Monte Carlo Tree Search with Robust Exploration

Takahisa Imagawa[1,2(✉)] and Tomoyuki Kaneko[1]

[1] Graduate School of Arts and Sciences, The University of Tokyo, Tokyo, Japan
{imagawa,kaneko}@graco.c.u-tokyo.ac.jp
[2] Research Fellow of Japan Society for the Promotion of Science, Tokyo, Japan

Abstract. This paper presents a new Monte-Carlo tree search method that focuses on identifying the best move. UCT which minimizes the cumulative regret, has achieved remarkable success in Go and other games. However, recent studies on simple regret reveal that there are better exploration strategies. To further improve the performance, a leaf to be explored is determined not only by the mean but also by the whole reward distribution. We adopted a hybrid approach to obtain reliable distributions. A negamax-style backup of reward distributions is used in the shallower half of a search tree, and UCT is adopted in the rest of the tree. Experiments on synthetic trees show that this presented method outperformed UCT and similar methods, except for trees having uniform width and depth.

1 Introduction

Monte Carlo tree search (MCTS) algorithms including UCT [5] have achieved remarkable success, especially in the game of Go [10].

UCT is an algorithm based on the minimization of cumulative regret [1,13], which is suitable for estimating the expected score at each node. However, in game-playing algorithms, it is more important to identify the best move at the root, than to identify its score. Both goals are closely related but still different, as MTD(f) [16] exploits this difference in the context of $\alpha\beta$ search. Recent studies have shown that the performance of MCTS is improved by focusing on simple regret instead of on cumulative regret [6,14,15,19]. However, it is also known to be difficult to directly minimize the simple regret in tree search algorithms.

This paper presents an alternative Monte-Carlo tree search method that focuses on the confidence when choosing the best move. Our work is based on the careful combination of two ideas, each of which can be found in existing work: (1) negamax-style backup of the distribution of rewards at each interior node [4,18] in a main search tree, and (2) hybrid MCTS on top of UCT [15,19]. By using reward distributions obtained with a negamax-style backup, we can estimate the probability that the current move will be superseded by another move by a deeper search [4]. By using these distributions, we can identify the best leaf that most influences the confidence at the root. The negamax-style backup

A. Plaat et al. (Eds.): CG 2016, LNCS 10068, pp. 34–46, 2016.
DOI: 10.1007/978-3-319-50935-8_4

also contributes to the convergence [18]. We adopted UCT on an extended tree in order to obtain the distribution at each leaf in the main search tree.

The experiments on incremental random trees show that the presented method outperforms UCT, except in trees with a uniform width and depth.

2 Background and Related Work

This section briefly reviews related work on best-first search methods including Monte-Carlo tree search and negamax-style backup of reward distributions.

Monte Carlo tree search (MCTS) [5] is a kind of best-first algorithm that iteratively expands and evaluates a game tree by using a random simulation. In this section, we consider a general framework of best-first search algorithms where each iteration consists of the following four steps.

1. leaf selection: a leaf with the highest priority is selected.
2. expansion: if necessary, the leaf is expanded, and a leaf is selected again among newly created leaves.
3. evaluation: the leaf is evaluated or the evaluation is elaborated (e.g., by random simulation in UCT).
4. backpropagation: the evaluated value is shared throughout the tree.

We followed the convention in which MCTS starts with a tree having only its root and the immediate successor nodes. Then a leaf is expanded when we visit the leaf for the first time in step 2 [3,5,8]. Sometimes, a search tree is fixed to simplify analysis [18,19]. This is necessary in typical game programs in order to handle expansion of the tree during the search process. The other steps—leaf selection, evaluation, and backpropagation—characterize the search algorithms as discussed in the Subsects. 2.1 to 2.3. In this paper, we use move a or the position after move a interchangeably for simplicity, because a position after move a is defined without ambiguity in deterministic games.

2.1 Monte Carlo Tree Search and UCT

UCT [13] has been applied to many games including Go and has achieved remarkable success. For the evaluation of a leaf, it conducts random play (called simulation, or roll-out) starting at the position corresponding to the leaf and observes its outcome as a *reward*. In this paper, we assume reward r to be a win (1), loss (0), or draw (0.5), following the usual convention. Focusing on wins/losses/draws instead of raw game scores is known to be effective in Go [3,8]. Note that reward r is replaced by $1 - r$ for nodes where the opponent player moves. The observed reward is shared among the nodes between the leaf and the root in the backpropagation step, and the average of the rewards $\overline{X}_{i,t_i} = \sum_{t'}^{t_i} r_{t'}/t_i$ is maintained for each node i, where t_i is the number of visits to the node.

