ON EFFECTIVE PARALLELIZATION OF MONTE CARLO TREE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite its groundbreaking success in Go and computer games, Monte Carlo Tree Search (MCTS) is computationally expensive as it requires a substantial number of rollouts to construct the search tree, which calls for effective parallelization. However, *how to design effective parallel MCTS algorithms* has not been systematically studied and remains poorly understood. In this paper, we seek to lay its first theoretical foundation, by examining the potential performance loss caused by parallelization when achieving a desired speedup. In particular, we discover two necessary conditions for achieving a desirable parallelization performance and highlight two of their practical benefits. First, by examining whether existing parallel MCTS algorithms satisfy these conditions, we identify key design principles that should be inherited by future algorithms. Specifically, we find that, although no existing algorithm satisfies both conditions in general (i.e., they are still far from optimal), pre-adjusting the nodes visit counts (used in VL-UCT (Segal, 2010) and WU-UCT (Liu et al., 2020)) is essential. In particular, WU-UCT pre-adjusts visit counts in a way that always satisfies one necessary condition while its other designs meet the other condition when the maximum tree depth is 2. We theoretically establish that they together facilitate $\mathcal{O}(\ln n + M/\sqrt{\ln n})$ cumulative regret under the depth-2 setting, where n is the number of rollouts and M is the number of workers. A regret of this form is highly desirable, as compared to $\mathcal{O}(\ln n)$ regret incurred by a sequential counterpart, its excess part approaches zero as n increases. Second, and more importantly, we demonstrate how the proposed necessary conditions can be adopted to design more effective parallel MCTS algorithms. To illustrate this, we propose a new parallel MCTS algorithm, called *BU-UCT*, by following our theoretical guidelines. The newly proposed algorithm, albeit preliminary, out-performs four competitive baselines on 11 out of 15 Atari games. We hope our theoretical results could inspire future work of more effective parallel MCTS.

1 INTRODUCTION

Monte Carlo Tree Search (MCTS) (Browne et al., 2012) algorithms have achieved unprecedented success in fields such as computer Go (Silver et al., 2016), card games (Powley et al., 2011), and video games (Schrittwieser et al., 2019). However, they generally require a large number of Monte Carlo rollouts to construct search trees, making themselves time-consuming. For this reason, parallel MCTS is highly appealing and has been successfully used in solving challenging tasks such as Go (Silver et al., 2017; Couëtoux et al., 2017) and mobile games (Poromaa, 2017; Devlin et al., 2016).

Despite their extensive usage, the performance of parallel MCTS algorithms (Chaslot et al., 2008) is not systematically understood from a theoretical perspective. There are empirical studies on the advantages (e.g., Yoshizoe et al. (2011); Gelly & Wang (2006)) and disadvantages (e.g., Mirsoleimani et al. (2017); Soejima et al. (2010); Bourki et al. (2010)) of existing approaches. However, they are mainly algorithm-specific analysis, which provides less *systematic* design principles on effective MCTS parallelization. As a consequence, practitioners still largely rely on the trial-and-error approach when designing a new parallel MCTS algorithm, which is time-wise costly.

In this paper, we seek to lay the first theoretical foundation for effective MCTS parallelization. Parallel MCTS algorithms generally exhibit different levels of performance loss compared to their

sequential counterparts, especially when a large number of workers are employed to achieve high speedups (Segal, 2010). It is highly desirable for algorithm designers to minimize this loss while still achieving high speedup, especially in solving challenging large-scale tasks. Therefore, we focus on examining the potential performance loss caused by the parallelization when achieving a desired speedup. And we measure the performance loss by *excess regret*, which is the extra cumulative regret of a parallel MCTS algorithm relative to its sequential counterpart. In particular, we will characterize the excess regret from a theoretical perspective and seek to answer the following key question: *under what conditions would the excess regret vanish when the number of rollouts increases?*

To this end, with the help of a unified algorithm framework that covers all major existing parallel MCTS algorithms as its special cases, we derive two necessary conditions for any algorithm specified by the framework to achieve vanishing excess regret when the number of rollouts increases (Thm. 1). We then highlight two practical benefits of the necessary conditions. First, the conditions allow us to identify *key design wisdom proposed by existing algorithms*, for example pre-adjusting the node visit counts, which is used in VL-UCT (Segal, 2010) and WU-UCT (Liu et al., 2020). Second, and more importantly, we show that the necessary conditions can provide concrete guidelines for designing better (future) algorithms, which is demonstrated through an example workflow of algorithm design based on the necessary conditions. The resulting algorithm, **B**alance the **U**nobserved in UCT (BU-UCT), out-performs four competitive baselines on 11 out of 15 Atari games. We hope this encouraging result could inspire more future work to develop better parallel MCTS algorithms with our theory.

2 Preliminary: MCTS and its Parallelization

Consider a *Markov Decision Process* (MDP) $\langle \mathcal{S}, \mathcal{A}, R, P \rangle$, where $\mathcal{S}$ denotes a *finite* state space, $\mathcal{A}$ is a *finite* action space, R is a *bounded* reward function, and P defines a *deterministic* state transition function. We additionally define $\gamma \in (0, 1]$ as the discount factor. At each time step t, the agent takes an action a_t when the environment is in a state s_t, causing it to transit to the next state s_{t+1} and emit a reward r_t. In the context of MCTS, P and R (or their approximations) are assumed to be known to the agent. By exploiting such knowledge, MCTS seeks to *plan* the best action a at a given state s to achieve the highest expected cumulative reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$. To this end, it constructs a search tree using a sequence of repeated Monte Carlo *rollouts*, where a node corresponds to a state, and an edge from s_t to s_{t+1} represents the action a_t that causes the transition from s_t to s_{t+1}. Each edge (s, a) in the search tree also stores a set of statistics $\{Q(s, a), N(s, a)\}$, where $Q(s, a)$ is the mean action value and $N(s, a)$ is the count of completed simulations. These statistics guide the construction of the search tree and are updated during the process. Specifically, during the *selection* phase, the algorithm traverses over the current search tree by using a *tree policy* (e.g., the Upper Confidence Bound (UCB) Auer (2002)) to iteratively select an action a_t that leads to a child node s_{t+1}:

