ON EFFECTIVE PARALLELIZATION OF MONTE CARLO TREE SEARCH

Anji Liu[†], **Yitao Liang**[†], **Ji Liu**[‡], **Guy Van den Broeck**[†] **& Jianshu Chen**[§]
[†]Department of Computer Science, University of California, Los Angeles
{liuanji, yliang, guyvdb}@cs.ucla.edu
[‡]Seattle AI Lab, Kwai Inc., Bellevue, WA 98004, USA
jiliu@kuaishou.com
[§]Tencent AI Lab, Bellevue, WA 98004, USA
jianshuchen@tencent.com

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 the necessary conditions of 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, for example tracking the unobserved samples (used in WU-UCT (Liu et al., 2020)). We theoretically establish this essential design facilitates $\mathcal{O}(\ln n + M/\sqrt{\ln n})$ cumulative regret when the maximum tree depth is 2, 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, outperforms 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 tracking the unobserved samples, which is proposed in 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

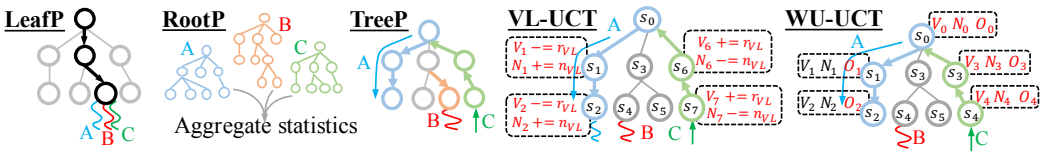


Figure 1: Typical existing parallel MCTS algorithms. VL-UCT and WU-UCT use virtual loss (i.e., r_{VL}) and number of on-going simulations (i.e., O) to pre-adjust node statistics, respectively.

to independently maintain different search trees, and the statistics are aggregated after all workers 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); for both $\widetilde{Q}$ and $\widetilde{N}$) and incomplete visit count (Liu et al. (2020); for $\widetilde{N}$). 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).

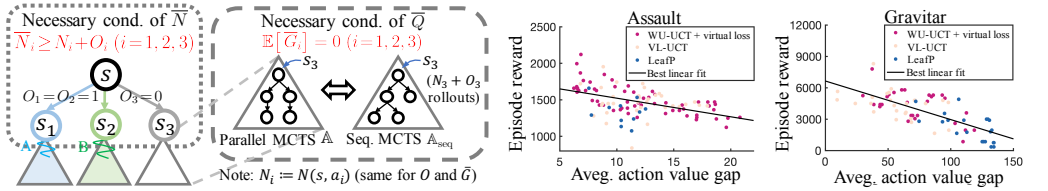
The two necessary conditions are best illustrated in Fig. 2(a). Consider node s in a search tree where we want to select one of its child nodes. Workers A and B are in their simulation steps, querying an offspring node of s_1 and s_2, respectively. To introduce the necessary condition of $\overline{N}$, 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 high incomplete visit count will

[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.

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. Note that VL-UCT with virtual loss (Segal, 2010; Silver et al., 2016) is the first to consider pre-adjusting action values and visit counts to improve the exploration-exploitation tradeoff in MCTS. We refer readers interested in the difference between VL-UCT and WU-UCT to Appendix A.1.

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*. We found that none of them satisfy the necessary condition on $\overline{Q}$ and only WU-UCT satisfies the necessary condition of $\overline{N}$. In hindsight, this implies that that the design of $\overline{N}$ in WU-UCT is 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. Comparing 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

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

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

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.

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):

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

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 $\mathbb{A}_{\mathrm{seq}}$ (i.e., $\mathrm{Regret}_{\mathbb{A}}(n) - \mathrm{Regret}_{\mathbb{A}_{\mathrm{seq}}}(n)$). 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 work (e.g., Hay et al. (2014); Tolpin & Shimony (2012)), simple regret and cumulative regret are both useful metrics that have a strong correlation with MCTS algorithms' performance, hence both are worth studying in the context of MCTS. In this paper, we are in particular focused 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 $\overline{Q}$:*** $\overline{G}(s,a) := \big|\mathbb{E}[\overline{Q}(s,a)] - \mathbb{E}[Q_m^{\mathbb{A}_{\mathrm{seq}}}(s,a)]\big| = 0 \ \big(m = N(s,a) + O(s,a)\big)$, (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.

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

[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.

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 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. Next, regarding $\overline{N}$, we found that WU-UCT satisfies its necessary condition by using the sum of complete and incomplete visit count as its modified visit count (i.e., $\overline{N}(s,a) := N(s,a)+O(s,a)$). Now a natural question to ask is *whether satisfying the necessary condition of $\overline{N}$ offers noticeable gain in WU-UCT's performance*, 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:*

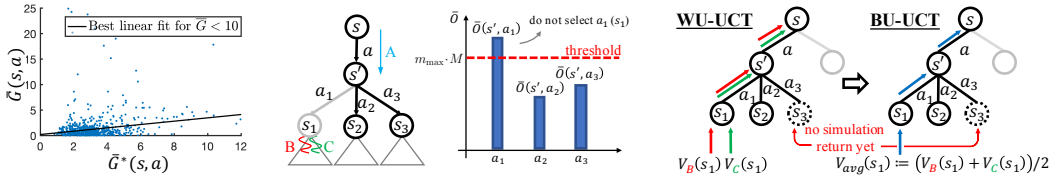
$$\underbrace{\sum_{k:\mu_k<\mu^*} \left(\frac{8}{\Delta_k} + 2\Delta_k\right) \ln n + \Delta_k}_{R_{\text{UCT}}(n)} + \underbrace{4M \sum_{k:\mu_k<\mu^*} \frac{\Delta_k^2}{\sqrt{\ln n}}}_{\text{excess regret}},$$

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. 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 $\overline{N}$ would be beneficial. This left us with the question *how to make use of the necessary condition of $\overline{Q}$ to further improve existing parallel MCTS algorithms*, which will be addressed in the following section.



(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.

5 Theory in Practice: A Promising Study

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

[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.

policy. Concretely, this is achieved with the following modified action value $\overline{Q}$:

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

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 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 are considered as 1 (i.e., $\forall a \in \mathcal{A}\ \ \overline{N}(s', a) = 1$) 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.

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

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

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.

Supplementary Material

In this supplementary material, we first give a more detailed review of existing parallel MCTS algorithms in Appendix A. We then formally introduce the general algorithm framework for parallel MCTS algorithms (see Figure 4 as well as Algorithm 1) in Appendix B, especially showing how existing algorithms fall into our general framework (Appendix B.3). Then we provide the detailed proofs of the parallel algorithms in Appendix C. The supplementary ends up with more details on the proposed algorithm BU-UCT (Appendix D), the surrogate statistics introduced in Section 5 (Appendix E), and additional details for the Atari experiments (Appendix F).

A Existing parallel MCTS algorithms

Leaf Parallelization (LeafP) (Cazenave & Jouandeau, 2007), Root Parallelization (RootP) (Cazenave & Jouandeau, 2007), and Tree Parallelization (TreeP) (Chaslot et al., 2008) develop different ways to cooperate among the workers.[8] As shown in Figure 1, ***LeafP*** and ***RootP*** parallelize MCTS from the leaf nodes and the root node, respectively. Specifically, in LeafP, only a main process performs sequential rollouts. However, during the simulation step, M workers simultaneously query the same node choosed in the selection and expansion steps, and after all simulations complete, the simulation returns are backpropagated to update node statistics along the selected path. In RootP, M workers independently run sequential MCTS and maintain different search trees, each with a predefined rollout budget. After all workers complete their jobs, the statistics are aggregated to make the final decision (i.e. which action to take). On the other hand, in TreeP, the workers independently perform sequential rollouts on a shared search tree. Node statistics updated by any worker are immediately observable by other workers.

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 before the respective simulation tasks are initiated. As shown in Figure 1, to encourage different workers to explore different nodes, VL-UCT penalizes the action value (i.e., $\overline{Q}$) of nodes that are currently being simulated by some workers so that other workers tend not to query this same set of nodes. Specifically, it has the following two variants. The *hard penalty* version (Chaslot et al., 2008) adds fixed virtual rewards r_{VL} directly to the average return and uses the following expression in the tree policy:

$$\overline{Q}(s,a) := Q(s,a) - O(s,a)\cdot r_{\text{VL}}. \quad \text{(VL-UCT hard penalty)}$$

Instead of directly penalizing $\overline{Q}$, when a node is being simulated by a worker, the *soft penalty* version (Silver et al., 2016) adds n_{VL} virtual simulation returns each with reward $-r_{\text{VL}}$:

$$\overline{Q}(s,a) := \frac{Q(s,a)\cdot N(s,a) - r_{\text{VL}}\cdot n_{\text{VL}}\cdot O(s,a)}{N(s,a) + n_{\text{VL}}\cdot O(s,a)},$$
$$\overline{N}(s,a) := N(s,a) + n_{\text{VL}}\cdot O(s,a). \quad \text{(VL-UCT soft penalty)}$$

Intuitively, the hard version of VL-UCT aggressively encourages different workers to explore different nodes, while the soft version has diminishing effect as the visit count grows to infinity.

In anticipation that the confidence in our estimates of $Q(s,a)$ will eventually increase if some child nodes of (s,a) are currently being simulated, Liu et al. (2020) proposes to only adjust the visit count $\overline{N}$:

$$\overline{Q}(s,a) := Q(s,a),$$
$$\overline{N}(s,a) := N(s,a) + O(s,a). \quad \text{(WU-UCT)}$$

A.1 VL-UCT and WU-UCT

As illustrated above, WU-UCT is similar to VL-UCT as they adjust the visit count (i.e., $\overline{N}$) in similar ways. Specifically, when $n_{VL}=1$, both methods adjust the visit count in the same way. However,

[8] LeafP and RootP are originally called "single-run" parallelization (Cazenave & Jouandeau, 2007).

note that the methods they adjust action values (i.e., $\overline{Q}$) is different — VL-UCT uses the adjusted visit count (i.e., $\overline{N}$) as well as a hyperparameter r_{VL} to adjust $\overline{Q}$ while WU-UCT proposes to keep $\overline{Q}$ unchanged. (Note that for VL-UCT $\overline{Q}(s,a)$ is always different from $Q(s,a)$ unless $n_{VL}=0$ and $r_{VL}=0$, which is equivalent to not adding virtual loss.)

B A General Framework for Parallel MCTS Algorithms

This section formally introduce a general algorithm framework for parallel MCTS algorithms, which is a critical component of Theorem 1. Specifically, since the necessary conditions stated in Theorem 1 only apply to algorithms that are specialized from the general algorithm framework, it is important that the framework covers all major existing parallel MCTS algorithms and their variants. In the following, we first introduce the general framework for parallel MCTS (Appendix B.1) and provide additional details (Appendix B.2). Appendix B.3 then explains how existing approaches fit in the general algorithm framework.

B.1 Formal Introduction of The General Algorithm Framework

In the following, we first provide an overview of the general framework for parallel MCTS algorithms, highlighting its two key modules, *statistics collection* and *statistics augmentation*, which allow it to represent various existing methods. We then discuss both modules in detail.

Overview The general framework consists of a master process and M simulator processes. Simulators perform simulations and return the outcomes (i.e. $V(s)$) back to the master. All simulators communicate *only* with the master and perform one simulation at a time. M search trees $\{\mathcal{T}_m\}_{m=1}^{M}$ are maintained to mimic M distinct sets of statistics stored in existing algorithms. For example, in RootP (Figure 1), each of the M workers maintain a search tree locally with different statistics, which can be represented by the M search trees in the general algorithm framework, respectively. As illustrated by the block diagram in Figure 4, the master performs *rollouts* repeatedly to gradually build the M search trees and the statistics in them.[9] During this process, *statistics collection* and *statistics augmentation* are two crucial modules in the rollout process that make the general framework flexible enough to represent various algorithms. Specifically, statistics collection consists of the *tree selection*, *simulation*, and *tree sync* steps, which characterize how the master employs the simulators to obtain simulation results and use them to update the M search trees. Statistics augmentation includes the *pseudo statistics pre-update* and *backpropagation* steps, both aiming to improve node statistics in individual search trees with additional side information to achieve better exploration-exploitation tradeoff during *node selection*.

We briefly go through the rollout process, where the important steps will be further discussed later. In Figure 4, at the beginning of each rollout, a search tree $\mathcal{T}_m$ is selected using the function f_{sel} in the *tree selection* step. Then, during *node selection*, $\mathcal{T}_m$ is traversed using a modified tree policy, where a set of modified statistics ($\overline{Q}_m$ and $\overline{N}_m$) are adopted. The modified statistics are defined as follows:

$$\overline{Q}_m(s,a) := \alpha_m(s,a)Q_m(s,a) + \beta_m(s,a)\widetilde{Q}_m(s,a), \tag{12}$$

$$\overline{N}_m(s,a) := N_m(s,a) + \widetilde{N}_m(s,a), \tag{13}$$

where Q_m and N_m are the original statistics used in the sequential MCTS algorithm (Eq. (1)); $\widetilde{Q}_m$ and $\widetilde{N}_m$ are a set of pseudo statistics that incorporate additional side information; α_m and β_m controls the ratio between Q_m and $\widetilde{Q}_m$. Note that Eqs. (12) and (13) resemble Eq. (4) in the main text. Then, after *expanding* a new node in a similar manner to the sequential algorithm, the *pseudo statistics pre-update* step[10] adjusts the pseudo statistics using the functions $f_{\widetilde{Q}}$ and $f_{\widetilde{N}}$. Afterwards, it assigns the *simulation* task to an idle simulator. Rollouts are started over again here unless all simulators are occupied or have completed task not yet processed by the master. Otherwise, the master *waits* for a completed simulation result and performs *backpropagation*, which consists of the traditional update (i.e. Eqs. (2)-(3)) and a *pseudo statistics post-update* step. In the post-update step, pseudo statistics

[9] A *rollout* represents the process of executing *all* steps in the block diagram illustrated in Figure 4 once, while a *simulation* refers to a step in the rollout process that queries a node's value (i.e. $V(s)$).

[10] The statistics $O_m(s,a)$ in Figure 4 will be introduced in the "statistics augmentation" paragraph.

