Resource Allocation in Multi-armed Bandit Exploration: Overcoming Sublinear Scaling with Adaptive Parallelism

Brijen Thananjeyan [1] **Kirthevasan Kandasamy** [1] **Ion Stoica** [1] **Michael I. Jordan** [1] **Ken Goldberg** [1] **Joseph E. Gonzalez** [1]

Abstract

We study exploration in stochastic multi-armed bandits when we have access to a divisible resource that can be allocated in varying amounts to arm pulls. We focus in particular on the allocation of distributed computing resources, where we may obtain results faster by allocating more resources per pull, but might have reduced throughput due to nonlinear scaling. For example, in simulation-based scientific studies, an expensive simulation can be sped up by running it on multiple cores. This speed-up however, is partly offset by the communication among cores, which results in lower throughput than if fewer cores were allocated per trial to run more trials in parallel. In this paper, we explore these trade-offs in two settings. First, in a fixed confidence setting, we need to find the best arm with a given target success probability as quickly as possible. We propose an algorithm which trades off between information accumulation and throughput and show that the time taken can be upper bounded by the solution of a dynamic program whose inputs are the gaps between the sub-optimal and optimal arms. We also prove a matching hardness result. Second, we present an algorithm for a fixed deadline setting, where we are given a time deadline and need to maximize the probability of finding the best arm. We corroborate our theoretical insights with simulation experiments that show that the algorithms consistently match or outperform baseline algorithms on a variety of problem instances.

1. Introduction

In multi-armed bandit exploration, an agent draws samples from a set of n arms, where, upon pulling arm i, it receives a stochastic reward drawn from a distribution with mean μ_i. The goal is to identify the best arm, i.e., $\mathrm{argmax}_i\, \mu_i$, by adaptively choosing which arms to pull. This problem is known in the literature as best-arm identification (BAI).

[1]University of California, Berkeley. Correspondence to: Brijen Thananjeyan <bthananjeyan@berkeley.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

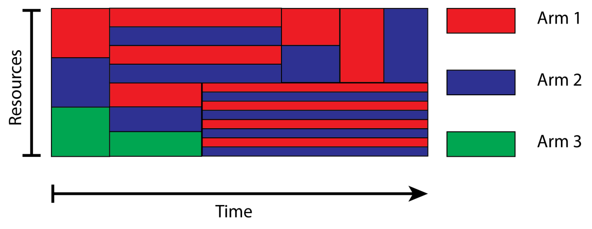

Figure 1. An example of the setting using $n = 3$ arms. An algorithm has a *divisible resource* (y-axis) that can be split up to generate samples by pulling the 3 arms. Each pull takes *time* (x-axis) to complete. Allocating more resources to a pull produces faster results, but at the cost of decreased throughput, e.g. using twice the resources for a pull will provide results only 1.5x faster. When a pull completes, the resources used are freed and can be reallocated to new pulls.

As an example, consider simulation-based studies in physics, which are used for estimating cosmological constants and controlling nuclear fusion reactors (Davis et al., 2007; Xing et al., 2019). Pulling an arm may correspond to running a stochastic simulation with specific values for parameters (cosmological constants or reactor parameters) which affect the output of the simulation. Here, searching for the optimal parameters is naturally modeled as a BAI problem. BAI is also used in model selection (Li et al., 2017), A/B testing (Howard & Ramdas, 2019), and other configuration tuning tasks. Traditionally, BAI is studied in two settings: fixed confidence and fixed budget. In the former, we must identify the best arm with a given target success probability, while keeping the number of arm pulls to a minimum. In the latter, we are given a budget of pulls and should maximize the probability of identifying the best arm.

The focus of work in the BAI literature has been the sequential setting in which the agent can pull only one arm at a time (Bubeck et al., 2009; Audibert & Bubeck, 2010; Even-Dar et al., 2002). There is also a line of work on the batch parallel setting, where the agent can pull a fixed number of arms at a time (Jun et al., 2016). These formulations

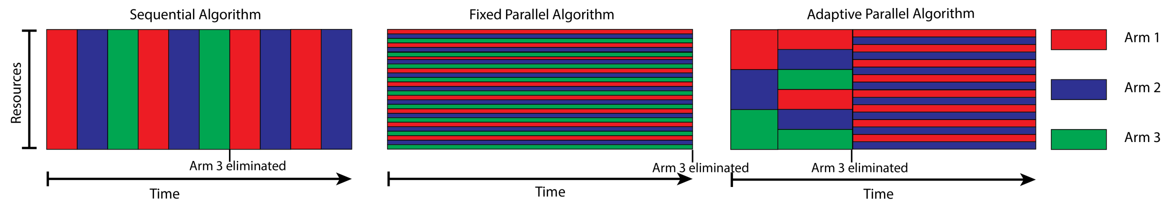

Figure 2. Left: A sequential strategy will allocate all resources to pulls one at a time. This enables the algorithm to obtain information about individual pulls sooner for replanning; after pulling all arms twice, it is able to eliminate the green arm. However, this reduces throughput due to sublinear scaling, which can be inefficient in the long run. *Middle:* A highly-parallel strategy with fixed batch size will have higher throughput. However, since it does not have feedback, it may evaluate the poor arms too many times and take longer to eliminate them. *Right:* Our algorithm, APR, for the fixed confidence setting, adaptively manages parallelism during execution based on the scaling function and task progress. In this figure, the green arm is eliminated early, and then throughput is increased for the remaining 2 arms which may be harder to distinguish. While the setting permits allocating the resources "asynchronously" (see Fig. 1), we show that an algorithm which operates synchronously, but chooses the amount of parallelism adaptively is minimax optimal for this problem.

have their limitations; in particular, modern infrastructures for parallel and distributed computing enable us to execute several arm pulls in parallel and control the duration of arm pulls by assigning multiple resources to a single pull. For example, physics simulations can exhibit strong elasticity in their resource requirements, where the number of CPUs allocated to the same simulation can range from a few tens to a few thousand CPUs. Using more CPUs results in speedier outputs, but this speedup can be sublinear due to communication and synchronization among the CPUs.

We develop algorithms for BAI where we are given access to a fixed amount of a divisible resource to execute the arm pulls. In addition to choosing which arms to pull, an algorithm must also determine how much of this resource to allocate for each arm pull. We have illustrated the setting in Fig. 1. While traditional formulations for BAI are stated in terms of the number of arm pulls, in our setting it is meaningful to formulate this in terms of *time constraints*.

By using more resources for a single arm pull, we obtain its result sooner; however, there is diminishing value to allocating more resources to the same arm pull, since typical distributed environments do not scale linearly (Venkataraman et al., 2016; Liaw et al., 2019). By allocating more resources for individual pulls and obtaining their results sooner, we can use that information to make refined decisions about which arms to try in subsequent iterations. However, due to sublinear scaling, by *parallelizing* arm pulls (executing many arm pulls simultaneously with fewer resources each), we can complete more pulls per unit time and obtain more information about the arms. This tradeoff between information accumulation (obtaining the results of a particular pull sooner) and throughput (number of arm pulls per unit of time) is fundamentally different from the usual explore-exploit trade-off encountered in bandit problems. In the latter, exploration is akin to testing several arms (and not arm pulls) and exploitation is akin to testing fewer carefully selected arms, and in our setting, an algorithm may choose to explore and/or exploit at different levels of parallelism. We have illustrated this trade-off in Figure 2.

