AC-Band: A Combinatorial Bandit-Based Approach to Algorithm Configuration

Jasmin Brandt[1], Elias Schede[2], Björn Haddenhorst[1], Viktor Bengs[3,4], Eyke Hüllermeier[3,4], Kevin Tierney[2]

[1]Department of Computer Science, Paderborn University, Germany
[2]Decision and Operation Technologies Group, Bielefeld University, Germany
[3]Institute of Informatics, LMU Munich, Germany
[4]Munich Center for Machine Learning (MCML), Germany
{jasmin.brandt, bjoernha}@upb.de, {elias.schede, kevin.tierney}@uni-bielefeld.de, {viktor.bengs, eyke}@ifi.lmu.de

Abstract

We study the algorithm configuration (AC) problem, in which one seeks to find an optimal parameter configuration of a given target algorithm in an automated way. Recently, there has been significant progress in designing AC approaches that satisfy strong theoretical guarantees. However, a significant gap still remains between the practical performance of these approaches and state-of-the-art heuristic methods. To this end, we introduce AC-Band, a general approach for the AC problem based on multi-armed bandits that provides theoretical guarantees while exhibiting strong practical performance. We show that AC-Band requires significantly less computation time than other AC approaches providing theoretical guarantees while still yielding high-quality configurations.

Introduction

Algorithm configuration (AC) is concerned with the task of automated search for a high-quality configuration (e.g., in terms of solution quality or runtime) of a given parameterized target algorithm. Parameterized algorithms are found in many different applications, including, for example, optimization (e.g., satisfiability (SAT) (Audemard and Simon 2018) or mixed-integer programming (MIP) solvers (IBM 2020)), simulation, and machine learning. Finding good configurations is a significant challenge for algorithm designers, as well as for users of such algorithms who may want to adjust the algorithm to perform well on data specific to their use case. Finding good configurations through trial and error by hand is a daunting task, hence automated AC methods have been developed on the basis of heuristics, such as ParamILS (Hutter, Hoos, and Stützle 2007; Hutter et al. 2009), GGA (Ansótegui, Sellmann, and Tierney 2009; Ansótegui et al. 2015), irace (Birattari et al. 2002; López-Ibáñez et al. 2016) or SMAC (Hutter, Hoos, and Leyton-Brown 2013, 2011).

While heuristic configurators have had great success at finding good configurations on a wide variety of target algorithms, they do not come with theoretical guarantees. To this end, the pioneering work of Kleinberg, Leyton-Brown, and Lucier (2017) proposes the first algorithm configurator with provable, theoretical guarantees on the (near-)optimality of the configuration returned. Further improvements and adjustments to those guarantees have followed (Weisz, György, and Szepesvári 2018, 2019; Kleinberg et al. 2019; Weisz et al. 2020). All of these works essentially consider the runtime as the performance objective and provide, in particular, an upper bound on the total runtime of the respective algorithm configurator that is (nearly) optimal in a worst-case sense.

Despite their appealing theoretical properties and the steady progress on their empirical performance, these approaches still cannot compete with heuristic approaches in terms of practical performance. The main issue of these theoretical approaches is that they are conservative in the process of discarding configurations from the pool of potential candidates, as pointed out in recent work (Weisz et al. 2020). This is indeed a key characteristic difference compared to the heuristic approaches, which discard configurations quite quickly after being used only on a couple of problem instances. From a practical point of view, this makes sense, as the risk of discarding all good configurations is generally lower than wasting lots of time looking at bad configurations.

In an attempt to further bridge the gap between heuristic configurators and theoretical approaches, we propose AC-Band, a general algorithm configurator inspired by the popular Hyperband (Li et al. 2016) approach based on multi-armed bandits (Lattimore and Szepesvári 2020). Hyperband is an algorithm for the hyperparameter optimization problem (HPO) (Yang and Shami 2020; Bischl et al. 2021b), which is essentially a subproblem of the general AC problem focusing on configuring solution quality of algorithms, with a particular focus on machine learning methods. While using HPO approaches for AC looks attractive at first, it is rather uncommon in practice due to the subtle differences between the two problems. These differences include potentially using runtime as a configuration metric and the existence of multiple *problem instances*, which are different settings or scenarios that an optimization method must solve.

Our suggested approach reconciles the basic idea behind the mechanism of Hyperband with the key characteristics of the AC problem and incorporates the advantageous property of discarding configurations quickly. This is achieved by first replacing the underlying bandit algorithm of Hyperband, Successive Halving (SH) (Karnin, Koren, and Somekh 2013), by a more general variant, Combinatorial Successive Elimination (CSE) (Brandt et al. 2022), and then carefully adapting the parameters of the iterative CSE calls over time. In contrast to SH, as well as to all other multi-armed bandit algorithms

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

for pure exploration with finite budget, CSE allows (i) to steer the aggressiveness of discarding arms (configurations in our terminology) from the set of consideration, (ii) to pull more than one arm simultaneously (run multiple configurations in parallel), and (iii) to work with observations either of quantitative or qualitative nature. As mentioned above, the first property seems to be of major importance in AC problems, but the other two properties will turn out to be particularly helpful as well. Indeed, (ii) obviously allows parallelization of the search process, while the generality regarding the nature of the observations in (iii) transfers quite naturally to the suggested method.

The interplay of the second and third properties allows us to instantiate AC-Band to obtain appealing practical performance compared to existing theoretical approaches regarding the total runtime. On the theoretical side, we derive (under mild assumptions on the underlying AC problem) that AC-Band is guaranteed to return a nearly optimal configuration with high probability if used on sufficiently many problem instances. Our theoretical result is quite general in the sense that the notion of optimality is not restricted to the runtime of the configurations, but also encompasses other target metrics such as solution quality or memory usage. The technical appendix can be found on arXiv[1].

