An Optimal Computing Budget Allocation Tree Policy for Monte Carlo Tree Search

Yunchuan Li, Michael C. Fu, *Fellow, IEEE*, and Jie Xu, *Senior Member, IEEE*

***Abstract*—We analyze a tree search problem with an underlying Markov decision process, in which the goal is to identify the best action at the root that achieves the highest cumulative reward. We present a new tree policy that optimally allocates a limited computing budget to maximize a lower bound on the probability of correctly selecting the best action at each node. Compared to widely used upper confidence bound (UCB) tree policies, the new tree policy presents a more balanced approach to manage the exploration and exploitation tradeoff when the sampling budget is limited. Furthermore, UCB assumes that the support of reward distribution is known, whereas our algorithm relaxes this assumption. Numerical experiments demonstrate the efficiency of our algorithm in selecting the best action at the root.**

***Index Terms*—Machine learning, Monte Carlo tree search (MCTS), optimization algorithms, stochastic optimal control.**

I. Introduction

We consider a reinforcement learning problem where an agent interacts with an underlying environment. A Markov decision process (MDP) with finite horizon is used to model the environment. In each move, the agent will take an action, receive a reward, and land in a new state. The reward is usually random, and its distribution depends on both the state of the agent and the action taken. The distribution of the next state is also determined by the agent's current state and action. Our goal is to determine the optimal sequence of actions that leads to the highest expected reward. The optimality of the decision policy will be evaluated by the probability of correctly selecting the best action in the first stage of the underlying MDP.

If the distributions and the dynamics of the environment are known, the optimal set of actions can be computed through dynamic programming [2]. Under more general settings where the agent does not have perfect information regarding the environment, the authors in [3] proposed an adaptive algorithm based on a multiarmed bandit (MAB) model and upper confidence bound (UCB) [4]. The authors in [5] and [6] applied UCB to tree search, and [6] invented the term Monte Carlo tree search (MCTS) and used it in a Go-playing program for the first time. Since then, MCTS has been developed extensively and applied to various games such as Othello [7] and Go [8]. To deal with different types of problems, several variations of MCTS have been introduced, e.g., flat UCB (and its extension bandit algorithm for smooth trees) [9] and single-player MCTS (for single-player games) [10].

However, most bandit-based MCTS algorithms are designed to minimize regret (or maximize the cumulative reward of the agent), whereas in many situations, the goal of the agent may be to efficiently determine the optimal set of actions within a limited sampling budget. To the best of our knowledge, there is limited effort in the MCTS literature that aims at addressing the latter problem. Teraoka *et al.* [11] first incorporated best arm identification (BAI) into MCTS for a MIN-MAX game tree, and provided upper bounds of play-outs under different settings. [12] had an objective similar to [11], but with a tighter bound. Their tree selection policy selects the node with the largest confidence interval, which can be seen as choosing the node with the highest variance. In some sense, this is a pure exploration policy and would not efficiently use the limited sampling budget. In our work, we are motivated to establish a tree policy that intelligently balances exploration and exploitation (analogous to the objective of UCB). The algorithms developed in [11] and [12] are only for MIN-MAX game trees, whereas our new tree policy can be applied to more general types of tree search problems. The MCTS algorithm in [13] is more general than [11] and [12], but its goal is to estimate the maximum expected cumulative reward at the root node, whereas we focus on identifying the optimal action.

Algorithms that focus on minimizing regret tend to discourage exploration. This tendency can be seen in two ways. Suppose at some point an action was performed and received a small reward. To minimize regret, the algorithm would be discouraged from

Manuscript received June 22, 2020; revised June 30, 2020, April 16, 2021, and May 16, 2021; accepted May 30, 2021. Date of publication June 14, 2021; date of current version May 31, 2022. This work was supported in part by the National Science Foundation under Grants CMMI-1434419 and DMS-1923145, in part by the Air Force Office of Scientific Research under Grants FA95502010211 and FA9550-19-1-0383, in part by the Defense Advanced Research Projects Agency (DARPA) under Grant N660011824024, and in part by the UChicago Argonne LLC under Grant 1F-60250. This paper was presented in part at the 58th IEEE Conference on Decision, and Control, Nice, France, December 2019. Recommended by Associate Editor Q.-S. Jia. *(Corresponding author: Yunchuan Li.)*

Yunchuan Li is with the Department of Electrical and Computer Engineering and the Institute for Systems Research, University of Maryland, College Park, MD 20742 USA (e-mail: yli93@umd.edu).

Michael C. Fu is with the R. H. Smith School of Business and the Institute for Systems Research, University of Maryland, College Park, MD 20742 USA (e-mail: mfu@umd.edu).

Jie Xu is with the Department of Systems Engineering and Operations Research, George Mason University, Fairfax, VA 22030 USA (e-mail: jxu13@gmu.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TAC.2021.3088792.

Digital Object Identifier 10.1109/TAC.2021.3088792

taking this action again. However, the small reward could be due to the randomness in the reward distribution. Mathematically, the authors in [14] showed that for MAB algorithms, the number of times the optimal action is taken is exponentially more than suboptimal ones, which makes sense when the objective is to maximize the cumulative reward, since the exploration of other actions is highly discouraged. Such exploration–exploitation balance may be optimal under a different objective, such as minimizing regret, but when the goal is to select the optimal action, as in our setting, MAB-based algorithms would overweight exploitation. This leads to our second motivation: Is there a tree policy that explores suboptimal actions more to ensure the optimal action is found?

Apart from the lack of exploration as a result of the underlying MAB model's objective to minimize regret or maximize cumulative reward, most MCTS algorithms assume that the support of the reward distribution is bounded and known (typically assumed to be $[0, 1]$). With the support of reward distribution being known, the parameter in the upper confidence term in UCB is tuned or the reward is normalized. However, a general tree search problem may likely have an unknown and practically unbounded range of rewards. In such case, assuming a range can lead to very poor performance. Therefore, the third motivation of our research is to relax the known reward support assumption.

To tackle the challenge in balancing exploration and exploitation with a limited sampling budget for a tree policy, we model the tree selection problem at each stage as a statistical *ranking & selection* (R&S) problem (which is referred to as BAI in the computer science community) and propose a new tree policy for MCTS based on an adaptive algorithm from the R&S community. Similar to the MAB problem, R&S (BAI) assumes that we are given a set of bandit machines (often referred to as alternatives in the R&S literature) with unknown reward distributions, and the goal is to select the machine with the highest mean reward. Developments of BAI under various settings such as linear bandits and infinitely many arms can be found in [15], [16], and [17]; see [18] for a review. In this work, we will develop an MCTS tree policy based on the popular optimal computing budget allocation (OCBA) framework [19], [20]. OCBA was first proposed in [21], and aims to maximize the probability of correctly selecting the action with highest mean reward using a limited sampling budget. More recent developments of OCBA under various optimization goals and simulation settings include addressing multiple objectives [22], subset selection [23], and input uncertainty [24]. Applications of OCBA in energy and semiconductor manufacturing include [25], [26].

The objective of the proposed OCBA tree policy is to maximize the approximate probability of correct selection (APCS), which is a lower bound on the probability of correctly selecting the optimal action at each node. Intuitively, the objective function of the new OCBA tree selection policy would lead to an optimal balance between exploration and exploitation with a limited sampling budget, and thus, help address the drawbacks of existing work that either pursues pure exploration [11], [12] or exponentially discourages exploration [14]. Our new OCBA tree policy also removes the known and bounded support assumption for the reward distribution, because the new OCBA policy determines the sampling allocation based on the posterior distribution of each action, which is updated adaptively according to samples.

To summarize, the contributions of our work are primarily algorithmic and computational, and include the following.

1) We propose a new tree policy for MCTS that focuses on efficiently selecting the optimal action, where UCB is replaced by OCBA. The new OCBA tree selection policy also relaxes the UCB assumption of known bounded support on the reward distribution.
2) We present a sequential algorithm to implement the new OCBA tree policy that maximizes the APCS at each sampling stage, where the selection policy converges to the optimal action.
3) To support the OCBA-based MCTS algorithm, we provide convergence results and bounds based on OCBA theory, which highlights the exploration-exploitation tradeoff of OCBA that is more suitable than UCB for identifying the best action.
4) We demonstrate the efficiency of our algorithm through numerical experiments.

Remark 1: In much of the computer science/artificial intelligence literature, an algorithm that focuses on determining the optimal set of actions under a limited budget is defined as a pure exploration algorithm (see, e.g., [27]–[29]), whereas we view such algorithms as retaining a balance between exploration and exploitation, as the analysis in Section III shows. In statistical R&S, pure exploration algorithms generally imply sampling based primarily on the variance of each action, which often leads to sampling suboptimal actions more. This will become clearer in the Section V, where we show that OCBA-MCTS actually samples less those highly suboptimal actions and "exploits" those potential actions more.

The rest of the article is organized as follows. We present the problem formulation in Section II, and review the proposed OCBA-MCTS algorithm in Section III. Theoretical analyses, including convergence results and exploration–exploitation analysis, are carried out in Section IV. Proofs are given in the Appendix. Numerical examples are presented in Section V to evaluate the performance of our algorithm. Section VI concludes this article and points to future research directions.

A preliminary version of this work was presented in [1], where a simpler tree policy (not employing the node representation adopted in the current work) was used. In addition to improving the efficiency of the MCTS algorithm, here we show that our proposed algorithm converges asymptotically to the optimal action, and provide an exploration–exploitation tradeoff analysis, both analytically and through a more comprehensive set of numerical experiments.

II. Problem Formulation

Consider a finite horizon MDP $M = (X, A, P, R)$ with horizon length H, finite state space X, finite action space A with $|A| > 1$, bounded reward function $R = \{R_t, t = 0, 1, \ldots H\}$, such that R_t maps a state-action pair to a random variable (r.v.), and transition function $P = \{P_t, t = 0, 1, \ldots H\}$, such that P_t maps a state-action pair to a probability distribution over X. We assume that P_t is unknown and/or $|X|$ and $|A|$ are very large,

and hence it is not feasible to solve the problem by dynamic programming. Further define X_a and A_x as the available child states when taking action a and available actions at state x, respectively. Denote by $P_t(x, a)(y)$ the probability of transitioning to state $y \in X_a$ from state $x \in X$ when taking action $a \in A_x$ in stage t, and $R_t(x, a)$ the reward in stage t by taking action a in state x. Let Π be the set of all possible nonstationary Markovian policies $\pi = \{\pi_i | \pi_i : X \to A, i \geq 0\}$.

Bandit-based algorithms for MDPs seek to minimize the expected cumulative regret, whereas our objective is to identify the best action that leads to maximum total expected reward given by $\mathrm{E}[\sum_{t=0}^{H-1} R_t(x_t, \pi_t(x_t))]$ for given $x_0 \in X$. We first define the optimal reward-to-go value function for state x in stage i by

$$V_i^*(x) = \max_{\pi \in \Pi} \mathrm{E}\left[\sum_{t=i}^{} R_t(x_t, \pi_t(x_t)) \;\middle|\; x_i = x\right], \quad i = 0, 1, \ldots, H-1 \tag{1}$$

with $V_H^*(x) = 0$ for all $x \in X$. Also define

$$Q_i(x, a) = \mathrm{E}[R_i(x, a)] + \sum_{y \in X_a} P_t(x, a)(y) V_{i+1}^*(y)$$

with $Q_H(x, a) = 0$. It is well known [2] that (1) can be written via the standard Bellman optimality equation

$$\begin{aligned} V_i^*(x) &= \max_{a \in A_x} (\mathrm{E}[R_i(x, a)] + \mathrm{E}_{P_i(x,a)} V_{i+1}^*(Y)) \\ &= \max_{a \in A_x} (\mathrm{E}[R_i(x, a)] + \sum_{y \in X_a} P_t(x, a)(y) V_{i+1}^*(y)) \\ &= \max_{a \in A_x} (Q_i(x, a)), \quad i = 0, 1, \ldots, H-1 \end{aligned}$$

where $Y \sim P_i(x, a)(\cdot)$ represents the random next state.