$$a_t = \arg\max_{a \in \mathcal{A}} \left\{ Q(s_t, a) + c \sqrt{\frac{2 \ln \sum_{a'} N(s_t, a')}{N(s_t, a)}} \right\}, \tag{1}$$

where the first term estimates the utility of executing a at s_t, the second term represents the uncertainty of that estimate, and the hyperparameter c controls the tradeoff between exploitation (term 1) and exploration (term 2). The selection process is performed iteratively until arriving at a node s_{T-1} where some of its actions are not expanded. Then, the algorithm selects an unexpanded action a_{T-1} at s_{T-1} and adds a new leaf node s_T (corresponds to the next state) to the search tree at the *expansion* phase, followed by querying its value $V(s_T)$ through *simulation*, where a *default policy* repeatedly interacts with the MDP starting from s_T. Finally, in *backpropagation*, the statistics along the selected path are recursively updated from s_{T-1} to s_0 (i.e., from $t = T - 1$ to $t = 0$) by

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1, \quad V(s_t) = R(s_t, a_t) + \gamma V(s_{t+1}), \tag{2}$$

$$Q(s_t, a_t) \leftarrow \frac{N(s_t, a_t) - 1}{N(s_t, a_t)} Q(s_t, a_t) + \frac{V(s_{t+1})}{N(s_t, a_t)}, \tag{3}$$

where the recursion starts from the simulation return value $V(s_T)$.

Parallel MCTS algorithms seek to speedup their sequential counterparts by distributing workloads stemmed from the simulation steps to multiple workers, aiming to achieve the same performance with less computation time. Fig. 1 presents five typical parallel MCTS algorithms. Among them, Leaf Parallelization (LeafP) (Cazenave & Jouandeau, 2007) assigns multiple workers to simulate the same node simultaneously; Root Parallelization (RootP) (Cazenave & Jouandeau, 2007) adopts the workers to independently maintain different search trees, and the statistics are aggregated after all workers

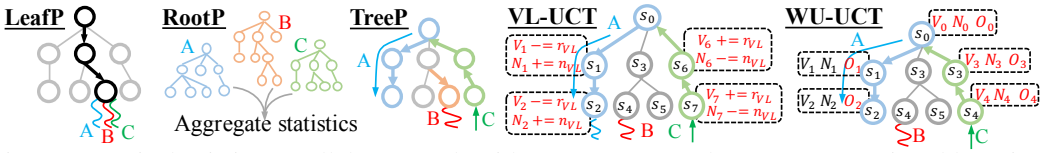


Figure 1: Typical existing parallel MCTS algorithms. VL-UCT and WU-UCT use virtual loss (i.e., r_{VL}) and number of on-going simulations (i.e., O) to pre-adjust node statistics, respectively.

complete their jobs; in Tree Parallelization (TreeP) (Chaslot et al., 2008), the workers independently perform rollouts on a shared search tree; TreeP with Virtual Loss (VL-UCT) (Segal, 2010; Silver et al., 2016) and Watch the Unobserved in UCT (WU-UCT) (Liu et al., 2020) pre-adjust the node statistics with side information to achieve a better exploration-exploitation tradeoff. Please refer to Appendix A for a more detailed and thorough discussion of existing parallel MCTS algorithms.

Main challenges Since parallel MCTS algorithms have to initiate new rollouts before all assigned simulation tasks are completed, they are generally not able to incorporate the information from *all* initiated simulations into its statistics (i.e., Q and N). As demonstrated in previous studies (e.g., Liu et al. (2020)), this could lead to significant performance loss compared to sequential MCTS algorithms since the tree policy (Eq. (1)) cannot properly balance exploration and exploitation when using such statistics. Therefore, most existing algorithms seek to improve their performance by augmenting the statistics Q and N used by the tree policy, which is done by either adjusting how statistics possessed by different workers are synchronized/aggregated (e.g., LeafP, RootP) or adding additional side information (e.g., VL-UCT, WU-UCT). Specifically, this can be formalized by introducing a set of *modified statistics* (defined as $\overline{Q}$ and $\overline{N}$) in replacement of Q and N in the tree policy (Eq. (1)):

$$\overline{Q}(s,a) := \alpha(s,a)\cdot Q(s,a) + \beta(s,a)\cdot \widetilde{Q}(s,a), \qquad \overline{N}(s,a) := N(s,a) + \widetilde{N}(s,a)^{1}, \tag{4}$$

where $\widetilde{Q}$ and $\widetilde{N}$ are a set of *pseudo statistics* that incorporate additional side information; α and β control the ratio between Q and $\widetilde{Q}$. Common choices of the pseudo statistics include virtual loss (Segal, 2010; Silver et al., 2016) and incomplete visit count (Liu et al., 2020). Given this formulation, a natural question is *how to design $\overline{Q}$ and $\overline{N}$ in order to achieve good parallel performance in MCTS?*

3 Overview of Our Main Theoretical Results

The main objective of this paper is to answer the above question by identifying key necessary conditions of $\overline{Q}$ and $\overline{N}$ to achieve desirable performance[2] in parallel MCTS algorithms. Throughout the paper, we highlight two benefits of our theoretical results: in hindsight, they help identify beneficial design principles used in existing algorithms (Sec. 4.3); furthermore, they offer simple and effective guidelines for designing better (future) algorithms (Sec. 5).