In the selection step, the algorithm descends from the root to a leaf by recursively selecting the most urgent child at each node. The urgency of a node among its siblings is determined by UCB defined by the following equation:

$$\mathrm{UCB} = \overline{X}_{i,t_i} + \sqrt{2 \ln t / t_i}, \tag{1}$$

where t is the number of times the parent of node i is visited up to now. UCT models a move in a game as an arm in a multi-armed bandit problem [1], assuming that the reward is stochastically determined by a distribution with mean μ_i when arm i is pulled. In typical game playing programs, we need to handle terminal or solved positions where the reward is fixed [20]. We need to ignore losing moves in the selection step by propagating win/loss information that is found to ancestors.

2.2 Improvements in UCT

UCT works so that the cumulative regret of the root is minimized [1,13]. Cumulative regret is the summation of the difference between the best choice, which is unknown to the agent, and his or her actual choice over time, $\sum_t^T (r^* - r_t)$, where r^* is the mean reward of the best arm and r_t is the observed reward at time t. Therefore, it is suitable for the estimation of the expected reward at a root. Simple regret is an alternative criterion for the accuracy of the final choice, $r^* - r_T$, where r_T is the mean reward of the agent's final choice after time T [6]. Informally, if one does more exploration of sub-optimal moves, simple (cumulative) regret decreases (increases).

Recent studies suggest that a hybrid approach is effective. This is a primary strategy for a root (or shallower part of a tree) to reduce simple regret at the root, and a sub-strategy to reduce cumulative regret in the rest of the tree. For a primary strategy, the approximated value of information (VOI) is reported to be effective in MCTS SR+CR [19], while SHOT [7] is used in H-MCTS [15]. H-MCTS has been tested on various domains; however, it inherits the limitation of SHOT in which the number of simulations must be fixed in advance. Our approach adopts alternative primary strategies and achieves an anytime algorithm. Additionally, Liu and Tsuruoka presented an adjustment in confidence bounds to reduce simple regret [14].

2.3 Minimax Backup of Reward Distribution

Usually, only the average of observed rewards is maintained at each node in MCTS. However, if a histogram of rewards is maintained in addition to the average, more information about positions can be extracted from the histogram[1] [11,12]. Moreover, reward distribution can be propagated in negamax style [4,18].

Bayes-UCT [18] is a Bayesian extension to UCT that selects a node having the maximum Bayes-UCT value among the siblings:

$$\text{Bayes-UCT1} := \hat{\mu}_i + \sqrt{2 \ln t / T_i(t)}, \quad \text{Bayes-UCT2} := \hat{\mu}_i + \sqrt{2 \ln t}\sigma_i, \qquad (2)$$

where $\hat{\mu}_i$ is the estimated mean of child i's reward distribution obtained by negamax backup, i.e., $\hat{\mu}_i = \int_x (1-x) p_i(x)$, where $p_i(x)$ is the probability that node i

[1] http://www.althofer.de/crazy-shadows.html.

Table 1. Comparison of similar work and our work. The top half of the table summarizes existing methods. The column "minimax backup" indicates how the distribution of each node is presented, where "-" means the minimax backup of reward distributions is not adopted. The column "primary strategy" gives the main strategy of choosing a leaf to be expanded or that for a playout. The column "sub-strategy" lists a substrategy for the deeper part of a tree if a hybrid approach is adopted, or "-" otherwise. The bottom half of the table summarizes our approaches discussed in this paper, which respectively adopt discrete and UCT for minimax backup and hybrid.

Existing method	minimax backup	primary strategy	sub-strategy
Baum and Smith [4]	discrete	QSS	-
BayesUCT [18]	Beta distribution	UCB (Bayes)	-
H-MCTS [15]	-	SHOT [7]	UCT
MCTS SR+CR [19]	-	ϵ-greedy, UCB$_{\sqrt{(\cdot)}}$, VOI	UCT
Yokoyama and Kitsuregawa [21]	discrete [4]	QSS [4], UCT	$\alpha\beta$ search
Work in this paper			
HB+E$^{\text{Expected}}$	discrete [4]	E$^{\text{Expected}}$	UCT
HB+E$^{\text{Robust}}$		E$^{\text{Robust}}$	
HB+E$^{\text{Terminal}}$		E$^{\text{Terminal}}$=QSS [4]	
HB+BayesUCT1		$\hat{\mu}_i + \sqrt{2 \ln t / t_i}$ [18]	
HB+BayesUCT2		$\hat{\mu}_i + \sqrt{2 \ln t}\sigma_i$ [18]	