Since we are considering a tree search problem, some additional notation and definitions beyond MDP settings are needed. Define a state node by a tuple that contains the state and the stage number:

$$\mathbf{x} = (x, i) \in \mathbf{X} \quad \forall x \in X, \; 0 \leq i \leq H$$

where $\mathbf{X}$ is the set of state nodes. Similarly, we define a state-action node by a tuple of state, stage number, and action (i.e., a state node followed by an action)

$$\mathbf{a} = (\mathbf{x}, a) = (x, i, a) \quad \forall x \in X, \; 0 \leq i \leq H, \; a \in A_x.$$

Now, we can rewrite the immediate reward function, value function for state, state-action pair with state node and state-action node, and state transition distribution, respectively, by

$$\begin{aligned} R(\mathbf{a}) &= R(\mathbf{x}, a) := R_i(x, a) \\ V^*(\mathbf{x}) &:= V_i^*(x) \\ Q(\mathbf{a}) &= Q(\mathbf{x}, a) := Q_i(x, a) \\ P(\mathbf{a}) &= P(\mathbf{x}, a) := P_i(x, a). \end{aligned}$$

Similarly, $V^*(\mathbf{x})$ and $Q(\mathbf{x}, a)$ are assumed to be zero for all terminal state nodes $\mathbf{x}$. To make our presentation clearer, we adopt the following definitions based on nodes: Define $N(\mathbf{x})$ and $N(\mathbf{x}, a)$ the number of visits to node $\mathbf{x}$ and $(\mathbf{x}, a)$, respectively, $\mathbf{X_a}$ the set of child state nodes given parent nodes, and $A_\mathbf{x}$ the set of available child actions at node $\mathbf{x}$, respectively.

Traditionally, MCTS algorithms aim at estimating $V^*(\mathbf{x})$ and model the selection process in each stage as an MAB problem, i.e., view $Q(\mathbf{x}, a)$ as a set of bandit machines, where $(\mathbf{x}, a)$ are child state-action nodes of $\mathbf{x}$ ([3], [5]), and minimize the *regret*, namely,

$$\begin{aligned} &\min_{a_1, \ldots, a_N \in A_\mathbf{x}} \{N \max_{a \in A_\mathbf{x}} (Q(\mathbf{x}, a)) - \sum_{k=1} Q(\mathbf{x}, a_k)\} \\ &\qquad = \{N V^*(\mathbf{x}) - \sum_{k=1} Q(\mathbf{x}, a_k)\} \end{aligned}$$

for $\mathbf{x}$ in stage $1, 2, \ldots H$, where N and a_k are the number of rollouts/simulations (also known as total sampling budget in much of R&S literature) and the kth action sampled at state node $\mathbf{x}$ by the tree policy, respectively. The meaning of rollout will be clearer in Section III. In this article, our goal is to identify the optimal action that achieves the highest cumulative reward at the root with initial state x, that is, find

$$a_{\mathbf{x}_0}^* = \arg \max_{a \in A_{\mathbf{x}_0}} Q(\mathbf{x}_0, a)$$

where the root state node $\mathbf{x}_0 = (x, 0)$. Let $\hat{Q}(\mathbf{x}, a) = R(\mathbf{x}, a) + V^*(\mathbf{y})$ be the random cumulative reward by taking action a at state node $\mathbf{x}$, where $\mathbf{y}$ is the random state node reached. Clearly, $\hat{Q}(\mathbf{x}, a)$ is a random variable. We assume $\hat{Q}(\mathbf{x}, a)$ is normally distributed with known variance, and its mean $\mu(\mathbf{x}, a)$ has a conjugate normal prior with a mean equals $Q(\mathbf{x}, a)$. Hence we have

$$Q(\mathbf{x}, a) = \mathrm{E}[\mathrm{E}[\hat{Q}(\mathbf{x}, a) | \mu(\mathbf{x}, a)]].$$

Remark 2: For our derivations, we assume the variance of the sampling distribution of $\hat{Q}(x, a)$ is known; however, in practice, the prior variance may be unknown, in which case estimates such as the sample variance are used [20].

Consider the noninformative case, i.e., the prior mean $Q(\mathbf{x}, a)$ is unknown, it can be shown that [30] the posterior of $\mu(\mathbf{x}, a)$ given observations (i.e., samples) is also normal. For convenience, define the tth sample by $\hat{Q}^t(\mathbf{x}, a)$. Then the conditional distribution of $\mu(\mathbf{x}, a)$ given the set of samples $(\hat{Q}^1(\mathbf{x}, a), \hat{Q}^2(\mathbf{x}, a), \ldots, \hat{Q}^{N(\mathbf{x},a)}(\mathbf{x}, a))$ is

$$\tilde{Q}(\mathbf{x}, a) \sim N(\bar{Q}(\mathbf{x}, a), \frac{\sigma^2(\mathbf{x}, a)}{N(\mathbf{x}, a)}) \tag{2}$$

where

$$\bar{Q}(\mathbf{x}, a) = \frac{1}{N(\mathbf{x}, a)} \sum_{t=1}^{N} \hat{Q}^t(\mathbf{x}, a)$$

$$\tilde{Q}(\mathbf{x}, a) = \mu(\mathbf{x}, a) | (\hat{Q}^1(\mathbf{x}, a), \hat{Q}^2(\mathbf{x}, a), \ldots, \hat{Q}^{N(x,a)}(\mathbf{x}, a))$$

and $\sigma^2(\mathbf{x}, a)$ is the variance of $\hat{Q}(\mathbf{x}, a)$ and can be approximated by the sample variance

$$\hat{\sigma}^2(\mathbf{x}, a) = \frac{1}{N(\mathbf{x}, a)} \sum_{t=1}^{N} \hat{Q}^t(\mathbf{x}, a) - \bar{Q}(\mathbf{x}, a) \quad .$$

Remark 3: If the samples of $Q(\mathbf{x}, a)$ are not normally distributed, the normal assumption can be justified by batch sampling and the central limit theorem.

Under these settings, our objective is to maximize the probability of correct selection (PCS) defined by

$$\text{PCS} = P\left(\bigcap_{a \in A, a \neq \hat{a}^*_{\mathbf{x}}} (\tilde{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) \geq \tilde{Q}(\mathbf{x}, a))\right) \tag{3}$$

for a state node $\mathbf{x}$, where $\hat{a}^*_{\mathbf{x}}$ is the action that achieves the highest mean sample Q-value at such node, i.e., $\hat{a}^*_{\mathbf{x}} = \arg\max_{a \in A_{\mathbf{x}}} \bar{Q}(\mathbf{x}, a)$.

PCS is hard to compute because of the intersections in the (joint) probability. We seek to simplify the joint probability by changing the intersections to sums using the Bonferroni inequality to make the problem tractable. By the Bonferroni inequality, PCS is lower bounded by the APCS, that is,

$$\text{PCS} \geq 1 - \sum_{a \in A_{\mathbf{x}}, a \neq \hat{a}^*_{\mathbf{x}}} P\left(\tilde{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) \leq \tilde{Q}(\mathbf{x}, a)\right) =: \text{APCS}. \tag{4}$$

The objective of our new tree policy is to maximize APCS as given in (4). Compared to MAB's objective of minimizing the expected cumulative regret, this objective function will result in an allocation of sampling budget to alternative actions in a way that optimally balances exploration and exploitation. This objective function is motivated by the OCBA algorithm [19], [20] in the R&S literature. We will present and analyze our OCBA tree policy in the following sections.

III. Algorithm Description

In this section, we first briefly describe the main four phases, i.e., *selection*, *expansion*, *simulation*, and *backpropagation*, in an MCTS algorithm. Then, we propose a novel tree policy in the selection stage that aims at finding the optimal action at each state node.

A. Canonical MCTS Algorithm

Here we briefly summarize the four phases in a typical MCTS algorithm. We refer readers to [31] for a complete illustration of these phases. Algorithm 1 represents a canonical MCTS, with detailed descriptions of the main phases below.

1) Selection: In this phase, the algorithm will navigate down the tree from the root state node to an expandable node, i.e., a node with unvisited child nodes. We assume that expansion is automatically followed when a state-action is encountered. Therefore, when determining the path down, there are three possible situations as follows.

a) If a state-action node is encountered (denoted by $(\mathbf{x}, a)$), we will land into a new state node $\mathbf{y}$ which is obtained by calling the expansion function. Then, we continue with the selection algorithm.

b) If an expandable state node (which could be a leaf node) is encountered, we call the expansion function to add a new child state-action node and a state node (by automatically expanding the state-action node) to the path. Then, we stop the selection phase and return the path from the root to this state node. Finally, we proceed with the simulation and backpropagation phase.

c) If an unexpandable state node is encountered (denoted by $\mathbf{x}$), we employ a *tree policy* to determine which child action to sample. Then we enter the new state-action node $(\mathbf{x}, a)$ and continue the selection algorithm with this state-action node. The tree policies can be briefly categorized into two types: Deterministic, such as UCB1 and several of its variants (e.g., UCB-tuned, UCB-E), and stochastic, such as ϵ-greedy and EXP3; see [31] for a review.

2) Expansion: In this phase, a random child state or state-action node of the given node is added. If the incoming node is a state node $\mathbf{x}$, the next node is selected randomly (usually uniform) from those unvisited child state-action nodes. If the incoming node is a state-action node $(\mathbf{x}, a)$, the subsequent state node is found by simply sampling from distribution $P(\mathbf{x}, a)(\cdot)$.

3) Simulation: In some literature, this phase is also known as "rollout." The simulation phase starts with a state node. The purpose of this step is to simulate a path from this node to a terminal node and produce a sample of cumulative reward by taking this path (which is a sample of the value for this node). The simulated path is taken by a *default policy*, which is to usually sample the feasible child sate-action nodes uniformly. With this node's value sample, we may proceed to the backpropagation phase.

4) Backpropagation: This phase simply takes the simulated node value and updates the values of the nodes in the path (obtained in selection step) backward.

In the next section, we will propose our tree policy based on OCBA and illustrate the detailed implementations of the four phases.

B. OCBA Selection Algorithm

We now present an efficient tree policy to estimate the optimal actions in every state node by estimating $V^*(\mathbf{x})$ and $Q(\mathbf{x}, a)$ for all possible $a \in A_{\mathbf{x}}$ at the state node. Denote the estimates of $V^*(\mathbf{x})$ at node $\mathbf{x}$ by $\hat{V}^*(\mathbf{x})$, which is initialized to 0 for all state nodes. Our algorithm estimates $Q(\mathbf{x}, a)$ for each action a by its sample mean, and selects the action that maximizes the sample mean as $\hat{a}^*_{\mathbf{x}}$. During the process, the estimate of $Q(\mathbf{x}, a)$ is given by (2) and the proposed new OCBA tree policy is applied. Our algorithm follows the algorithmic framework described in Section III-A, with the tree policy changed to OCBA and other mild modifications.

The structure of the proposed OCBA-MCTS algorithm is shown in Algorithms 1 to 6. There are two major characteristics: The first is to use the proposed OCBA algorithm for the tree policy. The second is to require each state-action node to be expanded $n_0 > 1$ times, because we need a sample variance

for each state-action node, which will become clearer after the tree policy illustration. The process is run for a prespecified N times (which will be later referred to as number of rollouts or sampling budget) from the root state node $\mathbf{x}_0$, after which a partially expanded tree is obtained and the optimal action $\hat{a}^*_{\mathbf{x}_0}$ can be derived.