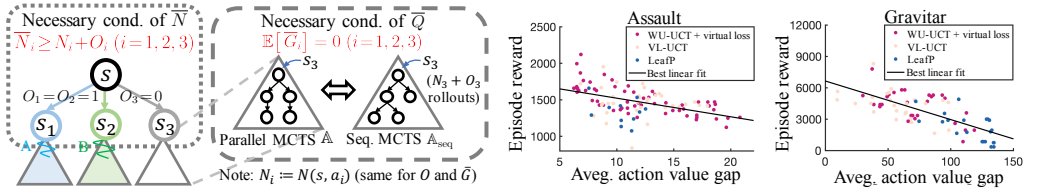
The two necessary conditions are best illustrated in Fig. 2(a). Consider node s in a search tree where we want to select one of its child nodes. Workers A and B are in their simulation steps, querying an offspring node of s_1 and s_2, respectively. To introduce the necessary condition of $\overline{N}$, we define the *incomplete visit count* $O(s,a)$, which was introduced by Liu et al. (2020) to track *the number of simulation tasks that has been initiated but not yet completed*. For example, in Fig. 2(a), both the edges associated with s_1 and s_2 have incomplete visit counts of 1 since workers A and B are still simulating their offspring nodes. The necessary condition regarding $\overline{N}$ is stated as follows:

$$\forall (s,a) \in \{\text{edges in the search tree}\} \quad \overline{N}(s,a) \geq N(s,a) + O(s,a). \tag{5}$$

One potential benefit of adding incomplete visit count (i.e., O) into $\overline{N}$ is to improve the diversity of exploration (Liu et al., 2020). Specifically, since increasing O leads to a decrease of the exploration bonus (the second term) in the tree policy (Eq. (1)), nodes with a high incomplete visit count will be less likely to be selected by other workers, which increase the diversity of exploration. In our example, the chance of selecting s_3 is increased due to the introduction of O. In conclusion, this necessary condition suggests that pre-adjusting the visit count is essential for effective MCTS parallelization.

[1] Given the N-related are counts of simulations, which means $\overline{N}$ shall never be smaller than N, this equation is sufficient for the general purpose and there is no need for weights before N and $\widetilde{N}$.

[2] The notion of "desirable performance" will be formalized in Sec. 4.1.



(a) Illustration of the two necessary conditions in Theorem 1. (b) The aveg. action value gap is negatively correlated w. episode reward.

Figure 2: The necessary conditions to achieve vanishing excess regret and their implications.

The necessary condition of $\overline{Q}$ focuses on the similarity between the action value maintained by the parallel MCTS algorithm $\mathbb{A}$ and its sequential counterpart $\mathbb{A}_{\text{seq}}$. Formally, it requires the following *action value gap* $\overline{G}(s,a)$ to be *zero* for each edge (s,a) in the search tree:

$$\overline{G}(s,a) := \left|\mathbb{E}\left[\overline{Q}(s,a)\right] - \mathbb{E}\left[Q_m^{\mathbb{A}_{\text{seq}}}(s,a)\right]\right| \quad (m = N(s,a) + O(s,a)), \tag{6}$$

where $\overline{Q}(s,a)$ is generated by the parallel MCTS algorithm $\mathbb{A}$, $Q_m^{\mathbb{A}_{\text{seq}}}(s,a)$ represents the action value of a sequential MCTS algorithm $\mathbb{A}_{\text{seq}}$ that starts from the child node of (s,a) and runs for m rollouts, and $\mathbb{E}[\cdot]$ averages over the randomness in the simulation returns. Although seemingly nontrivial to satisfy, we will show that it indeed provides important insights for designing better algorithms.

Finally, we revisit both necessary conditions and give a preview of their two benefits: (i) identifying useful designs in existing algorithms that should be inherited by future algorithms (Section 4), and (ii) revealing new design principles for future algorithms (Section 5). First, we identify key techniques used in existing algorithms that are *aligned with our theoretical findings*. Although none of them satisfy both necessary conditions in the general MCTS setup, WU-UCT satisfies both conditions when the maximum depth of the search tree is 2. In hindsight, this implies that the design of using incomplete visit count to *only* adjust the modified visit count $\overline{N}$ (Eq. (4)) is (partly) consistent with our theoretical guidelines. And we further confirm the benefit of this essential design by showing that it facilitates WU-UCT to achieve a cumulative regret of $\mathcal{O}(\ln n + M/\sqrt{\ln n})$ when the maximum tree depth is 2 (Thm. 2), where n is the total number of rollouts and M is the number of workers. Compared to sequential UCT, whose cumulative regret is $\mathcal{O}(\ln n)$, WU-UCT merely incurs an excess regret of $\mathcal{O}(M/\sqrt{\ln n})$ that goes to zero as n increase. Second and more importantly, the necessary condition of $\overline{Q}$ can guide us in designing better (future) algorithms. Specifically, we show in Fig. 2(b) that the action value gap $\overline{G}$ is a strong performance indicator of parallel MCTS algorithms. The scatter plots obtained from two Atari games demonstrate that, regardless of algorithms and hyperparameters, there is a strong negative correlation between the action value gap $\overline{G}$ and the performance. In Sec. 5, we will demonstrate that finding a surrogate gap to approximate $\overline{G}$ and reducing its magnitude could lead to significant performance improvement across a large number of Atari games.

4 Parallel MCTS: Theory and Implications

Following our aforementioned takeaways, we start this section with formalizing the evaluation criteria of MCTS parallelization before presenting the rigorous development of our theoretical results.

4.1 What is Effective Parallel MCTS?

We analyze the performance of parallel MCTS algorithms by examining their *performance loss* under a fixed *speedup* requirement. To begin with, we define the following metrics.

Speedup The speedup of a parallel MCTS algorithm $\mathbb{A}$ using M workers[3] is defined as

$$\text{speedup} = \frac{\text{runtime of the sequential MCTS}}{\text{runtime of algorithm } \mathbb{A} \text{ using } M \text{ workers}},$$

where the runtime of both the sequential and the parallel algorithms is measured by the duration of performing the same fixed number of rollouts. Assuming simulation is much more time-consuming compared to other steps,[4] parallel MCTS algorithms have a speedup *close to* M (see also Section 5) since all M workers will be occupied by simulation tasks most of the time.

[3] A worker refers to a computation unit in practical algorithms that performs simulation tasks *sequentially*.