has the game theoretical value x for player to move, and σ_i is the standard deviation of the distribution. Note that reward $1-x$ in a node corresponds to reward x in a parent node because the player whose turn it is to move changes with each move. Here, the observed average X_i in UCT (Eq. (1)) is replaced by $\hat{\mu}_i$, and it is shown that $\hat{\mu}_i$ converges more rapidly to the game theoretical reward of node i. There are many differences between this approach and our work, including exploration strategies and the adoption (or not) of a prior distribution or continuous probability distribution. In this work, we used $\hat{\mu}_i = \sum_{x \in \{0,0.5,1\}} (1-x) p_i(x)$ instead.

Before the introduction of MCTS, Baum and Smith presented a best-first search method that utilizes the reward distribution at each node [4].

Bayes-UCT as well as Baum and Smith's method assumes the existence of an external function to assign a probability distribution to each leaf, as a prior or special kind of evaluation function. In this work, the assumption is not necessary due to the use of UCT to yield the distribution. We also present improved strategies for exploration. The upper half of Table 1 summarizes the existing methods.

3 Exploration with Refinement of Confidence

We present a new hybrid MCTS algorithm that iteratively refines the confidence of the choice at the root. Our algorithm maintains the main tree for our primary

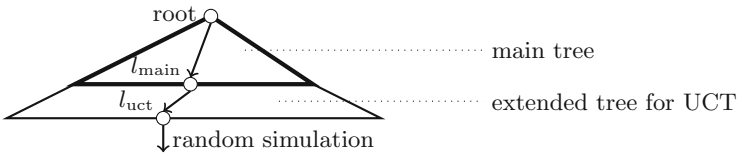

Fig. 1. Two parts of a search tree maintained with our algorithm

1. leaf selection: the most valuable leaf in the main tree is selected, with respect to the exploration strategy defined in Sect. 3.2.
2. evaluation: evaluation of the leaf l_{main} is elaborated by *budget* times of internal playouts in UCT with root l_{main}:
   (a) leaf selection: the most valuable leaf l_{uct} is identified by recursively descending successor node having the highest UCB1 value from l_{main}.
   (b) expansion: if the same condition in step 2 in Sect. 2 is satisfied, the leaf l_{uct} is expanded, and a new l_{uct} is then selected among newly created leaves.
   (c) evaluation: the evaluation of the leaf is elaborated by a random simulation that starts from the leaf.
   (d) backpropagation: the reward is stored in each node between l_{main} and l_{uct}. In addition to the usual back up of the reward, we maintain a histogram of the rewards (i.e., the frequencies of {0, 0.5, 1}) in each node.
3. expansion: after evaluation, if there appear such nodes in the UCT tree that meet the conditions described in Sect. 3.3, they are promoted into the main tree.
4. backpropagation: the distribution of the rewards is updated for each node between l_{main} and the root, by using Eq. (4).

Fig. 2. Outline of our algorithm

strategy introduced in the next subsections, as well as the extended tree for our sub-strategy, UCT, as depicted in Fig. 1. The algorithm iteratively extends the whole tree and refines evaluations in a best-first manner as listed in Fig. 2. Steps 1 through 4 are for the primary strategy, and steps 2(a) through 2(d) are for the sub-strategy, UCT (see Sect. 3.1). We first select leaf l_{main} in the main tree by using a strategy introduced in Sect. 3.2. Then, it runs playouts according to UCT *budget* times, where the budget is usually one. A newly created leaf is first added to the UCT tree and will be incorporated into the main tree when they meet the conditions described in Sect. 3.3.

3.1 Hybrid Backup of Reward Distribution

For each node in a search tree, an array of size 3 is assigned to present the reward distribution of the node. We introduce different backup procedures for the main tree and extended tree and call the scheme hybrid backup (HB). In each node in the extended tree, we maintain a histogram that holds the frequency of each reward value (i.e., $\{0, 0.5, 1\}$) and adapt it for the reward distribution of the node. When a playout is completed, the corresponding frequency count in the histogram in each node involved in the playout is updated by one.

