Finding Optimal Arms in Non-stochastic Combinatorial Bandits with Applications in Algorithm Configuration

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

Abstract. We consider the combinatorial bandit problem with semi-bandit feedback under finite sampling budget, where the action is to choose a set of arms in a non-stochastic setting with subset-dependent feedback. We propose an algorithmic framework to solve it, which, additionally, can be leveraged for the algorithm configuration problem, where the goal is to find an optimal parameter configuration for a given target algorithm. We showcase that our introduced algorithm requires significantly less computation time than other existing theoretically-grounded approaches while still yielding high-quality configurations.

1 Introduction and Related Work

Algorithm configuration (AC) is the task of automated search for a high-quality parameter configuration of a parameterized algorithm for solving specific computational problems. Typical examples are optimization tasks such as Boolean satisfiability (SAT) or mixed-integer programming (MIP) problems. Finding good parameters by hand is a daunting or even infeasible task, but a good parameter configuration is crucial for achieving promising results of the used algorithm. Accordingly, there is a great need to automate this process, due to the growing demand for automated algorithmic solutions in industry and academia.

Traditional AC methods such as ParamILS, GGA, irace or SMAC, are based on heuristics to guide the search for promising configurations, however, there are also theoretically-grounded approaches such as LeapsAndBounds, Structured Procrastination and (I)CAR. For a full overview of existing AC methods we refer to [4]. The AC problem can also be seen as a pure-exploration multi-armed bandit (MAB) problem with an infinite number of arms. However, only the special case of hyperparameter optimization (HPO) has been tackled by means of MAB algorithms [3], while there is no existing approach for the more general AC problem. Although there exists a large body of literature on combinatorial bandits with preference-based feedback [1], there seems to be no existing work regarding a pure-exploration setting for numerical rewards and finite budget in combinatorial bandits.

2 Non-stochastic Combinatorial Bandits

Problem formulation. We consider a set of n arms $[n] := \{1, \dots, n\}$ and a fixed subset size k that represents the maximal number of arms we can compare in parallel. Also, we are given a fixed budget B, which corresponds to the maximum number of allowed comparisons. Each time a subset of arms $Q \subset [n]$ with $2 \leq |Q| \leq k$ is queried, we observe feedback $o_{i|Q}$ for each arm $i \in Q$, e.g. a numerical reward or a winner information. By means of a given statistic s, the observations of an arm $i \in Q$ are mapped to a numerical value representing the "attractiveness" of arm i. For example, s might be the arithmetic mean, i.e., $s_{i|Q}(t) = s(o_{i|Q}(1), \dots, o_{i|Q}(t)) := t^{-1} \sum_{t'=1}^{t} o_{i|Q}(t')$. We assume that s converges against a limit denoted by $S_{i|Q} \overset{t\to\infty}{\longrightarrow} s_{i|Q}(t)$. The goal is to find the *generalized Condorcet winner* (GCW) i^* defined as the arm that has the highest limit statistic in each subset in which it is included. We assume in our setting, that such a GCW exists, which is a common assumption in the literature [1].

Algorithm. For the considered problem we propose the Combinatorial Successive Elimination (CSE) algorithm in Algorithm 1. The main idea is to iteratively (in rounds) partition all arms that are still possible winner candidates into query sets of size k. Then, in each round, each query set is queried a carefully chosen number of times b_r before the worst $f(k)$ arms regarding the statistic s are discarded from each query set. The discarding function $f : [k] \to [k]$ has to be defined by the user, while b_r guarantees that the overall budget is (i) not exceeded, and (ii) distributed evenly among the queried subsets and the number of overall rounds.

It is worth mentioning that CSE offers a lot of generality: on the one side, due to the variable discard portion $f(k)$ and the statistic s, and, on the other side, due to the possibility to consider numerical rewards or preference-based feedback as observations. In addition, we provide theoretical guarantees for CSE in [2].