In this work, we assume a *known* scaling function λ, where $\lambda(1/\alpha)$ is the time taken to complete a pull using a fraction α of the resource. To model the above trade-off, we will assume a diminishing returns property on λ.

Our contributions are as follows. First, in the fixed confidence setting, we propose an algorithm, Adaptive Parallel Racing (APR), and bound its time complexity by the solution to a dynamic program (DP) whose inputs are the inverse squared gaps between the optimal arm and the other arms. We also prove a hardness result which shows that the expected time taken by any algorithm is lower bounded by this DP, demonstrating that this DP is fundamental to this problem. Second, while our primary focus is in the fixed confidence setting, we also study a fixed deadline version of this problem, where we are given a time deadline, and wish to maximize the probability of identifying the best arm. We propose Staged Sequential Halving (SSH), a simple variant of the popular sequential-halving strategy for BAI and bound its failure probability. Third, we corroborate these theoretical insights with empirical simulations on some synthetic problems. We observe that APR performs as well as the best task-tuned baseline algorithms with fixed levels of parallelism in the fixed confidence setting. In the fixed deadline setting, SSH succeeds $20-90\%$ more often than baselines which do not account for nonlinear scaling.

Related Work

Since multi-armed bandits were introduced by Thompson (1933), they have been widely studied as an abstraction that formalizes the exploration-exploitation trade-offs that arise in decision-making under uncertainty (Robbins, 1952; Auer, 2003). Best arm identification (BAI) is a special case of bandits which is generally studied in the sequential setting in which an agent adaptively evaluates one arm at a time in order to identify the best arm (Bubeck et al., 2009; Gabillon et al., 2012; Karnin et al., 2013; Russo, 2016; Bubeck et al., 2013; Kalyanakrishnan & Stone, 2010; Jamieson et al., 2014). However, this sequential setting can fall short of capturing the full range of trade-offs that arise when an agent may be able to evaluate several arms concurrently.

In recent work, Jun et al. (2016) formulate a parallel version of BAI in the fixed confidence setting using the confidence intervals from Jamieson et al. (2014). In their work, the level of parallelism remains fixed; i.e., the agent is allowed to select a batch of arms to pull at each round. In the current paper, we consider a setting that allows adaptive parallelism using a fixed resource, where we explicitly consider *execution time* as a function of the batch size. Grover et al. (2018) study a similar setting where there is delayed feedback in BAI. While their model allows handling parallel arm pulls, it does not allow adaptive resource allocation. In addition, a line of work has studied parallel bandits in the regret minimization setting using either Bayesian optimization (Desautels et al., 2014; Kandasamy et al., 2018) or different notions of resource scaling (Lattimore et al., 2014; Dagan & Koby, 2018; Verma et al., 2019). In this work, we study the BAI setting, which is a pure exploration problem and we do not assume a prior over the arms.

Our algorithm for the fixed confidence setting proceeds by constructing confidence intervals for the arms' mean values, and then eliminates those arms which can be concluded to be non-optimal based on these confidence intervals. The construction of these confidence intervals is based on the law of the iterated logarithm (Jamieson et al., 2014). We also establish a hardness result in this setting which requires establishing lower bounds on the sample complexity, and then translating this to a lower bound on the time. For the sample complexity bounds, we use ideas from Kaufmann et al. (2016). In the fixed deadline setting, our algorithm is based on Karnin et al. (2013), who proposed a sequential-halving (SH) strategy for sequential BAI. SH splits the budget of arms into stages and eliminates the worst half in each stage. This allows the algorithm to make more pulls of the promising arms. We show that a naive extension of this strategy can perform poorly if the scaling function is poor and propose an alternative algorithm that eliminates arms at a rate determined by the scaling function.

2. Description of the Environment

We begin by describing the bandit environment which will be used in both settings. There are n arms, denoted $[n] = \{1, \dots, n\}$. When we pull arm $i \in [n]$, we observe a reward from a 1-sub-Gaussian distribution with expectation $\mu_i \in [0,1]$. The goal is to identify the best arm, i.e., we wish to find $\mathrm{argmax}_i\, \mu_i$. We will assume that the best arm is unique, and, for ease of exposition, that the arms are ordered in decreasing order. Therefore, $\mu_1 > \mu_2 \geq \cdots \geq \mu_n$. We define the *gaps* Δ_i as follows:

$$\Delta_1 = \mu_1 - \mu_2, \qquad \Delta_i = \mu_1 - \mu_i,\ \forall\, i \geq 2. \tag{1}$$

Deviating from prior work on BAI, we assume that we have access to a divisible resource which is to be used to execute the arm pulls. The time taken to execute a single pull using a fraction α of this resource is given by $\lambda(1/\alpha)$. Here, $\lambda : \mathbb{R}_+ \to \mathbb{R}_+$ is an application-specific *scaling function* which is assumed to be known. In particular, this implies that the time taken to execute $m \in \mathbb{N}$ arm pulls by *evenly* dividing the entire resource is $\lambda(m)$; this interpretation will be useful in understanding our assumptions going forward. It is reasonable to assume that scaling characteristics are known as they can either be modeled analytically (Zahedi et al., 2018), or can be profiled from historical experiments (Venkataraman et al., 2016; Liaw et al., 2019). If λ is unknown, we believe that there are significant limitations on what can be achieved in this setting. Prior work in the operations research literature studying scheduling in distributed systems also assumes that scaling characteristics are known (Berg et al., 2018; 2020).

While λ depends on the application, we will make some assumptions to model practical use cases. First, λ is an increasing function with $\lambda(0) = 0$, which simply states that executing more arm pulls requires more time and that zero pulls takes no time. Second, $\mathrm{Range}(\lambda) = \mathbb{R}_+$, which states that we cannot execute an unbounded number of pulls in a bounded amount of time; hence, $\lambda^{-1} : \mathbb{R}_+ \to \mathbb{R}_+$. Third, sublinear scaling—i.e., the diminishing returns of allocating more resources to a single pull—can be captured via the following assumption. For all $m_2 \geq m_1 > 0$, $\delta_1, \delta_2 > 0$,

$$\frac{\lambda(m_1 + \delta_1) - \lambda(m_1)}{\delta_1} \geq \frac{\lambda(m_2 + \delta_2) - \lambda(m_2)}{\delta_2}. \tag{2}$$

That is, the change in the average time taken to do additional arm pulls is smaller when there are more pulls already in the system. This assumption is equivalent to concavity. The above assumptions hold true for many choices for λ in practice. For instance, it is true for Amdahl's law and its variants that are popularly used to analytically model speedups in multi-core and distributed environments (Amdahl, 1967; Hill & Marty, 2008; Zahedi et al., 2018). They have also been found to be empirically true in other application-specific use cases (Venkataraman et al., 2016; Liaw et al., 2019). We also wish to mention that such concavity assumptions are common in the operations research literature when modeling diminishing returns (Berg et al., 2018; 2020).

