Contextual Preselection Methods in Pool-based Realtime Algorithm Configuration

Jasmin Brandt[1], Elias Schede[2], Shivam Sharma[1], Viktor Bengs[3,4], Eyke Hüllermeier[3,4] and 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, Germany

Abstract

Realtime algorithm configuration is concerned with the task of designing a dynamic algorithm configurator that observes sequentially arriving problem instances of an algorithmic problem class for which it selects suitable algorithm configurations (e.g., minimal runtime) of a specific target algorithm. The Contextual Preselection under the Plackett-Luce (CPPL) algorithm maintains a pool of configurations from which a set of algorithm configurations is selected that are run in parallel on the current problem instance. It uses the well-known UCB selection strategy from the bandit literature, while the pool of configurations is updated over time via a racing mechanism. In this paper, we investigate whether the performance of CPPL can be further improved by using different bandit-based selection strategies as well as a ranking-based strategy to update the candidate pool. Our experimental results show that replacing these components can indeed improve performance again significantly.

Keywords

Algorithm Configuration, Pool-based Realtime Algorithm Configuration, multi-armed bandits

CEUR-WS.org/Vol-3630/LWDA2023-paper43.pdf

1. Introduction

Algorithm Configuration (AC) is the task of automated search for a high-quality configuration of a given parameterized target algorithm. Such parameterized algorithms are often used to solve computationally hard problems like Boolean satisfiability problems (SAT), constraint satisfaction problems or vehicle routing problems. Finding good parameter configurations has a significant influence on the performance of these algorithms regarding their runtimes and/or their solution quality. However, searching for good configurations by hand is a complex or even infeasible task that motivates the development of automated AC methods.

The field of AC can be distinguished into two problem variants regarding the learning setting, namely offline learning and realtime algorithm configuration (RAC). The former corresponds to the batch learning scenario in machine learning: One has access to (a batch of) data in the form of triples $\{(i_j, \theta_j, m_j)_{j=1}^N\}$, where θ_j is the parameter configuration of the parameterized solver algorithm applied to the problem instance i_j, which results in the performance m_j. Here,

LWDA'23: Lernen, Wissen, Daten, Analysen. October 09–11, 2023, Marburg, Germany
jasmin.brandt@upb.de (J. Brandt); elias.schede@uni-bielefeld.de (E. Schede); sshivam95@gmail.com (S. Sharma); viktor.bengs@ifi.lmu.de (V. Bengs); eyke@ifi.lmu.de (E. Hüllermeier); kevin.tierney@uni-bielefeld.de (K. Tierney)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)
CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

performance can refer to runtime, solution quality, storage complexity, etc. The aim is thus to find a parameter configuration that generalizes across the whole problem instance distribution.

In contrast, the RAC problem is an online learning problem: There is generally no access to a data set in advance, and problem instances arrive sequentially for which parameters must be chosen in real time. For example, imagine a logistics company that is constantly faced with solving vehicle routing problems (VRPs), each of which requiring a high-quality solution as quickly as possible. The goal is thus to choose the best parameter configuration for the current problem instance in each time step. This is a challenging task as the nature of the incoming problem instances, i.e., the distribution over the problem class, might change over time (drastically or gradually). In the VRP example, the choice could depend, for instance, on the current weather and traffic situation, both of which change over time.

Thanks to modern, multi-core computer architectures, the computation of a suitable solution can be done in parallel, i.e., one can apply several different parameter configurations to the current problem instance simultaneously. In light of this, it makes sense to maintain a pool of (temporarily) good parameter configurations that is updated from time to time to react to possible changes in the problem distribution. Current algorithm configurators for the RAC problem all use such an approach, but differ in terms of the key operations such as updating the pool or choosing configurations from the pool (see Chapter 7 in [1]).

The most recent approach for the RAC problem in [2] uses methodologies from the multi-armed bandit literature for these key operations. More specifically, their approach is based on the idea that once a parameter configuration has found a solution, the others running in parallel can be stopped. This results in a preference observation in form of a top-1 ranking or winner feedback. Accordingly, the question of how to select a good subset from the pool can be viewed as a specific bandit problem, namely the preselection bandit [3]. In this learning scenario, the learner sequentially (pre-)selects a subset of objects (choice alternatives) for a user, from which the user selects a single object that has a certain probability of being the most preferred. The learner tries to find out the user's preferences from the provided feedback, and its main task corresponds to preselecting subsets that eventually lead to highly preferred choices of the user. Since context information, e.g., about the user or the objects, is often available, the preselection bandit setting has been extended in this respect in [4]. Considering the parameter configurations in the pool as the objects to be selected and the problem instances as the users, the CPPL method in [2] leverages the UCB strategy suggested in [4] to select configurations from the pool, while discarding configurations from the pool based on the racing principle [5].

Even though the UCB strategy is a well-established family of selection strategies in the bandit literature and the CPPL algorithm based on it also yields first promising results for the RAC problem, the question is whether other selection strategies can improve performance. For example, it has been empirically observed in various works that Bayesian selection strategies or stochastic selection strategies in general can improve the performance in the respective application scenarios, such as news article recommendation [6], planning problems [7] or online algorithm selection [8]. This motivates us to try one strategy at a time from the class of Bayesian or stochastic selection strategies recently proposed in the bandit literature, once in their context-dependent as well as context-free variants. In addition, we also investigate a new variant of performing the pool update within CPPL, i.e., removing configurations from the pool over time, which is based on the used ranking of objects in the respective selection strategies.

In our experimental study, we find that both strategies combined with the new variant of the pool update can indeed improve the performance of CPPL.