When steering down the tree and a state node $\mathbf{x}$ is visited, the selection phase, which is illustrated in Algorithm 2, will first determine if there is a child state-action node that was visited for less than n_0 times at the given state node. If there is, then the state-action node will be sampled and added to the path. In other words, we try to expand each state node when it is visited, and require each node to be expanded n_0 times. If all the state-action nodes are well expanded, Algorithm 2 will call Algorithm 3 (OCBA Selection), which calculates the allocation of samples to child state-action nodes of the current state node for a total sampling budget $\sum_{a\in A} N(\mathbf{x}, a) + 1$. To determine the number of samples allocated to each state-action node, denoted by $(\tilde{N}(\mathbf{x}, a_1), \tilde{N}(\mathbf{x}, a_2), \ldots, \tilde{N}(\mathbf{x}, a_{|A_\mathbf{x}|}))$ (where $a_i \in A_\mathbf{x}$, $i = 1, \ldots, |A_\mathbf{x}|$), the OCBA tree policy first identifies the child state-action node with the largest sample mean (sample optimal) and finds the difference between the sample means of the sample optimum and all other nodes

$$\hat{a}^*_\mathbf{x} := \arg\max_a \bar{Q}(\mathbf{x}, a),$$
$$\delta_\mathbf{x}(\hat{a}^*_\mathbf{x}, a) := \bar{Q}(\mathbf{x}, \hat{a}^*_\mathbf{x}) - \bar{Q}(\mathbf{x}, a),\ \forall a = \hat{a}^*_\mathbf{x}.$$

The set of allocations $(\tilde{N}(\mathbf{x}, a_1), \tilde{N}(\mathbf{x}, a_2), \ldots, \tilde{N}(\mathbf{x}, a_{|A|}))$ that maximizes APCS can be obtained by solving the following set of equations:

$$\frac{\tilde{N}(\mathbf{x}, a_{n+1})}{\tilde{N}(\mathbf{x}, a_n)} = \left(\frac{\sigma(\mathbf{x}, a_{n+1})/\delta_\mathbf{x}(\hat{a}^*_\mathbf{x}, a_{n+1})}{\sigma(\mathbf{x}, a_n)/\delta_\mathbf{x}(\hat{a}^*_\mathbf{x}, a_n)}\right)^2$$
$$\forall a_n, a_{n+1} = \hat{a}^*_\mathbf{x},\ a_n, a_{n+1} \in A_\mathbf{x} \quad (5)$$

$$\tilde{N}(\mathbf{x}, \hat{a}^*_\mathbf{x}) = \sigma(\mathbf{x}, \hat{a}^*_\mathbf{x}) \sqrt{\sum_{a\in A, a=\hat{a}^*_\mathbf{x}} \frac{(\tilde{N}(\mathbf{x}, a))^2}{\sigma^2(\mathbf{x}, a)}} \quad (6)$$

$$\sum_{a\in A} \tilde{N}(\mathbf{x}, a) = \sum_{a\in A} N(\mathbf{x}, a) + 1. \quad (7)$$

The derivations of (5) to (7) are illustrated in the Appendix.

After the new budget allocation is computed, the algorithm will select the "most starving" action to sample [20], i.e., sample

$$\hat{a} = \arg\max_{a\in A_\mathbf{x}} (\tilde{N}(\mathbf{x}, a) - N(\mathbf{x}, a)). \quad (8)$$

An alternative approach recently proposed in [32] optimizes a one-step look ahead value function to sequentially determine the action to sample, and asymptotically achieves the optimal sampling ratios.

We highlight some major modifications to the canonical MCTS in the proposed algorithm. First, in the selection phase, we will try to expand all "expandable" nodes visited when obtaining a path to leaf. Since the variances of the values of a state node's child nodes are required in the proposed tree policy, we define a state node as expandable if it has child nodes that are visited less than $n_0 > 1$ times. State-action nodes are always expandable.

At the expansion phase as shown in Algorithm 4, a state-action node is expanded by simply sampling the transition distribution $P(\mathbf{x}, a)(\cdot)$, and the resulting state node is subsequently added to the path. The reward by taking the action in the state node is also recorded and will be used in the backpropagation stage.

In the simulation and backpropagation phases illustrated in Algorithm 5 and 6, a leaf-to-terminal path is simulated, and its reward is used to update the value for the leaf node. If we denote the leaf node and the reward from the simulated path by $\mathbf{x}_l$ and r, respectively, the leaf node value estimate is updated by

$$\hat{V}^*(\mathbf{x}_l) \leftarrow \frac{N(\mathbf{x}_l) - 1}{N(\mathbf{x}_l)} \hat{V}^*(\mathbf{x}_l) + \frac{1}{N(\mathbf{x}_l)} r. \quad (9)$$

After updating the leaf state node, we update the nodes in the path collected in selection stage in reversed order. Suppose we have a path

$$(\mathbf{x}_0, (\mathbf{x}_0, a_0), \ldots, \mathbf{x}_i, (\mathbf{x}_i, a_i), \mathbf{x}_{i+1}, \ldots, \mathbf{x}_l)$$

and the node values of $\mathbf{x}_{i+1}, \ldots, \mathbf{x}_l$ have been updated, the preceding nodes $\mathbf{x}_i$ and $(\mathbf{x}_i, a_i)$ are updated through

$$\hat{Q}^{N(\mathbf{x},a)}(\mathbf{x}_i, a_i) = R(\mathbf{x}_i, a) + \hat{V}^*(\mathbf{x}_{i+1}) \quad (10)$$

$$\bar{Q}(\mathbf{x}_i, a_i) \leftarrow \frac{N(\mathbf{x}_i, a_i) - 1}{N(\mathbf{x}_i, a_i)} \bar{Q}(\mathbf{x}_i, a_i) + \frac{1}{N(\mathbf{x}_i, a_i)} \hat{Q}^{N(\mathbf{x},a)}(\mathbf{x}_i, a_i) \quad (11)$$

$$\bar{V}(\mathbf{x}_i) \leftarrow \frac{N(\mathbf{x}_i) - 1}{N(\mathbf{x}_i)} \bar{V}^*(\mathbf{x}_i) + \frac{1}{N(\mathbf{x}_i)} \bar{Q}(\mathbf{x}_i, a_i) \quad (12)$$

$$\hat{V}(\mathbf{x}_i) \leftarrow (1 - \alpha_{N(\mathbf{x}_i)}) \bar{V}(\mathbf{x}_i) + \alpha_{N(\mathbf{x}_i)} \max_{a\in A_{\mathbf{x}_i}} \bar{Q}(\mathbf{x}_i, a) \quad (13)$$

where $\bar{V}(\cdot)$ is an intermediate variable that records the average value of the node through the root-to-leaf path, and $\alpha_{N(\mathbf{x}_i)} \in [0, 1]$ is a smoothing parameter. The updates are performed backward to the root node.

Details of the OCBA tree policy are shown in Algorithms 1 to 6.

There are a few points worth emphasizing in Algorithm 3. First, $\tilde{N}(\mathbf{x}, a_i)$ is the total number of samples for each action i after the allocation. Given present information, i.e., all samples state node $\mathbf{x}$, OCBA-MCTS assumes now a total number of $\sum_{a\in A} N(\mathbf{x}, a) + 1$ samples available. By solving (5) to (7), the new budget allocation $(\tilde{N}(\mathbf{x}, a_1), \tilde{N}(\mathbf{x}, a_2), \ldots, \tilde{N}(\mathbf{x}, a_{|A|}))$ that maximizes APCS is calculated. Afterward, one action based on (8) is selected to sample and move to the next stage. This "most-starving" implementation of the OCBA policy [19] as given in Algorithm 3 is fully sequential, as each iteration allocates only one sample to an action before the allocation decision is recomputed. It is also possible to allocate the sampling budget in a batch of size $\Delta > 1$. We use the "most-starving" scheme, because it has been shown to be more efficient than the

Algorithm 1: MCTS.

Input: Simulation budget (roll-out number) N, root state node $\mathbf{x}_0$
Output: $\hat{a}^*_{\mathbf{x}_0}$, $\hat{V}^*(\mathbf{x}_0)$
Set simulation counter $n \leftarrow 0$
while $n < N$ **do**
  $path \leftarrow selection(x_0)$
  $leaf \leftarrow path[end]$
  $r \leftarrow simulate(leaf)$
  $backpropagate(path, r)$
  $n \leftarrow n+1$
end
return action $\hat{a}^*_{\mathbf{x}_0} = \arg\max_{a \in A} \bar{Q}(\mathbf{x}_0, a)$

Algorithm 2: *Selection*($\mathbf{x}_0$).

Input: root state node $\mathbf{x}_0$
Sample a root-to-leaf path.
$path \leftarrow ( )$
$\mathbf{x} \leftarrow \mathbf{x}_0$
while *True* **do**
  Append state node $\mathbf{x}$ to $path$
  $N(\mathbf{x}) \leftarrow N(\mathbf{x})+1$
  if $\mathbf{x}$ *is a terminal node* **then**
    return $path$
  end
  if $\mathbf{x}$ *is expandable* **then**
    $\hat{a} \leftarrow expand(\mathbf{x})$
    $\mathbf{y} \leftarrow expand((\mathbf{x}, \hat{a}))$
    Append state-action node $(\mathbf{x}, \hat{a})$ and leaf state node $\mathbf{y}$ to $path$
    $N(\mathbf{x}, \hat{a}) \leftarrow N(\mathbf{x}, \hat{a})+1$
    $N(\mathbf{x}) \leftarrow N(\mathbf{x})+1$
    return $path$
  else
    $\hat{a} \leftarrow OCBAselection(\mathbf{x})$
    Append state-action node $(\mathbf{x}, \hat{a})$ to $path$
    $N(\mathbf{x}, \hat{a}) \leftarrow N(\mathbf{x}, \hat{a})+1$
    $\mathbf{x} \leftarrow expand((\mathbf{x}, \hat{a}))$
  end
end

Algorithm 3: OCBA Selection($\mathbf{x}$).

Input: state node $\mathbf{x}$
Identify $\hat{a}^*_{\mathbf{x}} = \arg\max_a \bar{Q}(\mathbf{x}, a)$;
$\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, a) \leftarrow \bar{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) - \bar{Q}(\mathbf{x}, a)$;
Compute new sampling allocation $(\tilde{N}(\mathbf{x}, a_1), \tilde{N}(\mathbf{x}, a_2), \ldots, \tilde{N}(\mathbf{x}, a_{|A|}))$ by solving (5) to (7);
$\hat{a} \leftarrow \arg\max_{a \in A}(\tilde{N}(\mathbf{x}, a) - N(\mathbf{x}, a))$;
return $\hat{a}$;

Algorithm 4: *Expand*($\mathbf{x}$ *or* $(\mathbf{x}, a)$).

Input: a state node $\mathbf{x}$ *or* a state-action node $(\mathbf{x}, a)$
Output: child node to be added to the tree
if *the input node is a state node* $\mathbf{x}$ **then**
  $S \leftarrow$ {feasible actions of state x that has been sampled less than n_0 times}
  $\hat{a} \leftarrow$ random choice of S
  Add $(\mathbf{x}, \hat{a})$ to the tree if it is unvisited
  return $\hat{a}$
else
  Sample node $(\mathbf{x}, a)$ at state node $\mathbf{x}$ and obtain the child state node $\mathbf{y} \sim P(\mathbf{x}, a)(\cdot)$
  Add $\mathbf{y}$ to the tree if it is unvisited
  return $\mathbf{y}$.
end

Algorithm 5: *Simulate*($\mathbf{x}$).

Input: state node $\mathbf{x}$
$r \leftarrow 0$
while *True* **do**
  if $\mathbf{x}$ *is not terminal* **then**
    find a random child state-action node $(\mathbf{x}, a)$ of $\mathbf{x}$
    $r \leftarrow r + R(\mathbf{x}, a)$
    sample a and obtain the child state node $\mathbf{y} \sim P(\mathbf{x}, a)(\cdot)$
    $\mathbf{x} \leftarrow \mathbf{y}$
  else
    return r
  end
end

Algorithm 6: *ackpropagate*($path, reward$)

Input: path to a leaf node $path$, simulated reward $reward$
for *node in reversed(path)* **do**
  Update node values through Equations (9) to (13).
end

batch sampling scheme [33]. However, the benefit of sampling in batches for MCTS is that in one iteration, multiple root-to-leaf paths can be examined, enabling parallelization of the algorithm.

Second, updating $\hat{V}(\mathbf{x}_i)$ involves two stages: Updating the value estimate along the path (12) and taking the maximum over the values of the child state-action nodes (canonical way to update). Then the two values are mixed through $\alpha_{N(\mathbf{x}_i)}$ to update $\hat{V}(\mathbf{x}_i)$, as prior research (e.g., [6], [34]) suggests mixing with $\alpha_{N(\mathbf{x}_i)} \to 1$ (i.e., asymptotically achieves Bellman update) ensures more stable updates.

Finally, although we present our algorithm in the context of solving an MDP, it can be applied to other tree structures such as MIN-MAX game trees or more general game trees, by setting the reward function and the max and min operators accordingly.

IV. Analysis of OCBA-MCTS

In this section, we first discuss how the OCBA tree policy in OCBA-MCTS balances exploration and exploitation mathematically. Then, we present several theoretical results regarding OCBA-MCTS. The proofs are given in the Appendix.

Equations (5) to (7) determine the new sampling budget allocation. First, (5) shows that the suboptimal state-action nodes should be sampled proportional to their variances and inversely proportional to the squared differences between their sample means and that of the optimal state-action node. This OCBA property represents a different type of tradeoff between exploration (sampling actions with high variances) and exploitation (sampling actions with higher sample means) compared to bandit-based algorithms.

In this part, we present results for OCBA-MCTS. The first proposition ensures the estimate of the value-to-go function converges to the true value. The second proposition establishes that OCBA-MCTS will select the correct action, i.e., the PCS converges to 1.