Finally, as mentioned in Section 1, we will allow an algorithm to allocate its resources asynchronously, where a fraction of the resource may be allocated for one set of pulls to be completed at a certain throughput, and the remaining fraction for another set of pulls at a different throughput.

Thus, if both these sets of pulls start at the same time, they may finish at different times. As we will see shortly, at least in the fixed confidence setting, a synchronous algorithm may be sufficient as it matches a hardness result.

3. Fixed Confidence Setting

In the fixed confidence setting, the decision-maker is given a target failure probability δ, and must find the best arm with probability of error at most δ while minimizing the time required to do so. A common approach for sequential fixed-confidence BAI maintains confidence intervals for the mean values of the arms based on past observations; when the upper confidence bound of an arm falls below the lower confidence bound of any other arm, the algorithm eliminates that arm until eventually there is only one arm left (Jamieson et al., 2014; Gabillon et al., 2012).

There are two major challenges in applying such confidence-interval-based algorithms in our setting. They both arise due to nonlinear scaling and are a consequence of the trade-off between information accumulation and throughput. The first of these challenges is due to the fact that the arm mean values, or equivalently the gaps, are unknown. To illustrate this, consider an example with $n = 2$ arms and let $\Delta = \mu_1 - \mu_2$. It is well known that differentiating between these two 1-sub-Gaussian distributions requires $N_\Delta \approx \Delta^{-2}$ samples from each arm. For the purpose of this example[1], let us assume $N_\Delta = 32$ and assume that the scaling function is $\lambda(m) = m^{1/2}$. If we pull each arm one at a time, allocating all resources to each pull, this will require $2 \times N_\Delta = 64$ arm pulls and hence take time $2 \times N_\Delta \times \lambda(1) = 64$. If instead all 64 pulls are executed simultaneously with $1/64$ of the resources for each pull, this takes time $\lambda(64) = 8$, which is significantly less. However, knowing the right amount of parallelism requires information about the Δ value which is not available to the algorithm. If we parallelize more than necessary, we will be executing more arm pulls than necessary which can take more time. For instance, pulling each arm 512 times will take time $\lambda(1024) = 32$. Therefore, the first challenge for an algorithm is to choose the right amount of parallelism without knowledge of the gaps.

The second challenge arises from the fact that the ordering of the arms is unknown. As an example, consider a problem with three arms with $\mu_1 > \mu_2 > \mu_3$ and $\Delta_1 = \Delta_2 = \mu_1 - \mu_2$, and $\Delta_3 = \mu_1 - \mu_3$. Let $N_{\Delta_2}\left(\approx \Delta_2^{-2}\right)$, $N_{\Delta_3}\left(\approx \Delta_3^{-2}\right)$, and $N_{\Delta_1} = \max(N_{\Delta_2}, N_{\Delta_3}) = N_{\Delta_2}$ denote the number of pulls required from the second, third, and first arms, respectively. For simplicity, let us assume that the algorithm is aware of the N_{Δ_i} values, but does not know the ordering; i.e., which arms are the first, second, and third. Let $\lambda(m) = m^{1/2}$. First, let $N_{\Delta_1} = N_{\Delta_2} = 300$ and let $N_{\Delta_3} = 5$. As N_{Δ_3} is small, it is efficient to eliminate the third arm first and then differentiate between the top two arms. However, since the permutation of the arms is unknown, the algorithm will first pull each arm N_{Δ_3} times, eliminate the third arm, and then pull the remaining two arms $N_{\Delta_2} - N_{\Delta_3}$ times each. This takes time $\lambda(3N_{\Delta_3}) + \lambda(2(N_{\Delta_2} - N_{\Delta_3})) \approx 28.16$. Alternatively, consider a different example where $N_{\Delta_3} = 100$; i.e., the third arm is harder to distinguish. Here, first eliminating the third arm and then proceeding to the remaining arms takes time $\lambda(3N_{\Delta_3}) + \lambda(N_{\Delta_2} - N_{\Delta_3}) \approx 37.32$. However, simply pulling all three arms $3N_{\Delta_2}$ times, thus eliminating both the second and the third arm simultaneously, takes time $\lambda(3N_{\Delta_2}) = 30$ which is faster. This example illustrates that due to nonlinear scaling, it might be better to pull even the sub-optimal arms a larger number of times as the improved throughput may result in less time. This phenomenon becomes even more challenging when the Δ_i's are unknown and when there are multiple arms.

While the gaps and the ordering are also unknown in sequential and batch parallel BAI formulations, the above considerations are a direct consequence of nonlinear scaling when allocating resources, which is the focus of this work.

The second challenge motivates defining the following dynamic program for n arms which takes inputs $\{z_i\}_{i=2}^n \in \mathbb{R}^{n-1}$ with $z_2 \geq z_3 \geq \cdots \geq z_n$. First define $z_{n+1} = 0$ and $\mathcal{T}_{n+1}(\varnothing) = 0$. Then, recursively define $\mathcal{T}_j : \mathbb{R}^{n-j+1} \to \mathbb{R}_+$ for $j = n, n-1, \ldots, 2$ as follows:

$$\mathcal{T}_j\left(\{z_i\}_{i=j}^n\right) = \min_{k\in\{j,\ldots,n\}} \Big(\lambda(k(z_j - z_{k+1})) + \mathcal{T}_{k+1}\left(\{z_i\}_{i=k+1}^n\right)\Big). \tag{3}$$

This program is best understood from the point of view of a planner who knows the number of pulls necessary to eliminate arms $2, \ldots, n$, i.e., $\{z_i\}_{i=2}^n$, but is unaware of the ordering. The planner's goal is to optimally schedule the pulls so as to minimize the total time. Then, $\mathcal{T}_j$ is the minimum time taken to eliminate the worst $n - j + 1$ arms. For example, $\mathcal{T}_n = \lambda(nz_n)$, since the worst arm can be eliminated by pulling all n arms z_n times. Similarly, for $\mathcal{T}_j$, to eliminate the $n - j + 1$ worst arms, the planner may first eliminate the first $n - j$ arms in time $\mathcal{T}_{j+1}$, and then pull each of the remaining arms $(z_j - z_{j+1})$ times; alternatively, she may eliminate the first $n - j - 1$ arms in time $\mathcal{T}_{j+2}$, and then pull each of the remaining arms $(z_j - z_{j+2})$ times etc. There are $n - j + 1$ such options, and the planner will choose the one that takes the least time, as indicated in (3).

Given (3), we define $T^\star$ as follows. Recalling the definitions

[1]In this and the following example, we make simplifying assumptions in our treatment of N_Δ. For instance, we can usually only upper or lower bound N_Δ in terms of Δ^{-2}. Additionally, in adaptive settings, it may be a random variable. These simplifications are made to illustrate the challenges in our setup. Our subsequent analysis will be rigorous.