Related Work

Theoretical advances in AC. The field of AC has grown to include many different methods and settings; we refer to Schede et al. (2022) for a full overview, especially with regard to the heuristic methods previously mentioned. Inspired by Kleinberg, Leyton-Brown, and Lucier (2017), who introduced *Structured Procrastination* together with a non-trivial worst-case runtime bound, more and more algorithms with even better theoretical guarantees with respect to the runtime have been proposed. *LeapsAndBounds* (Weisz, György, and Szepesvári 2018) tries to guess an upper bound on the optimal runtime by doubling the last failed guess, whereas *Structured Procrastination with confidence* (Kleinberg et al. 2019) works by delaying solving hard problem instances until later. Rather, it first runs the configurations with the smallest lower confidence bound of its mean runtime on instances that are easy to solve. Another recent method, *CapsAndRuns* (Weisz, György, and Szepesvári 2019), first estimates a timeout for each configuration and afterwards performs a Bernstein race over the configurations. As a follow up method, *Impatient-CapsAndRuns* (Weisz et al. 2020) uses a more aggressive elimination strategy by filtering unlikely optimal configurations in a preprocessing. Further theoretical progress has been made regarding the analysis of the estimation error in AC settings (Balcan et al. 2019; Balcan, Sandholm, and Vitercik 2020), the distribution of the computational budget (Liu et al. 2020) and the understanding of heuristic methods (Hall, Oliveto, and Sudholt 2019, 2020).

HPO. As a subset of AC, HPO involves setting the *hyperparameters* of algorithms, in particular machine learning approaches. The term hyperparameter differentiates parameters that change the behavior of the algorithm being configured from parameters that are induced or learned from data and are thus not set by a user. In contrast, AC focuses on configuring algorithms that solve instances of a dataset independently. We refer to Bischl et al. (2021a) for a full overview of HPO.

Bandit methods for AC. Classically, methods for multi-armed bandits (MAB) (Lai and Robbins 1985; Bubeck and Cesa-Bianchi 2012; Lattimore and Szepesvári 2020) are designed to find a good balance between exploration-exploitation of specific choice alternatives (e.g., configurations or hyperparameters). The pure exploration setting, however, has attracted much research interest as well (Bubeck, Munos, and Stoltz 2009; Karnin, Koren, and Somekh 2013; Aziz et al. 2018), especially for HPO (Jamieson and Talwalkar 2015; Li et al. 2016). However, up to now bandit algorithms making single choice alternative decisions have been leveraged, although the parallel execution of configurations (or hyperparameters) in the AC (or HPO) setting seems to be predetermined for the combinatorial bandit variant (Cesa-Bianchi and Lugosi 2012; Chen, Wang, and Yuan 2013; Jourdan et al. 2021). In light of this, the recently proposed CSE algorithm (Brandt et al. 2022) seems promising as a generalization of the popular SH approach.

Problem Formulation

We adopt the formulation of the problem as by Schede et al. (2022). Let $\mathcal{I}$ be the space of problem instances and $\mathcal{P}$ an unknown probability distribution over $\mathcal{I}$. Suppose $\mathcal{A}$ is an algorithm that can be run on any problem instance $i \in \mathcal{I}$, and has different parameters p_j coming from a known domain Θ_j for each $j \in \{1, \dots, m\}$. We call $\mathcal{A}$ the *target algorithm* and the Cartesian product of its parameter domains $\Theta = \Theta_1 \times \cdots \times \Theta_m$ the *configuration* or *search space* consisting of all feasible parameter *configurations*. For a configuration $\theta \in \Theta$, we denote by $\mathcal{A}_\theta$ an instantiation of the target algorithm $\mathcal{A}$ with configuration θ. Running the target algorithm $\mathcal{A}$ with configuration θ on a specific problem instance $i \in \mathcal{I}$ results in costs specified by an unknown, and possibly stochastic, function $c : \mathcal{I} \times \Theta \to \mathbb{R}$, i.e., $c(i, \theta)$ represents the costs of using $\mathcal{A}_\theta$ for problem instance i. Here, the costs can correspond to the runtime of $\mathcal{A}_\theta$ for i, but also to other relevant target metrics such as the solution quality or the memory usage.

Algorithm Configurator. The goal in algorithm configuration is, roughly speaking, to find a configuration that is optimal, or at least nearly optimal, with respect to the costs in a certain sense, which we specify below. The search for such configurations is achieved by designing an algorithm configurator $\mathcal{AC}$ that (i) selects specific configurations in Θ and (ii) runs them on some (perhaps randomly) chosen problem instances in $\mathcal{I}$. To this end, the algorithm configurator uses a statistic $s : \bigcup_{t \in \mathbb{N}} \mathbb{R}^t \to \mathbb{R}$ that maps the observed cost of a configuration θ used for a set of problem instances $i_1, \dots, i_t$ to a representative numerical value $s(c(i_1, \theta), \dots, c(i_t, \theta))$, that guides the search behavior of $\mathcal{AC}$. For example, s could be the arithmetic mean, i.e., $s(c(i_1, \theta), \dots, c(i_t, \theta)) = t^{-1} \sum_{s=1}^{t} c(i_s, \theta)$.

In this work, we are interested in algorithm configurators that can run several different configurations, up to a certain

[1] https://arxiv.org/abs/2212.00333

size k, in parallel on a selected problem instance. The algorithm configurator is given a fixed computational budget B, which represents the maximum number of such parallel runs and is set externally. For this purpose, let $\Theta_{[2,k]} = \{\tilde{\Theta} \subset \Theta \mid 2 \leq |\tilde{\Theta}| \leq k\}$ be the set of all possible subsets of parameter configurations that have at least size 2 and at most size k. Furthermore, let $\Theta_{[2,k]}(\theta) = \{\tilde{\Theta} \in \Theta_{[2,k]} \mid \theta \in \tilde{\Theta}\}$ be the set of all possible subsets of parameter configurations containing the configuration $\theta \in \Theta$. Note, that the observed cost $c(i, \theta)$ for running θ along with other configurations on an instance $i \in \mathcal{I}$ could depend on the respectively chosen configuration set $\tilde{\Theta} \in \Theta_{[2,k]}(\theta)$. For example, the algorithm configurator could stop the parallel solution process as soon as one of the configurations provides a solution, and set a default cost (penalty) for the configurations that did not complete. Hence, we write $c_{\tilde{\Theta}}$ for the cost function in the following to take this contingency into account. Finally, we introduce $s_{\theta|\tilde{\Theta}}(t) = s(c_{\tilde{\Theta}}(i_1, \theta), \ldots, c_{\tilde{\Theta}}(i_t, \theta))$ which is the statistic of θ after running it in parallel with the configurations in $\tilde{\Theta} \backslash \{\theta\}$ on t problem instances $i_1, \ldots, i_t$.