[4] This holds in general since *only* the simulation step requires massive interactions with the environment.

Performance loss We measure the performance of a parallel MCTS algorithm $\mathbb{A}$ by *expected cumulative regret*, a common metric also used in related theoretical studies (Kocsis et al., 2006; Auer et al., 2002; Auer, 2002):

$$\text{Regret}_{\mathbb{A}}(n) := \sum_{i=1}^{n} \mathbb{E}\big[V_i^*(s_0) - V_i(s_0)\big], \tag{7}$$

where s_0 is the root state of the search tree; n is the number of rollouts; $V_i(s_0)$ is the value estimate of s_0 obtained in the ith rollout of algorithm $\mathbb{A}$, which is computed according to Eq. (2); similarly, $V_i^*(s_0)$ is the estimated value of s_0 acquired in the ith rollout of an oracle algorithm that always select the highest-rewarded action; the expectation is performed to average over the randomness in the simulation returns. Intuitively, cumulative regret measures the expected regret of not having selected the optimal path. We measure the *performance loss* of a parallel MCTS algorithm $\mathbb{A}$ by *excess regret*, which is defined as the difference between the regret of $\mathbb{A}$ and its sequential counterpart $\mathbb{A}_{\text{seq}}$ (i.e., $\text{Regret}_{\mathbb{A}}(n) - \text{Regret}_{\mathbb{A}_{\text{seq}}}(n)$). We say algorithm $\mathbb{A}$ has *vanishing excess regret* if and only if its excess regret converges to zero as n goes to infinity. Roughly speaking, having vanishing excess regret means the parallel algorithm is almost as good as sequential MCTS under large n.[5]

Beside cumulative regret, simple regret is also widely used in related studies. While it is generally agreed that simple regret is preferable in the Multi-Armed Bandit (MAB) setting given only the final recommendation affect the performance, it is still debatable whether MCTS should seek to minimize simple or cumulative regret (Pepels et al., 2014). Specifically, nodes in the search tree need both good final performance (corr. to simple regret) to make good recommendations and good any-time performance (corr. to cumulative regret) to backpropagate well-estimated values $V(s)$. In particular, Tolpin & Shimony (2012) highlighted this contradiction in MCTS and proposed a simple yet effective solution: minimizing simple regret when selecting children of the root node while minimizing cumulative regret at non-root nodes. Following this high-level idea, recently proposed (sequential) MCTS algorithms largely use hybrid approaches that seek to minimize *both* simple regret and cumulative regret (Feldman & Domshlak, 2014b;a; Kaufmann & Koolen, 2017; Liu & Tsuruoka, 2015; Hay et al., 2014). As argued in these works (e.g., Hay et al. (2014); Tolpin & Shimony (2012)), both types of regret are useful metrics that have a strong correlation with MCTS performance, and both are worth studying in the context of MCTS. In this paper, we focus on excess cumulative regret, and leave analysis revolving around simple regret to future work.

4.2 When Will Excess Regret Vanish?

This section examines *what conditions should be satisfied for a parallel MCTS algorithm to achieve vanishing excess regret*. To perform a unified theoretical analysis of existing parallel MCTS algorithms, we introduce a general algorithm framework (formally introduced in Appendix B) that covers most existing parallel MCTS algorithms and their variants as its special cases. Specifically, Appendix B.3 provides a rigorous justification of how the general framework can be specialized to LeafP, RootP, TreeP, VL-UCT, and WU-UCT. The following theorem gives two necessary conditions for any algorithm specialized from the general framework to achieve vanishing excess regret.

Theorem 1. *Consider an algorithm $\mathbb{A}$ that is specified from the general parallel MCTS framework formally introduced in Appendix B. Choose $\widetilde{N}(s,a)$ as a function of $O(s,a)$. If there exists an edge (s,a) in the search tree such that $\mathbb{A}$ violates any of the following conditions:*

- ***Necessary cond. of $\overline{Q}$:*** $\overline{G}(s,a) := \big|\mathbb{E}[\overline{Q}(s,a)] - \mathbb{E}[Q_m^{\mathbb{A}_{\text{seq}}}(s,a)]\big| = 0 \ \big(m = N(s,a) + O(s,a)\big)$, (8)
- ***Necessary cond. of $\overline{N}$:*** $\overline{N}(s,a) \geq N(s,a) + O(s,a)$, (9)

then there exists an MDP $\mathcal{M}$ such that the excess regret of running $\mathbb{A}$ on MDP $\mathcal{M}$ does not vanish.

Proof of the above theorem is provided in Appendix C.1. While the necessary condition of $\overline{N}$ is rather straightforward, suggesting that the modified visit count $\overline{N}(s,a)$ should be no less than the total number of simulations *initiated* (regardless of completed or not) from offspring nodes of (s,a) (i.e., $N(s,a)+O(s,a)$), the necessary condition of $\overline{Q}$ needs further elaboration. Intuitively, the action value gap $\overline{G}(s,a)$ measures *how well the modified action value $\overline{Q}(s,a)$ of $\mathbb{A}$ approximates the action value computed by its sequential counterpart* $\mathbb{A}_{\text{seq}}$ (i.e., $Q_m^{\mathbb{A}_{\text{seq}}}(s,a)$). There are two main obstacles toward lowering the action value gap and satisfy its necessary condition (i.e., Eq. (8)). First, as demonstrated in Sec. 3 as well as previous studies (Chaslot et al., 2008; Liu et al., 2020), the statistics

[5] Note that with relatively small n, parallel MCTS is in general inferior to their sequential counterpart since they are not able to collect sufficient information for effective exploration-exploitation tradeoff during selection.

used by the tree policy (Eq. (1)) in parallel MCTS algorithms tend to harm the effectiveness of the tree policy, which leads to suboptimal node selections and hence biases the simulation outcomes compared to that of the sequential algorithm. Second, as hinted by our notation, while the modified action value $\overline{Q}(s,a)$ incorporates information from $N(s,a)$ simulation returns, $Q_m^{\mathbb{A}_{\text{seq}}}(s,a)$ is the average of $N(s,a)+O(s,a)$ simulation outcomes. This requires the modified action value $\overline{Q}$ to incorporate additional information that helps "anticipate" the outcomes of incomplete simulations through the pseudo action value $\widetilde{Q}$.