For nodes in the main tree, a negamax-style backup is adopted. Let $p_i(x)$ be the probability that the game theoretical value of node i is x, and $\underline{c}^{(i)}(x)$ ($\overline{c}^{(i)}(x)$) be the cumulative distribution function, CDF, for probability $p_i(x)$ being less (more) than or equal to x:

$$\underline{c}^{(i)}(x) := \sum_{k:k\leq x} p_i(k), \quad \text{and} \quad \overline{c}^{(i)}(x) := \sum_{k:k\geq x} p_i(k). \tag{3}$$

Inversely, probability $p_i(x)$ is easily computed by function $\underline{c}^{(i)}(x)$ or $\overline{c}^{(i)}(x)$. We introduce the negamax backup of probability distributions, following Baum and Smiths method [4]. The distribution of internal node n is defined based on those of the successor nodes assuming that there is no correlation between sibling nodes:

$$\underline{c}^{(n)}(1-x) = \prod_{c\in \text{successor}(n)} \overline{c}^{(c)}(x). \tag{4}$$

The CDF $\underline{c}^{(i)}(x)$ in each node in the main tree is updated by using Eq. (4), so that it represents the negamax CDF of distributions of its successors. Recall that we limit the reward in $\{0, 0.5, 1\}$ in this work. Therefore, p_i or $\underline{c}^{(i)}(\cdot)$ can be stored in an array of size three. Also, $\sum_x p_i(x) = 1$ holds for any node i.

The intuition behind this design is that the estimated probability distribution is not accurate without a carefully designed prior distribution when the number of samples remains small. Therefore, we count the frequencies of reward values in the deeper part of the tree to average the results. If the number of samples is sufficient, the estimated distribution is accurate. Consequently, the negamax backup of probability distributions is adopted in the shallower part of the tree to achieve better convergence.

3.2 Exploration Strategy

Here, we introduce the primary strategy to select leaf l_{main} in the main tree to be explored next in step 1 in Fig. 2. Let m_0 be the best move at the root, estimated so far by searching. We define the uncertainty U as the difference between the estimated mean reward of root R and that of m_0:

$$U := \hat{\mu}'_R - \hat{\mu}'_{m_0}, \tag{5}$$

where each of $\hat{\mu}'_R$ ($= 1 - \hat{\mu}_R$) and $\hat{\mu}'_{m0}$ ($= \hat{\mu}_{m0}$) is the mean of the distribution at corresponding node (with respect to the root player). We can focus on the identification of the best move by minimizing U rather than by minimizing the variance of $\hat{\mu}_R$. For example, if U is zero, we can be confident that the best move is m_0 regardless of the value of the leaves, under the given distributions. Otherwise, there would be another move m_i that potentially has a better value than m_0.

Our goal is to select leaf l with the highest priority in the main tree to continue the exploration. Let U_l be the value of U after the exploration of leaf l, which is not known until the exploration is completed. When U_l is much smaller than U ($U_l \ll U$), it means that the best move become clearly distinguished from other moves. When U_l is much larger than U ($U_l \gg U$), it means that the

previous estimation of U was inaccurate and needed to be updated. Therefore, in the both cases, such l has a high priority for exploration, as discussed in the literature [4].

Below we discuss four strategies to select l (strategy four has two variants).

HB + E$^{\mathbf{Terminal}}$. One reasonable strategy is to select l that has the maximum absolute difference $|U_l - U|$, averaging over all possible U_l values with its probability. Let us assume that the exploration of leaf l reveals that l is terminal or solved with a game-theoretical value r according to its current distribution. Let $\hat{U}_l^{=r}$ be the value U_l when leaf l is found to be terminal with value r. Further assuming that the distributions of other leaves are not changed during the exploration of l, we can directly calculate $\hat{U}_l^{=r}$. Then, in strategy HB+E$^{\mathrm{Terminal}}$, we select l that has the maximum absolute difference $|U_l^{=r} - U|$ averaging over all possible $U_l^{=r}$ values with its probability:

$$\arg\max_l \textstyle\sum_{r\in\{0,0.5,1\}} p_l(r)|\hat{U}_l^{=r} - U| \tag{6}$$

This strategy is equivalent to QSS [4], though their work does not involve MCTS.