ϵ-optimal Configurations. Since the observed costs are potentially dependent on the chosen set of configurations to evaluate, we first introduce the following assumption on the limit behavior of the statistics.

$$(A1) \ : \ \forall \tilde{\Theta} \in \Theta_{[2,k]} \, \forall \theta \in \tilde{\Theta} \ : \ S_{\theta|\tilde{\Theta}} = \lim_{t \to \infty} s_{\theta|\tilde{\Theta}}(t) \text{ exists.}$$

In words, for each possible set of configurations the (possibly dependent) statistic of each configuration involved converges to some limit value if run on infinitely many problem instances. Recalling the example of s being the arithmetic mean, this is arguably a mild assumption and implicitly assumed by most approaches for AC problems, due to the considered i.i.d. setting. Since our assumption is more general, it would also allow considering non-stationary scenarios of AC.

The natural notion of an optimal configuration θ^* is a configuration that has the largest (configuration-set dependent) limit value over all configurations. Indeed, if we would replace $\Theta_{[2,k]}$ by the singleton sets of all configurations in Θ, this would correspond to the commonly used definition of the optimal configuration (see (Schede et al. 2022)), as the limit value would be then $\mathbb{E}[c(i, \theta)]$[2]. However, in our case this notion of optimality has two decisive drawbacks, as first of all such a θ^* may not exist. Moreover, even if it exists, the search for it might be hopeless as the configuration space is infinite (or very large). The latter issue arises in the "usual" AC problem scenario as well, and is resolved by relaxing the objective to finding a "good enough" configuration. We adapt this notion of near optimality by resorting to the definition of an ϵ-best arm from the preference-based bandit literature (Bengs et al. 2021). For some fixed relaxation parameter $\varepsilon > 0$, we call a configuration θ an *ϵ-best configuration* iff

$$\forall \tilde{\Theta} \in \Theta_{[2,k]}(\theta) \ : \ S_{\theta|\tilde{\Theta}} \geq S_{(1)|\tilde{\Theta}} - \epsilon, \tag{1}$$

where $S_{(1)|\tilde{\Theta}} \geq \ldots \geq S_{(|\tilde{\Theta}|)|\tilde{\Theta}}$ is the ordering of $\{S_{\theta|\tilde{\Theta}}\}_{\theta \in \tilde{\Theta}}$.

[2]The expectation is w.r.t. $\mathcal{P}$ and the possible randomness due to $\mathcal{A}_\theta$ and/or the cost generation.

Although we have relaxed the notion of optimality, finding ϵ-best configurations is still often like searching for a needle in a haystack. Hence, we need to ensure that there is a sufficiently high probability that an ϵ-best configuration is included in a large random sample set of Θ:

$$(A2) : \text{the proportion of } \epsilon\text{-best configurations is } \alpha \in (0, 1).$$

Note this assumption is once again inspired by the bandit literature dealing with infinitely many arms (de Heide et al. 2021). By fixing the probability for the non-occurrence of an ϵ-best configuration to some $\delta \in (0, 1)$, Assumption $(A2)$ ensures that a uniformly at random sampled set of configurations with size $N_{\alpha,\delta} = \lceil \log_{1-\alpha}(\delta) \rceil$ contains at least one ϵ-best configuration with probability at least $1 - \delta$.

Of course, an efficient algorithm configurator $\mathcal{AC}$ that aims to find an ϵ-best configuration θ^* cannot verify the condition $S_{\theta^*|\tilde{\Theta}} \geq S_{(1)|\tilde{\Theta}} - \epsilon$ for every possible query set $\tilde{\Theta} \in \Theta_{[2,k]}(\theta^*)$, in particular when the number of configurations, and thus the cardinality of $\Theta_{[2,k]}(\theta^*)$, is infinite. Instead, $\mathcal{AC}$ can only guarantee the above condition for a finite number of query sets and therefore it will always find a proxy for an ϵ-best configuration that does not have to be a true ϵ-best configuration. To guarantee that $\mathcal{AC}$ finds a true ϵ-best configuration with high probability, we introduce the following assumption.

$$(A3) : \forall M \in \mathbb{N}, \forall \theta \in \Theta :$$
$$\mathbb{P}\Big((\forall i \in [M] : S_{\theta|\tilde{\Theta}_i} \geq S_{(1)|\tilde{\Theta}_i} - \epsilon)$$
$$\Rightarrow \ (\forall \tilde{\Theta} \in \Theta_{[2,k]}(\theta) : S_{\theta|\tilde{\Theta}} \geq S_{(1)|\tilde{\Theta}} - \epsilon) \Big) \geq 1 - \psi(M),$$

where $\tilde{\Theta}_1, \ldots, \tilde{\Theta}_M \sim \text{Uniform}(\Theta_{[2,k]}(\theta))$ and $\psi : \mathbb{N} \to [0, 1]$ is a strictly monotone decreasing function. In words, the probability that a configuration θ is a global ϵ-best configuration increases with the number of (randomly chosen) configuration sets on which it is a local ϵ-best configuration, i.e., the characteristic condition in (1) is fulfilled.

Note that $(A3)$ is a high-level assumption on the difficulty of the underlying AC problem. The "easier" the problem, the steeper the form of ψ and vice versa. As we do not impose further assumptions on ψ other than monotonicity, our theoretical results below are valid for a variety of AC problems.

AC-Band Algorithm

The AC-Band algorithm consists of iterative calls to CSE (Brandt et al. 2022) that allow it to successively reduce the size of a candidate set of configurations. We first describe how CSE works, then elaborate on its use in the AC-Band approach.

Combinatorial Successive Elimination. CSE (Algorithm 1) is a combinatorial bandit algorithm that, given a finite set of arms (configurations), finds an optimal arm[3] defined similarly to (1) for $\epsilon = 0$ using only a limited amount of feedback observations (budget). To this end, CSE proceeds on a round-by-round basis, in each of which (i) the non-eliminated arms

[3]Its existence is simply assumed.