Proposition 1 (Asymptotic consistency): Assume the expected cumulative reward at state-action node $(\mathbf{x}, a)$ is a random variable with nonzero finite variance. Suppose the proposed OCBA-MCTS algorithm is run with a sampling budget N at root state node $\mathbf{x}_0$. Then at any subsequent nodes $\mathbf{x}$ and $(\mathbf{x}, a)$

$$\lim_{N\to\infty} \bar{Q}(\mathbf{x}, a) = \mathrm{E}[\hat{Q}(\mathbf{x}, a)] = Q(\mathbf{x}, a)$$

$$\lim_{N\to\infty} \hat{V}(\mathbf{x}) = V^*(\mathbf{x}),\ \forall\, \mathbf{x} \in \mathbf{X},\ (\mathbf{x}, a) \in \mathbf{X} \times A_{\mathbf{x}}.$$

Proposition 2 (Asymptotic correctness): Assume the expected cumulative reward at state-action node $(\mathbf{x}, a)$ is a normal random variable with mean $\mu(\mathbf{x}, a)$ and variance $\sigma^2(\mathbf{x}, a) < \infty$, i.e., $\hat{Q}(\mathbf{x}, a) \sim N(\mu(\mathbf{x}, a), \sigma^2(\mathbf{x}, a))$ for $0 \le i < H$ and that $\mu(\mathbf{x}, a)$ is normally distributed with unknown mean and known variance. Then the PCS converges to 1 for any state node $\mathbf{x} \in \mathbf{X}$, i.e.,

$$P\left(\bigcap_{a\in A_{\mathbf{x}}, a\neq \hat{a}^*_{\mathbf{x}}} \left(\lim_{N\to\infty} \tilde{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) - \lim_{N\to\infty} \tilde{Q}(\mathbf{x}, a)\right) \ge 0\right) = 1$$

where $\hat{a}^*_{\mathbf{x}} = \arg\max_{a\in A_{\mathbf{x}}} \bar{Q}(\mathbf{x}, a)$.

The allocation rule obtained by solving (5) and (6) can be derived using a similar analysis as that in [19], which shows that at each point of the algorithm when a decision needs to be made, the action that maximizes the APCS (asymptotically, i.e., as $N \to \infty$) will be selected and sampled. Therefore, the OCBA tree policy gradually maximizes the overall APCS at the root, which is a lower bound for PCS.

A lower bound on PCS for the algorithm in the known variance setting can be derived using (4) by substituting the OCBA allocation into the APCS expression (right-hand side of the inequality) and incorporating the normal distribution assumptions on the Q functions.

Proposition 3 (Lower bound on the probability of correct selection): Under the same assumptions of Theorem 2, the PCS at each stage and state is lower bounded by

$$PCS \ge 1 - \sum_{a\in A_{\mathbf{x}}, a\neq \hat{a}^*_{\mathbf{x}}} \Phi\left(-\frac{\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, a)\ \sqrt{N(\mathbf{x}, \hat{a}^*_{\mathbf{x}})}}{\sqrt{\sigma^2(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) + \sigma(\mathbf{x}, \hat{a}^*_{\mathbf{x}})\sigma^2(\mathbf{x}, a)}}\sqrt{\sum_{\bar{a}\in A_{\mathbf{x}}, \bar{a}=\hat{a}^*_{\mathbf{x}}} \frac{r_{\mathbf{x}}(\bar{a}, a)}{\sigma(\mathbf{x}, \bar{a})}}\right)$$

where $\Phi(\cdot)$ is the cdf of standard normal distribution and

$$r_{\mathbf{x}}(\bar{a}, a) = \frac{\sigma(\mathbf{x}, \bar{a})\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, a)^2}{\sigma(\mathbf{x}, a)\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, \bar{a})}.$$

Similar to solving (5) to (7) in the selection stage, the true variance may not be known in practice, but can be approximated with the sample variance, which provides an estimate for the PCS lower bound.

V. Numerical Examples

In this section, we evaluate our proposed OCBA-MCTS on two tree search problems against the well-known UCT [5]. The effectiveness is measured by PCS, which is estimated by the fraction of times the algorithm chooses the true optimal action. We first evaluate our algorithm on an inventory control problem with random nonnormal rewards. Then we apply our algorithm to the game of tic-tac-toe. The code is available at [35].

For convenience, we restate the UCT tree policy here. At a state node $\mathbf{x}$, the UCT policy will select the child state-action node with the highest upper confidence bound, i.e.,

$$\hat{a} = \arg\max_{a\in A_{\mathbf{x}}} \left\{ \bar{Q}(\mathbf{x}, a) + w_e \sqrt{\frac{2\log \sum_{a\in A_{\mathbf{x}}} N(\mathbf{x}, a)}{N(\mathbf{x}, a)}} \right\} \tag{14}$$

where w_e is the "exploration weight." The original UCT algorithm assumes the value function in each stage is bounded in [0,1] because it sets $w_e = 1$, whereas the support is unknown in many practical problems. Therefore, in general, w_e needs to be tuned to encourage exploration.

For all experiments, we set the smoothing parameter in (13) in the backpropagation phase to $\alpha_{N(\mathbf{x})} = 1 - \frac{1}{5N(\mathbf{x})}$. Since initial estimates of sample variance can be less accurate with small n_0, we add an initial variance $\sigma_0^2 > 0$, which decays as the number of visits grows, to the sample variance to encourage exploration. Specifically, we set

$$\hat{\sigma}^2(\mathbf{x}, a) = \frac{1}{N(\mathbf{x}, a)} \sum_{t=1}^{N} \left(\hat{Q}^t(\mathbf{x}, a) - \bar{Q}(\mathbf{x}, a)\right)^2 + \sigma_0^2/N(\mathbf{x}, a)$$

where the first term is the sample variance, and second term vanishes as $N(\mathbf{x}, a)$ grows.

A. Inventory Control Problem

We now evaluate the performance of OCBA-MCTS using the inventory control problem in [3]. The objective is to find the initial order quantity that minimizes the total cost over a finite horizon. At decision period i, we denote by D_i the random demand in period i, $\mathbf{x}_i = (x_i, i)$ the state node, where x_i is the inventory level at the end of period i (which is also the inventory at the beginning of period $i+1$), $(\mathbf{x}_i, a_i)$ the corresponding child state-action node with a_i being the order amount in period i, p the per period per unit demand lost penalty cost, h the per period per unit inventory holding cost, K the fixed (setup) cost per order, M the maximum inventory level (storage capacity),

and H the number of simulation stages. We set $M = 20$, initial state $x_0 = 5$, $h = 1$, $H = 3$, $D_i \sim DU(0, 9)$ (discrete uniform, inclusive), and consider two different settings for p and K as follows.

1) Experiment 1: $p = 10$ and $K = 0$.
2) Experiment 2: $p = 1$ and $K = 5$.

The reward function, which in this case is the negative of the inventory cost in stage i, is defined by

$$R(\mathbf{x}_i, a_i) = -(h \max\{0, x_i + a_i - D_i\} + p \max\{0, D_i - x_i - a_i\} + K1_{\{a_i>0\}})$$

where 1 is the indicator function, and the state transition follows:

$$x_{i+1} = \max(0, x_i + a_i - D_i)$$

where

$$a_i \in A_{x_i} = \{a | x_i + a \le M\}.$$

For UCT, to accommodate the reward support not being [0, 1], we adjust the exploration weight when updating a state-action node, i.e., set w_e initially to 1, then in the backpropagation step, update w_e by

$$w_e = \max(w_e, |\hat{Q}^{N(\mathbf{x},a)}(\mathbf{x}, a)|)$$

where $\hat{Q}^{N(\mathbf{x},a)}(\mathbf{x}, a)$ is obtained in (10). The initial variance σ_0^2 is set to 100. For both OCBA-MCTS and UCT, we set the number of expansions (n_0) to 4 for depth 1 state-action nodes (i.e., the child nodes of the root) and to 2 for all other state action nodes in Experiment 1, and set n_0 to 2 for all nodes in Experiment 2. The different values of n_0 are due to the variance decreasing with the depth of a node, and Experiment 2 is a relatively easier problem.

For both experiment settings, each algorithm is repeated 2000 times at each simulation budget level N to estimate PCS. Since Experiment 1 is a much harder problem compared to Experiment 2, more rollouts (budget) are required. Therefore, N ranges from 14 000 to 24 000 and from 50 to 200 for Experiments 1 and 2, respectively. The estimated PCS curves for both experiments are illustrated in Fig. 1, where the standard error ($= \sqrt{\text{PCS}(1 - \text{PCS})/N}$) is small and thus omitted for clarity. OCBA-MCTS achieves better PCS for both experiment setups. For Experiment 1 (optimal action $a_0^* = 4$), as shown in Fig. 1(a), OCBA-MCTS achieves a 5% higher PCS (absolute) compared to UCT. For Experiment 2 (optimal action $a_0^* = 0$), we see a 15% performance gap between UCT and OCBA-MCTS when the number of samples is less than 80, after which UCT gradually closes the gap as OCBA already reaches PCS> 95%.

It is also beneficial to compare the distribution of budget allocation of OCBA-MCTS and UCT to show the exploration–exploitation balance of OCBA-MCTS. For convenience, we label the child actions of the root node from 0 to 15, where action i denotes ordering i units. Figs. 2 and 3 illustrate the average number of visits, average estimated value function, and average estimated standard deviation of all child state-action nodes of the root node over 1000 repeated runs with 24 000 and 170 rollouts for Experiment 1 and Experiment 2, respectively. Note that although the estimated standard deviation does not play a role in determining the allocation for UCT, we still plot it for reference. Both figures show that the number of visits to children nodes is, to some extent, proportional to the estimated value of the node for UCT. On the other hand, OCBA-MCTS puts more effort on the estimated optimal and second optimal actions (actions 4 and 3 for Experiment 1 and actions 0 and 1 for Experiment 2, respectively), as illustrated in Figs. 2(b) and 3(b).

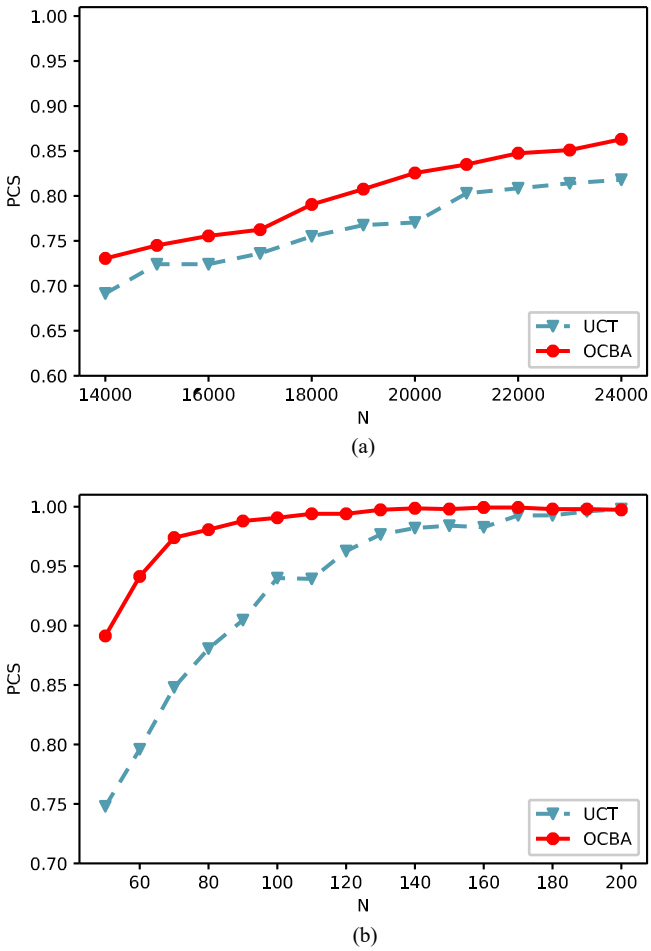

Fig. 1. Estimated PCS as a function of sampling budget achieved by UCT-MCTS and OCBA-MCTS for inventory control problem, averaged over 2000 runs. (a) Experiment 1: $p = 10$, $K = 0$ and (b) Experiment 2: $p = 1$, $K = 5$.