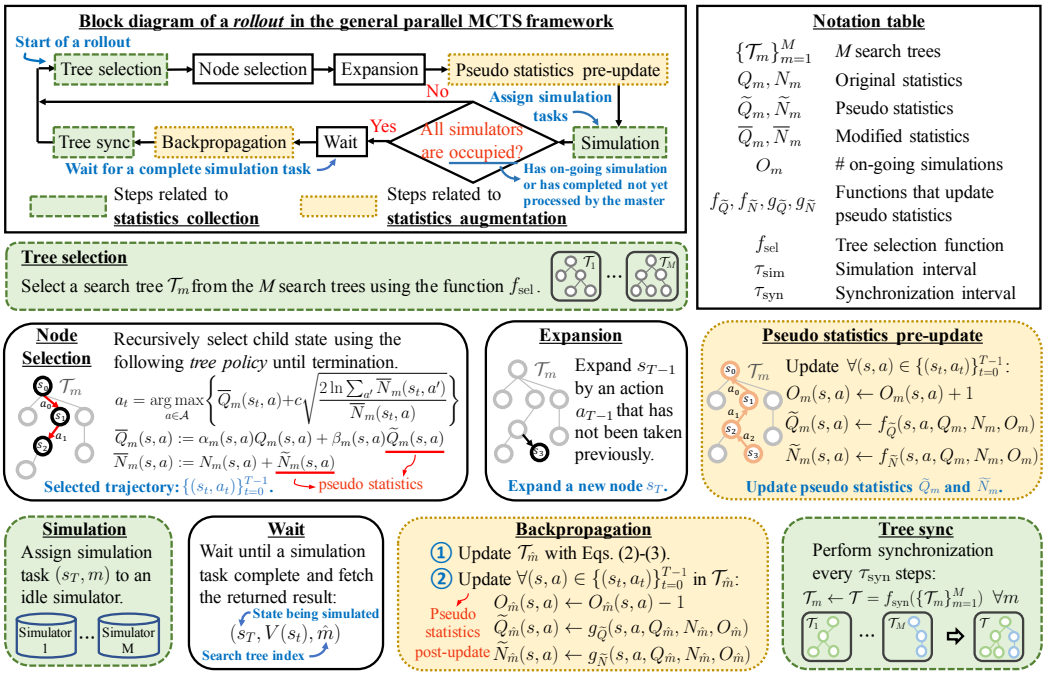


Figure 4: The proposed general framework that covers existing parallel MCTS algorithms. The overall diagram is on the top-left, the notation is on the top-right, and the details are in other boxes.

are adjusted with $g_{\widetilde{Q}}$ and $g_{\widetilde{N}}$. Finally, information from different search trees are synchronized every τ_{syn} rollouts, where τ_{syn} is defined as the *synchronization interval*.

Statistics collection By choosing different f_{sel} and τ_{syn} (in the tree selection and tree sync step, respectively), the simulators cooperate in different collaboration models that appear in various existing algorithms. Specifically, if tree sync happens in all rollout steps (i.e. $\tau_{\text{syn}} = 1$), then the M search trees are always identical during node selection, which can be regarded as M workers performing sequential rollouts (Section 2) on a shared search tree, representing TreeP. On the other hand, when having no communication between the search trees until finishing the last rollout (i.e. $\tau_{\text{syn}} = n_{\max}$, the total number of rollouts), and letting f_{sel} choose the search tree that is updated in the backpropagation step of the previous rollout (i.e. $\mathcal{T}_{\hat{m}}$), then the M search trees can be regarded as search trees maintained by M independent sequential MCTS algorithms, which resembles RootP.[1]

Statistics augmentation Statistics augmentation focuses on using extra side information besides the simulation returns to improve the tree policy. Specifically, besides the statistics extracted from completed simulations (i.e. Q_m and N_m), the general framework also uses pseudo statistics (i.e. $\widetilde{Q}_m$ and $\widetilde{N}_m$) to incorporate information from on-going simulations into its tree policy. Central to the pseudo statistics is the *incomplete sample count* O_m that keeps track of the number of initiated but not yet completed simulations for each node Liu et al. (2020). It is used to construct pseudo statistics through the *pseudo statistics pre-/post-update* steps. For example, by choosing $\alpha_m(s,a) = 1$, $\beta_m(s,a) = 0$, $f_{\widetilde{N}}(s,a) = g_{\widetilde{N}}(s,a) = O_m(s,a)$, the general framework is specialized into WU-UCT.[1]

Finally, Table 2 summarizes how different hyperparameter choices in the general parallel MCTS framework specialize the general algorithm to various existing parallel MCTS algorithms. Please refer to Appendix B.3 for rigorous justifications for such specializations.

B.2 Additional Details of the General Algorithm Framework

In this subsection, we provide additional details for the general framework of parallel MCTS algorithm. Specifically, we introduce the general framework using Algorithm 1, highlighting details that are not

[1] The functions α, β, f_{sel}, $f_{\widetilde{Q}}$, $f_{\widetilde{N}}$, $g_{\widetilde{Q}}$, $g_{\widetilde{N}}$ and the synchronization interval τ_{syn} are hyperparameters. When set differently, the algorithm can be specialized to different parallel MCTS algorithms.

Table 2: Different choices of the parameters in the general parallel MCTS algorithm framework correspond to various existing parallel MCTS algorithms. N_m and O_m are abbreviation of $N_m(s,a)$ and $O_m(s,a)$, respectively. $n_{\max}$ is the total number of rollouts. r_{VL} and n_{VL} are hyperparameters specific to VL-UCT. m' and $\hat{m}$ are the index of the search tree selected in the previous tree selection step and updated in the previous backpropagation step, respectively.

Algorithm	$f_{\mathrm{sel}}(m', \hat{m})$	τ_{syn}	$\alpha_m(s,a)$	$\beta_m(s,a)$	$\widetilde{Q}_m(s,a)$	$\widetilde{N}_m(s,a)$
UCT	1	1	1	0	0	0
LeafP	$(m'+1)\%M$	M	1	0	0	0
RootP	$\hat{m}$	$n_{\max}$	1	0	0	0
TreeP	$\mathrm{randint}(M)$	1	1	0	0	0
WU-UCT	$\mathrm{randint}(M)$	1	1	0	0	O_m
VL-UCT (hard)	$\mathrm{randint}(M)$	1	1	O_m	$-r_{\mathrm{VL}}$	0
VL-UCT (soft)	$\mathrm{randint}(M)$	1	$\frac{N_m}{N_m+n_{\mathrm{VL}}\cdot O_m}$	$\frac{n_{\mathrm{VL}}\cdot O_m}{N_m+n_{\mathrm{VL}}\cdot O_m}$	$-r_{\mathrm{VL}}$	$n_{\mathrm{VL}}\cdot O_m$

stated clearly enough in the main text. We proceed by introducing each of the steps shown in the block diagram in Figure 4.

Tree selection The tree selection function f_{sel} takes m' and $\hat{m}$ as input. According to Line 15, m' denotes the index of the search tree selected in the previous rollout. $\hat{m}$ is the index of the search tree being updated in the backpropagation step during the previous rollout (see Lines 10 and 11).

Node selection Note that the terminal conditions can be customized. Here we adopt a widely used set of terminal conditions: either the node contains unexpanded child nodes or its depth exceed $d_{\max}$.

Expansion Identical to the expansion step in sequential MCTS.

Pseudo statistics pre-update Although explicitly written here, $\widetilde{Q}_m$ and $\widetilde{N}_m$ may not need to be explicitly stored during implementation since this computation may be done during node selection.

Simulation The search tree index m is passed to the simulator as record. Recall that the M search trees maintained by the master respectively mimic the "search trees" maintained by the M workers in practical algorithms, the index m helps Algorithm 1 to mimic the activation of different "workers".

Wait Similarly, the search tree index $\hat{m}$ is returned so that the algorithm knows which search tree to update the statistics.

Backpropagation Additional to the updation of Q_m and N_m, pseudo-statistics are also updated.

Tree sync We provide a formal definition of the synchronization function f_{syn}. Note that the following descriptions are only for rigorous purpose, practical algorithms do not need to actually implement the following algorithm.

The input of f_{syn} is a set of M search trees $\{\mathcal{T}_m\}_{m=1}^M$ and the output is a synchronized search tree $\mathcal{T}$. Intuitively, f_{syn} performs union of the M individual search trees and aggregate their newly acquired statistics after the previous synchronization (see Algorithm 2). Therefore, it can be divided into two steps: *topology construction phase* and the *statistics aggregation phase*. The topology construction phase generates a new tree topology for $\mathcal{T}$ by taking the union of the topologies from $\{\mathcal{T}_m\}_{m=1}^M$. It can be implemented by the following steps. We begin with a search tree $\mathcal{T}$ with only one root node representing the initial state (i.e., the input s_0 in Algorithm 1). In addition, we initialize a set of node, $\mathcal{V}_{\mathrm{syn}}$, with the root node s_0. We then repeat the following steps until $\mathcal{V}_{\mathrm{syn}}$ is empty: (i) (randomly) take out an element s from $\mathcal{V}_{\mathrm{syn}}$ and delete it from $\mathcal{V}_{\mathrm{syn}}$, (ii) for all $a \in \mathcal{A}$, if the edge (s,a) exists in at least one of the M search trees $\{\mathcal{T}_m\}_{m=1}^M$, we grow the tree $\mathcal{T}$ by attaching this edge (s,a) along with its next state s' to node s, and (iii) add node s' to the set $\mathcal{V}_{\mathrm{syn}}$.

To explain the *statistics aggregation phase* of f_{syn} (i.e., the second phase), we have to introduce two sets of additional statistics associated with each edge (s,a) at the M input search trees $\{\mathcal{T}_m\}_{m=1}^M$. Specifically, for each edge (s,a) in the search tree $\mathcal{T}_m$, let $\mathcal{R}_m(s,a)$ be a set that consists of elements in the following form and is constructed in a recursive manner (to be explained later):

$$\mathcal{R}_m(s,a) = \{(V_{s,a}, \xi_{s,a}) : \ V_{s,a} := V(s'), \ \xi_{s,a} \in \{0,1\}\}, \tag{17}$$

Algorithm 1 A general framework of parallel MCTS algorithms.

input: Environment $\mathcal{M}$; number of simulator processes M; number of rollouts N; functions $\alpha_m(\forall m)$, $\beta_m(\forall m)$, f_{sel}, $f_{\widetilde{Q}}$, $f_{\widetilde{N}}$, $g_{\widetilde{Q}}$, $g_{\widetilde{N}}$; synchronization interval τ_{syn}; initial state s_0; maximum depth $d_{\max}$.[11]

initialize: number of completed simulations $n_{\text{complete}} \leftarrow 0$; search tree No. $m' \leftarrow M$ and $\hat{m} \leftarrow 1$; M search trees $\{\mathcal{T}_m\}_{m=1}^M$, each with node set $\mathcal{V}_m \leftarrow \{s_0\}$ and edge set $\mathcal{E}_m \leftarrow \emptyset$:

$$\mathcal{T}_m := \langle (\mathcal{V}_m, \mathcal{E}_m), \{Q_m, \widetilde{Q}_m, N_m, \widetilde{N}_m, O_m\} \rangle,$$

where the statistics $\{Q_m, \widetilde{Q}_m, N_m, \widetilde{N}_m, O_m\}$ are initialized to zero.

while $n_{\text{complete}} < N$ **do**

(Tree selection) Select a search tree $\mathcal{T}_m$ where $m = f_{\text{sel}}(m', \hat{m}) \in \{1, \dots, M\}$.

(Node selection) Traverse over $\mathcal{T}_m$ according to the following tree policy and collect a sequence of traversed state action pair $\{(s_t, a_t)\}_{t=0}^{T-1}$, where s_0 is the root node and s_{T-1} is the state that satisfies one of the following conditions: (i) it contains unexpanded child nodes, (ii) its depth exceed $d_{\max}$:

$$a_t = \arg\max_{a \in \mathcal{A}} \left\{ \overline{Q}_m(s_t, a) + c\sqrt{\frac{2\ln \sum_{a'} \overline{N}_m(s_t, a')}{\overline{N}_m(s_t, a)}} \right\}, \tag{14}$$

where the adjusted statistics $\overline{Q}_m$ and $\overline{N}_m$ are given by

$$\overline{Q}_m(s, a) := \alpha_m(s, a) Q_m(s, a) + \beta_m(s, a) \widetilde{Q}_m(s, a), \tag{15}$$

$$\overline{N}_m(s, a) := N_m(s, a) + \widetilde{N}_m(s, a). \tag{16}$$

(Expansion) Pick an expandable action a_{T-1} at s_{T-1} and add node s_T (the next state following (s_{T-1}, a_{T-1})) to tree $\mathcal{T}_m$.

(Pseudo statistics pre-update) *Pre-update* pseudo statistics for all $(s, a) \in \{(s_t, a_t)\}_{t=0}^{T-1}$:

$$O_m(s, a) \leftarrow O_m(s, a) + 1,$$

$$\widetilde{Q}_m(s, a) \leftarrow f_{\widetilde{Q}}(s, a, Q_m, N_m, O_m),$$

$$\widetilde{N}_m(s, a) \leftarrow f_{\widetilde{N}}(s, a, Q_m, N_m, O_m).$$

(Simulation) Assign simulation task (s_T, m) to a simulator process.

if there exist simulators without an assigned task **then continue**

(Wait) Wait until a simulation task completes and fetch the simulation return $(s_T, V(s_T), \hat{m})$.