Algorithm 1 Adaptive Parallel Racing (APR)

```
1: Input: confidence $1-\delta$, (constant) parameter $\beta > 1$.
2: $r \leftarrow 1, N_i(0) \leftarrow 0, \forall i \in [n]$
3: $A_1 \leftarrow \varnothing,\ S_1 \leftarrow [n], t_1 = \lambda(n), q_1 = 1$
4: while $|A_r| = 0$ do
5:     Pull each arm in $S_r$, $q_r$ times, taking $\lambda(|S_r|q_r)$ time.
6:     $N_i(r) = N_i(r-1) + q_r \mathbb{1}\{i \in S_r\}\ \forall i \in [n]$
7:     $\hat{\mu}_{i,r} \leftarrow$ empirical mean of arm $i$ at round $r$, $\forall i \in [n]$
8:     $A_{r+1} \leftarrow$                                    # See (5)
           $\{i \in S_r | L_i(r,\delta) > \max_{j \in S_r \setminus \{i\}} U_j(r,\delta)\}$
9:     $S_{r+1} \leftarrow$
           $\{i \in S_r | U_i(r,\delta) > \max_{j \in S_r} L_j(r,\delta)\} \setminus A_{r+1}$
10:    $q_{r+1} \leftarrow \lfloor \lambda^{-1}(\beta^r t_1)/|S_{r+1}| \rfloor$
11:    $r \leftarrow r+1$.
12: Return $A_r$
```

of the gaps Δ_i from (1), we have:

$$T^\star = \mathcal{T}_2\left(\{\Delta_i^{-2}\}_{i=2}^n\right) \tag{4}$$

Shortly, we will prove upper and lower bounds which depend on the gaps via $T^\star$ and differ only in factors that are doubly logarithmic in the gaps and sub-polynomial in n.

3.1. An Algorithm for the Fixed Confidence Setting

Algorithm 1, called Adaptive Parallel Racing (APR), maintains confidence intervals for the mean values, and adaptively tunes the amount of parallelism by starting with a few pulls and then increasing the level of parallelism during execution. It operates over a sequence of rounds, indexed r, with the first round being allocated $t_1 = \lambda(n)$ time and round r being allocated $\beta^{r-1}t_1$ time. Here, β is an input parameter; the algorithm and the analysis work for any constant value of $\beta > 1$. When $\beta = 2$, this is akin to the doubling trick seen frequently in bandit settings, except here we apply it in time-scale, instead of the number of pulls.

The algorithm maintains a subset $S_r \subseteq [n]$ of surviving arms at round r. At the beginning of each round, it pulls each arm in S_r a total of q_r times, such that $|S_r|q_r$ is equal to the maximum number of arm pulls that can be executed in $\beta^{r-1}t_1$ time. At the end of each round, it constructs confidence intervals $\{(L_i, U_i)\}_{i\in[n]}$ for the mean values $\{\mu_i\}_{i\in[n]}$ as we will describe in (5). In lines 8 and 9, it updates the set of surviving arms by eliminating those arms whose upper confidence bounds are less than the highest lower confidence bound. This strategy implies that as the algorithm progresses, it spends more time per batch of pulls by favoring throughput over information accumulation.

Our confidence intervals are based on the law of the iterated logarithm and were first proposed by Jamieson et al. (2014). To describe them, let $N_i(r)$ denote the number of times we have pulled arm i in the first r rounds, and $\widehat{\mu}_i(r)$ denote the empirical mean of the samples collected. If we pulled each arm q_r times in round r, we have:

$$N_i(r) = \sum_{s=1}^r q_s \mathbb{1}\{i \in S_s\},\ \ \widehat{\mu}_i(r) = \frac{1}{N_i(r)}\sum_{s=1}^r\sum_{j=1}^{q_s} X_{i,s,j}.$$

Here, $\{X_{i,r,j}\}_{j=1}^{q_r}$ are the samples collected from arm i in round r. Next, let $D(N,\delta) = \sqrt{4\log(\log_2(2N)/\delta)/N}$ represent the uncertainty in using $\widehat{\mu}_{i,n}$ as an estimate for μ_i. Then, we can compute a confidence interval $(L_i(r,\delta), U_i(r,\delta))$ for arm i as follows:

$$\begin{aligned} L_i(r,\delta) &= \widehat{\mu}_i(r) - D(N_i(r), \sqrt{\delta/(6n)}) \\ U_i(r,\delta) &= \widehat{\mu}_i(r) + D(N_i(r), \sqrt{\delta/(6n)}). \end{aligned} \tag{5}$$

3.2. Upper Bound

We now state our main result for the proposed algorithm. Theorem 1 shows that Algorithm 1 finds the best arm with probability greater than $1-\delta$ and bounds its execution time.

Theorem 1. *Assume λ satisfies the assumptions in Section 2. Let $\beta \in (1, n]$. Let $\omega = \sqrt{\delta/(6n)}$ and define $\bar{N}_i := 1 + \lfloor 64\Delta_i^{-2}\log((2/\omega)\log_2(192\Delta_i^{-2}/\omega))\rfloor\ \forall i \in [n]$. Let $\mathcal{T}_2$ be as defined in (3). With probability at least $1-\delta$, Algorithm 1 outputs the best arm and the total execution time of the algorithm T satisfies:*

$$\begin{aligned} T \ &\leq\ 4\frac{\beta^{3+4\sqrt{\log_\beta(n)}}}{\beta-1}\mathcal{T}_2\left(\{\bar{N}_i\}_{i=2}^n\right) \\ &\leq 512\frac{\beta^{3+4\sqrt{\log_\beta(n)}}}{\beta-1} \\ &\qquad \left(\log\left(\frac{2}{\omega}\log_2\left(\frac{192}{\Delta_2^2\omega}\right)\right) \vee \frac{1}{64}\right)T^\star. \end{aligned}$$

While the first bound is tight, the second bound shows that that the runtime is bounded by essentially $T^\star \log(1/\delta)$; note $\log(1/\omega) \asymp \log(1/\delta)$. All other terms are small: $256\beta^3/(\beta-1)$ is a constant as β is a constant, the additional dependence on the gaps $\{\Delta_i\}_i$ and δ is doubly logarithmic, and $\beta^{4\sqrt{\log_\beta(n)}}$ is sub-polynomial, seen via the following simple calculation. Let $\alpha > 0$. Then,

$$\lim_{n\to\infty}\frac{\beta^{4\sqrt{\log_\beta(n)}}}{n^\alpha} = \beta^{\left(\lim_{n\to\infty} 4\sqrt{\log_\beta(n)} - \alpha\log_\beta(n)\right)} = 0.$$

Example 1. *Let us compare the above result with an algorithm which operates sequentially taking $\lambda(1)$ time for each pull. The sequential algorithm of (Jamieson et al., 2014) terminates in time at most $\lambda(1)(\bar{N}_2 + \sum_{i=2}^n \bar{N}_i) = \lambda(1)\sum_{i=2}^n i(\bar{N}_i - \bar{N}_{i+1})$ w.p. at least $1-\delta$.*

On the other hand, Algorithm 1 terminates in at most $C(n)\mathcal{T}\left(\{\bar{N}_i\}_{i=2}^n\right) \leq C(n)\sum_{i=2}^n \lambda(i(\bar{N}_i - \bar{N}_{i+1}))$, where $C(n) \in o(\mathrm{poly}(n))$. The second (looser) bound is obtained by considering one of the cases over which the minimum is taken in the DP. Additionally, by nonlinear scaling, we have $\lambda(n) \leq n\lambda(1)$ (Lemma 4); once again, when the scaling is poor this inequality is loose. Even with these two loose inequalities, we have a runtime bound of $C(n)\lambda(1)\sum_{i=2}^n i(\bar{N}_i - \bar{N}_{i+1})$ for Algorithm 1, which is not worse than the sequential version up to lower order terms.