2. Problem Formulation

In the following, we define the *algorithm configuration* (AC) problem and adopt the notation in [1]. Let $\mathscr{A}$ be a parameterized algorithm, called the *target algorithm* in the following, and $\mathscr{I}$ the space of *problem instances* that $\mathscr{A}$ can solve. We denote the different internal parameters of $\mathscr{A}$ as p_i each with domain Θ_i for $i \in \{1, \dots, m\}$, such that $\Theta = \Theta_1 \times \cdots \times \Theta_m$ is the set of all feasible parameter configurations for $\mathscr{A}$ called the *configuration* or *search space*. Running $\mathscr{A}$ on a specific problem instance $i \in \mathscr{I}$ with a specific parameter configuration $\theta \in \Theta$ results in a cost $c(i, \theta)$ that is specified by an unknown and possibly stochastic cost function $c : \mathscr{I} \times \Theta \to \mathbb{R}$. The costs can, for example, correspond to the runtime of using $\mathscr{A}$ with configuration θ on instance i, but also to other metrics like the solution quality or the memory usage.

In the *realtime algorithm configuration* (RAC) scenario the problem instances arrive sequentially and the goal is to design a realtime algorithm configurator that chooses each time a configuration that performs best in terms of low cost each time. Unlike the classical variant of algorithm configuration, where a fixed but unknown distribution over the problem instances is assumed, in the RAC setting the distribution might change over time.

In the special case of *pool-based RAC*, the realtime algorithm configurator maintains a pool of configurations $P = \{\theta_1, \dots, \theta_n\}$ from which it selects then a subset of configurations that can be run in parallel on the current problem instance. The size of the subset is, for instance, determined by the number of cores of the available computing resources. Typically, the parallel computing process is stopped as soon as an algorithm configuration of the selected subset terminates. In this case, we get two types of information: Explicit numerical information, namely the cost of the first-terminated configuration, and implicit information, namely that the remaining configurations would have incurred higher costs. In the following, we shall focus on runtime as the costs, since this is the natural choice when using such a racing strategy with early stopping.

3. The CPPL Framework

In pool-based RAC the goal is to find an appropriate tradeoff between (i) gathering more information about the quality of the configurations that are contained in the pool and (ii) simultaneously selecting configurations from the pool that performed well in the past to ensure low costs on the recently seen problem instance. This motivates the use of multi-armed bandit (MAB) algorithms that are designed to solve exactly this exploration-exploitation tradeoff.

However, the observed runtimes of the configurations are right-censored since we usually use a timeout in the AC setting and in addition, we stop the run once the first configuration in the selected subset finishes and thus cannot observe the exact runtimes of all other configurations. These censored observations can introduce a bias in the estimated mean runtimes, see also [9] and [10]. To circumvent this problem, we can simply interpret the observations as a winner feedback and use preference-based bandit methods to deal with this.

3.1. Contextual Preselection Bandits

The above problem can be reduced to a *contextual preselection* problem, a sequential online decision-making problem. We are provided with context information $\mathbf{X}_t = (\mathbf{x}_{t,1}, \dots, \mathbf{x}_{t,n})$ in each time step $t \in \mathbb{N}$, where each $\mathbf{x}_{t,j}$ contains the contextual information for one of n different alternatives called arms that we denote in the following by their indices $[n] := \{1, \dots, n\}$. Using this context, the learner has to select a subset $S_t \subset [n]$ of size $|S_t| = k < n$ in each time step $t \in \mathbb{N}$. After the choice of the subset, the winner information among the selected arms is observed.

We consider the contextual preselection problem under the *Plackett-Luce* (PL) model. In this parameterized probability distribution on the set of all rankings over the arms, each arm $j \in [n]$ has an underlying utility or strength, represented by a parameter $v_j \in \mathbb{R}$. The probability under a PL model for a ranking $r : [n] \to [n]$ is

$$\mathbb{P}(r|v) = \prod_{i=1}^{n} \frac{v_{r^{-1}(i)}}{\sum_{j=i}^{n} v_{r^{-1}(j)}}.$$

To take the context information $\mathbf{x}_j \in \mathbb{R}^d$ for each arm $j \in [n]$ into account, we replace the utility for each arm $j \in [n]$ with the following log-linear function

$$v_j = v_j(\mathbf{X}) = \exp(\mathbf{w}^\top \mathbf{x}_j),$$

where $\mathbf{w} \in \mathbb{R}^d$ is a fixed but unknown weight vector. We can easily derive the log-likelihood function for a weight vector $\mathbf{w}$ for the observation that arm $\hat{\theta} \in S \subset [n]$ wins with context information $\mathbf{X}$ by

$$\mathscr{L}(\mathbf{w} \mid \hat{\theta}, S, \mathbf{X}) = \mathbf{w}^\top \mathbf{x}_{\hat{\theta}} - \log\left(\sum_{j \in S} \exp(\mathbf{w}^\top \mathbf{x}_j)\right).$$

Note that the probability to observe a winner under a PL model is the same as selecting an object out of a choice set in the well-known multinomial logit (MNL) model in discrete choice models [11].

The **goal** in our setting is to select, in each time step $t \in \mathbb{N}$, the subset $S_t \subset P$ of the recent pool of parameter configurations that contains the best possible configuration for the currently observed context $\mathbf{X}_t$. In addition, we regularly update the pool P by discarding the worst parameter configurations and fill up the newly available spots with new ones. In this way, we aim to to fill the pool P with increasingly good configurations from which our preselection strategy can select from in each time step. The overall goal is that the selected subset always contains a configuration whose cost function is as low as possible.