(Backpropagation) Update $Q_{\hat{m}}$ and $N_{\hat{m}}$ in the search tree $\mathcal{T}_{\hat{m}}$ using the same rule as Eqs. (2) and (3); perform *pseudo-statistics post-update* on the search tree $\mathcal{T}_{\hat{m}}$ for all $(s, a) \in \{(s_t, a_t)\}_{t=0}^{T-1}$:

$$O_{\hat{m}}(s, a) \leftarrow O_{\hat{m}}(s, a) - 1,$$

$$\widetilde{Q}_{\hat{m}}(s, a) \leftarrow g_{\widetilde{Q}}(s, a, Q_{\hat{m}}, N_{\hat{m}}, O_{\hat{m}}),$$

$$\widetilde{N}_{\hat{m}}(s, a) \leftarrow g_{\widetilde{N}}(s, a, Q_{\hat{m}}, N_{\hat{m}}, O_{\hat{m}}).$$

if $n_{\text{complete}} \equiv \tau_{\text{syn}} - 1 \pmod{\tau_{\text{syn}}}$ **then**

(Tree sync) Synchronize the statistics in different search trees such that:

$$\mathcal{T}_m \leftarrow \mathcal{T} = f_{\text{syn}}(\{\mathcal{T}_m\}_{m=1}^M) \quad m = 1, \dots, M.$$

end if

$n_{\text{complete}} \leftarrow n_{\text{complete}} + 1$; $m' \leftarrow m$

end while

return $\mathcal{T} = f_{\text{syn}}(\{\mathcal{T}_m\}_{m=1}^M)$ (or return the "best" action for the initial state s_0)

where s' is the next state of (s, a), and $V(s')$ is recursively defined (over $\mathcal{T}_m$) according to Eq. (2) starting from the simulation return $V(s_T)$.[12] $\xi_{s,a} = 1$ means that $V_{s,a}$ at this edge (s, a) has been synchronized in the previous synchronization cycles and 0 otherwise. When an edge (s, a) is initialized (e.g., expanded), an empty set $\mathcal{R}_m(s, a)$ will be initialized accordingly. During the

[12] We drop the dependency on m in the above set of $\mathcal{R}_m(s, a)$ for simplicity of notation.

backpropagation phase of Algorithm 1, for each traversed edge corresponding to the complete simulation with return $(s_T, V(s_T), \hat{m})$ (assume the traversed edges are $\{(s_t, a_t)\}_{t=1}^{T-1}$), we update the sets $\mathcal{R}_{\hat{m}}(s_t, a_t)$ $(t = 0, \dots, T)$ by recursively computing $V(s_t)$ using Eq. (2) and add the element $(V(s_{t+1}), 0)$ into the set $\mathcal{R}_{\hat{m}}(s_t, a_t)$.

During the statistics aggregation phase, for each edge $(s, a) \in \mathcal{T}$, we perform the following steps to construct the set $\mathcal{R}(s, a)$: (i) initialize an empty set $\mathcal{R}(s, a)$, (ii) traverse all elements $(V_{s,a}, \xi_{s,a}) \in \mathcal{R}_1(s, a)$ and add it to $\mathcal{R}(s, a)$ if $\xi_{s,a} = 1$, (iii) traverse all elements $(V_{s,a}, \xi_{s,a}) \in \cup_{m=1}^{M} \mathcal{R}_m(s, a)$[13] and add $(V_{s,a}, 1)$ to $\mathcal{R}(s, a)$ if $\xi_{s,a} = 0$. The intuition of the above procedure is that both the synchronized elements ($\xi_{s,a} = 1$) and elements that have not been synchronized yet ($\xi_{s,a} = 0$) are added to $\mathcal{R}(s, a)$ only once. We then calculate the statistics Q and N at the output search tree $\mathcal{T}$ as follows:

$$Q(s, a) := \frac{1}{|\mathcal{R}(s, a)|} \sum_{\langle V, \xi \rangle \in \mathcal{R}(s,a)} V, \tag{18}$$

$$N(s, a) := |\mathcal{R}(s, a)|, \tag{19}$$

where $|\mathcal{R}(s, a)|$ denotes the cardinality of the set $\mathcal{R}(s, a)$. Finally, the synchronization of the on-going simulation count $O(s, a)$ is performed in the following manner: for each edge $(s, a) \in \mathcal{T}$,

$$O(s, a) \leftarrow \sum_{m=1}^{M} O_m(s, a), \tag{20}$$

where $O_m(s, a)$ is set to zero if this particular edge (s, a) does not appear in $\mathcal{T}_m$. The details for the implementation of f_{syn} are summarized in Algorithm 2.

B.3 Specialization of the General Framework into Existing Parallel MCTS Algorithms

In this subsection, we show how the existing algorithms introduced in Appendix A could be viewed as special cases of Algorithm 1. Table 2 demonstrates how different choices of the hyperparameters in Algorithm 1 could lead to different parallel algorithms. The functions $f_{\widetilde{Q}}$, $f_{\widetilde{N}}$, $g_{\widetilde{Q}}$, and $g_{\widetilde{N}}$ are omitted in Table 2 since they can be inferred from $\widetilde{Q}_m$ and $\widetilde{N}_m$. Note that for some methods the equivalence exists only when the simulation phase takes much more time than the other phases. Nevertheless, this holds in general (Liu et al., 2020; Chaslot et al., 2008) and therefore does not affect our analysis.

LeafP Consider the following identification in Algorithm 1: $f_{\text{sel}}(m', \hat{m}) := (m' + 1)\%M$, where $\%$ denotes the modulo operator, $\alpha_m(s, a) = 1$, and $\beta_m(s, a) = \widetilde{Q}_m(s, a) = \widetilde{N}_m(s, a) = 0$. If we further choose $\tau_{\text{syn}} = M$, Algorithm 1 will be equivalent to LeafP for the following reasons. First, since synchronization happens at time steps $\tau_{\text{syn}}, 2\tau_{\text{syn}}, \dots$ (i.e., $M, 2M, \dots$), the search trees $\{\mathcal{T}_m\}_{m=1}^{M}$ are identical at the end of these time steps. We now show that the algorithm status at the ends of the rollouts $M, 2M, \dots$ in Algorithm 1 is equivalent to the algorithm status of LeafP at the ends of the rollouts $1, 2, \dots$, respectively (note that in each rollout of LeafP, M simulation returns of the same node is acquired). Specifically, during the M rollouts in the general framework (i.e., Algorithm 1), each search tree is selected only once due to the specific setting of f_{sel} (i.e., sequentially select all search trees). Since the M trees are identical and the tree policy (Eq. (14)) is deterministic, each of the M rollouts will independently expand and simulate one unique search tree among the M trees at the *same leaf node position*, which keeps all the M trees having an identical topology. Finally, the synchronization step aggregates the M simulation returns into a single search tree. As a result, it becomes equivalent to having M workers to simulate the same node in the simulation step of LeafP. Figure 5 illustrates the above equivalence between LeafP and the general framework under this identification.

TreeP Consider the choice of $\alpha_m(s, a) = 1$ and $\beta_m(s, a) = \widetilde{Q}_m(s, a) = \widetilde{N}_m(s, a) = 0$, and let the synchronization be excuted at each rollout cycle in Algorithm 1 (i.e., $\tau_{\text{syn}} = 1$, also see Table 2). We now show that this resembles the TreeP algorithm. First, since synchronization happens at every rollout cycle, the M search trees are identical at the beginning of each rollout cycle in Algorithm 1,

[13] $\cup$ refers to the set union.

Algorithm 2 The synchronization function f_{syn}

input: M search trees $\{\mathcal{T}_m\}_{m=1}^M$.
initialize: a search trees $\mathcal{T} := \langle(\mathcal{V}, \mathcal{E}), \{Q, N\}\rangle$, where $\mathcal{V} \leftarrow \{s_0\}$ is the set of nodes, and $\mathcal{E} \leftarrow \emptyset$ is the set of edges (s_0 is the root node of $\mathcal{T}$).

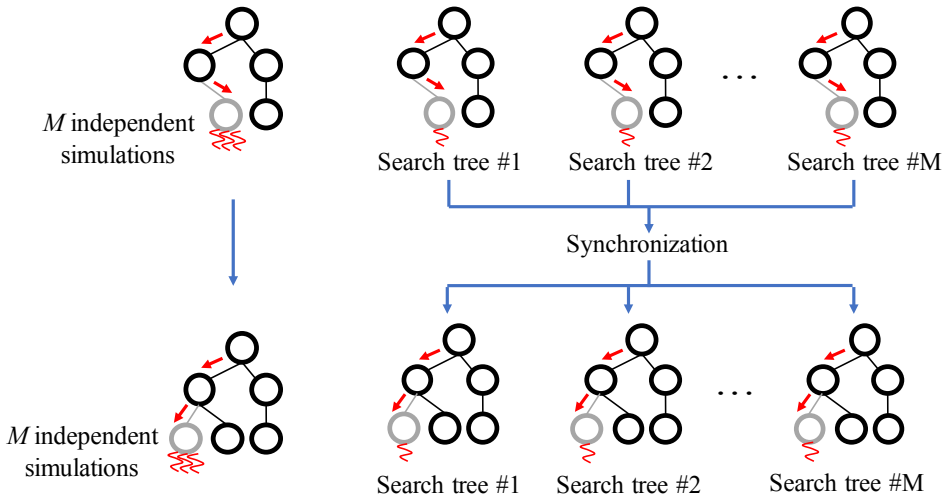
`# Phase 1: Topology construction`
Initialize $\mathcal{V}_{\text{syn}} := \{s_0\}$.
while $\mathcal{V}_{\text{syn}}$ not empty **do**
  $s \leftarrow \text{pop}(\mathcal{V}_{\text{syn}})$
  for $a \in \mathcal{A}$ **do**
    if (s, a) exists in at least one of the M search trees $\{\mathcal{T}_m\}_{m=1}^M$ **then**
      $s' \leftarrow$ the next state following (s, a)
      Add edge (s, a) and node s' to $\mathcal{T}$
      Add s' to the set $\mathcal{V}_{\text{syn}}$
    end if
  end for
end while

`# Phase 2: Statistics aggregation`
For all trees $\mathcal{T}_m$ and edges (s, a), define $\mathcal{R}_m(s, a)$ according to equation (17). $\mathcal{R}_m(s, a)$ is maintained during rollouts as described in Section B.2.
for all edges (s, a) in $\mathcal{T}$ **do**
  $\mathcal{R}(s, a) := \emptyset$
  for all $(V_{s,a}, \xi_{s,a}) \in \mathcal{R}_1(s, a)$ **do**
    if $\xi_{s,a} = 1$ **then**
      Add $(V_{s,a}, \xi_{s,a})$ to $\mathcal{R}(s, a)$
    end if
  end for
  for $m = 1, \ldots, M$ **do**
    for all $(V_{s,a}, \xi_{s,a}) \in \mathcal{R}_m(s, a)$ **do**
      if $\xi_{s,a} = 0$ **then**
        Add $(V_{s,a}, 1)$ to $\mathcal{R}(s, a)$
      end if
    end for
  end for
  Update $Q(s, a)$, $N(s, a)$, and $O(s, a)$ according to equations (18)-(20), respectively.
end for

return search tree $\mathcal{T}$

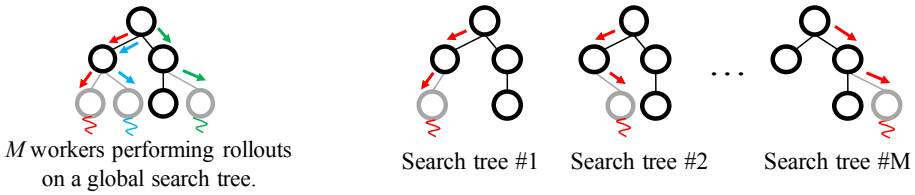
and can be regarded as a global search tree since all simulation returns are gathered immediately at the end of each rollout cycle (according to the definition of f_{syn}). Second, the simulator processes are independent, and whenever a simulator completes, its simulation return will be updated to the global search tree (in the backpropagation phase) by the synchronization step performed at every time step, which resembles TreeP. Finally, whenever a worker is idle, the algorithm will traverse the global search tree to assign a new simulation task to it, which mimics the setting in TreeP that each worker individually perform rollouts and update the global statistics. See Figure 6 for an illustration of the intuition for this equivalence.

RootP Consider the following choice of hyperparameters: $f_{\text{sel}}(m', \hat{m}) := \hat{m}$, (i.e., always select the search tree updated in the backpropagation step in the most recently completed rollout), $\alpha_m(s, a) = 1$, $\beta_m(s, a) = \widetilde{Q}_m(s, a) = \widetilde{N}_m(s, a) = 0$, and $\tau_{\text{syn}} = N_{\max}$ (i.e., synchronize after all the jobs at all the workers are totally completed). This setting is equivalent to RootP for the following reasons. First, since $\tau_{\text{syn}} = N_{\max}$, all the M search trees act independently (i.e., building their own search trees) and will not be aggregated by f_{syn} until all rollouts are completed. Second, we can show that the rollout cycles in Algorithm 1 will preserve the independence of the operations at these M search trees under the above identification. To see this, note that, by $f_{\text{sel}}(m, \hat{m}) := \hat{m}$, Algorithm 1 at the current rollout cycle will always select the search tree $\mathcal{T}_{\hat{m}}$ that has returned its simulation in the previous rollout cycle. This means that, in the current rollout cycle, Algorithm 1 will continue to perform rollouts and employ another worker to simulate this same search tree $\mathcal{T}_{\hat{m}}$. For this reason, it can be



(a) LeafP. (b) Equivalent LeafP by Algorithm 1.

Figure 5: Illustration of how LeafP can be viewed as a special case of Algorithm 1. In (b), each of the M trees initializes an identical simulation task to the simulator processes and synchronization happens after all M simulation tasks are completed. This is analogous to (a), where M workers are assigned to simulate a same node independently.



(a) TreeP. (b) Equivalent TreeP by Algorithm 1.

Figure 6: Illustration of how TreeP can be viewed as a special case of Algorithm 1. Performing M independent rollouts on M search trees and then synchronizing the statistics per rollout cycle ($\tau_{\text{syn}} = 1$) is equivalent to having M workers independently performing rollouts and updating the statistics on one global search tree in TreeP. This equivalence holds in general, regardless of whether virtual loss or pseudo-statistics are used. However, without them, the vanilla TreeP normally will quicly collapse into a mode that is similar to LeafP.

viewed as if we have M virtual "designated" workers to perform rollouts and simulations for these M search trees independently, which is exactly what RootP does. Since we assume other phases consume much less time than the simulation phase, these M virtual "designated" workers are almost bound to continuously performe rollouts and simulation process without long waits. Finally, different variants of RootP (e.g., certain workers only operate on some child nodes of the search tree) can also be modeled by Algorithm 1 by setting $\widetilde{Q}_m$ at these nodes. For instance, $\widetilde{Q}_m$ can be chosen to be big enough such that at the root node the algorithm will always choose these same child nodes. Figure 7 illustrates the equivalence between RootP and Algorithm 1 under the above identification.