HB + E$^{\mathbf{Expected}}$. In many games, terminal nodes are relatively rarer than non-terminal nodes. Following this observation, we introduce a model in which leaf l is assumed to be an unsolved node after exploration with an additional playout result r with probability $p_l(r)$. This assumption is natural when we adopt UCT as a sub-strategy. Let $\hat{U}_l^{+r}$ be the value of U_l when the distribution of leaf l is changed by observing an additional result r. Then, we select l that has the maximum absolute difference $|U_l^{+r} - U|$, averaging all over possible U_l^{+r} values with its probability:

$$\arg\max_l \textstyle\sum_{r\in\{0,0.5,1\}} p_l(r)|\hat{U}_l^{+r} - U| \tag{7}$$

HB + E$^{\mathbf{Robust}}$. This is our main strategy that identifies the worst playout result on any leaf l that maximizes U_l. Recall that a U_l larger than the current U suggests an error in the current estimation. To extend this idea further, HB+E$^{\mathrm{Robust}}$ explores l that can achieve the maximum U_l considering all U_l^{+r}:

$$\arg\max_l \max_{r\in\{0,0.5,1\}} U_l^{+r}. \tag{8}$$

A distribution obtained by the negamax procedure tends to be unstable in that a single playout may substantially modify the distribution. This strategy is expected to remedy the instability by exploring such nodes first.

Property 1. Value U_l^{+r} is maximized when r is 1 (0) for a leaf l that is (is not) the descendant of the current best move m_0 at the root. Note that reward r here is for a player to move at the root. When a player to move at leaf l differs from that at the root, reward r at the root, corresponds to $1 - r$ at leaf l.

HB+BayesUCT. All strategies introduced so far are aimed at minimizing the uncertainty U. Alternatively, we can adopt BayesUCT [18]. HB+BayesUCT descends from the root to a leaf, choosing the node having the largest Bayes-UCB value shown in Eq. (2). Here, $\hat{\mu}_i$ is the mean of probability $p(r)$ for $r \in \{0, 0.5, 1\}$.

The lower half of Table 1 summarizes our strategies.

3.3 Implementation Details

Nodes in the extended tree should be promoted to the main tree in step 3 in each iteration in Fig. 2, when the sub-trees under the nodes are sufficiently searched. In our experiments, all of leaf l_{main}'s children are promoted at the same time, when the number of visits to l_{main} reaches at least 20 and when the minimum number of visits to each of them reaches at least 2. The condition on the number of visits is crucial, especially in strategies HB+$\text{E}^{\text{Terminal}}$ and HB+$\text{E}^{\text{Expected}}$. In these strategies, such a node has the least priority and is rarely selected for exploration if it has a single reward of positive frequency. If the playout result of node l up to this point is always a win, for example the same result is assumed to be observed in the next playout in $\text{E}^{\text{Terminal}}$ and $\text{E}^{\text{Expected}}$, then it yields that $U_l = U$. Here we note that HB+E^{Robust} is free from this problem.

In strategies HB+$\text{E}^{\text{Terminal}}$, HB+$\text{E}^{\text{Expected}}$ and HB+E^{Robust}, the computation of the priority of each leaf can be accelerated by first identifying the influence of each leaf on the root [4]. We followed this technique in the current work. Still, it requires computation proportional to the number of nodes in the main tree. Note that this computational cost is usually concealed because the number of nodes in the main tree is much less than that in the extended tree.

Also, the balance in the computational costs in a primary strategy and in UCT can be adjusted by the budget, which is the number of internal playouts performed in step 2. When the budget is more than 1, we need to estimate U_l after multiple playouts. In HB+E^{Robust}, such U_l is estimated without additional computational costs because reward r giving the maximum $\hat{U}_l^{+r}$ remains the same, regardless of the budget, for each leaf l by Property 1 (see 3.2). Additionally, in HB+$\text{E}^{\text{Terminal}}$, we assumed the same U_l for multiple playouts. However, in HB+$\text{E}^{\text{Expected}}$, U_l must be computed with additional costs.

We handled solved nodes using the method by Winands et al. [20]. Note that the reward distribution of a solved node automatically converges when a negamax-style backup is adopted. However, we still need to maintain solved nodes for UCT in the extended tree. Also, in HB+BayesUCT, solved nodes of a draw reward may be chosen in Eq. (2). Therefore, such nodes should be excluded from the candidates for exploration because the exploration no longer contributes to updating the probability distribution. Further it is noted that in UCT, solved nodes of a draw reward should be kept as candidates for exploration so as to stabilize the reward average.