Algorithm 1 Combinatorial Successive Elimination

Input: arms $[n]$, subset size $k < n$, budget B, a function $f : [k] \to [k]$, sequence $\{P_r\}_r$ (number of partitions at round r), R (number of rounds)
Initialization: $\mathbb{A}_1 \leftarrow [n]$, $r \leftarrow 1$

while $|\mathbb{A}_r| \geq k$ **do**
  $b_r \leftarrow \lfloor B/(P_r R) \rfloor$, $J \leftarrow P_r$
  $\mathbb{A}_{r,1}, \mathbb{A}_{r,2}, \dots, \mathbb{A}_{r,J} \leftarrow \text{Partition}(\mathbb{A}_r, k)$
  if $|\mathbb{A}_{r,J}| < k$ **then**
    $\mathcal{R} \leftarrow \mathbb{A}_{r,J}$, $J \leftarrow J - 1$
  else
    $\mathcal{R} \leftarrow \emptyset$
  end if
  $\mathbb{A}_{r+1} \leftarrow \mathcal{R}$
  for $j \in [J]$ **do**
    $\mathcal{R} \leftarrow \text{Elimination}(\mathbb{A}_{r,j}, b_r, f(|\mathbb{A}_{r,j}|)), \mathbb{A}_{r+1} \leftarrow \mathbb{A}_{r+1} \cup \mathcal{R}$
  end for
  $r \leftarrow r + 1$
end while
$\mathbb{A}_{r+1} \leftarrow \emptyset$
while $|\mathbb{A}_r| > 1$ **do**
  $\mathbb{A}_{r+1} \leftarrow \text{Elimination}(\mathbb{A}_{r+1}, b_r, f(|\mathbb{A}_{r+1}|)), r \leftarrow r + 1$
end while

Output: The remaining item in $\mathbb{A}_r$

[1] Department of Computer Science, Paderborn University, Germany
[2] Decision and Operation Technologies Group, Bielefeld University, Germany
[3] Institute of Informatics, University of Munich (LMU), Germany
[4] Munich Center for Machine Learning (MCML)

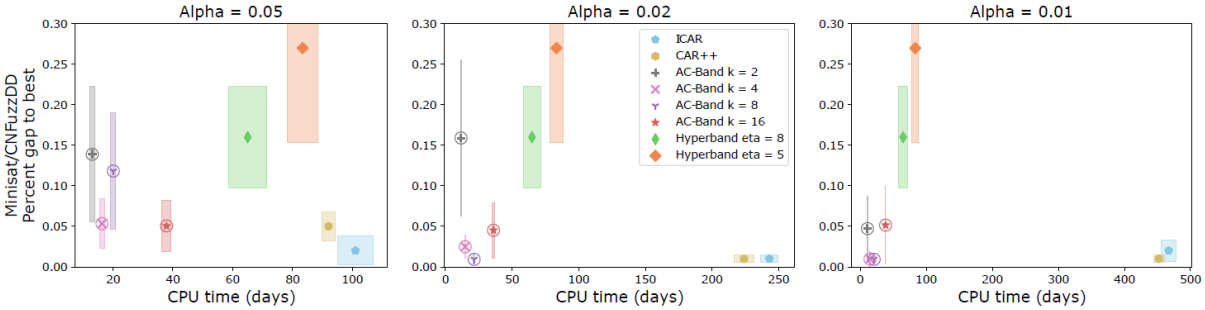

Figure 1. Mean CPU time and percent gap to best over 5 seeds for $\delta = 0.05$ and different α for AC-Band, ICAR, CAR++ and Hyperband on CNFuzzDD. Circles indicate variants of AC-Band. Rectangles represent the standard error over the seeds. The number of configurations tried for CAR++ are 97, for ICAR 134, for AC-Band 60, for Hyperband ($\eta = 5$) 842 and for Hyperband ($\eta = 8$) 618.

Algorithm 2 Elimination$(\mathbb{A}', b, l)$

Use $\mathbb{A}'$ for b times
For all $i \in \mathbb{A}'$, update $s_{i|\mathbb{A}'}(b)$
Choose an ordering $i_1, \ldots, i_{|\mathbb{A}'|}$ of $(s_{i|\mathbb{A}'}(b))_{i \in \mathbb{A}'}$
return $\{i_1, \ldots, i_l\}$

3 Algorithm Configuration

If we consider the parameter configurations for a parameterized algorithm for solving specific computational problems as the arms, we can apply the approach outlined in the previous section to the algorithm configuration problem.