VL-UCT Since it is a variant of TreeP, the workers' collaboration model in VL-UCT is identical to that of TreeP. Therefore, we can follow the same setting in $\tau_{\text{syn}}=1$ and f_{sel}. On the other hand, we choose the pseudo statistics as shown in Table 2. Specifically, for VL-UCT with hard penalty, we select (also see Table 2)

$$\alpha_m(s,a) = 1, \quad \beta_m(s,a) = O_m(s,a),$$
$$\widetilde{Q}_m(s,a) = -r_{\text{VL}}, \quad \widetilde{N}_m(s,a) = 0.$$

(a) RootP. (b) Equivalent RootP by Algorithm 1.

Figure 7: Illustration of how RootP could be viewed as a special case of Algorithm 1. Each subtree in RooP corresponds to one of the M search trees in Algorithm 1. Under a particular identification, Algorithm 1 can be viewed as having M virtual "designated" workers that operate independently on these M search trees, which is equivalent to what RootP does.

And for VL-UCT with soft penalty, we choose

$$\alpha_m(s,a) = \frac{N_m(s,a)}{N_m(s,a) + n_{\mathrm{VL}} \cdot O_m(s,a)},$$
$$\beta_m(s,a) = \frac{n_{\mathrm{VL}} \cdot O_m(s,a)}{N_m(s,a) + n_{\mathrm{VL}} \cdot O_m(s,a)},$$
$$\widetilde{Q}_m(s,a) = -r_{\mathrm{VL}},$$
$$\widetilde{N}_m(s,a) = n_{\mathrm{VL}} \cdot O_m(s,a).$$

WU-UCT Although not exactly based on TreeP, WU-UCT follows the same master-worker architecture as in Algorithm 1. We now show that WU-UCT can also be viewed as a special case of Algorithm 1 under the identification to be explained below. Similar to TreeP, we set $\tau_{\mathrm{syn}} = 1$, i.e., the statistics from the M search trees are synchronized at the end of each rollout cycle. Likewise, we set $f_{\mathrm{sel}}(m', \hat{m}) = \mathrm{randint}(M)$; that is, it selects a random search tree in the selection phase.[14] In addition, we make the following choices (see Table 2)

$$\alpha_m(s,a) = 1, \quad \beta_m(s,a) = \widetilde{Q}_m(s,a) = 0,$$
$$\widetilde{N}_m(s,a) = O_m(s,a).$$

C Proofs: Parallel Algorithms for Monte Carlo Tree Search

This section provides proofs for Theorems 1, and 2, which locate in Sections C.1 and C.2, respectively.

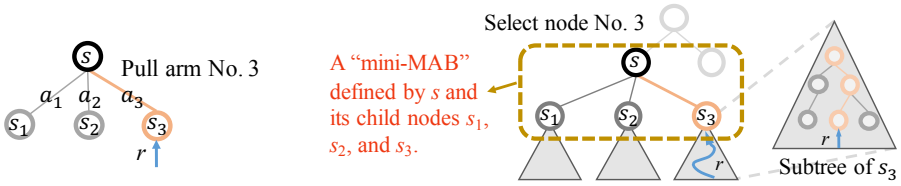
C.1 The Necessary Conditions

To help elaboration, we first introduce the following additional definitions. Define $\mathbb{A}_{\mathrm{seq}}$ as the sequential MCTS algorithm introduced in Section 2 (Kocsis et al., 2006). $\mathcal{T}^{\mathbb{A}}_{s,n}$ is defined as the search tree with root node s and is constructed by a (parallel) MCTS algorithm $\mathbb{A}$ with n rollouts. Whenever it is clear from context, we omit the subscript n for notation simplicity. Let $V^{\mathbb{A}}_{s,n}(s')$ be the cumulative reward $V(s')$ obtained in the backpropagation step (i.e. computed by Eq. (2)) when performing a rollout using algorithm $\mathbb{A}$ on the search tree $\mathcal{T}^{\mathbb{A}}_{s,n}$ (if s' is not selected during the rollout, $V^{\mathbb{A}}_{s,n}(s') := 0$). Note that $V^{\mathbb{A}}_{s,n}(s')$ is indeed a random variable due to the stochasticity in the simulation returns.

Following the above definitions as well as the terminology in the general algorithm framework (Appendix B.1), we give a formal version of Theorem 1.

Theorem 3 (A formal version of Theorem 1)**.** *Consider an algorithm $\mathbb{A}$ that is specified from the general parallel MCTS framework by choosing $\widetilde{N}_m(s,a) = f(O_m(s,a))$ $(m = 1, \ldots, M)$, where*

[14] In the original paper, WU-UCT also parallelizes the expansion step. However, since we assume the simulation phase is much more time-consuming then other phases, we ignore this detail.



(a) MAB with $K = 3$. (b) Equivalent mini-MAB in a search tree.

Figure 8: Demonstration of the mini-MABs in MCTS search trees that resembles a multi-armed bandit (MAB). (a): a MAB with three arms. (b): s, s_1, s_2, and s_3 define a *mini-MAB* that resembles the MAB in (a).

$f(\cdot) : \mathbb{Z}_0^+ \to \mathbb{R}$ is a function. If there exists an edge (s, a) in any of the M search trees $\{\mathcal{T}_m\}$ such that algorithm $\mathbb{A}$ violates any of the following conditions (with s' defined as the next state following (s, a)):

- ***Necessary cond. of $\overline{Q}$:*** $\mathbb{E}[\overline{Q}_m(s,a)] = \frac{1}{n}\sum_{n'=1}^{n}\mathbb{E}[V_{s',n'}^{\mathbb{A}_{\text{seq}}}(s')]$ *($n = N_m(s,a) + O_m(s,a)$),* (21)
- ***Necessary cond. of $\overline{N}$:*** $f(x) \geq x$ $(\forall x \in \mathbb{Z}_0^+)$, (22)

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

In the following, we provide the formal proof of Theorem 3, which states two necessary conditions for having vanishing excess regret in parallel MCTS algorithms. Before delving into the proof, we use Figure 8 to introduce the concept of *mini-MAB*. Specifically, in a search tree, each node and its child nodes represent a two-layer search tree that resembles a MAB with the same number of children. We define this two-layer search tree as mini-MAB. Note that one core difference between mini-MABs and MABs is that the reward acquired by a child node of mini-MABs are rewards obtained from a sub-tree rooted at the child node (Figure 8(b)), while for MAB all child nodes produce i.i.d. rewards following a pre-defined distribution.

Proof of Theorem 1 (Theorem 3. To obtain vanishing excess regret, it is necessary to show the following: *the excess regret of the mini-MABs that represent nodes on the optimal path in the search tree should decrease as t increases.* This necessary condition holds since all nodes on the optimal path will be visited $\Omega(t)$ times when t is sufficiently large (see Kocsis et al. (2006)), and if any of the nodes have nonvanishing excess regret, the tree search algorithm will suffer from nonvanishing regret. In the following, we derive the necessary conditions for algorithms that have vanishing excess regret in *mini-MABs*.

Consider a *mini-MAB* whose root node is s (assume it is on the optimal path). The actions are defined as $\{a_k\}_{k=1}^K$ and the next state following (s, a_k) is defined as s_k. In order to achieve vanishing excess regret during parallel, it is necessary to have vanishing excess regret when this mini-MAB is parallelized while rollouts its child nodes are performed sequentially. That is, assume we use the sequential algorithm $\mathbb{A}_{\text{seq}}$ to produce reward for all child nodes of the mini-MAB rooted at s. Correspondingly, we define $\mu_{k,n}$ as the expected reward obtained by executing action a_k for the nth time. That is,

$$\mu_{k,n} := \frac{1}{n}\sum_{n'=1}^{n}\mathbb{E}[V_{\tau_{s,a_k}(n')}^{\mathbb{A}_{\text{seq}}}(s')]. \tag{23}$$

Similarly, define $\mu_n^* := \max_k \mu_{k,n}$.

Define $\overline{Q}_{k,n,o}$ as the estimated value of the kth child node of the *mini-MAB* when there are n initialized simulations and o on-going simulations (which means that there are $n - o$ completed simulations). Formally, $\overline{Q}_{k,n,o}$ can be written as (reflects Eq. (13)/Eq. (4) in the general framework)

$$\overline{Q}_{k,n,o} := \alpha_k Q_{k,n-o} + \beta_k \widetilde{Q}_{k,n,o}, \tag{24}$$

where $\alpha_k := \alpha(s, a_k)$ and $\beta_k := \beta(s, a_k)$ (assume s as the root node of the *mini-MAB*); $\widetilde{Q}_{k,n,o} := \widetilde{Q}(s, a_k)$ is the pseudo value. Note that α_k and β_k might also depend on n and o.

We define $T_k(t)$ as the number of times action a_k is selected in the first t rollouts. According to the regret decomposition identity Mazumdar et al. (2017), $\mathrm{Regret}(t)$ can be decomposed with respect to different arms:

$$\mathrm{Regret}(t) = \sum_{k\in\{1,\dots,K\}, k\neq k^*} \Delta_k \mathbb{E}\left[T_k(t)\right]$$

where $\Delta_k := \max_{k'} \mathbb{E}[Q_t(s,a_{k'})] - \mathbb{E}[Q_t(s,a_k)]$ is the expected regret of selecting action a_k instead of the best action in the mini-MAB. Therefore, to achieve vanishing excess regret, it is necessary to show that the number of times a suboptimal action is chosen (i.e. $\mathbb{E}[T_k(t)]$) for a parallel MCTS algorithm should be the number of times such action is taken in sequential MCTS plus a term that vanishes as t goes to infinity.

Define $\overline{e}_{t,n,o} := \sqrt{(2\ln t)/(n+f(o))}$, where $f(\cdot)$ is defined in Theorem 3. We lower bound $T_k(t)$ by (k^* is the index of the optimal action)

$$\begin{aligned} T_k(t) &= \sum_{\tau=1}^{t} [\text{Action}_\tau = a_k] \\ &\geq \sum_{\tau=1}^{t} \min_{n,n'\in[0,\tau];o,o'\in[0,M-1]} \mathbb{1}\left[\overline{Q}_{k^*,n,o} + \overline{e}_{\tau,n,o} \leq \overline{Q}_{k,n',o'} + \overline{e}_{\tau,n',o'}\right] \end{aligned}$$

We lower bound the probability of the event $\overline{Q}_{k^*,n,o} + \overline{e}_{\tau,n,o} \leq \overline{Q}_{k,n',o'} + \overline{e}_{\tau,n',o'}$ using one minus the sum of the probability of the two following events:

$$\overline{Q}_{k^*,n,o} \geq \mu^*_n - \overline{e}_{\tau,n,o}, \tag{25}$$

$$\overline{Q}_{k,n',o'} \leq \mu_{k,n'} + \overline{e}_{\tau,n',o'}, \tag{26}$$

where $\mu_{k,n} := \frac{1}{n}\mathbb{E}[\sum_{t=1}^{n} Q_t(s,a_k)]$ is defined as the average value return of the first n times a_k is taken; μ^*_n denotes the same quantity defined for the optimal action $a^* := \arg\max_{a'} \mathbb{E}[\sum_{t=1}^{n} Q_t(s,a')]$. Note that this bound (i.e. Eqs. (25) and (26)) holds since by definition $\mu^*_{n-o} > \mu_{k,n-o}$.

We first give an outline of the proof regarding the necessary condition of $\overline{Q}$. We shall first show that if $\mathbb{E}[\overline{Q}_{k^*,n,O_{k^*,t}}] \geq \mu^*_n$ and $\mathbb{E}[\overline{Q}_{k,n',O_{k,t}}] \leq \mu_{k,n'}$ are not satisfied, there exists a mini-MAB task, n_0, and $p_\epsilon \in (0,1)$ such that for any $n > n_0$, the probability of both Eq. (25) and Eq. (26) are smaller than p_ϵ. Hence $T_k(t)$ will be lower bounded by $(1-2p_\epsilon)\cdot t$, meaning that the suboptimal arm k will be pulled $\Omega(t)$ times. Therefore, the algorithm cannot achieve vanishing excess regret. Next, given that (i) $\mathbb{E}[\overline{Q}_{k^*,n,O_{k^*,t}}] \geq \mu^*_n$ and $\mathbb{E}[\overline{Q}_{k,n',O_{k,t}}] \leq \mu_{k,n'}$ should be satisfied, and (ii) the algorithm does not know which arm is optimal (i.e., it cannot distinguish between k and k^*), the algorithm has to satisfy $\mathbb{E}[\overline{Q}_{k^*,n,O_{k^*,t}}] = \mu^*_n$ and $\mathbb{E}[\overline{Q}_{k,n',O_{k,t}}] = \mu_{k,n'}$, which gives the necessary condition of $\overline{Q}$. Details are provided as follows.

Define μ_{k,n_1,n_2} as the average reward of the kth arm of the *mini-MAB* from its n_1th rollout to its n_2th rollout ($n_1 \leq n_2$). Similarly $\mu^*_{n_1,n_2}$ defines the same quantity for the optimal arm k^*. We have the following results (for any integer $0 \leq o \leq n$):

$$\mu^*_n = \frac{n-o}{n}\mu^*_{n-o} + \frac{o}{n}\mu^*_{n-o+1,n}, \tag{27}$$

$$\mu_{k,n} = \frac{n-o}{n}\mu_{k,n-o} + \frac{o}{n}\mu_{k,n-o+1,n}. \tag{28}$$

Using the above results, Eqs. (25) and (26) can be equivalently written as

$$\alpha_{k^*}\left(Q_{k^*,n-o} - \mu^*_{n-o}\right) + \beta_{k^*}\widetilde{Q}_{k^*,n,o} + \alpha_{k^*}\mu^*_{n-o} - \mu^*_n \geq -\overline{e}_{\tau,n,o}, \tag{29}$$

$$\alpha_k\left(Q_{k,n'-o'} - \mu_{k,n'-o'}\right) + \beta_k\widetilde{Q}_{k,n',o'} + \alpha_k\mu_{k,n'-o'} - \mu_{k,n'} \leq \overline{e}_{\tau,n',o'}, \tag{30}$$

where ϕ_k is a variable that depend on k, n, and o. By definition, we have $\mathbb{E}[Q_{k^*,n-o}] = \mu^*_{n-o}$ and $\mathbb{E}[Q_{k,n'-o'}] = \mu_{k,n'-o'}$. We then focus on the following terms in the above equations:

$$\beta_{k^*}\widetilde{Q}_{k^*,n,o} + \alpha_{k^*}\mu^*_{n-o} - \mu^*_n, \tag{31}$$

$$\beta_k \widetilde{Q}_{k,n',o'} + \alpha_k \mu_{k,n'-o'} - \mu_{k,n'}. \tag{32}$$

We show that vanishing excess regret cannot be achieved unless Eqs. (31) and (32) have ≤ 0 and ≥ 0 expectation value, respectively. Otherwise, there exists n_0 such that for any $n > n_0$ and $o < M$ (by definition), $\overline{e}_{\tau,n,o}$ and $\overline{e}_{\tau,n',o'}$ will have smaller absolute value than Eqs. (31) and (32), respectively. For Eq. (30), this means that when $n > n_0$ its left-hand side has higher expectation value than its right-hand side, which means there exists $p_\epsilon \in (0,1)$ such that the probability of Eq. (30) is smaller than p_ϵ. This argument similarly applies to Eq. (29). As mentioned before, Eqs. (29) and (30) have probability upper bound means the suboptimal arm k will be pulled $\Omega(t)$ times, which makes the parallel MCTS algorithm fail to achieve vanishing excess regret.