3.2. The Algorithmic Framework

A state-of-the-art algorithm to solve the pool-based RAC problem is the Contextual Preselection with Plackett-Luce (CPPL) algorithm [2]. It was the first step in the use of multi-armed bandit methods for RAC by successfully connecting RAC to the online contextual preselection bandit setting. Experimental results have shown that CPPL can effectively compete with the performance of other RAC methods like ReACTR [12]. The CPPL algorithm is shown in detail in Algorithm 1 and follows the racing principle. First, a problem instance $i \in \mathscr{I}$ is observed in

Algorithm 1 CPPL($\Theta, n, k, \mathcal{I}, f$)

1: Initialize n random parameterizations $P = \{\theta_1, \dots, \theta_n\} \subset \Theta$ and initialize $\widehat{\mathbf{w}}_1$ randomly
2: $\overline{\mathbf{w}}_{1,j} = \widehat{\mathbf{w}}_1$ for each $j \in P$
3: **for** $t = 1, 2, \dots$ **do**
4: Observe problem instance $i \in \mathcal{I}$
5: $\hat{v}_{t,j} \leftarrow \exp\left(\mathbf{x}_{t,j}^\top \overline{\mathbf{w}}_{t,j}\right)$, where $\mathbf{x}_{t,j} = \Phi(f(\theta_j), f(i))$ for all $j = 1, \dots, n$
6: $S_t \leftarrow$ **ARM_SELECTION()**
7: Run parameterizations of S_t to solve i, observe parameterization $\hat{\theta}_t \in S_t$ terminating first
8: $\overline{\mathbf{w}}_{t,j} \leftarrow$ **UPDATE_WEIGHTS()** for each $j \in P$
9: $K \leftarrow$ **DISCARD_SET()**
10: $P \leftarrow P \setminus K$
11: Generate $|K|$ new parameterizations using the genetic approach as described.
12: **end for**

line 4, and the contextual information is built with a joint feature map of the instance features $f(i)$ and the configuration features $f(\theta_j)$ for each $\theta_j \in P$. In our setting, the joint feature map is modeled with a second-degree polynomial consisting of all possible polynomial combinations of the features with degree less than or equal to 2, i.e. for $\mathbf{x} \in \mathbb{R}^r$ and $\mathbf{y} \in \mathbb{R}^c$ we have

$$\Phi(\mathbf{x}, \mathbf{y}) = (1, x_1, \dots, x_r, y_1, \dots, y_c, x_1^2, \dots, x_r^2, y_1^2, \dots, y_c^2, x_1 y_1, x_1 y_2, \dots, x_r y_c).$$

By means of the contextual preselection method, a subset from the pool is chosen in each time step in line 6. Each configuration in this subset is then executed in parallel according to the racing principle and the winner feedback is observed in line 7. After updating the estimation of the weight vector $\overline{\mathbf{w}}$ in line 8, the pool of configurations P is also updated in line 10 using genetic engineering. Strictly speaking, a uniform crossover of the two configurations that are ranked best by the model is executed. For diversification, single genes (parameters) are mutated and some configurations are generated at random with low probability. All newly generated configurations are ranked by the recently learned model and only the best individuals are kept in the pool.

The original version of CPPL uses an upper confidence bound (UCB) method for the contextual preselection of the configurations. UCB implements the principle of optimism in the face of uncertainty and solves the exploration-exploitation tradeoff by constructing (context-dependent) confidence intervals around the estimated (context-dependent) utilities and choosing the most optimistic objects according to the intervals. More specifically, the **ARM_SELECTION** strategy in line 6 to choose the subset S_t is

$$S_t \leftarrow \operatorname{argmax}_{S \subset P,\ |S|=k} \sum_{j \in S} \left(\hat{v}_{t,j} + c_{t,j}\right),$$

where $\hat{v}_{t,j}$ are the estimated utilities from line 5 for each configuration in P and $c_{t,j} = \omega \cdot I(t, d, \mathbf{x}_{t,j})$ are confidence bounds. Here, $\omega > 0$ is a suitable constant and I is a function depending on the gradient and the Hessian matrix of the log-likelihood function with respect to the observation at time step t (see [4] for details).

The estimator of the unknown weight vector is updated in **UPDATE_WEIGHTS** using the Polyak-Ruppert averaged stochastic gradient descent method as follows.

$$\overline{\mathbf{w}}_{t+1,j} \leftarrow t\frac{\overline{\mathbf{w}}_{t,j}}{t+1} + \frac{\widehat{\mathbf{w}}_{t+1,j}}{t+1} \text{ with } \widehat{\mathbf{w}}_{t+1,j} = \widehat{\mathbf{w}}_{t,j} + \gamma(t+1)^{-\alpha}\nabla\mathscr{L}\left(\widehat{\mathbf{w}}_{t,j} \mid \hat{\theta}_t, S_t, \mathbf{X}_t\right) \text{ for each } j \in S_t.$$

Moreover, **DISCARD_SET** in line 9 also depends on the confidence bounds. To be more precise, all configurations that have a smaller upper confidence bound than the lower confidence bound of another configuration in the pool are discarded from the pool P. Formally,

$$K \leftarrow \{\theta_i \in P \mid \exists \theta_j \neq \theta_i \text{ s.t. } \hat{v}_{t,j} - c_{t,j} \geq \hat{v}_{t,i} + c_{t,i}\}.$$

4. Modifying CPPL Components

The above CPPL algorithm already works well in practice, and the question is whether one can adjust the two crucial components, the selection strategy and the pool update, to achieve improvements. To this end for the former, we consider several bandit-based selection strategies that have been proposed recently. For the pool update, we use a ranking-based strategy to update the candidate pool based on the internal rankings of the configurations that all selection strategies maintain.