In Experiment 1, where there are two competing actions with similar estimated values (actions 3 and 4, with action 4 being the optimal), OCBA-MCTS will spend most of its sampling budget on those two potential actions and put much lesser effort on clearly inferior actions, such as actions 6 to 14, compared to UCT. This strategy makes more sense when the objective is to identify the best action, and thus, is more suitable for MCTS problems, as the ultimate goal is to make a decision. It is also interesting to note that OCBA-MCTS actually allocates slightly more visits to the competing suboptimal action than the optimal one (mean 10 781 and 10 350 for actions 3 and 4, respectively), which will not happen in bandit-based policies, as their goal is to minimize regret, and thus, will put more effort on exploiting the estimated optimal action. In Experiment 2, where the optimum is

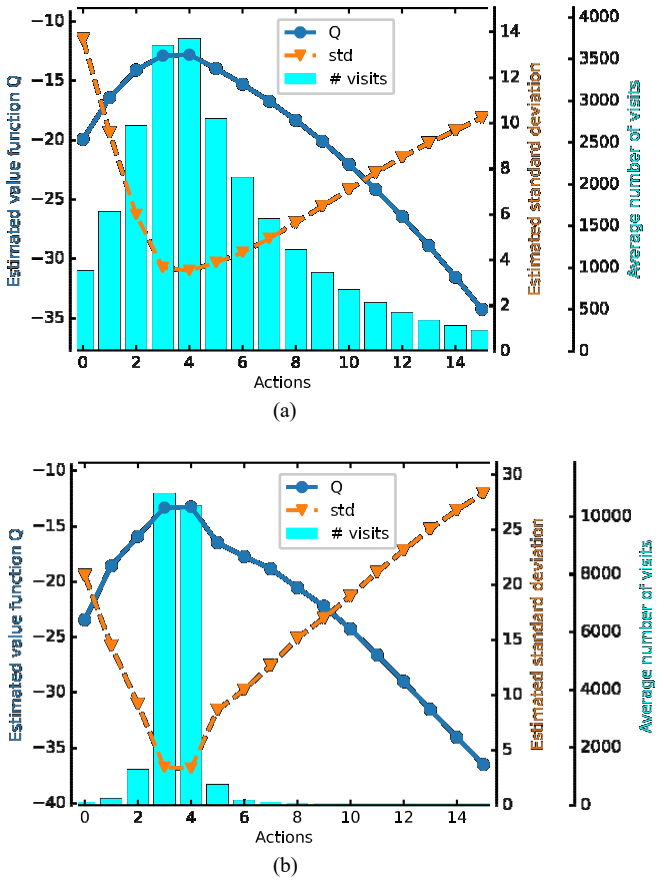

Fig. 2. Sampling distribution for Experiment 1 with $N = 24\,000$, averaged over 2000 runs (action 4 optimal, action 3 near optimal). (a) UCT and (b) OCBA.

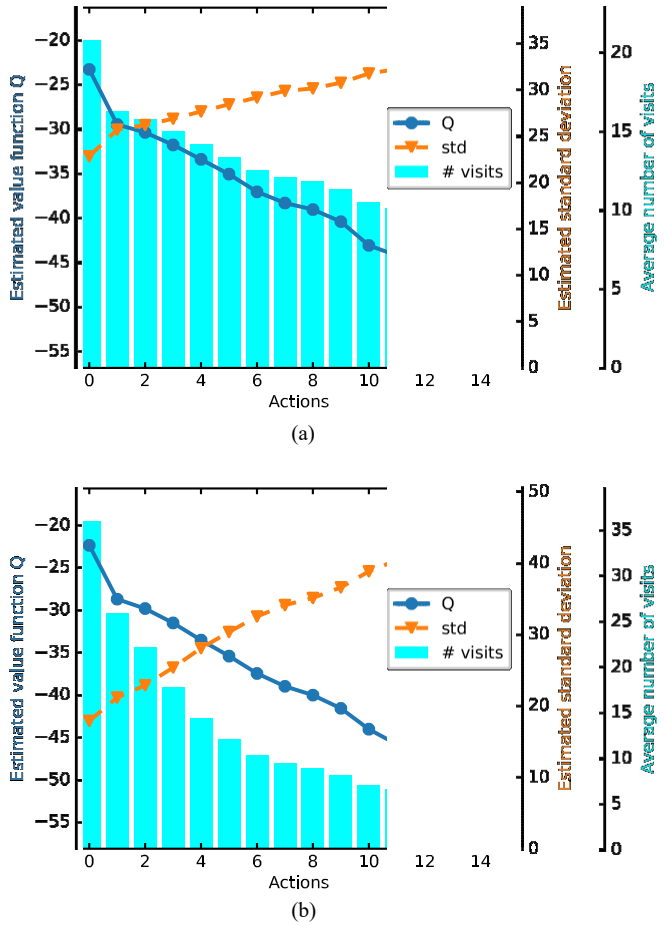

Fig. 3. Sampling distribution for Experiment 2 with $N = 200$, averaged over 2000 runs (action 0 optimal). (a) UCT and (b) OCBA.

slightly easier to find, although OCBA-MCTS allocates a larger fraction of samples to suboptimal actions compared to that in Experiment 1, most of the samples are still allocated to the top 2 actions as shown in Fig. 3(b), whereas UCT performs similar to that in Experiment 1.

B. Tic-Tac-Toe

In this section, we apply OCBA-MCTS and UCT to the game of tic-tac-toe to identify the optimal move. Tic-tac-toe is a game for two players who take turns marking "X" (Player 1) and "O" (Player 2) on a 3 × 3 board. The objective for Player 1 (Player 2) is to mark 3 consecutive "X" ("O") in a row, column, or diagonal. If both players act optimally, the game will always end in a draw.

For ease of presentation, we number the spaces sequentially as shown in Fig. 4. We use OCBA-MCTS and UCT to represent Player 2 (who marks "O" on the board). We consider two board setups: Player 1 already marked "X" space 0 (setup 1) and space 4 (setup 2). The root nodes and optimal actions for Player 2 are illustrated in Figs. 5 and 6, respectively. Since in setup 2, the optimal move for Player 2 will be marking any of the corner spaces [shown in Fig. 6(b)] due to symmetry, we consider it a correct selection if the algorithm returns any one of the optimal moves. For both setups, taking any of the suboptimal actions will end up in losing the game if Player 1 plays optimally. In

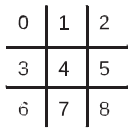

Fig. 4. Action layout.

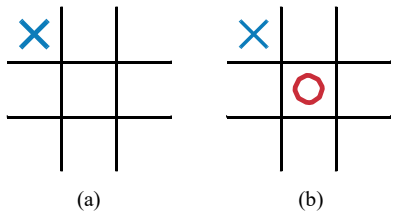

Fig. 5. Tic-tac-toe board setup 1. (a) Root node and (b) Optimal.

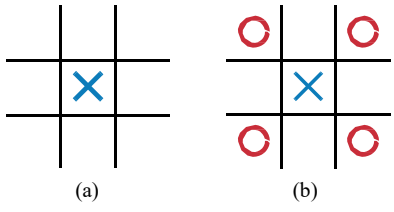

Fig. 6. Tic-tac-toe board setup 2. (a) Root node and (b) Optimal (any one of the corner spaces).

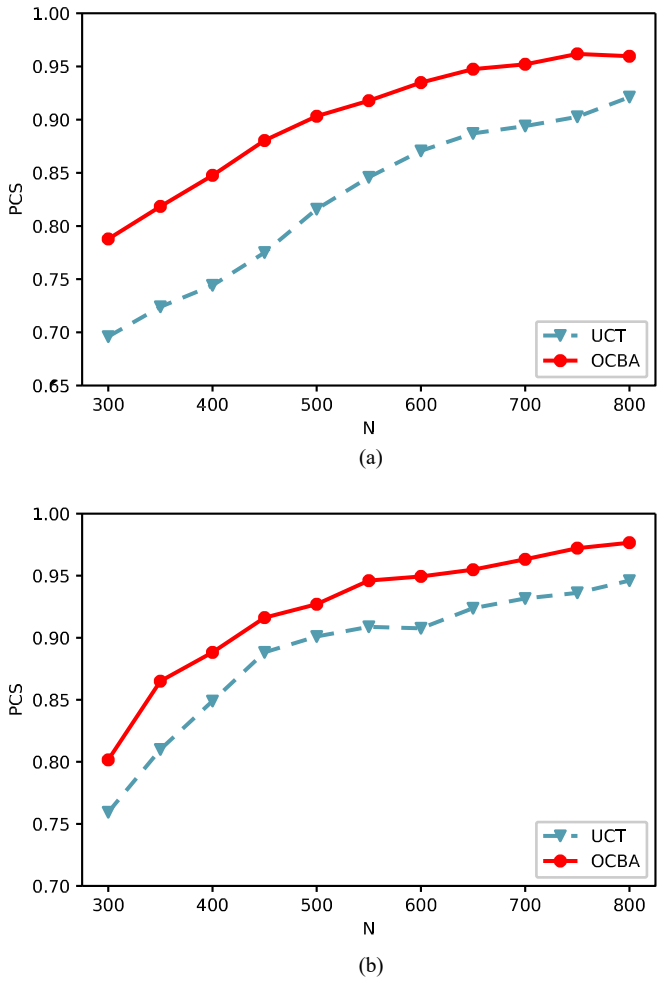

Fig. 7. Estimated PCS as a function of sampling budget achieved by UCT-MCTS and OCBA-MCTS for tic-tac-toe under setup 1, averaged over 5000 runs. (a) Experiment 3: Player 1 plays randomly and (b) Experiment 4: Player 1 plays UCT.

this game, Player 2 (MCTS algorithm) makes decisions at even stages $(2, 4, \ldots)$ and Player 1 makes decisions at odd stages $(3, 5, \ldots)$. The state transitioning is deterministic and Player 1's move is modeled using a randomized policy. We consider two different policies for Player 1 under both setups.

1) Experiment 3: Under setup 1, Player 1 plays randomly, i.e., with equal probability to mark any feasible space.
2) Experiment 4: Under setup 1, Player 1 plays UCT.
3) Experiment 5: Under setup 2, Player 1 plays randomly, i.e., with equal probability to mark any feasible space.
4) Experiment 6: Under setup 2, Player 1 plays UCT.

We compare the performance of OCBA-MCTS and UCT on Player 2 in all four experiments. At state node $\mathbf{x}$, the reward function for taking action a is defined according to the following rules: Immediately after taking the action, if Player 2 wins the game, $R(\mathbf{x}, a) = 1$, if it leads to a draw, $R(\mathbf{x}, a) = 0.5$; otherwise (Player 2 loses or in any nonterminating state), $R(\mathbf{x}, a) = 0$. n_0 is set to 2 across all nodes for both UCT and OCBA-MCTS. Since the value function for all state-action nodes is now bounded in $[0, 1]$, we set $w_e = 1$ throughout the entire experiment for UCT policies. The initial variance σ_0^2 is set to

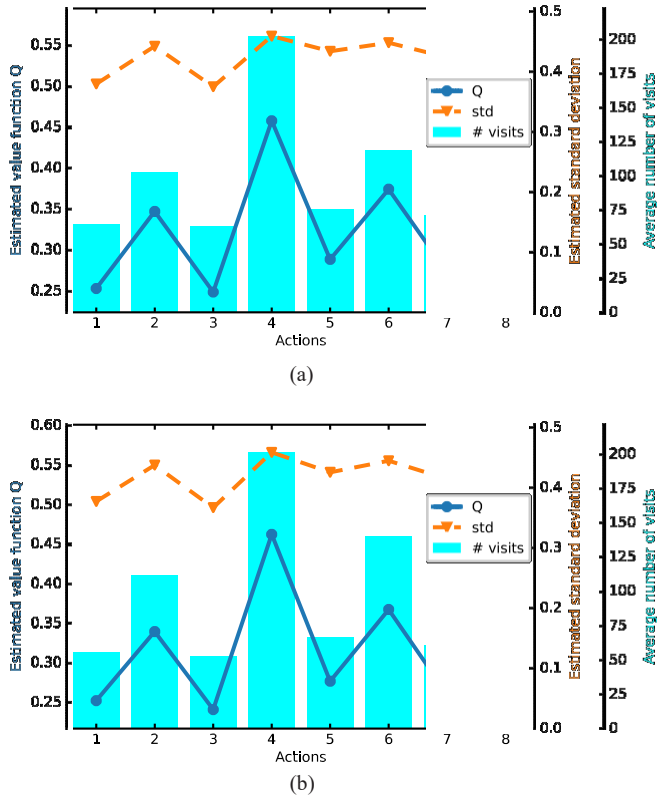

Fig. 8. Sampling distributions for Experiment 3, averaged over 5000 runs (action 4 optimal). (a) UCT and (b) OCBA.