Given that the parallel MCTS algorithm belongs to the general framework (Algorithm 1), there are three types of pseudo statistics that can be added to $\widetilde{Q}$, which are (i) statistics related to all complete simulations, (ii) statistics related to all incomplete simulations, and (iii) statistics non-related to simulation returns. Given this, we decompose the pseudo value $\widetilde{Q}_{k,n,o}$ into three terms:

$$\widetilde{Q}_{k,n,o} := \widetilde{Q}_{k,n,o}^{\mu_{k,n-o}} + \phi_k \cdot \widetilde{Q}_{k,n,o}^{\mu_{k,n-o+1,n}} + \widetilde{Q}_{k,n,o}^{R},$$

where $\mathbb{E}[\widetilde{Q}_{k,n,o}^{\mu_{k,n-o}}] = \mu_{k,n-o}$, $\mathbb{E}[\widetilde{Q}_{k,n,o}^{\mu_{k,n-o+1,n}}] = \mu_{k,n-o+1,n}$, and $\widetilde{Q}_{k,n,o}^{R}$ is independent of both $\mu_{k,n-o}$ and $\mu_{k,n-o+1,n}$. For the optimal arm k^* (the following holds for other arms as well), we have

$$\begin{aligned}
&\beta_{k^*}\widetilde{Q}_{k^*,n,o} + \alpha_{k^*}\mu^*_{n-o} - \mu^*_n \\
\overset{(a)}{=}&\beta_{k^*}(\widetilde{Q}_{k^*,n,o}^{\mu_{k^*,n-o}} + \phi_k \cdot \widetilde{Q}_{k^*,n,o}^{\mu_{k^*,n-o+1,n}} + \widetilde{Q}_{k^*,n,o}^{R}) + (\alpha_{k^*} - \frac{n-o}{n})\mu^*_{n-o} - \frac{o}{n}\mu^*_{n-o+1,n} \\
=&\left(\beta_{k^*}\widetilde{Q}_{k^*,n,o}^{\mu_{k^*,n-o}} + (\alpha_{k^*} - \frac{n-o}{n})\mu^*_{n-o}\right) + \left(\beta_{k^*}\phi_{k^*}\widetilde{Q}_{k^*,n,o}^{\mu_{k^*,n-o+1,n}} - \frac{o}{n}\mu^*_{n-o+1,n}\right) + \widetilde{Q}_{k^*,n,o}^{R},
\end{aligned} \tag{33}$$

where (a) uses the result of Eq. (27). The expectation value of Eq. (33) is

$$\left(\beta_{k^*} + \alpha_{k^*} - \frac{n-o}{n}\right)\mu^*_{n-o} + \left(\beta_{k^*}\phi_{k^*} - \frac{o}{n}\right)\mu^*_{n-o+1,n} + \mathbb{E}[\widetilde{Q}^R_{k^*,n,o}]. \tag{34}$$

Similarly, for arm k we have

$$\begin{aligned}
&\beta_k\widetilde{Q}_{k,n',o'} + \alpha_k\mu_{k,n'-o'} - \mu_{k,n'} \\
\overset{(a)}{=}&\beta_k(\widetilde{Q}_{k,n',o'}^{\mu_{k,n'-o'}} + \phi_k \cdot \widetilde{Q}_{k,n',o'}^{\mu_{k,n'-o'+1,n'}} + \widetilde{Q}_{k,n',o'}^{R}) + (\alpha_k - \frac{n'-o'}{n'})\mu_{k,n'-o'} - \frac{o'}{n'}\mu_{k,n'-o'+1,n'} \\
=&\left(\beta_k\widetilde{Q}_{k,n',o'}^{\mu_{k,n'-o'}} + (\alpha_k - \frac{n'-o'}{n'})\mu_{k,n'-o'}\right) + \left(\beta_k\phi_k\widetilde{Q}_{k,n',o'}^{\mu_{k,n'-o'+1,n'}} - \frac{o'}{n'}\mu_{k,n'-o'+1,n'}\right) + \widetilde{Q}_{k,n',o'}^{R},
\end{aligned} \tag{35}$$

and its expectation value is

$$\left(\beta_k + \alpha_k - \frac{n-o}{n}\right)\mu_{k,n-o} + \left(\beta_k\phi_k - \frac{o}{n}\right)\mu_{k,n-o+1,n} + \mathbb{E}[\widetilde{Q}^R_{k,n,o}]. \tag{36}$$

We now argue that in order for the ***mini-MAB*** to achieve vanishing excess regret, the expectation of the three terms in the above equation should all be 0. Specifically, previously we have shown that Eqs. (31) and (32) should have ≤ 0 and ≥ 0 expected values. However, as suggested by Eqs. (34) and (36), since the algorithm does not know which arm is optimal, and both equations have the same form, it is impossible to have Eq. (34) < 0 while Eq. (36) > 0. Hence, both equations should be equal to zero. Since μ^*_{n-o} and $\mu^*_{n-o+1,n}$ ($\mu_{k,n-o}$ and $\mu_{k,n-o+1,n}$) are task-specific, to make Eqs. (34) and (36) equals to zero, we should have ($\forall k$):

$$\begin{aligned}
\beta_k + \alpha_k - \frac{n-o}{n} &= 0, \\
\beta_k\phi_k - \frac{o}{n} &= 0, \\
\mathbb{E}[\widetilde{Q}^R_{k,n,o}] &= 0.
\end{aligned}$$

Plug in the above results into Eq. (24), we conclude that one necessary condition for having vanishing excess regret in the *mini-MAB* is ($\forall k$)

$$\begin{aligned}\mathbb{E}[\overline{Q}_{k,n,o}] &= \alpha_k \mu_{k,n-o} + \beta_k \Big(\widetilde{Q}_{k,n,o}^{\mu_{k,n-o}} + \phi_k \widetilde{Q}_{k,n,o}^{\mu_{k,n-o+1,n}} + \widetilde{Q}_{k,n,o}^{R} \Big) \\ &= (\alpha_k + \beta_k)\mu_{k,n-o} + \beta_k \phi_k \mu_{k,n-o+1,n} + \mathbb{E}[\widetilde{Q}_{k,n,o}^{R}] \\ &= \frac{n-o}{n}\mu_{k,n-o} + \frac{o}{n}\mu_{k,n-o+1,n} \\ &\overset{(a)}{=} \mu_{k,n},\end{aligned}$$

where (a) uses the result in Eq. (27).

According to the definition of $\mu_{k,n}$, the necessary condition for having vanishing excess regret is

$$\mathbb{E}[\overline{Q}_{k,n,o'}] = \mu_{k,n} = \frac{1}{n}\sum_{n'=1}^{n} \mathbb{E}[V^{\mathbb{A}_{\text{seq}}}_{\tau_{s,a_k}(n')}(s')].$$

Equivalently, it can be written as

$$\mathbb{E}[\overline{Q}(s,a_k)] = \mu_{k,n'} = \frac{1}{n}\sum_{n'=1}^{n} \mathbb{E}[V^{\mathbb{A}_{\text{seq}}}_{\tau_{s,a_k}(n')}(s')],$$

where $n = N(s,a_k) + O(s,a_k)$. This completes the proof of the first necessary condition in Theorem 1.

Assuming the first necessary condition is satisfied, we proceed to prove the second necessary condition. Note that according to the assumption on $\widetilde{N}$ made in the theorem, we have:

$$\overline{N}(s_0,a_k) = N(s_0,a_k) + f(O(s_0,a_k)),$$

where f can be any function whose domain is $\{x \mid 0 \le x \le M-1, x \in \mathbb{Z}\}$ and whose range is $[0,+\infty)$.

Suppose at time step τ_0 ($\tau_0 < \tau_{\text{sim}}$), arm k has been visited n_0+1 times (one of them is the done at the initialization phase of the corresponding edge (s,a_k)). We consider the quantity $\Pr(\overline{Q}_{k,n_0+1,n_0} \le \mu_k + \overline{e}_{\tau_0,n_0+1,n_0})$, which represents the probability of Eq. (26) in the circumstance specified by k, τ_0, and n_0:

$$\begin{aligned}&\Pr(\overline{Q}_{k,n_0+1,n_0} \le \mu_{k,n_0+1} + \overline{e}_{\tau_0,n_0+1,n_0}) \\ \overset{(a)}{=}\ &\Pr(\overline{Q}_{k,n_0+1,0} \ge \mu_{k,n_0+1} + \overline{e}_{\tau_0,n_0+1,n_0}) \\ \overset{(b)}{\le}\ &\exp(-\frac{n_0+1}{2}\overline{e}^2_{\tau_0,n_0+1,n_0}) \\ \overset{(c)}{=}\ &\exp(-\frac{n_0+1}{2}\frac{2\log g(\tau_0)}{f(n_0)+1}) \\ =\ &\frac{1}{g(\tau_0)^{\frac{n_0+1}{f(n_0)+1}}},\end{aligned} \tag{37}$$

where (a) uses the assumption that the first necessary condition is satisfied (i.e. $\overline{Q}_{k,n_0+1,n_0} = Q_{k,n_0+1}$), (b) follows the Chernoff-Hoeffding bound on the $\frac{1}{n_0+1}$-subgaussian random variable Q_{k,n_0+1}, and (c) expands the definition of $\overline{e}_{\tau_0,n_0+1,n_0}$ ($g(\cdot)$ is defined as follows). Suppose we have $f(n_0) = n_0' < n_0$ (without loss of generality assume $f(x) = x$ ($x \ne n_0$)). We first show that in this case $g(\tau_0) = \tau_0 - n_0 + n_0'$. Specifically, according to the tree policy (Eq. (14)), we have

$$\begin{aligned}g(\tau) &= \sum_k \overline{N}(s,a_k) \\ &= \sum_k N(s,a_k) + f(O(s,a_k)) \\ &= \sum_k N(s,a_k) + O(s,a_k) + \Big(f(O(s,a_k)) - O(s,a_k) \Big).\end{aligned}$$

First, notice that at rollout step τ_0, we have $\sum_k N(s,a_k) + O(s,a_k) = \tau_0$. The reason is that at rollout step τ_0, Algorithm 1 has initialized τ_0 simulations in total, and each simulation is either observed (will be counted by $N(s,a_k)$) or unobserved (will be counted by $O(s,a_k)$). Next, we look at the last term $f(O(s,a_k)) - O(s,a_k)$. By assumption, it equals to $n_0' - n_0$ if and only if $O(s,a_k) = n_0$, and is otherwise zero. Therefore, at rollout step τ_0, as long as no other edges have n_0 on-going simulations, we can conclude that $g(\tau_0) = \tau_0 - n_0 + n_0'$. Therefore, Eq. (37) can be further simplified as

$$\Pr(\overline{Q}_{k,n_0+1,n_0} \leq \mu_{k,n_0+1} + \overline{e}_{\tau_0,n_0+1,n_0}) \leq (\tau_0 - n_0 + n_0')^{-\frac{n_0+1}{n_0'+1}}. \tag{38}$$

We focus on the condition of having nonvanishing excess regret. Specifically, we focus on the condition of Eq. (38) being greater than the upper bound in the sequential case (i.e. when all n_0 unobserved samples are observed), which is $\frac{1}{\tau_0}$ Chernoff-Hoeffding inequality of subgaussian variables:

$$(\tau_0 - n_0 + n_0')^{-\frac{n_0+1}{n_0'+1}} > \frac{1}{\tau_0} \Leftrightarrow \tau_0 > (\tau_0 - n_0 + n_0')^{\frac{n_0+1}{n_0'+1}}.$$

For any n_0 and n_0', there exists t_0 such that when $\tau_0 > t_0$, we have

$$\tau_0 - (\tau_0 - n_0 + n_0')^{\frac{n_0+1}{n_0'+1}} > n_0 - n_0', \tag{39}$$

where $n_0 - n_0'$ is nonvanishing as τ_0 increases. Therefore, it will incur a nonvanishing term in the probability of Eq. (26), which will result in a nonvanishing regret term. Therefore, to have vanishing cumulative regret, we should not have $f(n_0) = n_0' < n_0$. This confirms the necessary condition $f(x) > x$. □

C.2 Theoretical Justification of WU-UCT

This section provides formal proof of Theorem 2, which indicates WU-UCT achieves vanishing excess regret under the depth-2 setup. In the following, we first justify the statement "$R_{UCT}(n)$ is the cumulative regret of running the (sequential) UCT for n steps on $\mathbb{T}$", i.e., the expected cumulative regret of the UCT algorithm under the depth-2 setup.