4.1. Preselection Strategies

In the following, we take a look at some recent multi-armed bandit algorithms as an alternative to the UCB approach for the subset selection in CPPL.

CoLSTIM A recently proposed method for regret minimization in a dueling bandit setting with context information is the CoLSTIM algorithm [13]. It assumes an underlying linear stochastic transitivity model (LSTM) with contextualized utilities for the winning probabilities, and even experimentally outperforms Bayesian methods. By means of a thresholded noise term $\epsilon_{t,j}$ sampled from an underlying perturbation distribution G at time step t for each configuration $j \in P$, the skill of each configuration is computed as the estimated utility value and a perturbed confidence width, generated by multiplying the estimated confidence bounds with the sampled perturbation variable. Introducing such a perturbation variable allows us to consider the accuracy of the imitated contextualized LSTM. Note that LSTMs are a general class of probability models for a winner feedback scenario that also contain MNL as a special case for the problem of choosing one object out of a choice set with the Gumbel distribution as the underlying distribution for the perturbation variable.

The detailed **ARM_SELECTION** strategy is shown in Algorithm 2. Note that we need a bounding constant C_{tresh} to threshold the generated perturbation variable for technical reasons. In our experiments, we choose a large value as a default for C_{thresh}. The estimator of the underlying weight for each configuration is computed similarly as in the UCB approach, so we can use an analogue update rule to **UPDATE_WEIGHTS**.

$$\overline{\mathbf{w}}_{t+1,j} \leftarrow t\frac{\overline{\mathbf{w}}_{t,j}}{t+1} + \frac{\widehat{\mathbf{w}}_{t+1,j}}{t+1} \text{ with } \widehat{\mathbf{w}}_{t+1,j} = \widehat{\mathbf{w}}_{t,j} + \gamma(t+1)^{-\alpha} \sum\nolimits_{s_t \in S_t} \nabla l\left(\widehat{\mathbf{w}}_{t,j} \mid \hat{\theta}_t, \{\hat{\theta}_t, s_t\}, \mathbf{X}_t\right) \quad \forall j \in S_t$$

Algorithm 2 CoLSTIM ARM_SELECTION

1: Sample $\tilde{\epsilon}_{t,j} \sim G \qquad \epsilon_{t,j} \leftarrow \min\left\{C_{tresh}, \max\{-C_{tresh}, \tilde{\epsilon}_{t,j}\}\right\} \quad \forall j \in P$
2: Choose $S_t \leftarrow \mathrm{argmax}_{S \subset P, |S|=k} \sum_{j \in S} \left(\hat{v}_{t,j} + \epsilon_{t,j} \cdot \sqrt{\mathbf{x}_{t,j} \mathbf{M}_t^{-1} \mathbf{x}_{t,j}}\right)$

Algorithm 3 TS-MNL UPDATE_WEIGHTS

1: $\mathbf{M}_{t+1} = \mathbf{M}_t + \sum_{j \in S_t} \mathbf{x}_{t,j} \mathbf{x}_{t,j}^T$
2: **for** $j \in S_t$ **do**
3: $\quad \widehat{\mathbf{w}}_{t+1,j} \leftarrow \widehat{\mathbf{w}}_{t,j} + \gamma(t+1)^{-\alpha} \nabla \mathscr{L}\left(\widehat{\mathbf{w}}_{t,j} \mid \hat{\theta}_t, S_t, \mathbf{X}_t\right), \quad \mathbf{w}_{t+1,j} \leftarrow t\frac{\mathbf{w}_{t,j}}{t+1} + \frac{\widehat{\mathbf{w}}_{t+1,j}}{t+1}$
4: $\quad$ Sample $\overline{\mathbf{w}}_{t+1,j} \sim \mathcal{N}(\mathbf{w}_{t+1,j}, \sigma^2 \mathbf{M}_{t+1}^{-1})$
5: **end for**

and $\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t + \sum_{j_t \in S_t} (\mathbf{x}_{t,w_t} - \mathbf{x}_{t,j_t})(\mathbf{x}_{t,w_t} - \mathbf{x}_{t,j_t})^T$,

where we initialize $\mathbf{M}_1 = \mathrm{diag}(1, \dots, 1)$. Note that we use another loss function l here to incorporate the underlying contextual linear stochastic transitivity model. More specifically, we assume that the probability that configuration θ is preferred over configuration λ for a given comparison function F is defined as $\mathbb{P}(1_{[\theta \succ \lambda]} = 1 | \mathbf{X}) = F(v_\theta(\mathbf{X}) - v_\lambda(\mathbf{X}))$. That leads to the following log-likelihood function for a weight vector $\mathbf{w}$ and an observation in the form of $(\theta, \{\theta, \lambda\}, \mathbf{X})$

$$l(\mathbf{w} \mid \theta, \{\theta, \lambda\}, \mathbf{X}) = 1_{[\theta \succ \lambda]} \log(F(\mathbf{w}^T(\mathbf{x}_\theta - \mathbf{x}_\lambda))) + (1 - 1_{[\theta \succ \lambda]}) \log(F(\mathbf{w}^T(\mathbf{x}_\theta - \mathbf{x}_\lambda))).$$

The features of both the problem instance and the configuration can also be ignored entirely by considering a suitable dummy encoding for the context vectors. We will call this selection strategy simply CoLSTIM-mo-f.

