 8(a). For $\mathrm{Random}(4, 0)$, where the tree is not biased, about $1\,000$ playouts are sufficient for convergence, whereas the failure rate is still high even with $10\,000$ playouts for trees $\mathrm{Random}(4, 3)$, which have the maximum bias in these experiments. As shown in Figs. 7(b) and 8(b), MaxFrequency performed better than Plain in $\mathrm{Random}(4, 2)$ and $\mathrm{Random}(4, 3)$, while Plain performed better in $\mathrm{Random}(4, 0)$ and $\mathrm{Random}(4, 1)$. Therefore, the trade-off observed in the constant trees also existed in the random trees. The differences in failure rates for UCT with or without our enhancement were statistically significant for the points when the number of playouts was in $[16, 1024]$ for all trees. The differences were not clear in the points when the failure rates almost converged to zero. The results with MCTS with KL-UCB were almost the same and were therefore omitted due to space limitations. The results with MCTS with KL-UCB were almost the same and were therefore omitted due to space limitations.

To compare the failure rates of various enhancements, we show the results with $\mathrm{Random}(4, 3)$ in Fig. 9. We can see that MaxFrequency steadily improved the performance in random trees, though the differences are not dramatic compared to $\mathrm{Constant}(1, 5)$. Also, Value was effective here, while Score sometimes had a higher failure rate than that of Plain.

VI. Conclusion

This paper showed that the property, "bias in suboptimal moves," seriously degrades the performance of MCTS. This property means that the impact of missing the optimal move is much greater for one player than it is for the opponent. However, the game result of a position is still draw if both players play optimally. Therefore, it is a different situation from that in handicap games. In experiments with two kinds of synthetic models constant trees and incremental random trees, the degradation was widely observed in MCTS algorithms. The algorithms applied were UCT, which is based on UCB1, as well as MCTS based on KL-UCB or Thompson sampling, where KL-UCB and Thompson sampling are algorithms recently recognized to be effective in multi-armed bandit problems. We also presented an enhancement, the Maximum Frequency method, which alleviates the performance degradation by adjusting a threshold used in the conversion of playout scores into wins or losses. The method is based on the maximization of the difference between the expected reward of the optimal move and that of others. We have shown through our experiments that Maximum Frequency effectively solves the problem both for constant trees and for random trees.

References

[1] C. Browne, E. J. 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 Trans. Comput. Intellig. and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

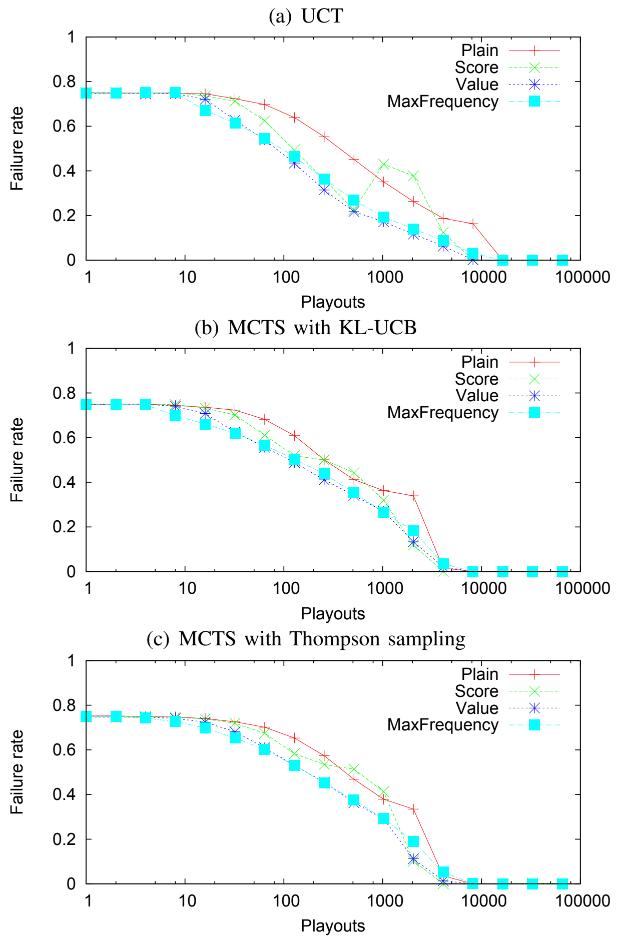

Fig. 9. Comparison of enhancements with MCTS algorithms in Random$(4, 3)$.

[2] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer go: Monte carlo tree search and extensions," *Commun. ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012.

[3] H. Finnsson and Y. Björnsson, "Game-tree properties and mcts performance," in *Proceedings of 2nd International General Game Playing Workshop (GIGA2011)*, 2011, pp. 23–30.