Cumulative regret upper bound of UCT in the depth-2 case Define a_{k^*} as the optimal action that leads to the highest expected reward. According to the regret decomposition identity Mazumdar et al. (2017), $\mathrm{Regret}(t)$ can be decomposed with respect to different arms:

$$\mathrm{Regret}(t) = \sum_{k \in \{1,\dots,K\}, k \neq k^*} \Delta_k \mathbb{E}\left[T_k(t)\right], \tag{40}$$

where $\Delta_k := \mu^* - \mu_k$, $\mu^* := \max_k \mu_k$, $k^* := \arg\max_k \mu_k$, and $T_k(t)$ is defined as the number of times arm k is selected in the first t rollouts. This suggests that we only need to bound the expected visit counts of all suboptimal arms (i.e., $\mathbb{E}\left[T_k(t)\right]\ (k \neq k^*)$). $Q_t(s_0,a_k)$ is defined as the reward estimate for arm k at the end of the tth rollout, and $N_t(s_0,a_k)$ denotes the visit count of arm k at the end of rollout step t. To simplify notation, we additionally define $Q_{k,n}$ as the (empirical) average reward of arm k after the nth observation of that arm (i.e., n simulation returns have been obtained).

The event $\mathrm{Arm}_\tau = k$ means the kth arm is pulled at time t. According to the definition of $T_k(t)$, we have (define l as an arbitrary positive integer; $e_{t,n} := \sqrt{(2\ln t)/n}$)

$$\begin{aligned}
T_k(t) &= 1 + \sum_{\tau=K+1}^{t} \mathbb{1}\left[\mathrm{Arm}_\tau = k\right] \\
&\leq l + \sum_{\tau=K+1}^{t} \mathbb{1}\left[\mathrm{Arm}_\tau = k, T_k(\tau-1) \geq l\right] \\
&\overset{(a)}{\leq} l + \sum_{\tau=K+1}^{t} \mathbb{1}\left[Q_{\tau-1}(s_0,a_{k^*}) + e_{\tau,N_{\tau-1}(s_0,a_{k^*})} \leq Q_{\tau-1}(s_0,a_k) + e_{\tau,N_{\tau-1}(s_0,a_k)}\right] \\
&\leq l + \sum_{\tau=K+1}^{t} \mathbb{1}\left[\min_{0<n<\tau} Q_{k^*,n} + e_{\tau,n} \leq \max_{l \leq n' < \tau} Q_{k,n'} + e_{\tau,n'}\right]
\end{aligned}$$

$$\leq l + \sum_{\tau=1}^{t} \max_{n\in[1,\tau-1]} \max_{n'\in[1,\tau-1]} \mathbb{1}\left[Q_{k^*,n} + e_{\tau,n} \leq Q_{k,n'} + e_{\tau,n'}\right]. \tag{41}$$

where (a) uses the fact that the necessary condition of choosing arm k at rollout step τ is that the upper confidence bound of the kth arm is greater than or equal to that of the optimal arm k^*.

We bound the probability of the event $Q_{k^*,n} + e_{\tau,n} \leq Q_{k,n'} + e_{\tau,n'}$ using the sum of the following three events' probability:

$$Q_{k^*,n} \leq \mu^* - e_{\tau,n}, \tag{42}$$
$$Q_{k,n'} \geq \mu_k + e_{\tau,n'}, \tag{43}$$
$$\mu^* < \mu_k + 2e_{\tau,n'}. \tag{44}$$

Since the rewards received from arm k minus its expectation (i.e., $R_k - \mu_k$) are independent 1-subgaussian random variables (by the assumption made in Theorem 2), we can show that $Q_{k,n}$ is $1/n$-subgaussian (since it is the average of n 1-subgaussian random variables Buldygin & Kozachenko (1980)). The Chernoff-Hoeffding bound for subgaussian random variables state that if random variable X is σ^2-subgaussian, we have $\Pr(X \geq \epsilon) \leq \exp\left(-\epsilon^2/(2\sigma^2)\right)$. Plug in Eqs. (42) and (43), we have

$$\Pr\left(Q_{k^*,n} \leq \mu^* - e_{\tau,n}\right) \leq 1/\tau, \tag{45}$$
$$\Pr\left(Q_{k,n'} \geq \mu_k + e_{\tau,n'}\right) \leq 1/\tau. \tag{46}$$

Next, we focus on Eq. (44):

$$\mu_k + 2e_{t,n} > \mu^* \quad \Leftrightarrow \quad \mu_k + 2\sqrt{\frac{2\ln t}{s_k}} > \mu^* \quad \Leftrightarrow \quad \sqrt{\frac{2\ln t}{s_k}} > \frac{\Delta_k}{2} \quad \Leftrightarrow \quad s_k < \frac{8\ln t}{\Delta_k^2}.$$

Therefore, when $n' \geq \left\lceil \frac{8\ln t}{\Delta_k^2} \right\rceil$, Eq. (44) is guaranteed to be false. So we have

$$\begin{aligned}
\mathbb{E}\left[T_k(t)\right] &\leq \left\lceil \frac{8\ln t}{\Delta_k^2} \right\rceil + \sum_{\tau=1}^{t} \left(\Pr\left(Q_{k^*,n} \leq \mu^* - e_{\tau,n}\right) + \Pr\left(Q_{k,n'} \geq \mu_k + e_{\tau,n'}\right)\right) \\
&\leq \left\lceil \frac{8\ln t}{\Delta_k^2} \right\rceil + \sum_{\tau=1}^{t} \frac{2}{\tau} \\
&\leq \left(\frac{8}{\Delta_k^2} + 2\right)\ln t + 1.
\end{aligned}$$

Plugging this result in Eq. (40) gives the regret upper bound

$$\mathrm{Regret}(t) \leq R_{UCT} := \sum_{k\in\{1,\ldots,K\}, k\neq k^*} \left[\left(\frac{8}{\Delta_k} + 2\Delta_k\right)\ln t + \Delta_k\right].$$

Next, we justify the cumulative regret upper bound of WU-UCT.

Formal proof of Theorem 2

Proof of Theorem 2. Before delving into the proof, we briefly review WU-UCT Liu et al. (2020). WU-UCT constructs a global search tree that is operated only by the main/master process. The master process repeatedly perform rollouts and assign simulation and expansion tasks to the workers and collect results from them. Specifically, the main process performs selection with the modified tree policy (14) (with the hyperparameter specified according to Table 2), where an *incomplete update* process increments the *incomplete visit count* $O(s_0, a_k)$ of the traversed nodes by one. Expansions and simulations are done in parallel by the workers, and we refer readers interested in the details to Liu et al. Liu et al. (2020). During backpropagation, an additional *complete update* process decrements $O(s, a)$ of the traversed nodes by one.

On the high level, WU-UCT has a parallel architecture similar to TreeP, where all statistics are globally available (thus $\tau_{\mathrm{syn}} = 1$). We start the proof by a high-level demonstration, and then follow the key intuitions to formalize it.

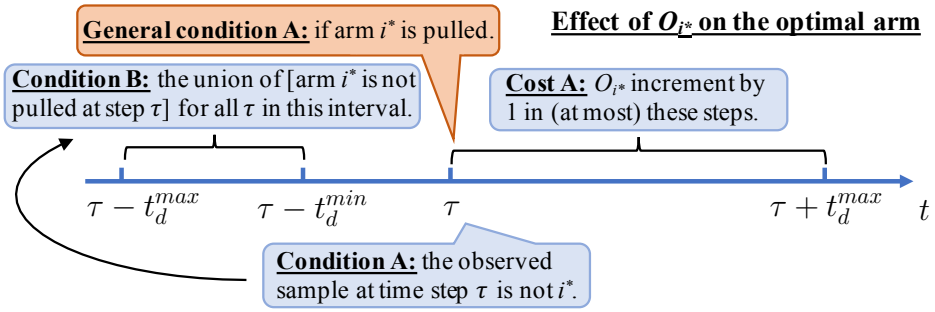


Figure 9: The influence of WU-UCT on the expected cumulative regret (comparing to the sequential case) by pre-updating O_{i^*}.

We argue that when dealing with the MAB problem, WU-UCT can be treated as a sequential UCT where some of the *observed samples* are replaced by *unobserved samples* without actual simulation return. First, note that with the help of the adjustment on the visit count (i.e., $\overline{N}(s_0, a_k) := N(s_0, a_k) + O(s_0, a_k)$), at time step τ, we can upper bound $T_k(t)$ by

$$T_k(t) \leq l + \sum_{\tau=K+1}^{t} \mathbb{1}\left[Q_{\tau-1}(s_0, a_{k^*}) + e_{\rho(\tau),\overline{N}_{\tau-1}(s_0,a_{k^*})} \leq Q_{\tau-1}(s_0, a_k) + e_{\rho(\tau),\overline{N}_{\tau-1}(s_0,a_k)}\right], \tag{47}$$

where $\rho(\tau) := \sum_{k=1}^{K} \overline{N}(s_0, a_k)$ according to the tree policy defined by Eq. (14). By the definition $\overline{N}(s_0, a_k) := N(s_0, a_k) + O(s_0, a_k)$ we can easily verify that $\rho(\tau) = \tau$: note that at the end of the τth rollout, there are τ assigned simulation tasks, and each task is either observed (is recorded in N) or unobserved (is recorded in O). Therefore, we can rewrite Eq. (47) as

$$T_k(t) \leq l + \sum_{\tau=K+1}^{t} \mathbb{1}\left[\overline{Q}_{\tau-1}(s_0, a_{k^*}) + e_{\tau,\overline{N}_{\tau-1}(s_0,a_{k^*})} \leq \overline{Q}_{\tau-1}(s_0, a_k) + e_{\tau,\overline{N}_{\tau-1}(s_0,a_k)}\right]. \tag{48}$$

The key observation we want to emphasize here is that with the help of the adjustment on the visit count (i.e., $\overline{N}(s_0, a_k) := N(s_0, a_k) + O(s_0, a_k)$), the time step represented by $\rho(\tau)$ has been calibrated to be the same with the sequential case, i.e., $\rho(\tau) := \tau$. In this way, as we shall proceed to show, though according to Table 2, WU-UCT has $\tau_{\text{sim}} = M$, it has vanishing regret. Under this observation, the main difference between WU-UCT and the sequential UCT is that its value estimates $\{Q_\tau(s_0, a_k)\}_{k=1}^{K}$ are less informative compared to UCT. Specifically, for the *incomplete simulations*, though $\overline{N}(s_0, a_k)$ is adjusted by $O(s_0, a_k)$ and resemble the sequential case, these simulation returns $\hat{V}(s_k)$ are not available and the variance of the estimate is relatively high compared to the sequential algorithm. Keep in mind this similarity between WU-UCT and UCT. In the following, we analyze the excess regret caused by the *inaccurate* $Q_\tau(s_0, a_k)$.

Following Eq. (48), we have

$$\begin{aligned}
T_k(t) &\leq l + \sum_{\tau=K+1}^{t} \mathbb{1}\left[\overline{Q}_{\tau-1}(s_0, a_{k^*}) + e_{\tau,\overline{N}_{\tau-1}(s_0,a_{k^*})} \leq \overline{Q}_{\tau-1}(s_0, a_k) + e_{\tau,\overline{N}_{\tau-1}(s_0,a_k)}\right] \\
&\overset{(a)}{\leq} l + \sum_{\tau=K+1}^{t} \mathbb{1}\left[Q_{\tau-1}(s_0, a_{k^*}) + e_{\tau,\overline{N}_{\tau-1}(s_0,a_{k^*})} \leq Q_{\tau-1}(s_0, a_k) + e_{\tau,\overline{N}_{\tau-1}(s_0,a_k)}\right] \\
&\overset{(b)}{\leq} l + \sum_{\tau=1}^{t} \max_{n\in[1,\tau-1]} \max_{n'\in[1,\tau-1]} \mathbb{1}\left[Q_{k^*,n} + e_{\tau,n+O_\tau(s_0,a_{k^*})} \leq Q_{k,n'} + e_{\tau,n'+O_\tau(s_0,a_k)}\right] \\
&\overset{(c)}{\leq} l + \sum_{\tau=1}^{t} \max_{n\in[1,\tau-1]} \max_{n'\in[1,\tau-1]} \mathbb{1}\left[Q_{k^*,n} + e_{\tau,n+O_\tau(s_0,a_{k^*})} \leq Q_{k,n'} + e_{\tau,n'}\right].
\end{aligned}$$

where (a) uses the fact that WU-UCT do not adjust the value (i.e. $\widetilde{Q}(s,a) = 0$), which results in $\overline{Q}_{\tau-1}(s_0, a_{k^*}) = Q_{\tau-1}(s_0, a_{k^*})$; (b) largely follows Eq. (41), and (c) is based on the fact that $e_{\tau,n_1} > e_{\tau,n_2}$ $(n_1 < n_2)$.

The main difference between the above upper bound and the corresponding upper bound of UCT (Eq. (41)) is the potential lag in Q, i.e., it has $O_\tau(s_0, a_k)$ less observed value estimates $\hat{V}(s_{k^*})$ compared to that expected by the confidence interval c. Similar to Eqs. (42)-(44), the probability of the event in the indicator function $\mathbb{1}[\cdot]$ can be bounded by the sum of the probability of the three following events:

$$Q_{k^*,n} \leq \mu^* - e_{\tau,n+O_\tau(s_0,a_{k^*})}, \tag{49}$$

$$Q_{k,n'} \geq \mu_k + e_{\tau,n'}, \tag{50}$$

$$\mu^* < \mu_k + 2e_{\tau,n'}. \tag{51}$$

Therefore, we only need to analysis the extra regret caused by $O_\tau(s_0, a_{k^*})$ in Eq. (49). Specifically, Figure 9 illustrate the affection on the regret caused by the existence of *incomplete simulations* (i.e., ongoing simulations whose return is currently unavailable) of the optimal arm. The remainder of the proof uses the following definition.

Definition 1 (Simulation interval τ_{sim})**.** *Between the period of a simulation task (s, m) being assigned to a worker in the simulation step and being returned in the wait step (i.e. the simulation completes), there are at most $\overline{\tau}_{\text{sim}} - 1$ and at least $\underline{\tau}_{\text{sim}} - 1$ other returned simulation result $(s', V, \hat{m})$ where $\hat{m} = m$.*

In the case of WU-UCT, since the algorithm contains only one global search tree, $\overline{\tau}_{\text{sim}}$ and $\underline{\tau}_{\text{sim}}$ measure the maximum and minimum rollout steps taken from a simulation task being assigned and being returned.