TS-MNL The multinomial logit (MNL) bandit problem is a class of bandit problems related to the preselection bandits, in which each object (arm) is equipped with a revenue and, accordingly, the goal is to maximize the cumulative expected revenue. Nevertheless, the algorithmic approaches for tackling the MNL bandits are quite similar to those in the preselection bandits. The Thompson Sampling (TS) MAB algorithm takes a Bayesian perspective, by assuming a prior loss distribution for every object (configuration), and in each time step selects an object according to its probability of being optimal. Thus, the exploration-exploitation trade-off is tackled through randomization coming from the posterior distribution. With this in mind, we modify the TS-MNL algorithm with optimistic sampling proposed in [14] so that it can be applied to a preselection bandit problem. We define an **ARM_SELECTION** strategy in which we do not need confidence bounds anymore as $S_t \leftarrow \mathrm{argmax}_{S \subset P,\ |S|=k} \sum_{j \in S} \hat{v}_{t,j}$.
In the procedure **UPDATE_WEIGHTS**, the covariance matrix $\mathbf{M}$ and the estimated mean of the weight vector $\mathbf{w}$ needs to be updated to learn the underlying multivariate normal distribution of the weight vector. After that, we can simply sample the weight vector $\overline{\mathbf{w}}$ that is used to compute the utilities $\hat{v}$ from the updated distribution. The procedure is given in detail in Algorithm 3.

Algorithm 4 ISS UPDATE_WEIGHTS

1: $W_{t+1,l} \leftarrow W_{t,l} + 1$, $F_{t+1,l} \leftarrow W_{t,l}$ for winning configuration l
2: $F_{t+1,j} \leftarrow F_{t,j} + \eta$, $W_{t+1,j} \leftarrow W_{t,j}$ for all $j \in S_t \setminus \{l\}$
3: Sample $\overline{w}_{t+1,j} \sim \text{Beta}(W_{t+1,j} + 1, F_{t+1,j} + 1)$ for each $j \in P$

In Algorithm 3, $\sigma > 0$ is a hyperparameter that needs to be chosen, for which smaller values (i.e., ≤ 2) are preferred. This is due to its similarity to the hyperparameter of the Thompson sampling approach in [8], where it has been empirically observed that smaller values of σ tend to lead to better results.

ISS Independent Self Sparring (ISS) is another algorithm in the spirit of Thompson Sampling that was proposed in [15] to solve the regret minimization task in a multi-dueling bandit scenario. However, it is a context-independent algorithm that simply keeps a belief on the pairwise winning probabilities of the underlying object (configurations), i.e., the probability that one object will win against another. For this purpose, a Beta prior distribution is used, as it is a conjugate prior and leads to a convenient update of the posterior distribution. In the **UPDATE_WEIGHTS** procedure, the Beta distribution for each configuration in the pool is updated as shown in Algorithm 4.
In the **ARM_SELECTION** strategy, we do not consider the contextual information anymore and simply use the samples from the Beta distribution as underlying utility. More specifically, we have $S_t \leftarrow \text{argmax}_{S \subset P,\ |S|=k} \sum_{j \in S} \overline{w}_{t,j}$.

4.2. Pool Update

Since we also compute confidence bounds in CoLSTIM, in principle we could also use the same condition to determine the **DISCARD_SET** as in the original framework of CPPL. Formally, this would look like

$$K \leftarrow \left\{ \theta_i \in P \mid \exists \theta_j \neq \theta_i \text{ s.t. } \hat{v}_{t,j} - \sqrt{\mathbf{x}_{t,j} \mathbf{M}_t^{-1} \mathbf{x}_{t,j}} \geq \hat{v}_{t,i} + \sqrt{\mathbf{x}_{t,i} \mathbf{M}_t^{-1} \mathbf{x}_{t,i}} \right\}.$$

However, such a racing strategy is not possible for TS-MNL and ISS. As an alternative, we can use a discard method based on rankings. In the ranking-based discard approach, a certain percentage $d \in [0, 1]$ of the configurations with the lowest rankings according to the estimated utilities, as determined by the model, are discarded after each iteration. More specifically, the **DISCARD_SET** in TS-MNL is selected as follows

$$K \leftarrow \text{argmin}_{\tilde{K} \subset P,\ |\tilde{K}| = \lfloor d \cdot |P| \rfloor} \sum_{j \in \tilde{K}} \hat{v}_{t,j}, \text{ resp. for ISS } K \leftarrow \text{argmin}_{\tilde{K} \subset P,\ |\tilde{K}| = \lfloor d \cdot |P| \rfloor} \sum_{j \in \tilde{K}} \overline{w}_{t,j}.$$

5. Related Work

The literature for offline AC methods can be partitioned into model-based, non-model-based and hybrid approaches. In non-model-based methods, racing is often used, in which a set of configurations race against each other on the same problem instances and poorly performing

ones are iteratively eliminated. Methods using this include F-Race [16], its successor Iterated F-Race [17] and GGA [18]. For a detailed overview over AC methods see [1].

The field of RAC developed to address the case of AC where the set of problem instances arrives sequentially. The ReACT algorithm [19] and its successor ReACTR [12] run a set of configurations in parallel on the instances to be solved. Both discard poorly performing configurations and replace them by new ones, but ReACTR incorporates in addition a ranking technique and a crossover mechanism to generate new configurations.

In the (basic) MAB problem, we consider a set of n different alternatives (called arms) from which one must be chosen in each time step. Next, a reward is observed for the pulled arm, generated from a fixed but unknown reward distribution for each arm. The goal is to maximize the reward or equivalently, to minimizing the regret that is defined as the difference between the sum of the rewards that we observe when we always pull the best arm and the sum of rewards we actually observe. Since MAB methods have already been successfully applied for algorithm selection [20] and hyperparameter optimization [21], [2] propose to also use them for RAC. To deal with the recently observed problem instances in the RAC problem, they consider a special case of MABs, called Contextual Bandits [22]. Moreover, they select in each time step a whole subset of arms respectively configurations which is covered by the preselection bandit scenario [3]. A combination of both the contextual bandits and the preselection bandits is considered in [4].