Algorithm 1: Combinatorial Successive Elimination (CSE)

Input: set of configurations $\tilde{\Theta}$ with $|\tilde{\Theta}| = n$, subset size $k \leq n$, budget B, $\rho \in (0, \log_2(k)]$, problem instances I with $|I| = B$ which can be partitioned into $R^{\rho,k,n} = \min_x\{g^{\circ x}(n) \leq k\} + \min_x\{f_\rho^{\circ x}(k) \leq 1\}$ problem instances I_r with $|I_r| = P_r^{\rho,k,n} \cdot b_r$ where $\left\{P_r^{\rho,k,n}\right\}_r = \left\{\left\lfloor n/k \left(f_\rho(k)/k\right)^{r-1}\right\rfloor\right\}_r$, and $g(x) = f_\rho(k) \cdot \lfloor x/k \rfloor + x \mod k$
Initialization: $\tilde{\Theta}_1 \leftarrow \tilde{\Theta}$, $r \leftarrow 1$

1: **while** $|\tilde{\Theta}_r| \geq k$ **do**
2: $\quad b_r \leftarrow \left\lfloor B/(P_r^{\rho,k,n} \cdot R^{\rho,k,n})\right\rfloor$, $J \leftarrow P_r^{\rho,k,n}$
3: $\quad \tilde{\Theta}_{r,1}, \tilde{\Theta}_{r,2}, \ldots, \tilde{\Theta}_{r,J} \leftarrow \mathrm{Partition}(\tilde{\Theta}_r, k)$
4: $\quad$ **if** $|\tilde{\Theta}_{r,J}| < k$ **then**
5: $\quad\quad \mathcal{R} \leftarrow \tilde{\Theta}_{r,J}$, $J \leftarrow J - 1$
6: $\quad$ **else**
7: $\quad\quad \mathcal{R} \leftarrow \emptyset$
8: $\quad$ **end if**
9: $\quad I_{r,1}, \ldots, I_{r,J} \leftarrow \mathrm{Partition}(I_r, b_r)$
10: $\quad \tilde{\Theta}_{r+1} \leftarrow \mathcal{R}$
11: $\quad$ **for** $j \in [J]$ **do**
12: $\quad\quad \mathcal{R} \leftarrow \mathrm{ArmElimination}(\tilde{\Theta}_{r,j}, b_r, f_\rho(|\tilde{\Theta}_{r,j}|), I_{r,j})$
13: $\quad\quad \tilde{\Theta}_{r+1} \leftarrow \tilde{\Theta}_{r+1} \cup \mathcal{R}$
14: $\quad$ **end for**
15: $\quad r \leftarrow r + 1$
16: **end while**
17: $\tilde{\Theta}_{r+1} \leftarrow \emptyset$
18: **while** $|\tilde{\Theta}_r| > 1$ **do**
19: $\quad \tilde{\Theta}_{r+1} \leftarrow \mathrm{ArmElimination}(\tilde{\Theta}_r, b_r, f_\rho(|\tilde{\Theta}_r|), I_r)$
20: $\quad r \leftarrow r + 1$
21: **end while**

Output: The remaining item in $\tilde{\Theta}_r$

Algorithm 2: ArmElimination$(\tilde{\Theta}', b, l, I')$

1: Use $\tilde{\Theta}'$ for b times on problem instances I'
2: For all $\theta \in \tilde{\Theta}'$, update $s_{\theta|\tilde{\Theta}'}(b)$
3: Choose an ordering $\theta_1, \ldots, \theta_{|\tilde{\Theta}'|}$ of $(s_{\theta|\tilde{\Theta}'}(b))_{\theta \in \tilde{\Theta}'}$
4: **Output:** $\{\theta_1, \ldots, \theta_l\}$

are partitioned into groups of a given size k (line 3) if possible (lines 4–8, 18–19) and (ii) feedback for each group is queried under a round-specific budget, which, once exhausted, leads to the elimination of a certain fraction of arms in each group (Algorithm 2 called in lines 12 and 19). Here, the feedback observed for an arm is mapped to a numerical value using a statistic s that indicates the (group-dependent) utility of the arm, and is used as the basis for elimination in each round. The fraction of eliminated arms is steered via a function $f_\rho : [k] \to [k]$ with $f_\rho(x) = \lfloor x/2^\rho \rfloor$, where the predetermined parameter $\rho \in (0, \log_2(k)]$ controls the aggressiveness of the elimination. A large value of ρ corresponds to a very aggressive elimination, retaining only the arm(s) with (the) highest statistics, while a small ρ eliminates only the arm(s) with the lowest statistics. The overall available budget is first split equally for each round, and then for all partitions in each round (line 2).

In light of the AC problem we are facing, *querying feedback for a group of arms* corresponds to running a subset of configurations in parallel on a problem instance, which results in observations in the form of costs. Moreover, we do not reuse a single problem instance for any parallel run so that the budget is in fact equal to the number of problem instances used. Accordingly, the overall budget of CSE corresponds to the number of problem instances used in total, which are split into disjoint problem instance sets of size b_r, i.e., the round-specific budget (line 9).

Since CSE initially assumes a finite set of arms and a fixed parameter ρ guiding the overall elimination aggressiveness, we face two trade-offs. The first is regarding the interplay between the number of initial arms and the round-wise budget:

(T1): If the initial number of configurations for CSE is small (large), the more (fewer) runs can be carried out on different instances, leading to potentially more reliable (unreliable) statistics, but only on a few (many) subsets of configurations.

The second trade-off arises through the interplay between the round-wise budget and the elimination aggressiveness:

(T2): If the elimination behavior of CSE is aggressive (conservative), then more (fewer) runs can be carried out on different instances, leading to potentially more reliable (unreliable) statistics, but only on a few (many) subsets of configurations.

The challenge now is to reconcile these two trade-offs and, above all, to take into account the specifics of AC problems.

Algorithm 3: AC-Band

Input: target algorithm $\mathcal{A}$, configuration space Θ, problem instance space $\mathcal{I}$, Budget B, subset size k, suboptimality ϵ of "good enough" configuration, proportion of ϵ-best configurations α, failure probability δ, $n_0 > \lceil \ln(\delta)/\ln(1-\alpha) \rceil \in \mathbb{N}$
Initialization: $E \leftarrow \lceil \log_2\left(n_0/n_0 - N_{\alpha,\delta}\right) \rceil$, $q \leftarrow 1 + k-1/E$
$C_1 \leftarrow \log_q(2)$, $C_2 \leftarrow 1 + \log_q\left(n_0 + \frac{4n_0}{n_0 - N_{\alpha,\delta}}\right)$,
$C_3 \leftarrow \lceil \log_q(k) \rceil$

1: sample $\theta_0 \in \Theta$
2: **for** $e \in [E]$ **do**
3: $\quad n_e = \lceil n_0/2^e \rceil + 1$, $\rho_e = \log_2\left(e+k-1/e\right)$,
4: $\quad$ sample $\theta_{e,1}, \ldots, \theta_{e,n_e-1} \in \Theta$
5: $\quad$ sample $I_e \subset \mathcal{I} \setminus \cup_{e'=1}^{e-1} I_{e'}$ with $|I_e| = B/c_e$, where $c_e = \frac{(C_1 E - (2^E - 1)(2C_1 - C_2 - C_3))2^e}{2^E(-eC_1 + C_2 + C_3)}$
6: $\quad \theta_e = \mathrm{CSE}(\{\theta_{e-1}, \theta_{e,1}, \ldots, \theta_{e,n_e-1}\}, k, |I_e|, \rho_e, I_e)$
7: **end for**

Output: θ_E

AC-Band. The design of AC-Band (Algorithm 3) seeks to find a good balance for both tradeoffs (T1) and (T2) by calling CSE iteratively. Initially, CSE is invoked with larger sets of configurations, and an aggressive elimination strategy is applied. Over time, the size of the candidate sets is successively reduced, and the aggressiveness of the elimination strategy is also gradually decreased. Roughly speaking, the idea is to have high exploration in the beginning, and thus more risk that good configurations are discarded, and become more and more careful towards the end.