4.3 Rethinking Existing Parallel MCTS Algorithms

In retrospect, we examine which techniques proposed in existing algorithms should be retained in future parallel MCTS algorithms by inspecting whether they satisfy the two necessary conditions. First, regarding $\overline{Q}$, existing algorithms either modify how simulation returns from different workers aggregate to generate action values (e.g., LeafP and RootP) or use virtual loss (Segal, 2010) to penalize action value and visit counts of nodes with high incomplete visit count (e.g., VL-UCT), which not necessarily minimize the action value gap $\overline{G}$. Hence, based on our knowledge, the necessary condition of $\overline{Q}$ is not satisfied by any existing algorithms in the general MCTS setup. Next, regarding $\overline{N}$, we found that properly pre-adjusting visit count satisfies the second necessary condition. While there are various approaches to pre-adjust visit counts, we found the design of WU-UCT particularly useful: by *only* adjust the modified visit count (i.e., $\overline{N}(s,a) := N(s,a)+O(s,a)$) and keep the modified action value $\overline{Q}$ unchanged, WU-UCT always satisfies the second necessary condition and satisfies the first necessary condition when the maximum depth of the search tree is 2. Now a natural question to ask is *whether satisfying both necessary conditions (in the depth-2 setup) offer noticeable gain in WU-UCT's performance from a theoretical perspective*, which can be answered in the affirmative. Specifically, besides its empirical success reported in the original paper, we demonstrate the superiority of WU-UCT from a theoretical perspective through the following theorem.

Theorem 2. *Consider a tree search task $\mathbb{T}$ with maximum depth $D=2$ (abbreviate as the depth-2 tree search task): it contains a root node s and K feasible actions $\{a_i\}_{i=1}^K$ at s, which lead to terminal states $\{s_i\}_{i=1}^K$, respectively. Let $\mu_i := \mathbb{E}[V(s_i)]$, $\mu^* := \max_i \mu_i$ and $\Delta_k := \mu^* - \mu_k$, and further assume: $\forall i, V(s_i) - \mu_i$ is 1-subgaussian (Buldygin & Kozachenko, 1980). The cumulative regret of running WU-UCT (Liu et al., 2020) with n rollouts on $\mathbb{T}$ is upper bounded by:*

$$\underbrace{\sum_{k:\mu_k<\mu^*}\left(\frac{8}{\Delta_k}+2\Delta_k\right)\ln n+\Delta_k}_{R_{\text{UCT}}(n)}+\underbrace{4M\sum_{k:\mu_k<\mu^*}\frac{\Delta_k^2}{\sqrt{\ln n}}}_{\text{excess regret}},$$

where $R_{\text{UCT}}(n)$ is the cumulative regret of running the (sequential) UCT for n steps on $\mathbb{T}$.

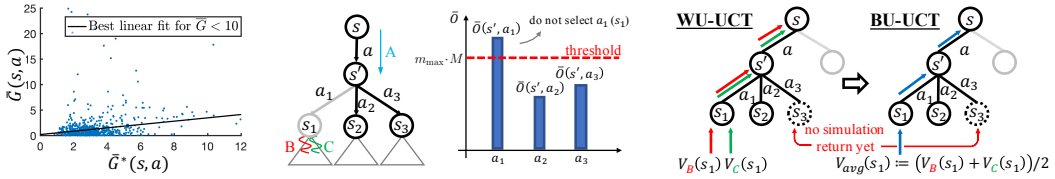
Proof of the above theorem is provided in Appendix C.2. Before interpreting the theorem, we emphasize that this result only apply to tasks where the maximum depth of the search tree is 2, which closely resembles the Multi-Armed Bandit (Auer et al., 2002; Auer, 2002) setup. In the general setting (i.e., depth > 2), since WU-UCT does not satisfy the first necessary condition, it will not have vanishing excess regret anymore. Therefore, although WU-UCT has some desirable properties that other existing algorithms do not, it is still far from optimal when considering MCTS tasks in general.

Thm. 2 indicates that the regret upper bound of WU-UCT in the depth-2 tree search task consists of two terms: the cumulative regret of the sequential UCT algorithm (i.e., R_{UCT}) and an excess regret term that *converges to zero as n increases*. Apart from showing a desirable theoretical property of WU-UCT, this result suggests that designing algorithms that satisfy the necessary conditions in Thm. 1 can potentially offers empirical as well as theoretical benefits.

In conclusion, by looking back at existing parallel MCTS algorithms, the necessary conditions suggest that retaining WU-UCT's approach to augment $\overline{N}$ while keep $\overline{Q}$ unchanged (i.e., $\overline{Q}(s,a) := Q(s,a)$) would be beneficial in the depth-2 setting. This left us with the question *how to make use of the necessary condition of $\overline{Q}$ to further improve existing parallel MCTS algorithms* in the general MCTS setting (i.e., maximum tree depth > 2), which will be addressed in the following section.

5 Theory in Practice: A Promising Study

In this section, we demonstrate that exploiting the proposed necessary conditions in Thm. 1 can immediately lead to a more effective parallel MCTS algorithms: **B**alance the **U**nobserved in UCT



(a) $\overline{G}^*(s,a)$ vs. $\overline{G}(s,a)$. (b) Key idea #1: thresholding $\overline{O}$. (c) Key idea #2: aggregate-and-backprop.

Figure 3: The BU-UCT algorithm. (a) Motivation: The statistics $\overline{G}^*$ is strongly correlated with the original gap $\overline{G}$, suggesting that it can be used as a surrogate gap to guide algorithm design. (b) Key idea #1: reducing $\overline{G}^*$ by thresholding $\overline{O}$ — only query nodes whose $\overline{O}$ is smaller than a threshold. (c) Key idea #2: aggregate the simulation returns on a same node (e.g., s_1) and then backpropagate.