In general, however, $\lambda(1)$ may be very large in our problem set up. We illustrate the advantages of using an adaptive parallel strategy via an example, for which we consider a scaling function of the form $\lambda(m) = m^q$, with $q \in [0,1]$, which satisfies the assumptions in Section 2. When $q = 1$, this corresponds to linear scaling, whereas when q approaches zero, the scaling becomes poor. Moreover, let us assume that the $(\bar{N}_i - \bar{N}_{i+1})$ is large for all i, so that we have $\mathcal{T}_2\left(\{\bar{N}_i\}_{i=2}^n\right) = \sum_{i=2}^n \left(i(\bar{N}_i - \bar{N}_{i+1})\right)^q$. By Theorem 1, ignoring constant and lower order factors, we have that Algorithm 1 terminates in at most $\sum_{i=2}^n \left(i(\bar{N}_i - \bar{N}_{i+1})\right)^q$ time, while the sequential algorithm terminates in $\sum_{i=2}^n i(\bar{N}_i - \bar{N}_{i+1})$ time. As q becomes smaller, the difference between these bounds becomes larger. Since the $\bar{N}_i - \bar{N}_{i+1}$ values are large, as per our example above, this difference is quite pronounced.

Proof Sketch: Designing algorithms in this setting is challenging, because adaptively finding the right level of parallelism can be expensive. In particular, there are two modes of failure. (i) Algorithms that take too long to "ramp-up", i.e. increase their parallelism, will spend too much time with low throughput, which can slow progress if many arm pulls are necessary to eliminate the next arms, (ii) Alternatively, algorithms that have too much parallelism could potentially overshoot, by over-pulling arms that could have been eliminated sooner. Algorithm 1 employs the doubling trick on *time* to avoid both of these situations: if all arms have not been eliminated, it multiplies the amount of time for the next round by a factor β. However, while the doubling trick on the number of pulls admits a fairly straightforward analysis in most bandit settings, the proof is significantly more challenging when it is used for a temporal criterion. The key technical challenge is to show that neither of the above two failure modes occurs frequently. We first show that the ramp-up time Algorithm 1 can be bounded within a constant factor of $4\frac{\beta^3}{\beta-1}$ of $T^\star$ in the runtime upper bound. The trickier scenario is to show that overshooting is not significant, and a naive analysis may result in a factor of n being produced. Our proof decomposes each stage, i.e., the rounds between arm eliminations, into two scenarios. In the first, the stage eliminates over a fraction f of the arms, and in the second the stage eliminates less than f. The first scenario can be expensive, as many arms that could have been eliminated earlier were over-pulled. However, we will carefully select f to bound the number of times this event can occur. The second scenario can happen many times, but we will show that each occurrence does not add too much to the runtime, again by carefully selecting f.

3.3. Lower Bound

We conclude this section by demonstrating that the quantity $T^\star$ in (4) is fundamental to this problem via a hardness result that matches the bound in Theorem 1. To state this theorem, let us define some quantities. For a set of n real-valued distributions $\theta = \{\theta_{(i)}\}_{i=1}^n$, let μ_j denote the j^{th} mean of these distributions in descending order; that is, $\mu_1 = \max_i \mathbb{E}_{\theta_{(i)}}[X] \geq \mu_2 \geq \cdots \geq \mu_{n-1} \geq \mu_n = \min_i \mathbb{E}_{\theta_{(i)}}[X]$. Let Θ denote the following class of sub-Gaussian distributions with a well-defined best arm:

$$\Theta = \Big\{ \{\theta_{(i)}\}_{i=1}^n \, : \mu_1 > \mu_2 \;\wedge \; \theta_{(j)} \text{ is 1-sub-Gaussian for all } j \Big\}.$$

Next, let $\mathcal{A}_{\delta,\lambda}$ denote the class of algorithms which can identify the best arm with probability at least $1-\delta$ for a given scaling function λ for *any* set of distributions in Θ. For an arbitrary algorithm $A \in \mathcal{A}_{\delta,\lambda}$ executed on a problem $\theta \in \Theta$, let $T(A,\theta)$ denote the time taken to stop. The theorem below provides a nonasymptotic lower bound on the expectation of $T(A,\theta)$.

Theorem 2. *Fix $\mu_1 > \mu_2 \geq \cdots \geq \mu_n$. Let Δ_i be as defined in (1), and $T^\star$ be as defined in (4). Assume λ satisfies the assumptions in Section 2, and additionally for some α_1, $\alpha_1 m \leq \lambda(m)$ for all $m \geq 1$. Then, there exists a set of distributions $\theta \in \Theta$ whose ordered mean values $\{\mu_i\}_i$ are such that for all $\delta \leq 0.15$,*

$$\inf_{A \in \mathcal{A}_{\delta,\lambda}} \mathbb{E}[T(A,\theta)] \geq 2c_\lambda \log\left(\frac{1}{2.4\delta}\right) T^\star.$$

Here, c_λ is a constant that depends only on λ.

The additional condition on λ captures the practical notion that each arm pull requires a minimum amount of work α_1 (fraction of resources $\times$ time) to execute: $\forall m \geq 0$, $\frac{1}{m}\lambda(m) \geq \alpha_1$. The above theorem states that any algorithm which identifies the best arm with probability at least $1-\delta$, has an expected runtime upper bounded by $\Omega(T^\star \log(1/\delta))$. Modulo lower order terms, the RHS of the above lower bound matches the RHS in the expression for the upper bound in Theorem 1. This demonstrates that $T^\star$ is a fundamental quantity in this setup.

It should be emphasized, however, that the upper and lower bounds are not entirely comparable. Theorem 1 is a high-probability result, guaranteeing that the best arm will be identified and the algorithm will terminate with probability at least $1-\delta$. In contrast, Theorem 2 lower bounds the *expected* run time of any algorithm that can identify the best arm with probability at least $1-\delta$. This discrepancy is common in fixed confidence BAI settings, with upper bounds tend to provide high-probability results for the number of arm pulls, while lower bounds are in expectation (e.g., Jamieson et al., 2014; Jun et al., 2016; Karnin et al., 2013; Kaufmann et al., 2016). Despite this discrepancy, no significant difference is yet to be observed, as is the case in this work. To our knowledge, only Kalyanakrishnan et al. (2012) upper bound the expected number of pulls.