4 Experimental Results

We conducted experiments on incremental random trees in order to compare the performance of various algorithms. A random value in a uniform distribution was assigned to each edge. The game score at a leaf was the summation of edge values from the root, and the reward of the leaf with respect to the root player was 1 for a positive score, 0.5 for score zero, and 0 for a negative score. All trees were generated with uniform width and depth, but some sub-trees were pruned in advance to introduce variations in width and depth. In the pruning procedure,

each internal node in a tree is set to be terminal with probability P, and the subtree under the node is pruned. Also, exactly one edge at each node is assigned value zero instead of a random value so that the score of the principal variation is zero, for simplicity of analysis [9]. However, it is known that the performance of UCT is sensitive to the rewards of moves at the root [2,12]. We introduced the bias parameter Bi that is added to the game scores of all leaves, in order to introduce diversity in the game score for the principal variation. Because each edge value is an integer, the reward of each leaf cannot be a draw when the bias is 0.5.

4.1 Failure Rates

We compared six algorithms: five variations of the proposed method with hybrid backup, and the usual UCT for reference. Algorithms HB+E$^{\text{Terminal}}$, HB+E$^{\text{Expected}}$, HB+E$^{\text{Robust}}$, HB+BayesUCT1, and HB+BayesUCT2 were introduced in Sect. 3. In this experiment, the budget in our algorithms was fixed in 1, and all algorithms incorporated MCTS Solver for handling solved nodes [20]. At the beginning of the search, we initialized the current best move randomly, because all moves have an equal mean of reward distribution. The current best move was replaced if and only if there appeared a better move with respect to the mean of its reward distribution.

Although we generated game trees in advance, each search algorithm starts with only its root with the immediate successor nodes and then gradually incorporates new leaves as explained in Sect. 3. We tested two configurations of branching factor and depth; (4, 12) and (8, 8). In addition, its terminal probability P was $0, 0.1$, or 0.2. For each configuration of trees, we generated 400 instances and ran each algorithm 200 times. We measured the effectiveness of each algorithm through the failure rate, following the method described by Kocsis and Szepesvári [13]. The failure rate is the rate in which the algorithm fails to choose the optimal move at the root.

The results are summarized in Table 2. The proposed method HB+E$^{\text{Robust}}$ achieved the best results among all methods in trees where terminal probability P was not zero, while UCT achieved the best results when P was zero. These results are consistent with previous reports that found the performance of UCT degrades when a tree has a non-uniform width or has traps each of which is a losing move [17,18]. The performance of both HB+E$^{\text{Robust}}$ and UCT improved when bias was added to the trees. The performance of BayesUCT was not as good as expected in these configurations. Although a detailed analysis of the reasons for this is beyond the scope of this paper, the differences in discrete histograms or Beta distributions in the representation of reward distributions may affect the performance. It might also depend whether a game tree is fixed or iteratively expanded during the search.

In Fig. 3, we can see how the failure rate decreases as the number of playouts increases. We can observe that the failure rate of HB+E$^{\text{Robust}}$ decreases faster than UCT and the other methods if the terminal probability is positive.

Table 2. Table of failure rates when there were 4000 playouts, where B is branching factor, D is maximum depth, P is terminal probability, and Bi is bias. In each setting, the lowest failure rate observed is indicated in bold.

B - D	P	Bi	Hybrid Backup (HB)					UCT
			$E^{Terminal}$	$E^{Expected}$	E^{Robust}	BayesUCT1	BayesUCT2	
4-12	0.0	0	0.079025	0.078487	0.078575	0.037750	0.129050	**0.013088**
		0.5	0.060987	0.058975	0.029712	0.057512	0.085550	**0.016987**
	0.1	0	0.001275	0.001212	**0.000188**	0.005525	0.017175	0.040137
		0.5	0.000163	0.000238	**0.000000**	0.001225	0.000988	0.000213
	0.2	0	**0.000000**	**0.000000**	**0.000000**	0.000650	0.000450	0.002288
		0.5	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
	0.0	1	0.050350	0.046487	0.025087	0.066387	0.145387	**0.023013**
	0.1	1	0.001213	0.001225	**0.000325**	0.034000	0.028837	0.007738
	0.2	1	**0.000000**	**0.000000**	**0.000000**	0.001088	0.002325	0.000025
	0.0	2	0.008500	0.008138	0.001512	0.003912	0.067475	**0.000000**
	0.1	2	0.000113	0.000100	**0.000000**	0.001438	0.007950	0.000013
	0.2	2	**0.000000**	**0.000000**	**0.000000**	**0.000000**	0.000450	**0.000000**
8-8	0.0	0	0.264350	0.251387	0.301200	0.777625	0.174887	**0.016825**
	0.1	0	0.090300	0.074125	**0.046825**	0.093362	0.068025	0.098437
	0.2	0	0.007125	0.004425	**0.001187**	0.036912	0.014563	0.067225