(BU-UCT). The newly proposed BU-UCT, albeit preliminary, is shown to outperform strong baselines (including WU-UCT, the current state-of-the-art) on 11 out of 15 Atari games. We want to highlight that BU-UCT is only used as an illustrative example about how to use our theoretical results in practice, and we hope this encouraging result could inspire more future work to develop better parallel MCTS algorithms with our theory.

Algorithm Design Thm. 1 suggests that parallel-MCTS algorithms should be designed to satisfy both necessary conditions. First, it is relatively easier to construct $\overline{N}$ to satisfy its necessary condition. For example, we can borrow wisdom from WU-UCT to choose $\overline{N}(s,a) := N(s,a) + O(s,a)$. On the other hand, however, the necessary condition on $\overline{Q}$ (i.e., $\overline{G}(s,a) = 0$) is more difficult to satisfy strictly. Nevertheless, we find that the magnitude of the average action value gap $\overline{G}(s,a)$ has a strong negative correlation with the actual performance (i.e., episode reward) — see Fig. 2(b).[6] And the behavior holds true regardless of the algorithms (points with different colors represent different algorithms) as well as the hyperparameters (points of the same color denote results obtained from different hyperparameters). The phenomenon suggests that *designing a parallel-MCTS algorithm that reduces $\overline{G}$ could lead to better performance in practice*. However, according to Eq. (8), directly using the original gap $\overline{G}(s,a)$ for algorithm design is not practical because it requires running the sequential UCT algorithm to compute $Q_m^{\mathbb{A}_{\text{seq}}}$. Therefore, a more realistic approach is to construct a *surrogate gap* to approximate $\overline{G}(s,a)$ based on the available statistics. In the following, we give one example to show how to construct such a surrogate gap for designing a better parallel MCTS algorithms. Please refer to Appendix E for more potential options for the surrogate gap.

Let $O_i(s,a)$ be the number of on-going simulations associated with the edge (s,a) at the ith rollout step. We consider using the following statistics $\overline{G}^*(s,a)$ as a surrogate gap to approximate $\overline{G}(s,a)$:

$$\overline{G}^*(s,a) := \max_{a' \in \mathcal{A}} \overline{O}(s',a') = \max_{a' \in \mathcal{A}} \left\{ \frac{1}{n} \sum_{i=1}^{n} O_i(s',a') \right\} \ (s' \text{ is the next state following } (s,a)), \quad (10)$$

where n is the number of rollouts. Before discussing its key insights, we first examine the correlation between $\overline{G}^*$ and the action value gap $\overline{G}$. As shown in Fig. 3(a), except for a few outliers, $\overline{G}^*(s,a)$ and $\overline{G}(s,a)$ have a strong positive correlation.[7] Motivated by this observation, we seek to design a better parallel MCTS algorithm by reducing the surrogate gap $\overline{G}^*(s,a)$. In the following, we introduce the proposed algorithm BU-UCT, and highlight how it lowers the surrogate gap $\overline{G}^*(s,a)$.

Algorithm Details Built on top of WU-UCT, BU-UCT proposes to lower $\overline{G}^*$ through (i) thresholding $\overline{O}$, and (ii) aggregating-and-backpropagating simulation returns. The first idea, thresholding $\overline{O}$, seek to explicitly set an upper limit to $\overline{O}$ (and hence $\overline{G}^*$). Specifically, BU-UCT keeps record of the $\overline{O}$ values on all edges and assure edges whose $\overline{O}$ is above a threshold will not be selected by the tree policy. Concretely, this is achieved with the following modified action value $\overline{Q}$:

$$\overline{Q}(s,a) := Q(s,a) + \mathbb{I}\left[\overline{O}(s,a) < m_{\max} \cdot M\right], \quad (11)$$

where $m_{\max} \in (0,1)$ is a hyperparameter and M is the number of workers; the indicator function $\mathbb{I}[\cdot]$ is defined to be zero when the condition holds and $-\infty$ otherwise. Consider the example given

[6] Note that each game step requires a new search tree. Hence the action value gap is averaged w.r.t. (i) search trees built at different game steps and (ii) different nodes in a search tree. See Appendix F.3 for more detail.

[7] Note that there are a few data points with $\overline{G}(s,a) > 10$ that the surrogate statistics cannot fit properly, which indicates that there could exist better surrogate gap that potentially leads to better parallel MCTS algorithms.

Table 1: Performance on 15 Atari games. Average episode return ($\pm$ standard deviation) over 5 trials are reported. The best average scores are highlighted in boldface. According to t-tests, BU-UCT significantly outperforms or is comparable with the existing alternative on 14 games, except RoadRunner where WU-UCT is better. "*", "†", "‡", and "§" denote BU-UCT's large-margin superiority (p-value < 0.05) over WU-UCT, VL-UCT, LeafP, and RootP, respectively.

Environment	BU-UCT (ours)		WU-UCT	VL-UCT	LeafP	RootP
Alien	5320±231	†‡	**5938**±1839	4200±1086	4280±1016	5206±282
Boxing	**100**±0	†‡§	**100**±0	99±0	95±4	98±1
Breakout	**425**±30	‡§	408±21	390±33	331±45	281±27
Centipede	**1610419**±338295	†‡§	1163034±403910	439433±207601	162333±69575	184265±104405
Freeway	**32**±0		**32**±0	**32**±0	31±1	**32**±0
Gravitar	**5130**±499	‡	5060±568	4880±1162	3385±155	4160±1811
MsPacman	17279±6136	‡§	**19804**±2232	14000±2807	5378±685	7156±583
NameThisGame	**47066**±5911	*†‡§	29991±1608	23326±2585	25390±3659	27440±9533
RoadRunner	44920±1478	†‡§	**46720**±1359	24680±3316	25452±2977	38300±1191
Robotank	**121**±18	†‡§	101±19	86±13	80±11	78±13
Qbert	**15995**±2635	§	13992±5596	14620±5738	11655±5373	9465±3196
SpaceInvaders	**3428**±525	§	3393±292	2651±828	2435±1159	2543±809
Tennis	3±1	†‡§	**4**±1	−1±0	−1±0	0±1
TimePilot	**111100**±58919	*†‡§	55130±12474	32600±2165	38075±2307	45100±7421
Zaxxon	**42500**±4725	‡§	39085±6838	39579±3942	12300±821	13380±769