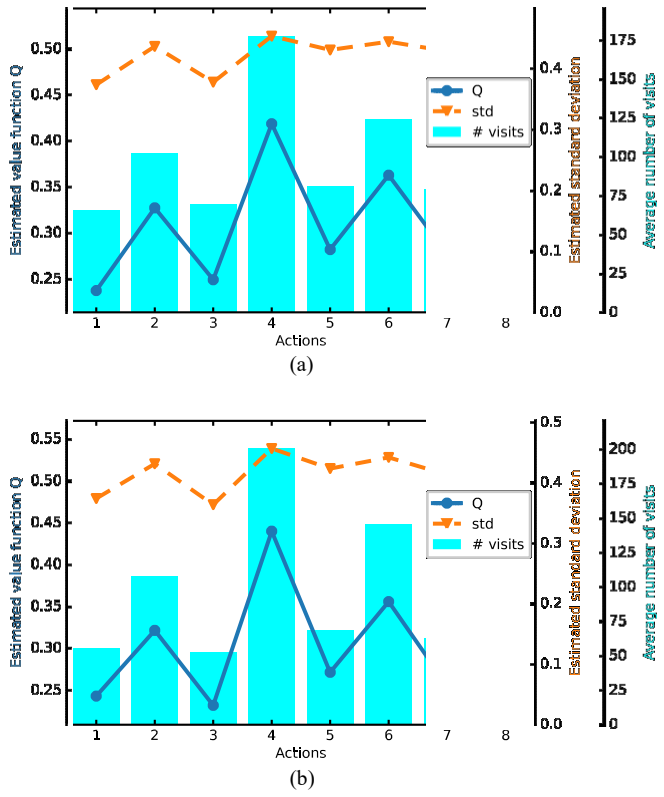

Fig. 9. Sampling distributions for Experiment 4, averaged over 5000 runs (action 4 optimal). (a) UCT and (b) OCBA.

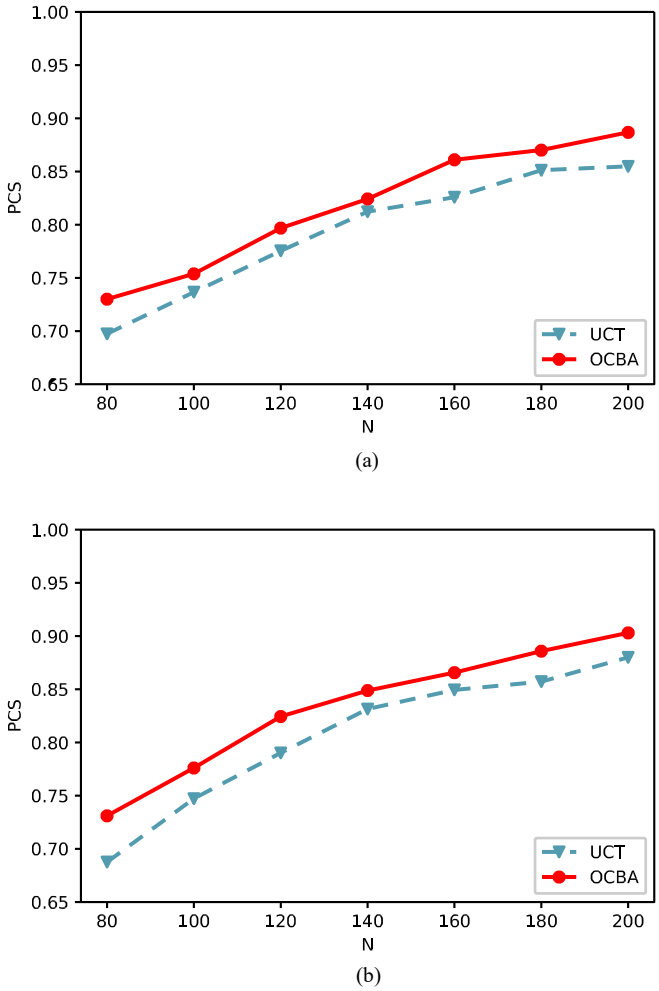

Fig. 10. Estimated PCS as a function of sampling budget achieved by UCT-MCTS and OCBA-MCTS for tic-tac-toe under setup 2, averaged over 5000 runs. (a) Experiment 5: Player 1 plays randomly and (b) Experiment 6: Player 1 plays UCT.

10. For Experiment 4, where Player 1 plays UCT, its goal is to *minimize* the reward, therefore, Player 1 will select the action that minimizes t[illegible]wer confidence bound. i.e.,

$$\hat{a} = \arg\min_{a \in A_{\mathbf{x}}} \left\{ \bar{} (\mathbf{x}, a) - w_e \sqrt{\frac{2 \log \sum_{a \in A_{\mathbf{x}}} N(\mathbf{x}, a)}{N(\mathbf{x}, a)}} \right\}$$

Similar to the previous section, we plot the PCS of the two algorithms as a function of the number of rollouts, which ranges from 300 to 800 for Experiments 3 and 4 and the PCS is estimated over 5000 independent experiments at each rollout level. The results are shown in Fig. 7, which indicates that the proposed OCBA-MCTS produces a more accurate estimate of the optimal action compared to UCT. In setup 1 (i.e., experiments 3 and 4), both experiments show that OCBA-MCTS is better at finding the optimal move when the sampling budget is relatively low. The performance of UCT and OCBA-MCTS becomes comparable when more samples become available. We also note that there is a greater performance gap between UCT and OCBA-MCTS in Experiment 3 than in Experiment 4: In Experiment 3, OCBA-MCTS achieves 15% better PCS, whereas in Experiment

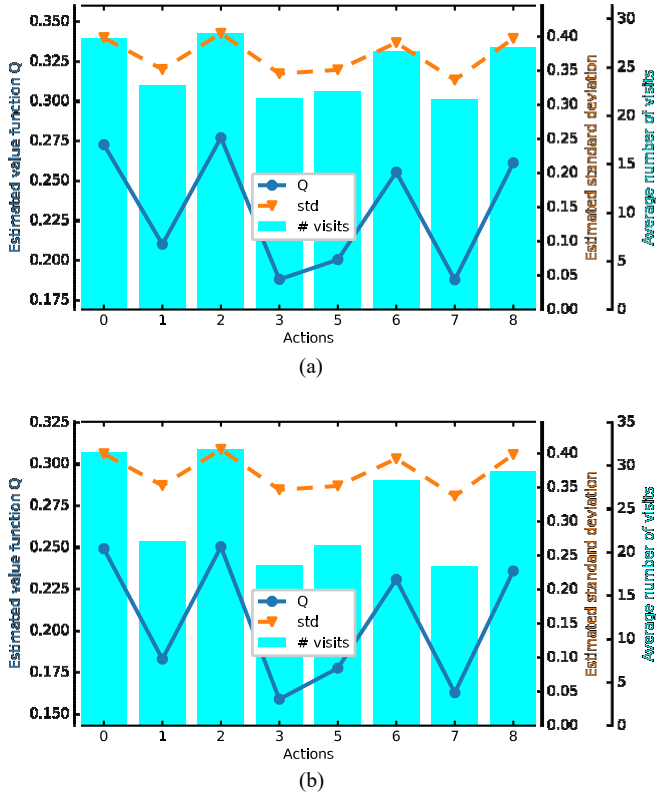

Fig. 11. Sampling distributions for Experiment 5, averaged over 5000 runs (actions 0, 2, 6, and 8 optimal). (a) UCT and (b) OCBA.

4, the difference is around 5% when $N < 500$ and soon catches up as N increases. This is expected, as it becomes easier to determine the optimal action when the opponent applies an AI algorithm (i.e., Player 1 has a better chance to take its optimal action). In this case, space 4 becomes a clear optimum and therefore Player 2's UCT algorithm tends to exploit it more, which leads to better performance.

The sampling distributions for OCBA-MCTS and UCT with $N = 700$ for both experiments are shown in Figs. 8 and 9. In this game, since a relatively clear optimum is available, OCBA-MCTS and UCT behaved differently compared to that in the inventory control problem. As shown in Figs. 8(a) and 9(a), UCT spends most of the sampling budget exploiting this action, whereas OCBA will still try to explore other promising suboptimal actions (e.g., action 2, 6, and 8) and pay less attention to inferior actions (e.g., 1, 3, and 5) due to its tendency to better balance exploration and exploitation.

The PCS estimation for setup 2 is shown in Fig. 10, where the budget ranges from 80 to 200. Since it is an easier setting (Player 2 has a 50% chance of marking an optimal space even if choosing randomly), the difference between OCBA-MCTS and UCT-MCTS is not as significant as that in Experiments 3 and 4, but OCBA still consistently performs better than UCT, especially when the budget is low. The sampling distribution for Experiments 5 and 6 are shown in Figs. 11 and 12, respectively. Similar to previous experiments, the OCBA-MCTS spent more effort on those equally optimal actions compared to UCT-MCTS,

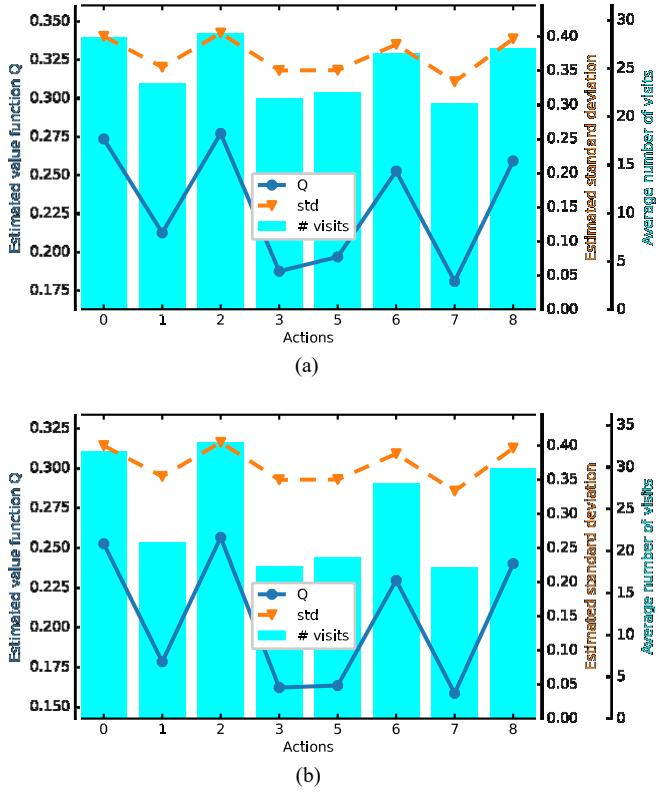

Fig. 12. Sampling distributions for Experiment 6, averaged over 5000 runs (actions 0, 2, 6, and 8 optimal). (a) UCT and (b) OCBA.

which, again, shows the unique property of OCBA-MCTS in finding an optimal action instead of minimizing regret.

In summary, the proposed OCBA-MCTS outperforms UCT in both experiments in finding the optimal action at the root. Since the objective of the proposed OCBA tree policy is to maximize PCS, it leads to different budget allocation and better PCS.

VI. Conclusion

In this article, we present a new OCBA tree policy for MCTS. Unlike bandit-based tree policies (e.g., UCT), the new policy maximizes PCS at the root node, and in doing so, balances the exploration and exploitation tradeoff differently. Furthermore, the new OCBA tree policy relaxes the assumption of known bounded support on the reward distribution, and thus, makes MCTS more generally applicable.

For future research, we intend to explore the use of a batch sampling scheme in Algorithm 2, which allocates a batch of $\Delta > 1$ samples at each node. With batch sampling and updating, we may exploit the power of parallel computing to more quickly identify the optimal action. On the other hand, this sequential algorithm could also benefit from a one-step look ahead policy, as suggested in [32]. Furthermore, establishing that our algorithm is $E - \delta$-correct and conducting time-complexity analysis are also important topics for future research. Another important future research direction is incorporating the uncertainty in sample variance estimation, as it has been shown that ignoring the estimation error could potentially lead to performance degradation [36]. Finally, it is worthwhile to investigate incorporating other R&S (or BAI) approaches with MCTS.

Algorithm 7: One-stage OCBA.