More specifically, AC-Band proceeds in epochs $e \in$

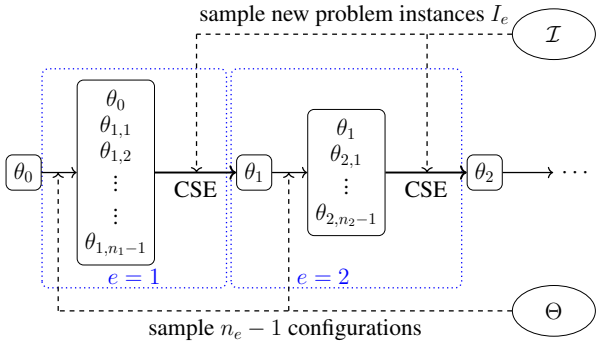

Figure 1: Illustration of AC-Band's solving process.

$\{1, \dots, E\}$, in each of which CSE is called on a specific set of problem instances using a specified degree of elimination aggressiveness ρ_e and a set of configurations of size n_e, with both ρ_e and n_e decreasing w.r.t. e (line 3). At the end of an epoch, i.e., when CSE terminates, a single high quality configuration among the considered set of configurations is returned (line 6). The set of configurations used in an epoch consists of the high quality configuration of the previous epoch, which is sampled randomly for the first epoch (line 1), and $n_e - 1$ randomly sampled configurations, which have not been considered before (line 4).

This epoch-wise procedure of AC-Band is depicted in Figure 1. Although AC-Band is similar in design to Hyperband in that it tries to find a good balance between specific trade-offs by successively invoking a bandit algorithm (CSE vs. SH), AC-Band differs in the way it defines the quality of the search objects[4]. Unlike Hyperband, we do not run configurations individually on problem instances and consider the cost of each configuration on its own, rather configurations are run in parallel and we consider potential interactions. Accordingly, we do not have one global quality value that we can compare for all configurations seen, but several at once.

The overall number of considered problem instances is B (the evaluation budget), which is a parameter that we analyze below. Besides the "usual" parameters of an algorithm configurator, i.e., the target algorithm $\mathcal{A}$, its configuration space Θ and the problem instance space $\mathcal{I}$, AC-Band requires:

- the maximum number of configurations that can be run in parallel k,
- some relevant summary statistic $s : \bigcup_{t \in \mathbb{N}} \mathbb{R}^t \to \mathbb{R}$ for the observed costs (see Problem Formulation),
- the theoretically motivated guarantee parameters $\epsilon > 0$, $\alpha \in (0, 1)$, and $\delta \in (0, 1)$ (see Problem Formulation)
- a reference size n_0 for the set of epoch-wise sampled configurations (must be $\geq \lceil \frac{\ln(\delta)}{\ln(1-\alpha)} \rceil$ for technical reasons).

With these parameters specified, AC-Band determines the overall number of epochs E and the sufficient number of problem instances B/c_e (line 5) for an epoch-wise CSE to return a high quality configuration (see Theoretical Guarantees). Moreover, the overall number of considered configurations is guaranteed to be at least $N_{\alpha,\delta} = \lceil \frac{\ln(\delta)}{\ln(1-\alpha)} \rceil$, which in light of the random sampling of the configurations ensures that at least one ϵ-best configuration is sampled with a probability of at least $1 - \delta$.

[4]Hyperband uses hyperparameters of machine learning models as search objects and AC-Band algorithm configurations.

Theoretical Guarantees

In Appendix C, we prove the following theoretical guarantee for AC-Band regarding the sufficient evaluation budget (or number of problem instances) to find an ϵ-best configuration with high probability w.r.t. $\mathcal{P}$ as well as the randomness invoked by AC-Band. For the proof, we need to extend the theoretical guarantees for CSE to the setting of finding ϵ-best configurations (see Appendix B).

Theorem 0.1. *Let R^e be the number of rounds of CSE in epoch $e \in \{1, \dots, E\}$, let C_1, C_2 and C_3 be as defined in Alg. 3 and further let $\mathbb{A}_r(\theta^*)$[5] be the partition in round r of CSE containing θ^* and*

$$\bar{\gamma}^{-1} = \max_{e \in [E], r \in [R^e]} \Big(1 + \bar{\gamma}^{-1}_{\mathbb{A}_r(\theta^*)}\Big(\max\Big\{\frac{\epsilon}{2}, \max_{r \in [R^e]} \frac{\Delta_{(f_\rho(|\mathbb{A}_r(\theta^*)|)+1)|\mathbb{A}_r(\theta^*)}}{2}\Big\}\Big)\Big),$$

$$\Delta_{(i)|\tilde{\Theta}} = S_{(1)|\tilde{\Theta}} - S_{(i)|\tilde{\Theta}},$$

$$\bar{\gamma}^{-1}_{\mathbb{A}_r(\theta^*)}(t) = \min_{\theta \in \mathbb{A}_r(\theta^*)} \inf_{t' \in \mathbb{N}} \{|s_{\theta|\mathbb{A}_r(\theta^*)}(t') - S_{\theta|\mathbb{A}_r(\theta^*)}| \leq t\}.$$

Under Assumptions $(A1)$–$(A3)$, Algorithm 3 used with a subset size k, an ϵ-best configuration proportion of α, and a failure probability of δ finds an ϵ-best configuration θ^* with probability at least $\min\{1 - \delta, 1 - \psi((R^E)^{-1})\}$ if

$$B \geq \bar{\gamma}^{-1} \cdot \frac{n_0}{k} \cdot \frac{C_1 E - (2^E - 1)(2C_1 - C_2 - C_3)}{2^E}.$$

Roughly speaking, AC-Band finds a near-optimal configuration with a probability depending on the allowed failure probability δ and the probability that a locally optimal configuration is also a globally optimal one (see Assumption (A3)) if the budget is large enough. The sufficient budget, in turn, is essentially driven by $\bar{\gamma}^{-1}$, which depends on the difficulty of the underlying AC problem by two characteristics: the maximal inverse convergence speed $\bar{\gamma}^{-1}_{\mathbb{A}_r(\theta^*)}$ of the used statistic s, and the maximal (halved) suboptimality gap Δ of the limits of the statistic between the best configuration and the best one that will be discarded from the query set θ^* is contained. The remaining terms of the sufficient budget can be computed explicitly once the theoretical guarantee parameters α and δ, as well as the subset size k, are fixed. The sufficient budget in dependence of the mentioned parameters is discussed and plotted in Appendix C.3.