Comparison to Prior Work: It is worth comparing the above results with prior work in the fixed confidence setting. First, for the upper bound, our algorithm uses similar confidence intervals to Jamieson et al. (2014) and Jun et al. (2016) who study the sequential and batch parallel settings respectively. However, unlike Jun et al. (2016), in our setting, we also need to choose the amount of resources to allocate for each pull. This determines the amount of parallelism to handle the tradeoff between information accumulation and throughput and depends on the scaling function λ. More importantly, much of our analysis in Appendix B is invested in managing this tradeoff which is not encountered in their settings. Similarly, for the lower bound, while we rely on some hardness results from Kaufmann et al. (2016), their result only captures the sample complexity of the problem and does not account for how resource allocation strategies may affect the time taken to collect those samples. The novelty of our work, relative to the above works, is further highlighted by the fact both the lower and upper bounds are given by a dynamic program which, as explained in the beginning of this section, characterises the tradeoff beween throughput and information accumulation.

4. Fixed Deadline Setting

While the primary focus of this paper is the fixed confidence setting, we also provide a simple algorithm for the fixed deadline version of this problem. Formally, we assume the same environment as described in Section 2, but now we have a time deadline T and wish to maximize the probability of finding the best arm under this deadline.

Our algorithm builds on the sequential-halving (SH) algorithm of Karnin et al. (2013). We begin with a brief review of SH in the sequential setting where we are given a budget on the *number of arm pulls*. SH divides this budget into $\log_2(n)$ equal stages. In the first stage, it pulls all arms an equal number of times and eliminates the bottom half of the arms, i.e., those arms whose empirical mean fall within in the bottom $n/2$ when ranked. It continues in this fashion for each subsequent stage, eliminating half of the surviving arms, until there is one arm left at the end of $\log_2(n)$ stages.

Now consider a naive extension of this algorithm that divides the *time budget* T into $\log_2(n)$ stages and pulls the surviving set of arms maximally before eliminating half of them—for simplicity, we will refer to this time-scale version as SH from now on. However, if the scaling function is sublinear, then splitting the time budget into larger stages can better take advantage of parallelism to execute more arm pulls and hence do better than an adaptive algorithm. For example, assume there are $n=4$ arms, let $\lambda(m)=m^{1/4}$, and let the deadline be $T=4$. If the budget was split into $\log_2(4)=2$ stages, where we eliminate two arms in the first stage and one in the second. Then, in each stage, $\lambda^{-1}(2)=16$ arm pulls can be executed with four pulls per arm in the first stage and eight pulls per arm in the second. If instead, we executed all arms in a single stage, then $\lambda^{-1}(4)=256$ arm pulls can be executed, with 64 pulls per arm. In the first strategy, attempting to accumulate information reduces throughput significantly. We should prefer the second option, as every arm is pulled more than it would be under the original SH strategy. In this section, we describe an algorithm that uses λ to balance between throughput and information accumulation by allocating resources to promising arms while maintaining high throughput.

Algorithm 2 Staged Sequential Halving (SSH)

```
1: Input: time budget $T$, number of stages to combine $k$
2: $m = \lambda^{-1}\left(T/\lceil \log_{2^k}(n)\rceil\right)$        #pulls per stage
3: $r_f \leftarrow \lceil \log_{2^k}(n)\rceil$, $S_0 \leftarrow [n]$
4: for $r \in \{0,\dots,r_f-1\}$ do
5:     Sample each arm $i \in S_r$, $t_r = \lfloor m/|S_r|\rfloor$ times.
6:     Let $S_{r+1}$ be the set of $\lceil |S_r|/2^k\rceil$ arms in $S_r$ with the
       highest empirical mean
7: Return The singleton element in $S_{r_f}$.
```

Our algorithm, outlined in Algorithm 2, takes a hyperparameter k. It splits T into $\lceil \log_{2^k}(n)\rceil$ stages, and pulls surviving arms maximally in each stage. Observe that using $k=1$ corresponds to running SH. It keeps the $\left\lceil \frac{1}{2^k}\right\rceil$ fraction of arms with highest empirical mean. Intuitively, increasing k will allow more time per stage, so throughput can be increased in each stage. However, increasing the time per stage will increase the time to obtain results and reduce opportunities to reallocate resources to promising arms. We propose using $k=k^\star$, obtained via the following optimization problem:

$$k^\star = \arg\max_{k\in\{1,\dots,\lceil \log_2(n)\rceil\}} x(k), \tag{6}$$

$$\text{s.t.} \quad x(k) = \left\lfloor \frac{\lambda^{-1}\left(\frac{T}{\lceil \log_{2^k}(n)\rceil}\right)}{2^{k\lceil \log_{2^k}(n)\rceil}(2^k-1)} \right\rfloor$$

$k^\star$ can be computed in practice since all parameters are known; moreover, this can be done inexpensively by evaluating $x(k)$ for all $\lceil \log_2(n)\rceil$ values for k. We show later that $k^\star$ increases as the scaling deteriorates, which means we should prioritize throughput over information accumulation.

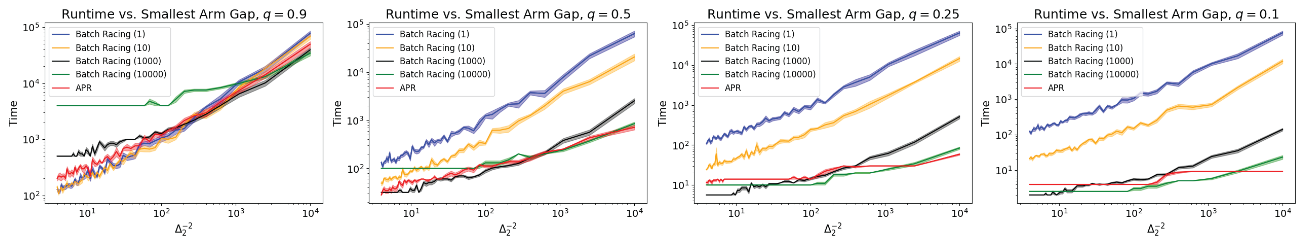

Figure 3. **Fixed confidence synthetic results:** We vary the first gap Δ_2 from 0.01 to 0.5 for $q \in \{0.9, 0.5, 0.25, 0.1\}$. A higher Δ_2^{-2} (right side) implies the problem is harder to solve as more arm pulls will be needed to separate the confidence bounds of the best arm from the rest. We present the mean and standard error across 10 runs for all experiments. We observe that Algorithm 1 (APR) is able to trade off between throughput and information accumulation effectively in most cases without additional hyperparameters, consistently performing near the best algorithms for each problem. The baseline algorithms perform well when their level of parallelism is well-suited to the scaling and Δ_i values of the problem, but are inconsistent across problems.

Ignoring rounding effects for simplicity, the numerator in (6) indicates the maximum number of pulls that can be completed during the stage. In the denominator, the first term is n, the number of arms. If a certain k' allows the algorithm to pull the arms at least $\sum_{i=0}^{k'-1} 2^i x(1)$ times in the first stage, then intuitively we should prefer $k = k'$ over $k = 1$, as all arms are pulled more times in the first stage of the former than any arm in the first k' stages of the latter. Observe now that this occurs if $x(k') \geq x(1)$. Extrapolating this argument, we have that we should choose the value k which maximizes $x(k)$. The following theorem bounds the probability of error for Algorithm 2.