Problem formulation. Let $\mathcal{A}$ be a parameterized algorithm and $\mathcal{I}$ be the problem instance space that consists of problems that algorithm $\mathcal{A}$ can solve. The *configuration space* of $\mathcal{A}$ is denoted by Θ, which consists of all feasible parameter configurations for $\mathcal{A}$. In addition, let $c : \mathcal{I} \times \Theta \to \mathbb{R}$ be an unknown cost function that measures the perhaps noisy cost $c(i, \theta)$ for running $\mathcal{A}$ with parameter configuration $\theta \in \Theta$ on an instance $i \in \mathcal{I}$. For example, this cost function can be the runtime of $\mathcal{A}$ with parameters θ for solving i or a numerical value that represents the quality of the returned solution by $\mathcal{A}$. The goal for an algorithm configurator is to find an ϵ-optimal parameter configuration θ^*. This is a configuration which, in each query set it is contained, has at most a difference of ϵ to the best configuration: $\forall Q \subset \Theta$ with $\theta^* \in Q$: $S_{\theta^*|Q} \geq S_{(1)|Q} - \epsilon$, where $S_{(1)|Q}$ denotes the best limit statistic in the query set Q.

Algorithm. To solve the AC problem we have to extend our CSE algorithm to handle an infinite number of possible configurations. To this end, we propose the AC-Band algorithm in Algo. 3. The main idea of AC-Band is to call CSE on an epoch-by-epoch basis, sampling new previously unused configurations of Θ in each epoch and using them as input to CSE. The proposed winner from CSE is then used as input for CSE in the next epoch together with new sampled configurations. The size of newly sampled configurations as well as the epoch-wise budget for CSE decrease geometrically over the epochs to reduce the degree of exploration of the configuration space.

4 Results

We examine AC-Band on a SAT dataset that has been used before to evaluate theoretical approaches for AC. We use the runtime as our cost function and consider two different metrics: (i) the CPU time needed by a method to return a promising configuration and (ii) *the percent gap to the best* which measures in percent how much more time the returned configuration needs to solve all instances compared to the best overall configuration for the dataset. We compare AC-Band against ICAR, CAR++ (see [4]) and Hyperband. AC-Band runs the configurations in each subset in parallel, stops as soon as the first one finishes and interprets the winner information as a preference feedback. As its statistic, it uses the relative frequency of how many times a configuration finishes first over a set of instances. We can see in Fig. 1 that AC-Band is up to 73% faster than ICAR and Hyperband while providing configurations that are only up to 7% worse. For most practical applications, this is an acceptable trade-off between time and quality.

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: $E = \lceil \log_2(\frac{n_0}{n_0 - N_{\alpha,\delta}}) \rceil$, $C_1 = \log_{1+\frac{k-1}{E}}(2)$,
$C_2 = 1 + \log_{1+\frac{k-1}{E}}(n_0 + 4\frac{n_0}{n_0 - N_{\alpha,\delta}})$, $C_3 = \lceil \log_{1+\frac{k-1}{E}}(k) \rceil$

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

5 Conclusion

We introduced a versatile approach for the combinatorial bandit problem and extended this to solve the AC problem. The experimental study showed that our proposed algorithm AC-Band can lead to a faster average CPU-time than state-of-the art theoretical AC methods, while still returning a suitable configuration.

REFERENCES

[1] Viktor Bengs, Róbert Busa-Fekete, Adil El Mesaoudi-Paul, and Eyke Hüllermeier, 'Preference-based online learning with dueling bandits: A survey', *Journal of Machine Learning Research*, (2021).
[2] Jasmin Brandt, Björn Haddenhorst, Viktor Bengs, and Eyke Hüllermeier, 'Finding optimal arms in non-stochastic combinatorial bandits with semi-bandit feedback and finite budget', *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, **36**, (2022).
[3] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar, 'Efficient hyperparameter optimization and infinitely many armed bandits', *CoRR*, (2016).
[4] Elias Schede, Jasmin Brandt, Alexander Tornede, Marcel Wever, Viktor Bengs, Eyke Hüllermeier, and Kevin Tierney, 'A survey of methods for automated algorithm configuration', *Journal of Artificial Intelligence Research (JAIR)*, (2022).