[4] N. Jouandeau and T. Cazenave, "Monte-carlo tree reductions for stochastic games," in *Technologies and Applications of Artificial Intelligence, 19th International Conference, TAAI 2014, Taipei, Taiwan, November 21-23, 2014. Proceedings*, ser. Lecture Notes in Computer Science, S. Cheng and M. Day, Eds., vol. 8916. Springer, 2014, pp. 228–238.

[5] P. Perick, D. St-Pierre, F. Maes, and D. Ernst, "Comparison of different selection strategies in monte-carlo tree search for the game of tron," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, Sept 2012, pp. 242–249.

[6] M. Lanctot, M. H. M. Winands, T. Pepels, and N. R. Sturtevant, "Monte carlo tree search with heuristic evaluations using implicit minimax backups," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*. IEEE, 2014, pp. 1–8.

[7] R. Ramanujan, A. Sabharwal, and B. Selman, "On the behavior of uct in synthetic search spaces," in *Proc. 21st Int. Conf. Automat. Plan. Sched., Freiburg, Germany*, 2011.

[8] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the success of perfect information monte carlo sampling in game tree search," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[9] N. Ikehata and T. Ito, "Monte-carlo tree search in ms. pac-man," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, Aug 2011, pp. 39–46.

[10] T. Graf, L. Schaefers, and M. Platzner, "On semeai detection in monte-carlo go," in *Computers and Games*. Springer, 2014, pp. 14–25.

[11] P. Baudiš, "Balancing mcts by dynamically adjusting the komi value," *ICGA Journal-International Computer Games Association*, vol. 34, no. 3, p. 131, 2011.

[12] O. Cappé, A. Garivier, O.-A. Maillard, R. Munos, and G. Stoltz, "Kullback-leibler upper confidence bounds for optimal sequential allocation," *Annals of Statistics*, vol. 41, no. 3, pp. 1516–1541, Jun. 2013.

[13] S. Agrawal and N. Goyal, "Analysis of thompson sampling for the multi-armed bandit problem," in *Proceedings of the 25th Annual Conference on Learning Theory (COLT)*, June 2012. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=166345

[14] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*. Springer, 2006, pp. 282–293.

[15] R. E. Korf and D. M. Chickering, "Best-first minimax search," *Artificial Intelligence*, vol. 84, pp. 299–337, 1996.

[16] T. Furtak and M. Buro, "Minimum proof graphs and fastest-cut-first search heuristics." in *IJCAI*, 2009, pp. 492–498.

[17] D. S. Nau, "An investigation of the causes of pathology in games," *Artificial Intelligence*, vol. 19, no. 3, pp. 257–278, 1982.

[18] T. Imagawa and T. Kaneko, "Improvement of performance of monte carlo tree search in positions where difficulty differs by turns," in *Proceedings of the Eighteenth Game Programming Workshop*, 2013, pp. 162–169, (in Japanese).

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[20] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, pp. 285–294, 1933.

[21] E. Kaufmann, N. Korda, and R. Munos, "Thompson sampling: An asymptotically optimal finite-time analysis," in *Algorithmic Learning Theory*, ser. Lecture Notes in Computer Science, N. Bshouty, G. Stoltz, N. Vayatis, and T. Zeugmann, Eds., vol. 7568. Springer Berlin Heidelberg, 2012, pp. 199–213.

[22] M. Enzenberger, M. Muller, B. Arneson, and R. Segal, "Fuego–an open-source framework for board games and go engine based on monte carlo tree search," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 4, pp. 259–270, 2010.

[23] T. Pepels, M. J. W. Tak, M. Lanctot, and M. H. M. Winands, "Quality-based rewards for monte-carlo tree search simulations," in *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, ser. Frontiers in Artificial Intelligence and Applications, T. Schaub, G. Friedrich, and B. O'Sullivan, Eds., vol. 263. IOS Press, 2014, pp. 705–710.

[24] M. H. Winands, Y. Bjornsson, and J.-T. Saito, "Monte-carlo tree search solver," in *Proceedings of the 6th international conference on Computers and Games*, ser. CG '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 25–36.

[25] T. Obata, T. Sugiyama, K. Hoki, and T. Ito, "Consultation algorithm for computer shogi: Move decisions by majority," in *Computers and Games*. Springer, 2011, pp. 156–165.

[26] L. S. Marcolino, A. X. Jiang, and M. Tambe, "Diversity beats strength?-towards forming a powerful team," in *15th International Workshop on Coordination, Organisations, Institutions and Norms (COIN 2013)*, 2013.