Note that the theoretical guarantee in Theorem 0.1 is not directly comparable to the ones by the theoretical AC approaches (Kleinberg et al. 2019; Weisz, György, and Szepesvári 2018, 2019; Weisz et al. 2020). This is due to the major differences of our approach and the later ones on how we approach the AC problem. Indeed, we do not restrict ourselves to runtime as the target metric (or the costs), and

[5]$\mathbb{A}_r(\theta) = \emptyset$ if θ is not contained anymore in the set of active configurations in round r.

we also take possible dependencies in the parallel runs into account. As a consequence, the notion of near optimality of a configuration in the other works is tailored more towards runtimes in an absolute sense, i.e., without considering interaction effects, while ours is more general and in particular focusing on such interaction effects. Thus, the theoretical guarantee in Theorem 0.1 in the form of a sufficient evaluation budget to obtain a nearly optimal configuration is sensible, as we do not commit to a specific target metric.

Experiments

We examine AC-Band on several standard datasets for evaluating theoretical approaches for AC. We note that while these datasets refer exclusively to runtimes, AC-Band is applicable to other target metrics (see Section Problem Formulation). In our experiments, we address the following two research questions: **Q1**: Is AC-Band able to find high quality configurations in less CPU time than state-of-the-art AC methods with guarantees? **Q2**: How does AC-Band scale with k?

Datasets. We use one SAT and two MIP datasets that have been used before to assess theoretical works on AC (Weisz et al. 2020). Due to space constraints, we only consider one of the MIP datasets here, while the appendix also discusses the other. The SAT dataset contains precomputed runtimes of configurations of the MiniSat SAT solver (Eén and Sörensson 2003) obtained by solving instances that are generated using CNFuzzDD (Weisz, György, and Szepesvári 2018). The dataset contains runtimes for 948 configurations on 20118 instances. The MIP datasets curated by Weisz et al. (2020) are generated using an Empirical Performance Model (EPM)(Hutter et al. 2014). In particular, the EPM is trained on the CPLEX solver (IBM 2020) separately using instances from a combinatorial auction (Regions200 (Leyton-Brown, Pearson, and Shoham 2000)) and wildlife conservation (RCW (Ahmadizadeh et al. 2010)) dataset. The resulting model is used to predict runtimes for 2000 configurations and 50000 and 35640 new instances, respectively. Since all runtimes are precomputed (a timeout of 900 seconds is used), the evaluation of configuration-instance pairs only required table look-ups in our experiments.

Evaluation. To compare methods, we consider two metrics: (i) the accumulated CPU time needed by a method to find a configuration, and (ii) the percent gap to the best configuration. This second metric measures in percent how much more time the configuration returned by the method needs to solve all instances compared to the best overall configuration for the dataset. Smaller values indicate that the configuration found is closer to the best configuration in terms of runtime and the best configuration has a value of 0. This allows for comparing the quality between methods, as well as to determine how "far" a configuration is from the optimal one. In practical applications of AC, wall-clock time is often a bottleneck, and speeding up the process of finding a suitable configuration is the main focus. For these speedups, practitioners are (usually) willing to sacrifice configuration quality to a certain extent. The other theoretical works use the R^δ metric (note that δ has a different meaning in this work) to evaluate the quality of a returned configuration. This metric is a variation of the mean runtime, where the mean runtime of a configuration is only computed over the $(1-\delta)$ portion of instances with the lowest runtime. In real-world settings, we do not have the luxury of ignoring a part of the instance set, thus we do not view this metric as suitable for evaluating our approach. For the sake of completeness, we nevertheless report the R^δ values in Appendix D.

Initialization of AC-Band. Due to the generality of our approach, a summary statistic s that measures the quality of a configuration needs to be determined. In our case, the k configurations in a subset of CSE can be evaluated in parallel for an instance given that k CPUs are available. When running k configurations in parallel, time can be saved by stopping all remaining configuration runs as soon as the first configuration finishes on the given instance. Through this capping, we obtain right-censored feedback where a runtime is only observed for the "finisher". A statistic that is able to deal with this censored feedback is needed to avoid using an imputation technique that could potentially add bias to the feedback. In line with Brandt et al. (2022), we interpret the obtained feedback as a preference: the finishing configuration is preferred over the remaining, unfinished configurations. Once we have obtained these preferences for multiple instances, we can use a preference-based statistic such as the relative frequency to establish an ordering over the configurations in k. In particular, we count how many times a configuration finishes first over a set of instances[6].

Competitors. AC-Band is compared against ICAR, CAR++ (Weisz et al. 2020) and Hyperband (Li et al. 2016). At the moment, ICAR is the best performing AC method that comes with theoretical guarantees. We use the implementation provided by Weisz et al. (2020) with the same hyperparameter settings. Since AC-Band is inspired by Hyperband, we also adapt Hyperband for the AC setting for a comparison. Specifically, we set the parameter R of Hyperband such that it uses the same total budget (number of instances) as AC-Band. In addition, we use the average runtime over instances as the validation loss and consequently return the configuration with smallest average runtime. Finally, we set $s_{max} = \lceil(\log_\eta(n_{max}))\rceil$, adjust the calculation of r_i slightly to account for instances that were already seen, and try different values of η.

Choice of δ. Varying δ can lead to significantly different performance of AC-Band and other techniques. Due to space constraints, we only show results for two datasets for $\delta = 0.05$ since this setting has also been used in previous work (Weisz et al. 2020). We note, however, that other settings are just as valid, and therefore also provide results for $\delta = 0.01$ and additional datasets in Appendix D. In fact, using $\delta = 0.01$ can result in finding better configurations, albeit it is up to the user of the AC approach to decide which δ best suits their needs.