Table 3. Average milliseconds consumed by single playout.

	budget	P = 0.0	P = 0.1	P = 0.2
$HB+E^{Robust}$	1	0.151095	0.079921	0.041580
	10	0.021505	0.015718	0.011130
	20	0.016167	0.012834	0.008930

	budget	P = 0.0	P = 0.1	P = 0.2
UCT	1	0.012968	0.008496	0.004249

4.2 Acceleration by Increasing Budget

We measured the computational efficiency of $HB+E^{Robust}$, which was the most effective algorithm in the previous experiments. The computational cost of the primary strategy in $HB+E^{Robust}$ is more expensive than that of UCT. Therefore, the computational efficiency per playout is improved by increasing the number of internal playouts and the budget at the expense of exploration accuracy.

We measured the average consumed time for a playout by dividing the total time consumed by the number of playouts. We used trees with branching factor 4 and depth 12 and performed 4000 playouts for each tree, which means that the main tree is explored 4000/budget times. In our hybrid algorithms, UCT sometimes identifies that the root of the current exploration, which is a leaf in the main tree, is solved. In such cases, exploration on the node is stopped, and the distributions of the main tree are updated, even before the number of playouts reaches a given budget. We used a computer equipped with an AMD Opteron Processor 6274, 2.2 GHz, running Linux for this experiment.

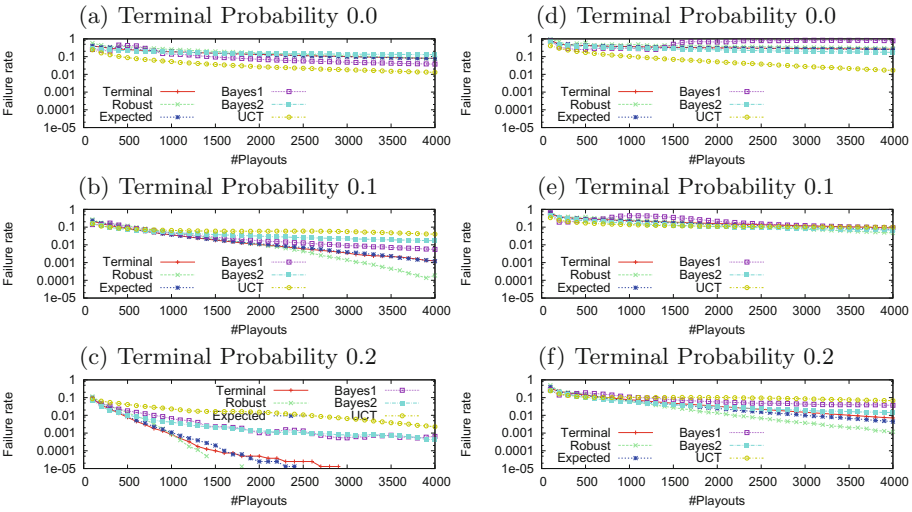

Fig. 3. Failure rates: (branching factor, depth) is $(4, 12)$ on left, and $(8, 8)$ on right.

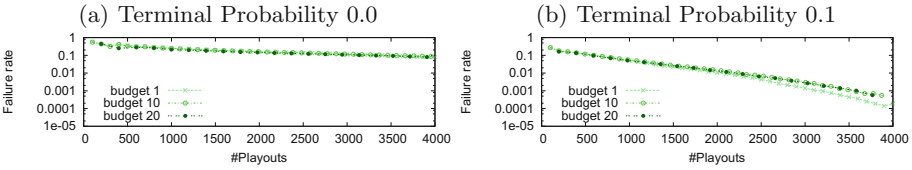

Fig. 4. Failure rates of HB+E$^{\text{Robust}}$ for budgets, 1, 10, and 20.

The failure rate is slightly increased by increasing the budget, although the difference is limited, as shown in Fig. 4. Table 3 lists the average time consumed per playout in HB+E$^{\text{Robust}}$ and in UCT. We can see that the efficiency improved by increasing the budget. Though UCT is still faster than HB+E$^{\text{Robust}}$ with a budget of 20, we argue that the difference is almost negligible in typical situations where a random simulation for each playout consumes about 1 ms.