in Fig. 3(b). Since $\overline{O}(s', a_1)$ is above the threshold $m_{\max} \cdot M$, its corresponding $\overline{Q}(s', a_1)$ becomes $-\infty$ due to the second term of Eq. (11) and hence the tree policy will not allow the new worker A to select a_1, which will eventually decrease $\overline{O}(s', a_1)$ (and hence lower $\overline{G}^*(s, a)$).

The second key idea, aggregating-and-backpropagating simulation returns, decreases $\overline{G}^*$ by reducing the maximum value in $\{\overline{O}(s', a') \mid a' \in \mathcal{A}\}$. Intuitively, $\overline{G}^*(s, a)$ will be large only if some child nodes of s' are *constantly* (reflected by the "average" operator in the definition of $\overline{O}$) selected by multiple workers. Although it is highly desirable for the optimal child node to be extensively queried constantly, it could be rather concerning if some nodes' incomplete visit count is high even in earlier stages, since this would suggest that its sibling nodes are insufficiently explored. Therefore, BU-UCT decreases the maximum $\overline{O}$ (and hence $\overline{G}^*$) by lowering $\overline{N}$ in earlier stages to encourage exploration of other nodes. Specifically, as shown in Fig. 3(c), we define "earlier stages" as the period when some child nodes (e.g., s_3) of s' have not received any simulation return yet. In this period, we aggregate all simulation returns originated from s''s same child node (e.g., $V_B(s_1)$ and $V_C(s_1)$) into their mean value (e.g., $V_{\text{avg}}(s_1)$). That is, if a node s' is in its "earlier stages", the visit counts of all its children that have received simulation returns are considered as 1 (i.e., $\overline{N}(s', a) = 1$) and their action values are the mean value of their received simulation returns, respectively. Compared to backpropagating all simulation returns individually, backpropagate aggregated statistics lowers $\overline{N}$ at all children of s', which encourage exploration in earlier stages and hence lowers $\overline{G}^*$. Please refer to Algorithm 3 and Appendix G for detailed explanation of the aggregation operation in BU-UCT.

Experiment setup We follow the experiment setup in the WU-UCT paper (Liu et al., 2020). Specifically, we compare BU-UCT with four baselines (i.e., LeafP, RootP, VL-UCT, and WU-UCT) on 15 Atari games (the same 15 Atari games selected in the WU-UCT paper). We use a pretrained PPO policy as the default policy during simulation. All experiments are performed with 128 rollouts and 16 workers. See Appendix F for more details.

Experiment results First, we verify speedup. Across 15 Atari games, BU-UCT achieves an average per-step speedup of 14.33 using 16 workers, suggesting that BU-UCT achieves (approximately) linear speedup even with a large number of workers. Next, we compare the performance, measured by average episode reward, between BU-UCT and four baselines. On Each task, we repeat 5 times with the mean and standard deviation reported in Table 1. Thanks to its efforts to lower the action value gap, BU-UCT outperforms all considered parallel alternatives in 11 out of 15 tasks. Pairwise student t-tests further show that BU-UCT performs significantly better (p-value < 0.05) than WU-UCT, TreeP, LeafP, and RootP in 2, 8, 12, and 12 tasks, respectively; note except in RoadRunner where WU-UCT tops the chart, in all other tasks BU-UCT performs statistically comparably to the baselines, which promisingly renders it as a potential default choice, if one wants to try one parallel MCTS algorithm.

6 Related Works

MCTS has a profound track record of being adopted to achieve optimal planning and decision making in complex environments (Schäfer et al., 2008; Browne et al., 2012; Silver et al., 2016). Recently, it has also been combined with learning methods to bring mutual improvements (Guo et al., 2014; Shen et al., 2018; Silver et al., 2017). To maximize the power of MCTS and enable its usage in time-sensitive tasks, effective parallel algorithms are imperative (Bourki et al., 2010; Segal, 2010). Specifically, leaf parallelization (Cazenave & Jouandeau, 2007; Kato & Takeuchi, 2010) manages to collect better statistics by assigning multiple workers to query the same node, at the expense of reducing the tree search diversity. In root parallelization, multiple trees are built and statistics are periodically synchronized. It promises better performance in some real-world tasks (Bourki et al., 2010), while being inferior on Go (Soejima et al., 2010). In contrast, tree parallelization assigns workers to traverse the same tree. To increase search diversity, Chaslot et al. (2008) proposes a virtual loss. Though having been adopted in some high-profile applications (Powley et al., 2011), virtual loss punishes performance under even four workers (Mirsoleimani et al., 2017). So far, WU-UCT (Liu et al., 2020) achieves the best tradeoff (i.e., linear speedup with small performance loss) by introducing statistics to track on-going simulations. Another related line of works focus on *distributed* multi-armed bandits (MAB) (Liu & Zhao, 2010; Hillel et al., 2013; Lai & Robbins, 1985; Martínez-Rubio et al., 2019), which is similar to parallel MCTS; in both multiple workers collaborate to improve the planning performance. Though inspiring, this line shares an overarching theme that highlights inter-agent communication, making their results not directly adaptable to our setting.

While this work focuses on minimizing the cumulative regret in parallel MCTS, simple regret has been considered as an alternative performance metric to analyze MCTS algorithms (Pepels et al., 2014) and have inspired a great amount of interesting work that seek to minimize both the simple regret and the cumulative regret in MCTS to achieve better performance (Tolpin & Shimony, 2012; Hay et al., 2014; Feldman & Domshlak, 2014a; Kaufmann et al., 2012; Liu & Tsuruoka, 2015).