Q1 Figure 2 shows the CPU time used by each method and the percent gap to best configuration. With a small value of k, AC-Band lies on the Pareto front of percent gap versus CPU time for both datasets (for the third, Regions200, as well). This allows us to answer **Q1** in the affirmative. In particular, with $k = 2$ AC-Band is 72% percent faster than ICAR and

[6]Code: https://github.com/DOTBielefeld/ACBand

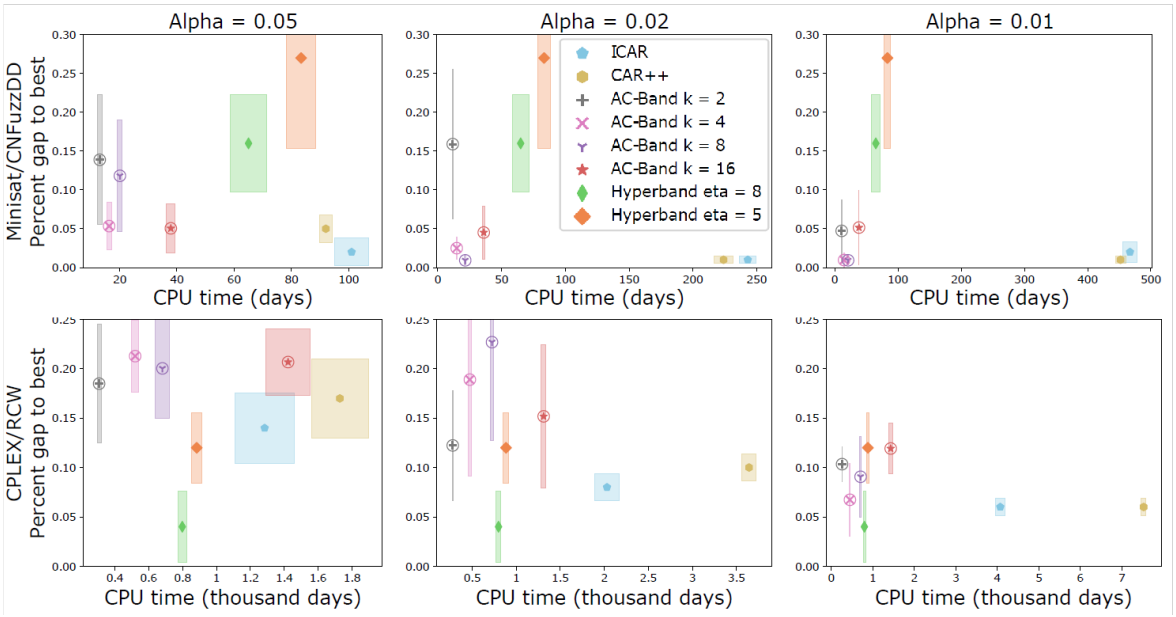

Figure 2: Mean CPU time and percent gap to best over 5 seeds for $\delta = 0.05$ and different α (columns) for AC-Band, ICAR, CAR++ and Hyperband on CNFuzzDD (top) and RCW (bottom). Circles indicate variants of AC-Band. Rectangles represent the standard error over the seeds. The number of configurations tried for CAR++: $\{97, 245, 492\}$, ICAR: $\{134, 351, 724\}$, AC-Band: $\{60, 153, 303\}$, Hyperband($\eta = 5$): $\{842\}$, Hyperband($\eta = 8$): $\{618\}$.

73% faster than Hyperband for $\delta = 0.05$ over all α and datasets, while providing configurations that are only 7% and 6% worse in terms of the gap to the best configuration. For most real-world applications, this is an acceptable trade-off in time versus quality. For all datasets, the percent gap to best decreases when AC-Band samples more configurations (smaller α). This increase in exploration does not lead to a significant increase in CPU time for a fixed k, since AC-Band still has the same budget, i.e., additional configurations are evaluated on fewer instances.

Hyperband samples more configurations in total than AC-Band, which helps it to achieve a better percent gap to best on datasets where the majority of configurations have a high mean runtime. On these datasets, only a few good configurations are present. This is the case for the RCW dataset (and the Regions200 dataset in the appendix) where only a few instances are needed to determine that a configuration should be discarded. On datasets where the runtime of configurations is more evenly distributed, such as the CNFuzzDD dataset, using too few instances may lead to discarding promising configurations early, giving AC-Band an edge by evaluating less configurations more thorough. Lastly, since Hyperband does not use capping, its CPU time deteriorates.

Q2 Our experiments clearly indicate that lower values of k are preferable. With $k = 2$, more CSE rounds are performed and thus the number of configurations decreases slower than with a higher k. With a larger k, the opposite occurs, and significant amounts of CPU time are expended with little information gain. However, note that higher k's have a lower wall-clock time, so a user would receive answers sooner.

Conclusion

In this paper we introduced AC-Band, a versatile approach for AC problems that comes with theoretical guarantees even for a range of target metrics in addition to runtime. We showed that AC-Band returns a nearly optimal configuration w.r.t. the target metric with high probability if used with a sufficient number of problem instances and the underlying AC problem satisfies some mild assumptions. In our experimental study, we considered an instantiation of AC-Band based on preference feedback, which generally leads to faster average CPU times than other theoretical approaches, while still returning a suitable configuration in the end.

Our results open up several possibilities for future work. First, it would be interesting to analyze AC-Band specifically for the case in which runtime is the relevant target metric and investigate whether a similar worst-case overall runtime guarantee can be derived as for the other theoretical approaches in this vein. Next, a theoretical as well as empirical analysis regarding the interplay between the explicit instantiation of AC-Band w.r.t. the underlying statistic s and the characteristics of the underlying AC would be desirable. In other words, on what types of AC problems does a specific instantiation of AC perform well or poorly? Also, our mild Assumption (A1) would even allow some leeway in the configuration or problem instance sampling strategy of the algorithm configurator, which is currently simply uniformly at random for AC-Band. Finally, real-world AC applications generally have only a handful of instances available, thus it would be advantageous to have strong theoretical guarantees even for scenarios without thousands of instances.

Acknowledgements

This research was supported by the research training group Dataninja (Trustworthy AI for Seamless Problem Solving: Next Generation Intelligence Joins Robust Data Analysis) funded by the German federal state of North Rhine-Westphalia and by the German Research Foundation (DFG) within the project "Online Preference Learning with Bandit Algorithms" (project no. 317046553).

References

Ahmadizadeh, K.; Dilkina, B.; Gomes, C. P.; and Sabharwal, A. 2010. An empirical study of optimization for maximizing diffusion in networks. In *International Conference on Principles and Practice of Constraint Programming*, 514–521. Springer.