As demonstrated by Figure 9, the main cost of the *on-going simulations* on the optimal arm is that it make the value estimate $\overline{Q}$ less accurate, and there will be an underestimation on the upper confidence bound since we shrinked the exploration term in 14 over-optimistically. Concretely, we formalize the condition of the loss and the cost/effect of it. To have $O_\tau(s_0, a_{k^*})$ increased at time step τ, the precondition should be that the arm k^* is pulled at that time step (i.e., general condition A). In addition to that, we have to make sure that the observed/returned task/simulation at time τ is not for arm k^* (i.e., condition A) since if that is the case, $O_\tau(s_0, a_{k^*})$ would not change before and after time τ, and thus no additional cost will be added. Since condition A is hard to directly quantify, we instead rely on a looser condition that has guaranteed larger probability of it. Specifically, condition B (Figure 9) is a quantifiable constraint that satisfies the above statement. Condition B is based on the fact that only the tasks initiated between time $\tau - \overline{\tau}_{\text{sim}}$ and $\tau - \underline{\tau}_{\text{sim}}$ is possible to terminate at time τ, where we define $\overline{\tau}_{\text{sim}}$ as the maximum simulation interval and $\underline{\tau}_{\text{sim}}$ as the minimum simulation delay. To justify $\Pr(\text{condition A}) \leq \Pr(\text{condition B})$, we can verify that the converse of condition B (i.e., arm i^* is pulled in all time steps between $\tau - \overline{\tau}_{\text{sim}}$ and $\tau - \underline{\tau}_{\text{sim}}$. Without loss of generality, in the following we assume the simulation interval is always equal to τ_{sim}.

Therefore, at an abstract level, the additional expected cumulative regret incurred by WU-UCT compared to the (sequential) UCT can be written as:

$$\Pr\left(\text{General condition A}\right) \cdot \Pr\left(\text{Condition B}\right) \cdot \Pr\left(\text{Cost A}\right). \tag{52}$$

We upper bound equation (52) by

$$\Pr\left(\text{Condition B}\right) \cdot \Pr\left(\text{Cost A}\right), \tag{53}$$

We now consider each of the probabilities.

Condition B As hinted by the description of condition B in Figure 9, the probability of condition B is upper bounded by

$$\begin{aligned}
&(\overline{\tau}_{\text{sim}} - \underline{\tau}_{\text{sim}} + 1) \cdot \max_{\tau_0 \in [\underline{\tau}_{\text{sim}}, \overline{\tau}_{\text{sim}}]} \max_{n \in [1,\tau-1]} \max_{n' \in [1,\tau-1]} \Big\{ \Pr(Q_{k^*,n} \leq \mu^* - e_{\tau-\tau_0,n}) \\
&\qquad\qquad + \Pr(Q_{k,n'} \geq \mu_k + e_{\tau-\tau_0,n'}) \Big\} \\
\leq & 2 \cdot \frac{\overline{\tau}_{\text{sim}} - \underline{\tau}_{\text{sim}} + 1}{\tau - \overline{\tau}_{\text{sim}}} \overset{(a)}{=} \frac{2}{\tau - \tau_{\text{sim}}},
\end{aligned}$$

where (a) uses our assumption that $\overline{\tau}_{\text{sim}} = \underline{\tau}_{\text{sim}} = \tau_{\text{sim}}$.[15] Note that in this case we do not need to consider the case where $O_\tau(s_0, a_k) > 0$ since they are bounded by the *cost A* term in previous time steps and would be redundant to consider again here. Specifically, the excess regret caused by the on-going simulation at time step $\tau - \tau_{\text{sim}}$ has been upper bounded by the *cost A* term in their respective rollout step that they are initialized.

Cost A The cost here refers to the additional expected regret incurred by using the adjusted confidence interval

$$\sqrt{\frac{2\ln\tau}{N_\tau(s_0, a_{k^*}) + O_\tau(s_0, a_{k^*})}} \quad (O_\tau(s_0, a_{k^*}) > 0)$$

instead of the optimistic one (in the sequential case) $\sqrt{\frac{2\ln\tau}{N_\tau(s_0,a_{k^*})}}$. Formally, cost A can be bounded by

$$\begin{aligned}
&\max_{O\in[1,M-1]} \sum_{t=\tau}^{\tau+\tau_{\text{sim}}} \max_{n\in[1,\tau-1]} \max_{n'\in[1,\tau-1]} \Big\{ \Pr(Q_{k^*,n+O} \le \mu^* - e_{t,n+O+1}) \\
&\qquad + \Pr(Q_{k,n'+O} \ge \mu_k + e_{t,n'+O+1}) \\
&\qquad - \Pr(Q_{k^*,n+O} \le \mu^* - e_{t,n+O}) \\
&\qquad - \Pr(V_{k,s'+O} \ge \mu_i + e_{t,n'+O}) \Big\} \\
&\le \max_{O\in[1,M-1]} \sum_{t=\tau}^{\tau+\tau_{\text{sim}}} 2\left[t^{-\frac{O}{O+1}} - t^{-1}\right] \\
&\le 2\left(\frac{\tau_{\text{sim}}}{\sqrt{\tau}} - \frac{\tau_{\text{sim}}}{\tau}\right). \qquad (54)
\end{aligned}$$

Finally, we plug in the upper bounds of the conditions and costs into Eq. (53), which gives

$$\frac{4}{\tau - \tau_{\text{sim}}}\left(\frac{\tau_{\text{sim}}}{\sqrt{\tau}} - \frac{\tau_{\text{sim}}}{\tau}\right).$$

Finally, we upper bound the total cost incurred on $\mathbb{E}\left[T_i(t)\right]$ by

$$\sum_{\tau=\lceil\frac{8\ln t}{\Delta_i^2}\rceil}^{t} \frac{4}{\tau - \tau_{\text{sim}}}\left(\frac{\tau_{\text{sim}}}{\sqrt{\tau}} - \frac{\tau_{\text{sim}}}{\tau}\right) \qquad (55)$$

Since τ_{sim} is not dependent on t, there exists $t^* \in \mathbb{Z}^+$ such that whenever $t > t^*$, we have $\tau_{\text{sim}} < \lceil\frac{8\ln t}{\Delta_k^2}\rceil$ / 2. Therefore, Eq. (55) is upper bounded by

$$4\tau_{\text{sim}} \sum_{\tau=\lceil\frac{8\ln t}{\Delta_k^2}\rceil/2}^{t} \left[\frac{1}{\tau\sqrt{\tau}} - \frac{1}{\tau^2}\right] \le 2\tau_{\text{sim}}\left[2\sqrt{\frac{\Delta_k^2}{4\ln t}} - \frac{\Delta_k^2}{4\ln t}\right]. \qquad (56)$$

Note that Eq. (48) is the regret bound of $\mathbb{E}[T_k(t)]$, plugging in Eq. (40) finishes the proof, that is, the cumulative regret of WU-UCT on the MAB case is upper bounded by

$$R_{\text{UCT}}(n) + 4\tau_{\text{sim}} \sum_{k:\mu_k<\mu^*} 2\Delta_k \sqrt{\frac{\Delta_k^2}{4\ln n}}.$$

Since in WU-UCT, $\tau_{\text{sim}} = M$, the above quantity is equal to

$$R_{\text{UCT}}(n) + 4M \sum_{k:\mu_k<\mu^*} 2\Delta_k \sqrt{\frac{\Delta_k^2}{4\ln n}}.$$

□

[15] If this assumption does not hold, it will only add a constant term (independent to the number of rollout steps) in the final regret, which will not affect our main result.

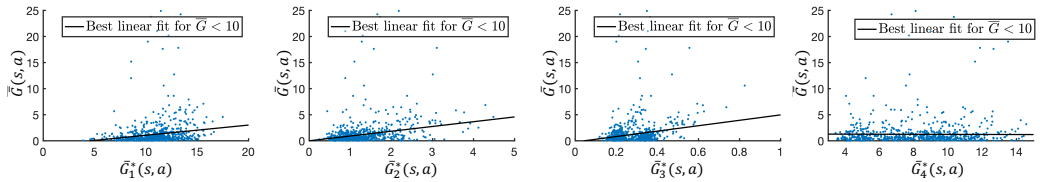


Figure 10: Relation between four additional surrogate gaps (i.e., $\overline{G}_1^*(s,a)$, $\overline{G}_2^*(s,a)$, $\overline{G}_3^*(s,a)$, and $\overline{G}_4^*(s,a)$) and the action value gap (i.e., $\overline{G}(s,a)$).

D Additional Details for BU-UCT

This section provides additional details of the BU-UCT algorithm, including an algorithm table (Appendix D.1) and introduction of all its hyperparameters (Appendix D.2).

D.1 Algorithm Table for BU-UCT

The algorithm table of BU-UCT is provided in Algorithm 3.

D.2 Hyperparameters of BU-UCT

The following provides a list of all hyperparameters in BU-UCT. We briefly discuss the recommended values for each hyperparameter.

- $m_{\max}$. $m_{\max} \in (0,1)$ is the hyperparameter that controls the degree we penalize $\overline{O}$. Specifically, if an edge (s,a) has $\overline{O}(s,a) \geq m_{\max} \cdot M$ (M is the number of workers), action a will not be selected by the tree policy when we are currently at node s. In our experiments, we choose $m_{\max} = 0.8$.

- Maximum tree depth/width. These hyperparameters should depend on the complexity of the tasks as well as the total computation budget. In our experiments, both the maximum tree depth and the maximum tree width are set to 100 and 20, respectively.

- Number of expansion/simulation workers. Expansion and simulation workers perform expansion and simulation tasks, respectively. In our experiments, we use 1 expansion worker and 16 simulation workers.

- The tree policy balancing factor c. c balance the exploration term (the second term) and the exploitation term (the first term) in the tree policy (Eq. (1)). In our experiments, it is selected as the standard deviation of the cumulative reward received by each node. For example, for node s, c is computed by the standard deviation of all cumulative reward received by s (i.e., all $V(s)$).

E Alternative Surrogate Statistics

Figure 10 presents four surrogate gaps (i.e., $\overline{G}_1^*(s,a)$, $\overline{G}_2^*(s,a)$, $\overline{G}_3^*(s,a)$, and $\overline{G}_4^*(s,a)$) that also exhibit positive correlation with the action value gap $\overline{G}(s,a)$. $\overline{G}_1^*(s,a)$, $\overline{G}_2^*(s,a)$, and $\overline{G}_3^*(s,a)$ all exhibit positive correlation with the action value gap (although their fitness scores are worse than $\overline{G}^*(s,a)$ introduced in the main text); $\overline{G}_4^*(s,a)$ is a very related statistics of $\overline{G}^*(s,a)$ but it is not correlated with $\overline{G}(s,a)$. Note that the surrogate gaps presented here are not exhaustive, and better statistics with stronger correlation with the action value gap could exists. The goal of presenting these additional gaps is to help inspire future work for designing more principled parallel MCTS algorithms. In the following, we introduce the three surrogate gaps in detail.

- $\overline{G}_1^*(s,a)$: $\overline{G}_1^*(s,a)$ is the standard deviation of the n (the number of rollouts) simulation returns related to the node s' (the next state following (s,a)):

$$\overline{G}_1^*(s,a) := \mathrm{Std}\big[\{V_i(s')\}_{i=1}^n\big],$$

where $\mathrm{Std}[A]$ denotes the standard deviation of all values in the set A.

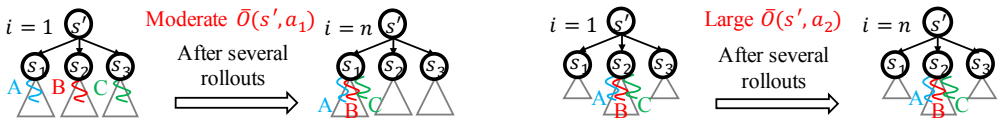


Figure 11: A key implication of the surrogate gap $\overline{G}^*$ introduced in the main text: high $\overline{G}^*$ potentially indicates overly exploitation of some (suboptimal) child nodes.

- $\overline{G}_2^*(s,a)$: Define $\overline{Q}_i(s,a)$ as the modified action value related to the edge (s,a) at the ith rollout step. $\overline{G}_2^*(s,a)$ denotes the standard deviation of the n modified action values $\{\overline{Q}_i(s,a)\}_{i=1}^n$:

$$\overline{G}_2^*(s,a) := \mathrm{Std}\big[\{\overline{Q}_i(s,a)\}_{i=1}^n\big].$$

- $\overline{G}_3^*(s,a)$: Similar to $\overline{G}_1^*(s,a)$, $\overline{G}_3^*(s,a)$ is the coefficient of variance (i.e., standard deviation divided by mean) of the n simulation returns $\{V_i(s')\}_{i=1}^n$:

$$\overline{G}_3^*(s,a) := \frac{\mathrm{Std}\big[\{V_i(s')\}_{i=1}^n\big]}{\mathrm{Aveg}\big[\{V_i(s')\}_{i=1}^n\big]},$$

where $\mathrm{Aveg}[A]$ denotes the average of all values in the set A.

- $\overline{G}_4^*(s,a)$: $\overline{G}_4^*(s,a)$ is defined as follows

$$\overline{G}_4^*(s,a) := \overline{O}(s,a) = \frac{1}{n}\sum_{i=1}^n O_i(s',a').$$

Despite its subtle difference with $\overline{G}^*(s,a)$ (recall that $\overline{G}^*(s,a) := \max_{a' \in \mathcal{A}} \overline{O}(s',a')$, where s' is the next state following (s,a)), $\overline{G}_4^*(s,a)$ is barely correlated with the action value gap $\overline{G}(s,a)$ while $\overline{G}^*(s,a)$ has strong (positive) correlation with it. Therefore, we conclude being extensively simulated by multiple workers does not necessarily results in high action value gap, which makes sense as the optimal child should be extensively exploited; instead, it is the maximum $\overline{O}$ among its child nodes that strongly correlates with $\overline{G}$, which might suggests that how well the tree policy can properly balance exploration and exploitation is of great importance. This point is further elaborated in the following.