Input: Total sampling budget T, initial sample size n_0
Output: Index of optimal action $\hat{b}$
Sample each of the k alternatives n_0 times;
Set counter $l_i \leftarrow n_0 \ \forall i = 1, 2, \ldots, k$;
$l \leftarrow kn_0$;
Calculate $\bar{J}_i$ and $\hat{\sigma}_i^2$, $\forall i = 1, 2, \ldots, k$;
while $l <= T$ **do**
  Compute new budget allocation $(\tilde{l}_1, \tilde{l}_2, \ldots, \tilde{l}_k)$ by solving eq. (15)-(17) with budget $l+1$;
  Sample $\hat{i} = \arg\max_{1 \le i \le k} (\tilde{l}_i - l_i)$;
  Update $\bar{J}_{\hat{i}}$ (and $\hat{\sigma}_{\hat{i}}^2$ if sample variance is used);
  $l_{\hat{i}} \leftarrow l_{\hat{i}} + 1$;
  $l \leftarrow l + 1$;
end
return $\hat{b} = \arg\max_{1 \le i \le k} \bar{J}_i$;

Appendix
Convergence Analysis

Proof of Proposition 1: To prove that our algorithm correctly selects the optimal action as the sampling budget goes to infinity, we first prove that at each stage, the PCS converges to 1. The process of our algorithm at each single stage is OCBA adapted from [19]. OCBA tries to identify the alternative with the highest mean from a set of alternatives with means J_i and known variances σ_i^2, $i = 1, 2, \ldots, k$ by efficiently allocating samples that maximizes APCS. Here we present OCBA again in Algorithm 7 for convenience. The budget allocation process is similar to (5) to (7). First define

$$\bar{J}_i := \frac{1}{l_i} \sum_{m=1}^{l_i} \hat{J}_i^m$$

$$b := \arg\max_i \bar{J}_i$$

$$\delta(b, i) := \bar{J}_b - \bar{J}_i, \ \forall i = b$$

where l_i is the number of samples for alternative i, $\hat{J}_i^m$ is the mth sample of J_i for $1 \le i \le k$, $1 \le m \le l_i$. The new allocations $(\tilde{l}_1, \tilde{l}_2, \ldots, \tilde{l}_k)$ with budget $T > \sum_i l_i$ can be obtained by solving the set of equations

$$\frac{\tilde{l}_i}{\tilde{l}_j} = \left(\frac{\sigma_i / \delta(b, i)}{\sigma_j / \delta(b, j)} \right)^2, \ \forall i = j = b \tag{15}$$

$$\tilde{l}_b = \sigma_b \sum_{i=1, i=b}^{k} \frac{\tilde{l}_i^2}{(\sigma_i)^2} \tag{16}$$

$$\sum_{i=1}^{k} \tilde{l}_i = T \tag{17}$$

where σ_i is the standard deviation of the ith reward distribution. As in Remark 2, σ_i is assumed to be known, but in practice can be unknown and approximated by sample standard deviation

$$\hat{\sigma}_i = \sqrt{\frac{1}{l_i - 1} \sum_{m=1}^{l_i} (\hat{J}_i^m - \bar{J}_i)^2}.$$

Since $\sum_{i=1}^{k} I_i = T$, when $T \to \infty$, at least one of the actions will be sampled infinitely many times, i.e., there exists an index i, such that $I_i \to \infty$. Then there are two possible cases: $i = b$ and $i = b$.

Case 1: $i = b$

According to (15)

$$I_j = \left(\frac{\sigma_j/\delta(b,j)}{\sigma_i/\delta(b,i)}\right)^2 I_i, \ \forall j = i, j = b.$$

Since σ_i and $\delta(b,i)$ are positive and finite for all i, $I_j \to \infty$, $\forall j = b$.

Therefore, by (16), $I_b \to \infty$. Thus, $I_i \to \infty$ for all $i = 1, 2, \ldots, k$.

Case 2: $i = b$ According to (16)

$$I_b = \sigma_b \sqrt{\sum_{i=1, i=b} \frac{I_i^2}{(\sigma_i)^2}} \to \infty.$$

Thus there exists an index $i = b$, such that $I_i \to \infty$. By a similar argument in Case 1, we can conclude that $I_i \to \infty$ for all $i = 1, 2, \ldots, k$.

Since $I_i \to \infty$, $\forall i$, by applying the strong law of large numbers, we have

$$\bar{J}_i \to \mathrm{E}[J_i] \text{ w.p. 1 as } T \to \infty, \ \forall i = 1, 2, \ldots, k. \tag{18}$$

With the convergence analysis from single-stage OCBA, Theorem 1 can be proved by induction.

First observe that since $N \to \infty$, each path is explored infinitely many times. Thus the number of samples in each stage also goes to infinity as $N \to \infty$.

panded. If the current state node $\mathbf{x}$ is at stage $H - 1$ (i.e., it will transit into a terminal node in the next transition), running Algorithm 3 reduces to a single-stage problem, which is the same as OCBA in Algorithm 7. $\hat{Q}(\mathbf{x}, a)$ can be viewed as a set of alternatives for $a \in A$. From Corollary 18, it is straightforward that

$$\lim_{N\to\infty} \bar{Q}(\mathbf{x}, a) = Q(\mathbf{x}, a).$$

Therefore, since the reward function is bounded

$$\begin{aligned}\lim_{N\to\infty} \hat{V}(\mathbf{x}) &= \lim_{N\to\infty} \max_{a\in A_{\mathbf{x}}} \bar{Q}(\mathbf{x}, a) \\ &= \max_{a\in A_{\mathbf{x}}} \lim_{N\to\infty} \bar{Q}(\mathbf{x}, a) \\ &= V^*(\mathbf{x}).\end{aligned}$$

Now suppose that the statement is true for all child state nodes $\mathbf{y}$ of a state $\mathbf{x}$, i.e., $\hat{V}(\mathbf{y}) \to V^*(\mathbf{y})$ and $\mathbf{y}$ could be achieved from $\mathbf{x}$. Then for $\mathbf{x}$, the algorithm also reduces to OCBA. Thus from Corollary 18 again

$$\begin{aligned}\lim_{N\to\infty} \bar{Q}(\mathbf{x}, a) &= \lim_{N(\mathbf{x},a)\to\infty} \bar{Q}(\mathbf{x}, a) \\ &= \mathrm{E}[R(\mathbf{x}, a)] + \mathrm{E}_{P(\mathbf{x},a)}[V^*(\mathbf{y})] \\ &= Q(\mathbf{x}, a)\end{aligned}$$

for all child state-action pair $(\mathbf{x}, a)$. It follows that:

$$\lim_{N\to\infty} \hat{V}(\mathbf{x}) \to V^*(\mathbf{x}).$$

□

Proof of Proposition 2: We start with the single-stage OCBA, which assumes that $\bar{J}_i$ is also normally distributed. Define the single-stage PCS

$$\mathrm{PCS}_{\mathrm{single}} = P\left\{\bigcap_{i=1, i=b} (\tilde{J}_b - \tilde{J}_i) \geq 0\right\}$$

where $\tilde{J}_i$ is the posterior distribution of J_i given I_i samples $\forall i = 1, 2, \ldots, k$. Then, we first show that $\mathrm{PCS}_{\mathrm{single}} \to 1$ as $T \to \infty$.

Similar to (4), $\mathrm{PCS}_{\mathrm{single}}$ can be lower bounded by $\mathrm{APCS}_{\mathrm{single}}$, i.e., by the Bonferroni inequality

$$\begin{aligned}\mathrm{PCS}_{\mathrm{single}} &= P\left\{\bigcap_{i=1, i=b} (\tilde{J}_b - \tilde{J}_i) \geq 0\right\} \\ &\geq 1 - \sum_{i=1, i=b} P\{\tilde{J}_b - \tilde{J}_i \leq 0\} \\ &= \mathrm{APCS}_{\mathrm{single}}.\end{aligned}$$

Thus, to prove that $\mathrm{PCS}_{\mathrm{single}} \to 1$, it suffices to prove $\mathrm{APCS}_{\mathrm{single}} \to 1$, i.e.,

$$\sum_{i=1, i=b} P\{(\tilde{J}_b - \tilde{J}_i) \leq 0\} \to 0 \text{ as } T \to \infty.$$

Based on the normality assumption, the posterior distribution is $\tilde{J}_i \sim N(\bar{J}_i, \sigma_i^2/I_i)$. Thus, $\tilde{J}_b - \tilde{J}_i \sim N(\bar{J}_b - \bar{J}_i, \sigma_b^2/I_b + \sigma_i^2/I_i)$. Therefore

$$P\{(\tilde{J}_b - \tilde{J}_i) \leq 0\} = \Phi\left(-\frac{\bar{J}_b - \bar{J}_i}{\sqrt{\sigma_b^2/I_b + \sigma_i^2/I_i}}\right) \tag{19}$$

where Φ is the cdf of the standard normal distribution.

Since $I_i \to \infty$ for all $i = 1, 2, \ldots, k$ and $\bar{J}_b$ is defined to be the maximum of all $\bar{J}_i$, i.e., $\bar{J}_b - \bar{J}_i \geq 0$ for all $i = b$, (19) becomes

$$\sum_{i=1, i=b} P\{(\tilde{J}_b - \tilde{J}_i) \leq 0\} = \sum_{i=1, i=b} \Phi\left(-\frac{\bar{J}_b - \bar{J}_i}{\sqrt{\sigma_b^2/I_b + \sigma_i^2/I_i}}\right) \to 0$$

as desired.

Since we assume $\hat{Q}(\mathbf{x}, a)$ is normally distributed with known variance, the posterior distribution of $\mu(\mathbf{x}, a)$ given observations $\hat{Q}(\mathbf{x}, a)$, i.e., $\tilde{Q}(\mathbf{x}, a)$, is also a normal random variable. Then, using the same analysis as that in proving $\mathrm{PCS}_{\mathrm{single}} \to 1$, we have

$$P\left\{\bigcap_{a\in A_{\mathbf{x}}, a=\hat{a}^*_{\mathbf{x}}} \left(\lim_{N\to\infty} \tilde{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) - \lim_{N\to\infty} \tilde{Q}(\mathbf{x}, a)\right) \geq 0\right\} = 1$$

$$\forall i = 1, \ldots, H, \ \mathbf{x} \in \mathbf{X}, \ a \in A.$$

□

Proof of Proposition 3: When the number of samples at node $\mathbf{x}$ is large, we assume that $N(\mathbf{x}, a)$ satisfies (5) to (6).

From (5), we have

$$N(\mathbf{x}, \tilde{a}) = \left(\frac{\sigma(\mathbf{x}, \tilde{a})\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, a)}{\sigma(\mathbf{x}, a)\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, \tilde{a})}\right)^2 N(\mathbf{x}, a)$$
$$\forall \tilde{a}, a = \hat{a}^*_{\mathbf{x}}. \tag{20}$$

In this way, we can express the budget allocation to any suboptimal action $\tilde{a}$ as the product of the budget allocation to a particular suboptimal action a and the factor

$$r_{\mathbf{x}}(\tilde{a}, a) = \left(\frac{\sigma(\mathbf{x}, \tilde{a})\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, a)}{\sigma(\mathbf{x}, a)\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, \tilde{a})}\right)^2.$$

From (6)

$$N(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) = \sigma(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) \sum_{\tilde{a}\in A_{\mathbf{x}}, \tilde{a}=\hat{a}^*_{\mathbf{x}}} \frac{N(\mathbf{x}, \tilde{a})}{\sigma^2(\mathbf{x}, \tilde{a})}.$$

Substituting $N(\mathbf{x}, \tilde{a})$ from (20) yields

$$N(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) = N(\mathbf{x}, a)\sigma(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) \sum_{\tilde{a}\in A_{\mathbf{x}}, \tilde{a}=\hat{a}^*_{\mathbf{x}}} \frac{(r_{\mathbf{x}}(\tilde{a}, a))^2}{\sigma^2(\mathbf{x}, \tilde{a})}$$

i.e.,

$$N(\mathbf{x}, a) = \frac{N(\mathbf{x}, \hat{a}^*_{\mathbf{x}})}{\sigma(\mathbf{x}, \hat{a}^*_{\mathbf{x}})} \Big/ \sum_{\tilde{a}\in A_{\mathbf{x}}, \tilde{a}=\hat{a}^*_{\mathbf{x}}} \frac{(r_{\mathbf{x}}(\tilde{a}, a))^2}{\sigma^2(\mathbf{x}, \tilde{a})}.$$