Theorem 3. *Assume λ satisfies the assumptions in Section 2. Algorithm 2 run with some $k \in \mathbb{N}_+$ identifies the best arm in time at most T with probability at least:*

$$1 - 3\lceil \log_2(n) \rceil \exp\left(-\frac{n x(k)}{8 H_2}\right), \tag{7}$$

where $H_2 = \max_{i \neq 1} i\Delta_i^{-2}$ and Δ_i is as defined in (1).

In Appendix D.1, we show that the success probability of SH is obtained by setting $k = 1$ in the expression for $x(k)$, and, by choosing $k = k^\star$, this probability is always better than SH. Unfortunately, is is not possible to simplify this further without additional assumptions on λ. Therefore, in order to illustrate the gains in using Algorithm 2 over SH, we consider a specific example.

Example 2. *Let us consider λ of the form $\lambda(m) = m^q$ for $q \in (0, 1]$. The scaling becomes poor as q approaches zero (recall footnote 1). In Appendix D.2, we show that* SSH *is quantitatively better than SH; moreover, this difference is magnified as the scaling becomes poor, i.e., q approaches zero. We sketch the argument here. First we show,*

$$x(k) \geq \frac{1}{4^k n}\left(\frac{Tk}{2\log_2(n)}\right)^{1/q}.$$

Therefore, the number of arm pulls per round increases as q decreases and as k increases (after multiplying $x(k)$ by $2^k - 1$ to de-normalize). Let $p_{\mathrm{ssh,k}}$ be the error probability of combining k stages in Algorithm 2. Applying Theorem 3,

$$p_{\mathrm{ssh,k}} = C\lceil \log_2(n) \rceil \cdot \exp\left(-D \cdot \left(\frac{k^{1/q}}{4^k}\right)\right),$$

where C and D are constants that do not depend on k. When the scaling is sufficiently poor, i.e., when $1/q$ is large, we have $\left(\frac{k^{1/q}}{4^k}\right) > \left(\frac{k'^{1/q}}{4^{k'}}\right)$ for any $k > k'$. So, the exponentially decaying term will favor larger k as the scaling becomes more poor. So, as scaling deteriorates, the error probability is lower for larger k values, which prioritizes throughput over information accumulation and resource reallocation. However, if k is too large for a given q, then the 4^k term will dominate, which occurs when the algorithm isn't reallocating enough resources to promising arms.

5. Simulations

We present an experimental evaluation in the fixed confidence setting and fixed deadline settings. The experiments evaluate whether the proposed algorithms in each setting can effectively trade off between information accumulation and throughput for a wide range of possible scaling functions and problem instances. We present additional experiments in the fixed deadline setting and some experimental details in Appendix F.

5.1. Fixed Confidence Experiments

We compare APR to a set of baselines that do not adaptively manage parallelism on an array of simulation experiments.

Baselines: We compare APR to Batch Racing (Jun et al., 2016), which is an algorithm developed for *fixed batch* parallel BAI. We apply Batch-Racing(m) with different batch sizes m, but with a fixed amount used throughout one execution of the algorithm.

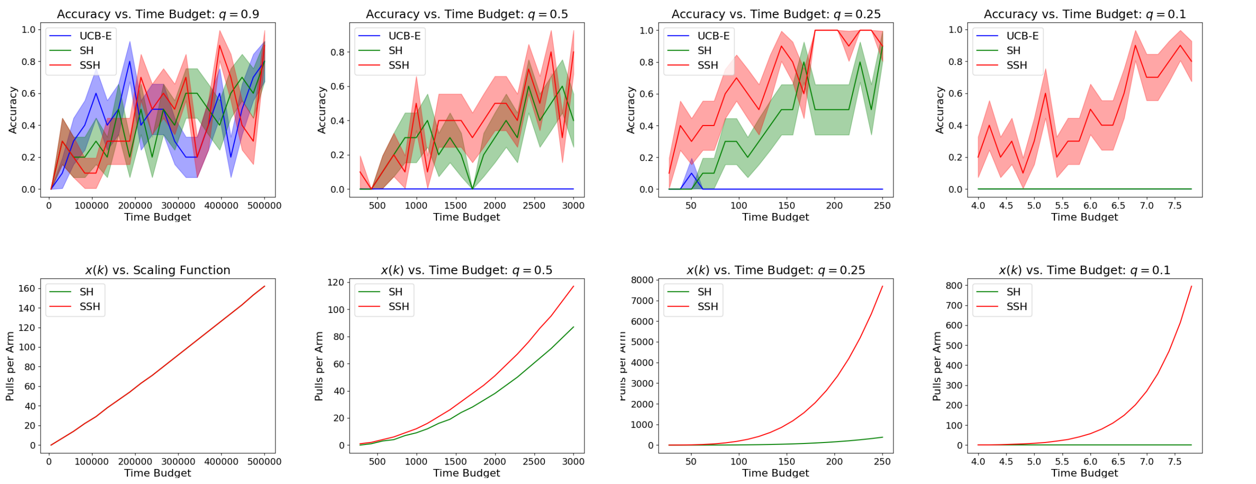

Figure 4. **Fixed deadline synthetic results:** We vary the time budget for $q \in \{0.1, 0.25, 0.5, 0.9\}$, running for 10 runs each and presenting the mean and standard error. We find that SSH (red) consistently outperforms SH (green) and UCB-E (blue). This difference is large when the scaling is poor (q is small) as SH does not pull each arm a sufficient number of times before each stage ends, and the cost of allocating all resources to individual pulls prevents UCB-E from making progress in comparable time. In contrast, SSH does well because it prioritizes high throughput when the scaling is poor. When $q = 0.1$, SH and UCB-E perform poorly whereas when $q \geq 0.5$, SSH has similar throughput as SH, resulting in similar performance.

Results: We evaluate Algorithm 1 with $\beta = 2$ on a synthetic domain consisting of $n = 16$ Bernoulli arms, $\delta = 0.1$, $\lambda(m) = m^q$, and with different values of $\Delta_2 = \mu_1 - \mu_2$. For a given Δ_2, the arm means have linearly interpolated values, $\mu_i = 0.9 - \Delta_2 - \frac{(0.9-0.1)(i-2)}{15}$ for $i \geq 2$ and $\mu_1 = 0.9$. This sets the best arm to 0.9 and sets the remaining means by linearly varying Δ_2 from 0.01 to 0.5. We evaluate all algorithms ten times on each setting. All algorithms almost always identify the best arm in the experiments. APR consistently does better than Batch Racing with a fixed amount of parallelism, and does as well as the best task-tuned batch size on the problem. See Figure 4.

5.2. Fixed Deadline Setting:

In this section, we present additional experiments evaluating SSH against baselines on a set of simulation experiments and on cosmological parameter estimation task.

Baselines: In the fixed deadline setting, we compare SSH to SH (Karnin et al., 2013) and the UCB-E algorithm (Audibert & Bubeck, 2010). UCB-E is maintains confidence bounds, and pulls the arm with the highest upper confidence bound. It unfortunately cannot be naturally extended to the parallel BAI setting, so it does not take advantage of parallelism.