Consider the two example tree search processes shown in Figure 11. On the left side, if the child nodes of s' (i.e., s_1, s_2, and s_3) are visited equally often at earlier stages (i.e., when s' has been visit for only a few times) and only start exploiting the nodes with higher action value (i.e., $\overline{Q}$) after certain number of rollouts, the statistics $\overline{O}(s', a_i)$ $(i = 1, 2, 3)$ will be relatively small since $\overline{O}$ is averaged across different rollout steps (i.e., different i for O_i). Hence, in this case, the surrogate gap $\overline{G}^*(s,a)$ ((s,a) is the edge that lead to s') will be relatively small, which suggests that the action value gap is also small. This match our intuition since properly explore all child nodes before exploiting the best one is beneficial and should lead to good performance. In contrast, consider a second case shown on the right side of Figure 10. In this case, exploitation happens even at the very beginning stage. The agent keeps assigning simulation tasks to query (offspring nodes of) s_2. In this case, $\overline{O}(s', a_2)$ is high according to its definition. This leads to high surrogate gap $\overline{G}^*(s,a)$, which suggests that the performance on this node is less desirable compared to the previous example. In fact, exploit certain nodes aggressively at earlier stages will cause other nodes under-explored, which makes the agent unable to recognize which child node is the most rewarding. If the agent does not happen to select the optimal node to exploit, it will fail to find an optimal action. Although implicit, the second modification (i.e., modification #2) proposed by BU-UCT try to solve this problem by penalizing over-exploitation (through lowering $\overline{N}$) in earlier stages.

Another advantage of BU-UCT's second modification is to encourage the algorithm to search deeper and wider. Again use Figure 10(left) as an example. For BU-UCT, if $\overline{N}(s', s_1) = m$, then m distinct offspring nodes of s_1 have been assigned simulation tasks. However, for its base algorithm WU-UCT, since s_1 might be queried by multiple workers, it will have less than m distinct offspring nodes of s_1 being assigned simulation tasks when $\overline{N}(s', s_1) = m$. This allows BU-UCT to explore deeper and

Table 3: Speedup achieved by BU-UCT using 16 workers on 15 Atari games. Elapsed time represents the average wall clock time to run a single tree search step (i.e., build a search tree with 128 rollouts). Speedup is calculated by dividing the (average) elapsed time using 1 worker by the (average) elapsed time using 16 workers.

Environment	Elapsed time/s (1 worker)	Elapsed time/s (16 workers)	Speedup (16 vs. 1 worker(s))
Alien	54.19	3.81	14.20
Boxing	54.71	3.55	15.37
Breakout	44.27	3.56	12.42
Centipede	50.18	3.34	15.01
Freeway	56.98	3.75	15.16
Gravitar	39.44	2.90	13.55
MsPacman	44.18	3.18	13.88
NameThisGame	43.36	3.06	14.13
RoadRunner	45.36	3.03	14.92
Robotank	54.08	3.80	14.19
Qbert	43.25	3.06	14.10
SpaceInvaders	45.11	3.14	14.36
Tennis	54.35	3.72	14.58
TimePilot	41.81	2.91	14.34
Zaxxon	46.09	3.09	14.88

potentially provide more accurate value estimate of s_1 compared to WU-UCT. In an extreme case, if WU-UCT assigns m workers to simulate s_1, it can only obtain the simulation return at node s_1, which makes its action value $\overline{Q}(s', a_1)$ less accurate.

F Additional Details for Experiments

This section provides additional experiment results and implementation details of the Atari experiments. First, Appendix F.1 provides results of BU-UCT's speedup test on 15 Atari games. Next, Appendix F.2 describes additional implementation details of the Atari experiments. Finally, Appendix F.3 provides details regarding the demonstrative experiment in Figure 2(b) (i.e., average action value gap vs. episode reward).

F.1 Speedup Test for BU-UCT

The speedup of BU-UCT with 16 workers compared to its sequential counterpart (i.e., 1 worker) is shown in Table 3. Across 15 Atari games, BU-UCT achieves on average 14.33 times speedup using 16 workers, which suggests that BU-UCT can better retain the performance of UCT compared to the baselines while achieving desired speedup.

F.2 Experiment Details of the Atari Games

MCTS simulation Each simulation worker is equipped with a pre-trained policy network (that predicts $\pi(a|s)$) and a pre-trained value network (that estimate $V(s)$). Both networks are pre-trained by the Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm. Table 4 summarizes the performance on the 15 Atari games using only the PPO policy. For a simulation started from state s_0, we use the PPO policy network to interact with the environment for 100 steps, which forms a trajectory $s_0, a_0, r_0, s_1, \ldots, s_{99}, a_{99}, r_{99}, s_{100}$. If the environment does not terminate, the full simulation return is computed by the intermediate rewards plus the value of s_{100}, i.e., the simulation return is $R_{\text{sim}} := \sum_{i=0}^{99} \gamma^i r_i + \gamma^{100} V(s_{100})$. To reduce the variance of Monte Carlo sampling, we average it with the value $V(s_0)$. The final simulation return is $R := 0.5 R_{\text{sim}} + 0.5 V(s_0)$.

Hyperparameters and experiment details for BU-UCT For all parallel MCTS algorithms, we choose the maximum tree depth/width as 100/20, respectively. The discount factor γ is set to 0.99 (note that the reported score is not discounted). Additional details regarding hyperparameters are shown in Appendix D.2. Experiments are deployed on machines with 88 CPU cores and 8 NVIDIA® P40 GPUs. To minimize speed fluctuation caused by difference in the machines' workload, we ensure that the total number of processes is smaller than the total number of CPU cores.

Table 4: Performance of the PPO policy on 15 Atari games.

Environment	PPO policy
Alien	850
Boxing	7
Breakout	191
Centipede	1701
Freeway	32
Gravitar	600
MsPacman	1860
NameThisGame	6354
RoadRunner	26600
Robotank	13
Qbert	12725
SpaceInvaders	1015
Tennis	-10
TimePilot	4400
Zaxxon	3504

Hyperparameters and experiment details for baseline algorithms For WU-UCT (Liu et al., 2020), we reuse their code provided on GitHub. We also reuse the implementation of the baseline algorithms (i.e., VL-UCT, LeafP, and RootP) provided by Liu et al. (2020). All algorithms are implemented in Python, especially utilizing its "multiprocessing" module. All hyperparametes of LeafP and RootP have been covered in the previous paragraph. For VL-UCT, we report the better performance among the following two hyperparameter setups: $r_{VL}=1.0$ and $r_{VL}=5.0$ (see Appendix A).

F.3 Details for the Action Value Gap vs. Performance Experiments

This section describes the experiment setup of the scatter plot between average action value gap and episode reward (i.e., Figure 2(b)).

General setup Each node in the scatter plots represents a full run in the corresponding Atari game. That is, at each time step, we use MCTS to plan for the best action to execute, until the game terminates. Note that for each time step we need to construct a search tree and perform rollouts on it.

Average action value gap The reported average action value is averaged across (i) search trees constructed at different time steps of the game, and (ii) (for a single search tree) the action value gap $\overline{G}(s,a)$ with respect to different edges (s,a). Note that to minimize noise, we use a weighted average over action value gap $\overline{G}(s,a)$ for different edges. The weight is the complete visit count of that node (i.e., $N(s,a)$).

Episode reward We adopt the most common performance measure used in Atari — the episode reward. It sums up the reward obtained at all time steps without discount.

Hyperparameters All experiments are run with a set of randomly selected hyperparameters. For all algorithms, we randomly select the number of workers from the range $[4, 32]$. All experiments perform in total 512 rollouts. A random default policy was used to reduce the time consumption. Maximum depth/width of the search tree is 100/20. For hard virtual loss (see Appendix A), r_{VL} is selected from the range $[0, 10]$; for soft virtual loss, r_{VL} is selected from the range $[0, 20]$ and n_{VL} is selected from the range $[1, 5]$.

G BU-UCT

Please refer to Algorithm 3 for an algorithm table of the proposed BU-UCT algorithm. In the following, we formally introduce the two key ideas proposed by BU-UCT — thresholding $\overline{O}$ and aggregating-and-backpropagating simulation returns.

Thresholding $\overline{O}$ We maintain an additional statistics $\overline{O}(s,a)$ (defined in Eq. (10)) for all edges (s,a) in the search tree:

$$\overline{O}(s,a) = \frac{1}{n}\sum_{i=1}^{n} O_i(s',a'),$$

where n is the current number of rollouts and $O_i(s,a)$ is the number of on-going simulations associated with the edge (s,a) at the ith rollout step. $\overline{O}$ is maintained by the **incomplete_update** function (Algorithm 4, Line 5). This function is called whenever a simulation task has been assigned to a worker, and is also used to update the incomplete visit count $O(s,a)$. $\overline{O}$ is then used to adjust the action value $\overline{Q}$ used by the selection step:

$$\overline{Q}(s,a) := Q(s,a) - \infty \cdot \mathbb{1}\left[\overline{O}(s,a) \geq m_{\max} \cdot M\right] \text{ (i.e., Eq. (11))}.$$

Aggregating-and-backpropagating simulation returns If a node s is in its "earlier stages" (i.e., some of its children have not received simulation returns), BU-UCT aggregates the simulation returns originated from each of its child node into a single simulation return. This is done by two modifications in Algorithm 3. First, in the selection step (i.e., Algorithm 3, Line 5), whenever we are at a node s that is in its "earlier stages" (i.e., $\exists a \in \mathcal{A}\ N(s,a) = 0$), then the visit count used in the tree policy is set as 1 for all $a \in \mathcal{A}$:

$$\overline{N}(s,a) := 1.$$

Otherwise the visit count $\overline{N}(s,a)$ is the sum of complete and incomplete visit count of edge (s,a):

$$\overline{N}(s,a) := N(s,a) + O(s,a).$$

The second modification is located in the **complete_update** function (Algorithm 5), which is called after a simulation has completed and is used to update node/edge statistics (i.e., the standard back-propagation step). Specifically, as shown in Lines 8-11 of Algorithm 5, when all children of node s have received at least one simulation return, we reset the complete visit count N of all child edges to 1:

$$\forall a' \in \mathcal{A} \quad N(s,a') \leftarrow 1.$$

Algorithm 3 BU-UCT

Input: environment emulator $\mathcal{E}$, root tree node s_{root}, maximum simulation step T_{max}, maximum simulation depth d_{max}, number of expansion workers N_{exp}, and number of simulation workers N_{sim}
Initialize: expansion worker pool $\mathcal{W}_{exp}$, simulation worker pool $\mathcal{W}_{sim}$, game-state buffer $\mathcal{B}$, $t \leftarrow 0$, and $t_{complete} \leftarrow 0$
while $t_{complete} < T_{max}$ **do**
  `# Selection`
  Traverse the tree top down from root node s_{root} with the tree policy shown below (i.e., Eq. (1)) until (i) its depth greater than d_{max}, (ii) it is a leaf node, or (iii) it is a node that has not been fully expanded and *random()* < 0.5. Specifically, when we are at node s_t, we select the following action a_t using the tree policy:

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left\{ \overline{Q}(s_t, a) + c \sqrt{\frac{2 \ln \sum_{a'} \overline{N}(s_t, a')}{\overline{N}(s_t, a)}} \right\} \ (\texttt{tree policy}),$$

  where $\overline{Q}(s,a) := Q(s,a) - \infty \cdot \mathbb{1}\left[\overline{O}(s,a) \geq m_{\max} \cdot M\right]$ (i.e., Eq. (11)); $\overline{N}(s,a) := N(s,a) + O(s,a)$ if $\forall a \in \mathcal{A}\ N(s_t, a) > 0$ and otherwise $\overline{N}(s,a) := 1$. After the selection step, we are at a leaf node s of the search tree.
  `# Assign expension and simulation tasks`
  if expansion is required **then**
    Assign expansion task (t, s) to pool $\mathcal{W}_{exp}$ **//** t **is the task index.**
  else
    Assign simulation task (t, s) to pool $\mathcal{W}_{sim}$ if episode not terminated
    Call **incomplete_update**(s); if episode terminated, also call **complete_update**$(t, s, 0.0)$
  end if
  $t \leftarrow t + 1$ **// Accumulate iteration count**
  `# Fetch expansion tasks`
  if $\mathcal{W}_{exp}$ fully occupied **then**
    Wait for a expansion task with return: (task index τ, current state s, expended action a, reward r, expended state s' (the next state following (s, a)), and terminal signal d); expand the tree according to τ, s, a, s', r, and d; assign simulation task (τ, s) to pool $\mathcal{W}_{sim}$
    Call **incomplete_update**(t, s)
  else continue
  `# Fetch simulation tasks and backpropagate`
  if $\mathcal{W}_{sim}$ fully occupied **then**
    Wait for a simulation task with return: (task index τ, node s, cumulative reward $\bar{r}$)
    Call **complete_update**$(\tau, s, \bar{r})$; $t_{complete} \leftarrow t_{complete} + 1$
  else continue
end while

Algorithm 4 incomplete_update

input: node s
while $n \neq$ null **do**
  $a \leftarrow$ `the previous action selected at this node`
  $O(s,a) \leftarrow O(s,a) + 1$ **// Update incomplete visit count**
  $\overline{O}(s,a) \leftarrow \frac{\overline{N}(s,a)-1}{\overline{N}(s,a)} \overline{O}(s,a) + \frac{1}{\overline{N}(s,a)} O(s,a)$ **// Update average incomplete visit count**
  $s \leftarrow \mathcal{PR}(s)$ **//** $\mathcal{PR}(s)$ **denotes the parent node of** s
end while

Algorithm 5 complete_update

input: task index t, node s, reward $\bar{r}$
while $n \neq$ null **do**
  Retrieve the selected action a and the corresponding reward r according to task index t
  $N(s,a) \leftarrow N(s,a) + 1$; $O(s,a) \leftarrow O(s,a) - 1$ **// Update complete and incomplete visit count**
  $\bar{r} \leftarrow r + \gamma \bar{r}$ **// Calculate cumulative reward**
  $Q(s,a) \leftarrow \frac{N(s,a)-1}{N(s,a)} Q(s,a) + \frac{1}{N(s,a)} \bar{r}$ **// Update action value**
  $s \leftarrow \mathcal{PR}(s)$ **//** $\mathcal{PR}(s)$ **denotes the parent node of** s
  // Now all child nodes of s **has at least one complete simulation return (note that this condition will be satisfied only once for each node)**
  if $N(s,a) = 1$ && $\forall a' \in \mathcal{A} \backslash \{a\}\ N(s,a') > 0$ **then**
    $\forall a' \in \mathcal{A} \quad N(s,a') \leftarrow 1$
  end if
end while