6. Experimental Comparison

We test the outlined bandit approaches on several synthetic datasets and target algorithms. We address the following research questions: **Q1**: How sensitive are CoLSTIM and TS-MNL to variations in their hyperparameter? **Q2**: How should configurations be discarded? **Q3**: Which bandit method provides the smallest overall cumulative runtime?

6.1. Experimental Setup

Datasets and Solvers We synthetically create three datasets, one for the CVRP using the CPLEX [23] solver, and two for SAT, using the Glucose [24] and Cadical [25] solvers. We seek instances that are neither trivial nor impossible to solve. To accomplish this, we execute each generated instance using the solvers' default configuration and retain only those instances that can be solved within a time range of 5 to 600 seconds. Each dataset contains a total of 1000 instances.

We introduce structural drift into all of our datasets. For SAT, we use the SHA-1 generator [26], which encodes attacks on the cryptographic hash function SHA-1 as a SAT problem. In the case of Glucose, we set the message-bits to 32 and the number of rounds to 22. To introduce drift, the hash-bits are increased by 8, starting from 64, every 78 instances. For Cadical, the message-bits are set to 0, and the number of rounds is set to 21. The hash-bits are increased by 5, starting from 115, every 100 instances. Glucose in version 4.2.1 has 16 continuous, 8 categorical, and 7 binary parameters. Cadical version 1.5.2 contains 30 continuous and 15 binary parameters. We use the SAT features listed in [27].

γ	0.001	0.01	0.1	1	10
Cadical	44.7	9.9	18.1	9.9	6.4
Glucose	118.0	91.9	83.4	88.9	88.2
CPLEX	142.4	68.1	55.8	39.0	47.9
Mean	101.7	57.6	52.4	46.9	49.6

(a) Variation in γ with $\alpha = 0.1$

α	0.001	0.01	0.1	1	10
Cadical	8.8	8.0	5.1	22.5	56.6
Glucose	71.0	92.7	59.9	117.3	117.3
CPLEX	40.8	43.6	43.1	80.1	85.4
Mean	40.2	48.1	36.0	73.3	86.4

(b) Variation in α with $\gamma = 1$

Table 1
Mean runtime in seconds for solving the first 200 instances of each dataset with CoLSTIM as selection mechanisms using different values for α and γ.

We create CVRP instances as in [28]. Customer locations are uniformly sampled at random within the unit square, and the demand at each customer is sampled uniformly between 1 and 10. We find that instances with $\geq$15 customers are hard for the default configuration of CPLEX to solve. To incorporate drift, we increment the vehicle capacity by one every 100 instances, starting from 12. CPLEX version 22.1.1.0 is employed, which has 35 continuous, 6 binary, and 54 categorical parameters. Instances features are implemented as in [29].

Initialization We compare CoLSTIM [13], its feature free variant (CoLSTIM-wo-f), CPPL [2], ISS [15], TS-MNL, ReACTR [12] and a random strategy against each other. For all methods, the pool size n is set to 32 and the subset size k to 16. To ensure a fair and consistent evaluation, all methods begin with an identical pool of configurations. To set the PCA dimensions used by methods to reduce the features, we follow [2] and use a value of 3 for the PCA dimension of instance features and 5 for the PCA dimension of algorithm features. To calibrate the PCA we use the features of the first 100 instances. For CoLSTIM we set G to be the standard Gumbel distribution. For ISS, we adopt the recommended value of $\eta = 3.5$. Similarly, for CPPL, we use the values $\alpha = 0.2$, $\gamma = 1$ and $\omega = 0.001$, as specified in [2]. As a baseline comparison, we include a random strategy where $k = 16$ configurations are randomly selected from the pool in each iteration. Finally, to control the execution time, a timeout of 300 seconds is set.

6.2. Q1: How sensitive are CoLSTIM and TS-MNL to variations in their hyperparameter?

The goal of this experiment is to identify appropriate hyperparameter values for CoLSTIM and TS-MNL. To this end, we run experiments with different values for α and γ for CoLSTIM and α, γ and σ for TS-MNL, where we use the first 200 instances of each dataset. The mean runtime is shown in Table 1a and Table 1b for CoLSTIM. Due to limited space, we are unable to display the results for TS-MNL [1]. The mean runtime encompasses both the time taken to solve an instance and the time needed for the model update after each solved instance. For CoLSTIM we achieve the best average runtime across our datasets by setting $\gamma = 1$ and $\alpha = 0.1$, although we note that these are determined independently from each other. Consequently, these values are selected for subsequent experiments for CoLSTIM. For TS-MNL we set $\gamma = 10$, $\alpha = 0.001$ and $\sigma = 0.0001$.

[1]Additional results and code are available under https://github.com/DOTBielefeld/colstim.

Discard	0	0.1	0.2	0.3	0.4	0.5	1	CB
CoLSTIM	57.89	45.37	36.25	43.89	38.52	42.57	97.91	44.53
CoLSTIM-wo-f	59.02	41.93	45.41	55.94	74.01	108.0	103.33	-
ISS	57.19	41.34	38.2	37.95	30.4	36.24	97.05	-
TS-MNL	56.15	55.92	63.78	71.48	76.39	76.4	91.92	-
CPPL	55.50	123.39	133.38	87.87	104.88	63.93	105.68	51.17
ReACTR	59.79	51.13	49.60	54.63	34.92	43.96	49.32	-
Random	77.52	79.61	78.59	80.14	78.66	80.62	81.35	-

Table 2
Mean runtime in seconds on the first 200 instances on CVRP 15. $[0, 0.1, 0.2, 0.3, 0.4, 0.5, 1]$: Fraction of pool that is discarded after each instance, CB: confidence bounds based discard.