Simulation Experiments We evaluate Algorithm 2 on a synthetic domain consisting of $n = 1024$ Bernoulli arms with means sampled uniformly from $[0, 1]$. We use a scaling function of the form $\lambda(m) = m^q$ for different choices of $q \in \{0.1, 0.25, 0.5, 0.9\}$. All settings are evaluated ten times, and we report the mean and standard error. We evaluate the accuracy of algorithms for varying time budgets. In Figure 4, we observe that Algorithm 2 (with $k = k^\star$) consistently matches or outperforms SH and UCB-E. This difference is more pronounced as the scaling is poor (smaller q).

6. Summary

We consider a novel setting for BAI, where arm pulls can be parallelized by dividing a fixed set of resources across them. While allocating more resources to a pull produces results sooner, this may result in lower throughput overall. So, algorithms must trade off between *information accumulation*, which allows us to invest resources in more promising candidates in future iterations, and *throughput*, which increases the overall number of samples. One avenue for future work is to study lower bounds for the fixed deadline setting.

Acknowledgements

This research was performed at the Real-Time Intelligent Secure Execution (RISE) Lab at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab. In addition to NSF CISE Expeditions Award CCF-1730628, this research is supported by gifts from Amazon Web Services, Ant Group, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware. We would like to thank Ellen Novoseller, Ashwin Balakrishna, Lawrence Chen, Jeffrey Ichnowski, and anonymous reviewers for their detailed feedback on this paper.

References

Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, 1967.

Audibert, J.-Y. and Bubeck, S. Best arm identification in multi-armed bandits. In *Conference on Learning Theory*, 2010.

Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 2003.

Berg, B., Dorsman, J.-P., and Harchol-Balter, M. Towards optimality in parallel job scheduling. In *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*, pp. 116–118, 2018.

Berg, B., Vesilo, R., and Harchol-Balter, M. hesrpt: Parallel scheduling to minimize mean slowdown. *Performance Evaluation*, pp. 102147, 2020.

Bubeck, S., Munos, R., and Stoltz, G. Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*, pp. 23–37. Springer, 2009.

Bubeck, S., Wang, T., and Viswanathan, N. Multiple identifications in multi-armed bandits. In *International Conference on Machine Learning*, pp. 258–265, 2013.

Dagan, Y. and Koby, C. A better resource allocation algorithm with semi-bandit feedback. In *Algorithmic Learning Theory*, pp. 268–320. PMLR, 2018.

Davis, T. M., Mörtsell, E., Sollerman, J., Becker, A. C., Blondin, S., Challis, P., Clocchiatti, A., Filippenko, A., Foley, R., Garnavich, P. M., et al. Scrutinizing exotic cosmological models using essence supernova data combined with other cosmological probes. *The Astrophysical Journal*, 666(2):716, 2007.

Desautels, T., Krause, A., and Burdick, J. W. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research*, 15:3873–3923, 2014.

Even-Dar, E., Mannor, S., and Mansour, Y. PAC bounds for multi-armed bandit and Markov decision processes. In *International Conference on Computational Learning Theory*, pp. 255–270. Springer, 2002.

Gabillon, V., Ghavamzadeh, M., and Lazaric, A. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Advances in Neural Information Processing Systems*, pp. 3212–3220, 2012.

Grover, A., Markov, T., Attia, P., Jin, N., Perkins, N., Cheong, B., Chen, M., Yang, Z., Harris, S., Chueh, W., et al. Best arm identification in multi-armed bandits with delayed feedback. *arXiv preprint arXiv:1803.10937*, 2018.

Hill, M. D. and Marty, M. R. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

Howard, S. R. and Ramdas, A. Sequential estimation of quantiles with applications to a/b-testing and best-arm identification. *arXiv preprint arXiv:1906.09712*, 2019.

Jamieson, K., Malloy, M., Nowak, R., and Bubeck, S. lil'UCB: An optimal exploration algorithm for multi-armed bandits. In *Conference on Learning Theory*, pp. 423–439, 2014.

Jun, K.-S., Jamieson, K. G., Nowak, R. D., and Zhu, X. Top arm identification in multi-armed bandits with batch arm pulls. In *AISTATS*, pp. 139–148, 2016.

Kalyanakrishnan, S. and Stone, P. Efficient selection of multiple bandit arms: Theory and practice. In *ICML*, volume 10, pp. 511–518, 2010.

Kalyanakrishnan, S., Tewari, A., Auer, P., and Stone, P. Pac subset selection in stochastic multi-armed bandits. In *ICML*, volume 12, pp. 655–662, 2012.

Kandasamy, K., Schneider, J., and Póczos, B. High dimensional Bayesian optimisation and bandits via additive models. In *International conference on machine learning*, pp. 295–304, 2015.

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Póczos, B. Parallelised Bayesian optimisation via Thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pp. 133–142, 2018.

Kandasamy, K., Vysyaraju, K. R., Neiswanger, W., Paria, B., Collins, C. R., Schneider, J., Poczos, B., and Xing, E. P. Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly. *arXiv preprint arXiv:1903.06694*, 2019.

Karnin, Z., Koren, T., and Somekh, O. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pp. 1238–1246, 2013.

Kaufmann, E., Cappé, O., and Garivier, A. On the complexity of best-arm identification in multi-armed bandit models. *The Journal of Machine Learning Research*, 17 (1):1–42, 2016.

Lattimore, T., Crammer, K., and Szepesvári, C. Optimal resource allocation with semi-bandit feedback. *arXiv preprint arXiv:1406.3840*, 2014.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Liaw, R., Bhardwaj, R., Dunlap, L., Zou, Y., Gonzalez, J. E., Stoica, I., and Tumanov, A. Hypersched: Dynamic resource reallocation for model development on a deadline. In *Proceedings of the ACM Symposium on Cloud Computing*, pp. 61–73, 2019.

Robbins, H. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 1952.

Russo, D. Simple Bayesian algorithms for best arm identification. In *Conference on Learning Theory*, pp. 1417–1418, 2016.

Shchigolev, V. Calculating luminosity distance versus redshift in flrw cosmology via homotopy perturbation method. *Gravitation and Cosmology*, 23(2):142–148, 2017.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 1933.

Venkataraman, S., Yang, Z., Franklin, M., Recht, B., and Stoica, I. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pp. 363–378, 2016.

Verma, A., Hanawal, M. K., Rajkumar, A., and Sankaran, R. Censored semi-bandits: A framework for resource allocation with censored feedback. *arXiv preprint arXiv:1909.01504*, 2019.

Wang, Z., Li, C., Jegelka, S., and Kohli, P. Batched high-dimensional Bayesian optimization via structural kernel learning. *arXiv preprint arXiv:1703.01973*, 2017.

Xing, Z., Eldon, D., Nelson, A., Eggert, W., Roelofs, M., Izacard, O., Glasser, A., Logan, N., Nazikian, R., Humphreys, D., et al. Automating kinetic equilibrium reconstruction for tokamak stability analysis. *Bulletin of the American Physical Society*, 64, 2019.

Zahedi, S. M., Llull, Q., and Lee, B. C. Amdahl's law in the datacenter era: a market for fair processor allocation. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–14. IEEE, 2018.