7 Conclusion

In this paper, we established the first theoretical foundation for parallel MCTS algorithm. In particular, we derived two necessary conditions for the algorithms to achieve a desired performance. The conditions can be used to diagnose existing algorithms and guide future algorithm design. We justify the first benefit (i.e., diagnosing existing algorithms) by identifying the key design wisdom inherent in existing algorithms. The second benefit (i.e., inspiring future algorithms) is demonstrated by constructing a new parallel MCTS algorithm, BU-UCT, based on our theoretical guidelines.

References

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Amine Bourki, Guillaume Chaslot, Matthieu Coulm, Vincent Danjean, Hassen Doghmen, Jean-Baptiste Hoock, Thomas Hérault, Arpad Rimmel, Fabien Teytaud, Olivier Teytaud, et al. Scalability and parallelization of monte-carlo tree search. In *International Conference on Computers and Games*, pp. 48–58. Springer, 2010.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Valerii V Buldygin and Yu V Kozachenko. Sub-gaussian random variables. *Ukrainian Mathematical Journal*, 32(6):483–489, 1980.

Tristan Cazenave and Nicolas Jouandeau. On the parallelization of uct. In *proceedings of the Computer Games Workshop*, pp. 93–101. Citeseer, 2007.

Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. Parallel monte-carlo tree search. In *International Conference on Computers and Games*, pp. 60–71. Springer, 2008.

Adrien Couëtoux, Martin Müller, and Olivier Teytaud. Monte carlo tree search in go, 2017.

Sam Devlin, Anastasija Anspoka, Nick Sephton, Peter I Cowling, and Jeff Rollason. Combining gameplay data with monte carlo tree search to emulate human play. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

Zohar Feldman and Carmel Domshlak. On mabs and separation of concerns in monte-carlo planning for mdps. In *ICAPS*, 2014a.

Zohar Feldman and Carmel Domshlak. Simple regret optimization in online planning for markov decision processes. *Journal of Artificial Intelligence Research*, 51:165–205, 2014b.

Sylvain Gelly and Yizao Wang. Exploration exploitation in go: Uct for monte-carlo go. In *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*, 2006.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pp. 3338–3346, 2014.

Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. *arXiv preprint arXiv:1408.2048*, 2014.

Eshcar Hillel, Zohar S Karnin, Tomer Koren, Ronny Lempel, and Oren Somekh. Distributed exploration in multi-armed bandits. In *Advances in Neural Information Processing Systems*, pp. 854–862, 2013.

Hideki Kato and Ikuo Takeuchi. Parallel monte-carlo tree search with simulation servers. In *2010 International Conference on Technologies and Applications of Artificial Intelligence*, pp. 491–498. IEEE, 2010.

Emilie Kaufmann and Wouter M Koolen. Monte-carlo tree search by best arm identification. In *Advances in Neural Information Processing Systems*, pp. 4897–4906, 2017.

Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On bayesian upper confidence bounds for bandit problems. In *Artificial intelligence and statistics*, 2012.

Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1, 2006.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

Anji Liu, Jianshu Chen, Mingze Yu, Yu Zhai, Xuewen Zhou, and Ji Liu. Watch the unobserved: A simple approach to parallelizing monte carlo tree search. In *International Conference on Learning Representations*, April 2020. URL https://openreview.net/forum?id=BJlQtJSKDB.

Keqin Liu and Qing Zhao. Distributed learning in multi-armed bandit with multiple players. *IEEE Transactions on Signal Processing*, 58(11):5667–5681, 2010.

Yun-Ching Liu and Yoshimasa Tsuruoka. 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. IEEE, 2015.

David Martínez-Rubio, Varun Kanade, and Patrick Rebeschini. Decentralized cooperative stochastic bandits. In *Advances in Neural Information Processing Systems*, pp. 4531–4542, 2019.

Eric Mazumdar, Roy Dong, Vicenç Rúbies Royo, Claire Tomlin, and S Shankar Sastry. A multi-armed bandit approach for online expert selection in markov decision processes. *arXiv preprint arXiv:1707.05714*, 2017.

S Ali Mirsoleimani, Aske Plaat, H Jaap van den Herik, and Jos Vermaseren. An analysis of virtual loss in parallel mcts. In *ICAART (2)*, pp. 648–652, 2017.

Tom Pepels, Tristan Cazenave, Mark HM Winands, and Marc Lanctot. Minimizing simple and cumulative regret in monte-carlo tree search. In *Workshop on Computer Games*, pp. 1–15. Springer, 2014.

Erik Ragnar Poromaa. Crushing candy crush: predicting human success rate in a mobile game using monte-carlo tree search, 2017.

Edward J Powley, Daniel Whitehouse, and Peter I Cowling. Determinization in monte-carlo tree search for the card game dou di zhu. *Proc. Artif. Intell. Simul. Behav*, pp. 17–24, 2011.

Jan Schäfer, Michael Buro, and Knut Hartmann. The uct algorithm applied to games with imperfect information. *Diploma, Otto-Von-Guericke Univ. Magdeburg, Magdeburg, Germany*, 11, 2008.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard B Segal. On the scalability of parallel uct. In *International Conference on Computers and Games*, pp. 36–47. Springer, 2010.

Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. In *Advances in Neural Information Processing Systems*, pp. 6786–6797, 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Yusuke Soejima, Akihiro Kishimoto, and Osamu Watanabe. Evaluating root parallelization in go. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):278–287, 2010.

David Tolpin and Solomon Eyal Shimony. Mcts based on simple regret. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Kazuki Yoshizoe, Akihiro Kishimoto, Tomoyuki Kaneko, Haruhiro Yoshimoto, and Yutaka Ishikawa. Scalable distributed monte-carlo tree search. In *Fourth Annual Symposium on Combinatorial Search*, 2011.