Ansótegui, C.; Malitsky, Y.; Samulowitz, H.; Sellmann, M.; and Tierney, K. 2015. Model-Based Genetic Algorithms for Algorithm Configuration. In *IJCAI*.

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. 142–157.

Audemard, G.; and Simon, L. 2018. On the glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27(01): 1840001.

Aziz, M.; Anderton, J.; Kaufmann, E.; and Aslam, J. 2018. Pure exploration in infinitely-armed bandit models with fixed-confidence. In *Algorithmic Learning Theory*, 3–24. PMLR.

Balcan, M.; DeBlasio, D. F.; Dick, T.; Kingsford, C.; Sandholm, T.; and Vitercik, E. 2019. How much data is sufficient to learn high-performing algorithms? *CoRR*, abs/1908.02894.

Balcan, M.; Sandholm, T.; and Vitercik, E. 2020. Refined bounds for algorithm configuration: The knife-edge of dual class approximability. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, volume 119, 580–590. PMLR.

Bengs, V.; Busa-Fekete, R.; El Mesaoudi-Paul, A.; and Hüllermeier, E. 2021. Preference-based Online Learning with Dueling Bandits: A Survey. *Journal of Machine Learning Research*, 22: 1–108.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A Racing Algorithm for Configuring Metaheuristics. In *Gecco*, 11–18.

Bischl, B.; Binder, M.; Lang, M.; Pielok, T.; Richter, J.; Coors, S.; Thomas, J.; Ullmann, T.; Becker, M.; Boulesteix, A.-L.; Deng, D.; and Lindauer, M. 2021a. Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges.

Bischl, B.; Binder, M.; Lang, M.; Pielok, T.; Richter, J.; Coors, S.; Thomas, J.; Ullmann, T.; Becker, M.; Boulesteix, A.-L.; et al. 2021b. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv preprint arXiv:2107.05847*.

Brandt, J.; Haddenhorst, B.; Bengs, V.; and Hüllermeier, E. 2022. Finding Optimal Arms in Non-stochastic Combinatorial Bandits with Semi-bandit Feedback and Finite Budget.

Bubeck, S.; and Cesa-Bianchi, N. 2012. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *CoRR*, abs/1204.5721.

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

Cesa-Bianchi, N.; and Lugosi, G. 2012. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5): 1404–1422.

Chen, W.; Wang, Y.; and Yuan, Y. 2013. Combinatorial Multi-Armed Bandit: General Framework and Applications. In *Proceedings of the 30th International Conference on Machine Learning*, 151–159.

de Heide, R.; Cheshire, J.; Ménard, P.; and Carpentier, A. 2021. Bandits with many optimal arms. In *Advances in Neural Information Processing Systems*, volume 34, 22457–22469. Curran Associates, Inc.

Eén, N.; and Sörensson, N. 2003. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*, 502–518. Springer.

Hall, G. T.; Oliveto, P. S.; and Sudholt, D. 2019. On the Impact of the Cutoff Time on the Performance of Algorithm Configurators. *CoRR*, abs/1904.06230.

Hall, G. T.; Oliveto, P. S.; and Sudholt, D. 2020. Analysis of the Performance of Algorithm Configurators for Search Heuristics with Global Mutation Operators. *CoRR*, abs/2004.04519.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2013. Bayesian Optimization With Censored Response Data. *arXiv preprint arXiv:1310.1947*.

Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *J. Artif. Intell. Res. (JAIR)*, 36: 267–306.

Hutter, F.; Hoos, H.; and Stützle, T. 2007. Automatic Algorithm Configuration based on Local Search. In *Proceedings of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization*, 507–523. Springer Berlin Heidelberg.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206: 79–111.

IBM. 2020. ILOG CPLEX Optimization Studio 20.1.0: User's Manual.

Jamieson, K. G.; and Talwalkar, A. 2015. Non-stochastic Best Arm Identification and Hyperparameter Optimization. *CoRR*, abs/1502.07943.

Jourdan, M.; Mutnỳ, M.; Kirschner, J.; and Krause, A. 2021. Efficient pure exploration for combinatorial bandits with semi-bandit feedback. In *Algorithmic Learning Theory*, 805–849. PMLR.

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

Kleinberg, R.; Leyton-Brown, K.; and Lucier, B. 2017. Efficiency Through Procrastination: Approximately Optimal Algorithm Configuration with Runtime Guarantees. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, 2023–2031. ijcai.org.

Kleinberg, R.; Leyton-Brown, K.; Lucier, B.; and Graham, D. 2019. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. *arXiv preprint arXiv:1902.05454*.

Lai, T.; and Robbins, H. 1985. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1): 4–22.

Lattimore, T.; and Szepesvári, C. 2020. *Bandit algorithms*. Cambridge University Press.

Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, 66–76.

Li, L.; Jamieson, K. G.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. 2016. Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits. *CoRR*, abs/1603.06560.

Liu, S.; Tang, K.; Lei, Y.; and Yao, X. 2020. On Performance Estimation in Automatic Algorithm Configuration. In *The Thirty-Fourth Conference on Artificial Intelligence, AAAI*, 2384–2391. AAAI Press.

López-Ibáñez, M.; Dubois-Lacoste, J.; Pérez Cáceres, L.; Birattari, M.; and Stützle, T. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3: 43–58.

Schede, E.; Brandt, J.; Tornede, A.; Wever, M.; Bengs, V.; Hüllermeier, E.; and Tierney, K. 2022. A Survey of Methods for Automated Algorithm Configuration. *CoRR*, abs/2202.01651.

Weisz, G.; György, A.; Lin, W.; Graham, D. R.; Leyton-Brown, K.; Szepesvári, C.; and Lucier, B. 2020. ImpatientCapsAndRuns: Approximately Optimal Algorithm Configuration from an Infinite Pool. In *Annual Conference on Neural Information Processing Systems, NeurIPS*.

Weisz, G.; György, A.; and Szepesvári, C. 2018. LEAPSANDBOUNDS: A Method for Approximately Optimal Algorithm Configuration. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80 of *Proceedings of Machine Learning Research*, 5254–5262. PMLR.

Weisz, G.; György, A.; and Szepesvári, C. 2019. CapsAndRuns: An Improved Method for Approximately Optimal Algorithm Configuration. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, volume 97, 6707–6715. PMLR.

Yang, L.; and Shami, A. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415: 295–316.