5 Conclusion

This paper presented a new anytime Monte-Carlo tree search method that iteratively refines the confidence on the best move. It is estimated by the reward distribution of each move at the root, where the distributions of interior nodes are obtained by a negamax-style backup in the main game tree and by UCT in the extended tree. In each iteration, the leaf that most contributes to the confidence is explored further by UCT. The experiments on synthetic trees showed that the presented method outperformed UCT and similar methods, except for trees having uniform width and depth. Among several strategies, the experimental results suggest that a strategy expecting the worst playout outcome performed the best.

Acknowledgement. This work was partially supported by Grant-in-Aid for JSPS Fellows 16J07455.

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**(2–3), 235–256 (2002)
2. Baudiš, P.: Balancing MCTS by dynamically adjusting the komi value. ICGA J. Int. Comput. Games Assoc. **34**(3), 131 (2011)
3. Baudiš, P., Gailly, J.: PACHI: state of the art open source go program. In: Herik, H.J., Plaat, A. (eds.) ACG 2011. LNCS, vol. 7168, pp. 24–38. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31866-5_3
4. Baum, E.B., Smith, W.D.: A bayesian approach to relevance in game playing. Artif. Intell. **97**(1), 195–242 (1997)
5. Browne, C., Powley, E.J., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intellig. AI Games **4**(1), 1–43 (2012)
6. Bubeck, S., Munos, R., Stoltz, G.: Pure exploration in finitely-armed and continuous-armed bandits. Theor. Comput. Sci. **412**(19), 1832–1852 (2011). Algorithmic Learning Theory (ALT 2009)
7. Cazenave, T.: Sequential halving applied to trees. IEEE Trans. Comput. Intellig. AI Games **7**(1), 102–105 (2015)
8. Enzenberger, M., Muller, M., Arneson, B., Segal, R.: Fuego-an open-source framework for board games and go engine based on Monte Carlo tree search. IEEE Trans. Comput. Intellig. AI Games **2**(4), 259–270 (2010)
9. Furtak, T., Buro, M.: Minimum proof graphs and fastest-cut-first search heuristics. In: IJCAI, pp. 492–498 (2009)
10. Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., Teytaud, O.: The grand challenge of computer go: Monte Carlo tree search and extensions. Commun. ACM **55**(3), 106–113 (2012)
11. Graf, T., Schaefers, L., Platzner, M.: On semeai detection in Monte-Carlo go. In: Herik, H.J., Iida, H., Plaat, A. (eds.) CG 2013. LNCS, vol. 8427, pp. 14–25. Springer, Heidelberg (2014). doi:10.1007/978-3-319-09165-5_2
12. Imagawa, T., Kaneko, T.: Enhancements in Monte Carlo tree search algorithms for biased game trees. In: IEEE Computational Intelligence and Games (CIG), pp. 43–50 (2015)
13. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006). doi:10.1007/11871842_29
14. Liu, Y.C., Tsuruoka, Y.: Regulation of exploration for simple regret minimization in Monte-Carlo tree search. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG), pp. 35–42, August 2015
15. Pepels, T., Cazenave, T., Winands, M.H.M., Lanctot, M.: Minimizing simple and cumulative regret in Monte-Carlo tree search. In: Cazenave, T., Winands, M.H.M., Björnsson, Y. (eds.) CGW 2014. CCIS, vol. 504, pp. 1–15. Springer, Heidelberg (2014). doi:10.1007/978-3-319-14923-3_1
16. Plaat, A., Schaeffer, J., Pijls, W., de Bruin, A.: Best-first fixed depth minimax algorithms. Artif. Intell. **87**, 255–293 (1996)
17. Ramanujan, R., Sabharwal, A., Selman, B.: On adversarial search spaces and sampling-based planning. In: ICAPS, pp. 242–245 (2010)

18. Tesauro, G., Rajan, V., Segal, R.: Bayesian inference in Monte-Carlo tree search. In: the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010) (2010)
19. Tolpin, D., Shimony, S.E.: MCTS based on simple regret. In: AAAI (2012)
20. Winands, M.H.M., Björnsson, Y., Saito, J.-T.: Monte-Carlo tree search solver. In: Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) CG 2008. LNCS, vol. 5131, pp. 25–36. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87608-3_3
21. Yokoyama, D., Kitsuregawa, M.: A randomized game-tree search algorithm for shogi based on Bayesian approach. In: Pham, D.-N., Park, S.-B. (eds.) PRICAI 2014. LNCS (LNAI), vol. 8862, pp. 937–944. Springer, Heidelberg (2014). doi:10.1007/978-3-319-13560-1_81