Since PCS is lower bounded by APCS, and the posterior $\tilde{Q}(\mathbf{x}, a)$ is normally distributed with

$$\tilde{Q}(\mathbf{x}, a) \sim N\left(\bar{Q}(\mathbf{x}, a), \frac{\sigma^2(\mathbf{x}, a)}{N(\mathbf{x}, a)}\right)$$

then

$$\begin{aligned} PCS &\geq APCS \\ &= 1 - \sum_{a\in A_{\mathbf{x}}, a=\hat{a}^*_{\mathbf{x}}} P\left(\tilde{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) \leq \tilde{Q}(\mathbf{x}, a)\right) \\ &= 1 - \sum_{a\in A_{\mathbf{x}}, a=\hat{a}^*_{\mathbf{x}}} \Phi\left(\frac{\bar{Q}(\mathbf{x}, a) - \bar{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}})}{\sigma_{\mathbf{x}}(a, \hat{a}^*_{\mathbf{x}})}\right) \\ &= 1 - \sum_{a\in A_{\mathbf{x}}, a=\hat{a}^*_{\mathbf{x}}} \Phi\left(-\frac{\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, a)}{\sigma_{\mathbf{x}}(a, \hat{a}^*_{\mathbf{x}})}\right) \end{aligned}$$

where the second equality is because $\tilde{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) - \tilde{Q}(\mathbf{x}, a)$ is normally distributed with mean $\bar{Q}(\mathbf{x}, a) - \bar{Q}(\mathbf{x}, \hat{a}^*_{\mathbf{x}})$ and variance

$$\begin{aligned} \sigma^2_{\mathbf{x}}(a, \hat{a}^*_{\mathbf{x}}) &= \frac{\sigma^2(\mathbf{x}, \hat{a}^*_{\mathbf{x}})}{N(\mathbf{x}, \hat{a}^*_{\mathbf{x}})} + \frac{\sigma^2(\mathbf{x}, a)}{N(\mathbf{x}, a)} \\ &= \frac{1}{N(\mathbf{x}, \hat{a}^*_{\mathbf{x}})} \sigma^2(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) + \\ &\quad \sigma(\mathbf{x}, \hat{a}^*_{\mathbf{x}})\sigma^2(\mathbf{x}, a) \sum_{\tilde{a}\in A_{\mathbf{x}}, \tilde{a}=\hat{a}^*_{\mathbf{x}}} \frac{(r_{\mathbf{x}}(\tilde{a}, a))^2}{\sigma^2(\mathbf{x}, \tilde{a})}. \end{aligned}$$

Applying inequality $\sum_{i=1}^n \overline{c_i^2} \leq \sum_{i=1}^n \overline{c_i^2} = \sum_{i=1}^n c_i$ for positive numbers c_i's yields

$$\begin{aligned} \sigma^2_{\mathbf{x}}(a, \hat{a}^*_{\mathbf{x}}) &\leq \frac{1}{N(\mathbf{x}, \hat{a}^*_{\mathbf{x}})} \sigma^2(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) + \\ &\quad \sigma(\mathbf{x}, \hat{a}^*_{\mathbf{x}})\sigma^2(\mathbf{x}, a) \sum_{\tilde{a}\in A_{\mathbf{x}}, \tilde{a}=\hat{a}^*_{\mathbf{x}}} \frac{r_{\mathbf{x}}(\tilde{a}, a)}{\sigma(\mathbf{x}, \tilde{a})}. \end{aligned}$$

Since APCS is decreasing in $\sigma^2_{\mathbf{x}}(a, \hat{a}^*_{\mathbf{x}})$, we have

$$PCS \geq 1 - \sum_{a\in A_{\mathbf{x}}, a=\hat{a}^*_{\mathbf{x}}} \Phi\left(-\frac{\delta_{\mathbf{x}}(\hat{a}^*_{\mathbf{x}}, a)\sqrt{N(\mathbf{x}, \hat{a}^*_{\mathbf{x}})}}{\sqrt{\sigma^2(\mathbf{x}, \hat{a}^*_{\mathbf{x}}) + \sigma(\mathbf{x}, \hat{a}^*_{\mathbf{x}})\sigma^2(\mathbf{x}, a) \sum_{\tilde{a}\in A_{\mathbf{x}}, \tilde{a}=\hat{a}^*_{\mathbf{x}}} \frac{r_{\mathbf{x}}(\tilde{a}, a)}{\sigma(\mathbf{x}, \tilde{a})}}}\right)$$

as desired. □

Acknowledgment

The views, opinions, and/or findings expressed are those of the authors, and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] Y. Li, M. C. Fu, and J. Xu, "Monte Carlo tree search with optimal computing budget allocation," in *Proc. 58th IEEE Conf. Decis. Control*, vol. 3, Dec. 2019, pp. 6332–6337.

[2] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA, USA: Athena Scientific, 1995.

[3] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, "An adaptive sampling algorithm for solving Markov decision processes," *Operations Res.*, vol. 53, no. 1, pp. 126–139, 2005.

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

[5] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. 17th Eur. Conf. Mach. Learn.*, pp. 282–293, 2006.

[6] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proc. Int. Conf. Comput. Games*, H. J. van den Herik, P. Ciancarini, and H. H. L. M. J. Donkers, Eds., 2007, pp. 72–83.

[7] P. Hingston and M. Masek, "Experiments with Monte Carlo Othello," in *Proc. IEEE Congr. Evol. Comput.*, Sep. 2007, pp. 4059–4064.

[8] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[9] P.-A. Coquelin and R. Munos, "Bandit algorithms for tree search," 2007, *arXiv:cs/0703062*.

[10] M. P. D. Schadd, "Selective search in games of different complexity," Ph.D. thesis, department of knowledge engineering, Universiteit Maastricht, Maastricht, The Netherlands, 2011.

[11] K. Teraoka, K. Hatano, and E. Takimoto, "Efficient sampling method for Monte Carlo tree search problem," *IEICE Trans. Inf. Syst.*, vol. 97, no. 3, pp. 392–398, 2014.

[12] E. Kaufmann and W. M. Koolen, "Monte-Carlo tree search by best arm identification," in *Proc. Adv. Neural Inf. Process. Syst.*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., , 2017, pp. 4897–4906.

[13] J.-B. Grill, M. Valko, and R. Munos, "Blazing the trails before beating the path: sample-efficient Monte-Carlo planning," in *Proc. Adv. Neural Inf. Process. Syst.*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 4680–4688.

[14] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Adv. Appl. Math.*, vol. 6, pp. 4–22, Mar. 1985.

[15] M. Soare, A. Lazaric, and R. Munos, "Best-arm identification in linear bandits," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, Cambridge, MA, USA, 2014, pp. 828–836.
[16] C. Tao, S. Blanco, and Y. Zhou, "Best arm identification in linear bandits with linear dimension dependency," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, J. Dy and A. Krause, Eds., 10–15 Jul. 2018, pp. 4877–4886.
[17] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvari, "X-armed bandits," *J. Mach. Learn. Res.*, vol. 12, pp. 1655–1695, Apr. 2011.
[18] S. Bubeck and C. Nicolò, *Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems*. Boston, MA, USA: Now, 2012.
[19] C.-H. Chen, J. Lin, E. Yücesan, and S. E. Chick, "Simulation budget allocation for further enhancing the efficiency of ordinal optimization," *Discrete Event Dyn. Syst.*, vol. 10, no. 3, pp. 251–270, 2000.
[20] C.-H. Chen and L. H. Lee, *Stochastic Simulation Optimization: An Optimal Computing Budget Allocation*. 1st ed., River Edge, NJ, USA: World Scientific, 2010.
[21] C.-H. Chen, "An effective approach to smartly allocate computing budget for discrete event simulation," in *Proc. 34th IEEE Conf. Decis. Control*, vol. 3, pp. 2598–2603 vol.3, Dec. 1995.
[22] L. H. Lee, E. P. Chew, S. Teng, and D. Goldsman, "Optimal computing budget allocation for multi-objective simulation models," in *Proc. Winter Simul. Conf.*, vol. 1, Dec., p. 594, 2004.
[23] S. Zhang, L. H. Lee, E. P. Chew, J. Xu, and C.-H. Chen, "A simulation budget allocation procedure for enhancing the efficiency of optimal subset selection," *IEEE Trans. Autom. Control*, vol. 61, no. 1, pp. 62–75, Jan. 2016.
[24] H. Xiao, F. Gao, and L. H. Lee, "Optimal computing budget allocation for complete ranking with input uncertainty," *IISE Trans.*, vol. 52, no. 5, pp. 489–499, 2020.
[25] A. E. Thanos, M. Bastani, N. Celik, and C. Chen, "Dynamic data driven adaptive simulation framework for automated control in microgrids," *IEEE Trans. Smart Grid*, vol. 8, no. 1, pp. 209–218, Jan. 2017.
[26] L. Y. Hsieh, E. Huang, and C. Chen, "Equipment utilization enhancement in photolithography area through a dynamic system control using multi-fidelity simulation optimization with big data technique," *IEEE Trans. Semicond. Manuf.*, vol. 30, no. 2, pp. 166–175, May 2017.
[27] A. Shleyfman, A. Komenda, and C. Domshlak, "On interruptible pure exploration in multi-armed bandits," in *Proc. 29th AAAI Conf. Artific. Intell.*, 2015. pp. 3592–3598.
[28] S. Chen, T. Lin, I. King, M. R. Lyu, and W. Chen, "Combinatorial pure exploration of multi-armed bandits," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 379–387.
[29] S. Bubeck, R. Munos, and G. Stoltz, "Pure exploration in multi-armed bandits problems," in *Proc. Algorithmic Learn. Theory*, R. Gavaldà, G. Lugosi, T. Zeugmann, and S. Zilles, Eds., Berlin, Germany, 2009, pp. 23–37.
[30] M. H. DeGroot, *Optimal Statistical Decisions*. vol. 82. Hoboken, NJ, USA: Wiley, 2005.
[31] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI in Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[32] Y. Peng, E. K. P. Chong, C.-H. Chen, and M. C. Fu, "Ranking and selection as stochastic control," *IEEE Trans. Autom. Control*, vol. 63, no. 8, pp. 2359–2373, Aug. 2018.
[33] C.-H. Chen, D. He, and M. Fu, "Efficient dynamic simulation allocation in ordinal optimization," *IEEE Trans. Autom. Control*, vol. 51, no. 12, pp. 2005–2009, Dec. 2006.
[34] D. R. Jiang, L. Al-Kanj, and W. B. Powell, "Monte Carlo tree search with sampled information relaxation dual bounds," *Operations Res.*, vol. 68, no. 6, pp. 1678–1697, 2020.
[35] Y. Li, "Monte Carlo tree search with optimal computing budget allocation." 2021, [Online]. Available: https://github.com/lyc192130/MCTS-with-OCBA
[36] I. O. Ryzhov, "On the convergence rates of expected improvement methods," *Operations Res.*, vol. 64, no. 6, pp. 1515–1528, 2016.

Yunchuan Li received the bachelor's degree in automation from University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2015, and the Ph.D. degree in electrical and computer engineering from the University of Maryland, College Park, MD, USA, in 2020.

His research interests include optimization and control with applications to operations research problems.

Michael C. Fu (Fellow, IEEE) received degrees in mathematics and electrical engineering and computer science from Massachusetts Institute of Technology, Cambridge, MA, USA, in 1985, and the Ph.D. degree in applied math from Harvard University, Cambridge, MA, USA, in 1989.

Since 1989, he has been with the University of Maryland, currently holding the Smith Chair of Management Science. He also served as the Operations Research Program Director with the National Science Foundation. His research interests include simulation optimization and stochastic gradient estimation.

Dr. Fu is a Fellow of the Institute for Operations Research and the Management Sciences (INFORMS).

Jie Xu (Senior Member, IEEE) received the B.S. degree in electrical engineering from Nanjing University, Nanjing, China, in 1999, the M.E. degree in electrical engineering from Shanghai Jiaotong University, Shanghai, China, in 2002, the M.S. degree in computer science from The State University of New York, Buffalo, NY, USA, in 2004, and the Ph.D. degree in industrial engineering and management sciences from Northwestern University, Evanston, IL, USA, in 2009.

He is currently an Associate Professor of Systems Engineering and Operations Research with George Mason University, Fairfax, VA, USA. His research interests are data analytics, stochastic simulation and optimization, with applications in cloud computing, manufacturing, and power systems.