6.3. Q2: How should configurations be discarded?

To determine a high-quality approach for discarding configurations from the pool, we conducted similar experiments as before. Specifically, we compare the racing strategies of CPPL and CoLSTIM, with the discard method based on rankings as described in section 4.2. Note that a discard value of 0 maintains a fixed pool of configurations across all instances, while a value of 1 completely replenishes the pool after each instance. We further note that ReACTR, ISS and TS-MNL use a ranking-based discard as a default strategy.

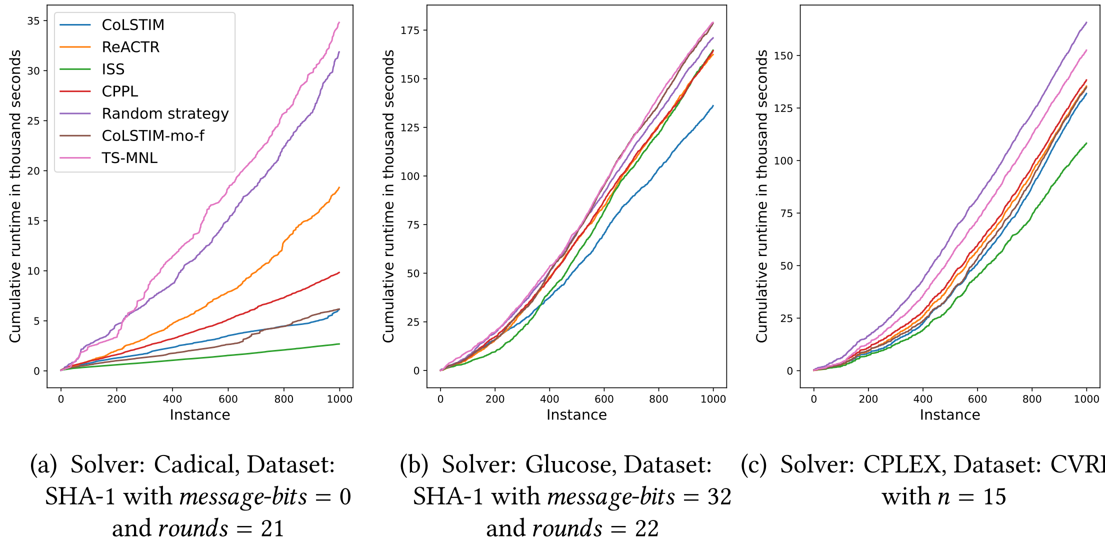
Table 2 presents the results for different discard strategies obtained on CVRP15 with CPLEX. Due to limited space, we are unable to display the results for the other two datasets here [2]. However, the results obtained from those datasets align with what is shown. Based on the findings, we set CoLSTIM to discard 20 percent of the pool after each instance, for ISS and ReACTR we set this value to 40 percent for the feature free versions of CoLSTIM and TS-MNL we use a value of 10 percent. For CPPL, we use the confidence bounds-based discard strategy.

6.4. Q3: Which bandit method provides the smallest cumulative runtime?

To determine which method performs best over all, we test the methods in question using all available instances of the respective datasets. Our experiments are performed on a 32 core AMD EPYC Milan 7763 2.45 GHz processor. We note that due to computational constraints we only perform one run per dataset and method. Throughout the figures, we present the accumulated runtime, which includes both the time taken to solve each instance and the time required for model updates. This approach reflects real-world scenarios where instances arrive rapidly, requiring prompt model updates. In addition, we display the average runtime across all instances and the number of timeouts.

Figure 1 showcases our results across the three benchmarks. While ISS does not take instance nor configuration features into account, it is able to outperform other methods that leverage this information on two of the three test cases. The observation that instance and configuration information may not boost performance in our set-up is strengthened when comparing CoLSTIM to CoLSTIM-mo-f. CoLSTIM is only clearly better on the Glucose dataset, which happens to be the most challenging test case among the three. This effect may very well be due to high

[2]We refer to the online appendix for additional results



(a) Solver: Cadical, Dataset: SHA-1 with *message-bits* = 0 and *rounds* = 21

(b) Solver: Glucose, Dataset: SHA-1 with *message-bits* = 32 and *rounds* = 22

(c) Solver: CPLEX, Dataset: CVRP with $n = 15$

Figure 1: Accumulated runtime (thousand seconds) over instances for each dataset, including both solver runtime and time taken for model updates.

information loss induced by the PCA. Among the tested methods, TS-MNL exhibits the weakest performance. It is constantly outperformed by all other bandit-based methods and for the Cadical and Glucose test cases even performs worse than the random strategy. Notably, all methods except for TS-MNL effectively handle the dataset drift, as evidenced by the absence of sudden spikes in the accumulated runtime at the drift points. The time taken to update the model after each instance is similar over all discussed bandit methods and datasets.

7. Conclusion

In this work, we established and compared different approaches for the contextualized preselection in the CPPL framework to automatically search for good parameter configurations in an online manner. Our experiments have shown that recent approaches like CoLSTIM or in particular ISS outperform the preselection method based on upper confidence bounds. In addition, our experiments show that features seem to be not necessary for a good performance. However, these results could also be reasoned by an inappropriate choice of the feature map or the PCA method for dimension reduction of the feature vectors.

In future work, we can further improve the joint feature map Φ and the dimensionality reduction of the features which was done with a simple PCA. Moreover, we can elaborate on how to create new configurations for the pool. Instead of genetic approaches, we can think about a kind of a continuity assumption in the space of configurations and do a guided search to find the optima.

Acknowledgments

This work was partially 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. The authors gratefully acknowledge the funding of this project by computing time provided by the Paderborn Center for Parallel Computing (PC2).

References

[1] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, K. Tierney, A Survey of Methods for Automated Algorithm Configuration, Journal of Artificial Intelligence Research 75 (2022) 425–487.

[2] A. El Mesaoudi-Paul, D. Weiß, V. Bengs, E. Hüllermeier, K. Tierney, Pool-Based Real-time Algorithm Configuration: A Preselection Bandit Approach, Springer International Publishing, 2020, p. 216–232.

[3] V. Bengs, E. Hüllermeier, Preselection Bandits, in: Proceedings of the 37th International Conference on Machine Learning, volume 119 of *Proceedings of Machine Learning Research*, PMLR, 2020, pp. 778–787.

[4] A. E. Mesaoudi-Paul, V. Bengs, E. Hüllermeier, Online Preselection with Context Information under the Plackett-Luce Model, CoRR abs/2002.04275 (2020). URL: https://arxiv.org/abs/2002.04275.

[5] O. Maron, A. W. Moore, Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation, in: Proceedings of Advances in Neural Information Processing Systems (NIPS), 1994, pp. 59–66.

[6] O. Chapelle, L. Li, An Empirical Evaluation of Thompson Sampling, Advances in Neural Information Processing Systems 24 (2011).

[7] A. Bai, F. Wu, X. Chen, Posterior sampling for Monte Carlo planning under uncertainty, Applied Intelligence 48 (2018) 4998–5018.

[8] A. Tornede, V. Bengs, E. Hüllermeier, Machine Learning for Online Algorithm Selection under Censored Feedback, Proceedings of the AAAI Conference on Artificial Intelligence 36 (2022) 10370–10380.

[9] J. Brandt, V. Bengs, B. Haddenhorst, E. Hüllermeier, Finding Optimal Arms in Non-stochastic Combinatorial Bandits with Semi-bandit Feedback and Finite Budget, in: Advances in Neural Information Processing Systems, volume 35, 2022, pp. 20621–20634.

[10] J. Brandt, E. Schede, B. Haddenhorst, V. Bengs, E. Hüllermeier, K. Tierney, AC-Band: A Combinatorial Bandit-Based Approach to Algorithm Configuration, Proceedings of the AAAI Conference on Artificial Intelligence 37 (2023) 12355–12363.

[11] K. E. Train, Discrete Choice Methods with Simulation, 2 ed., Cambridge University Press, 2009.

[12] T. Fitzgerald, Y. Malitsky, B. O'Sullivan, ReACTR: realtime Algorithm Configuration through Tournament Rankings, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, AAAI Press, 2015, pp. 304–310.

[13] V. Bengs, A. Saha, E. Hüllermeier, Stochastic Contextual Dueling Bandits under Linear Stochastic Transitivity Models, in: International Conference on Machine Learning, volume 162 of *Proceedings of Machine Learning Research*, PMLR, 2022, pp. 1764–1786.
[14] M.-h. Oh, G. Iyengar, Thompson Sampling for Multinomial Logit Contextual Bandits, Advances in Neural Information Processing Systems 32 (2019).
[15] Y. Sui, V. Zhuang, J. W. Burdick, Y. Yue, Multi-dueling Bandits with Dependent Arms, in: Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2017.
[16] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A Racing Algorithm for Configuring Metaheuristics, 2002, pp. 11–18.
[17] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-Race and Iterated F-Race: An Overview, Springer Berlin Heidelberg, 2009, pp. 311–336.
[18] C. Ansótegui, M. Sellmann, K. Tierney, A gender-based genetic algorithm for the automatic configuration of algorithms, in: I. P. Gent (Ed.), Principles and Practice of Constraint Programming - CP 2009, Springer Berlin Heidelberg, 2009, pp. 142–157.
[19] T. Fitzgerald, Y. Malitsky, B. O'Sullivan, K. Tierney, React: Real-Time Algorithm Configuration through Tournaments, in: Proceedings of the Seventh Annual Symposium on Combinatorial Search, AAAI Press, 2014.
[20] J. Wang, C. Tropper, Optimizing time warp simulation with reinforcement learning techniques, in: Proceedings Winter Simulation Conference, 2007, pp. 577–584.
[21] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization, Journal of Machine Learning Research 18 (2018) 1–52.
[22] V. Dani, T. P. Hayes, S. M. Kakade, Stochastic Linear Optimization under Bandit Feedback, in: Annual Conference Computational Learning Theory, 2008.
[23] IBM, ILOG CPLEX optimization studio 22.1.1: User's manual, 2022. URL: https://www.ibm.com/docs/en/icos/22.1.1?topic=cplex-optimizers.
[24] G. Audemard, L. Simon, On the glucose sat solver, International Journal on Artificial Intelligence Tools 27 (2018) 1840001.
[25] A. Biere, K. Fazekas, M. Fleury, M. Heisinger, CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020, in: T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions, volume B-2020-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2020, pp. 51–53.
[26] V. Nossum, Instance Generator for Encoding Preimage, Second-Preimage, and Collision Attacks on SHA-1, Proceedings of the SAT competition (2013) 119–120.
[27] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown, Features for sat, University of British Columbia,, Tech. Rep (2012).
[28] W. Kool, H. van Hoof, M. Welling, Attention, Learn to Solve Routing Problems!, in: International Conference on Learning Representations, 2019.
[29] J. Rasku, T. Kärkkäinen, N. Musliu, Feature extractors for describing vehicle routing problem instances, in: OASICS, Dagstuhl Publishing, 2016.