Contextual Preselection Bandits in Pool-Based Realtime Algorithm Configuration

Shivam Sharma

January 9, 2023

Department of Electrical Engineering,
Computer Science, and Mathematics
Warburger Straße 100
33098 Paderborn

Intelligent Systems Group (ISG)

Department of Computer Science
Heinz Nixdorf Institute
Intelligent Systems and Machine Learning (ISML)

Master's Thesis

Contextual Preselection Bandits in Pool-Based Realtime Algorithm Configuration

Shivam Sharma

1. Supervisor Dr. Theodor Lettmann
Department of Computer Science
Paderborn University

2. Supervisor Dr. Viktor Bengs
Institute of Informatics
Ludwig-Maximilians-Universität München

Advisor M.Sc. Jasmin Brandt

January 9, 2023

Shivam Sharma
Contextual Preselection Bandits in Pool-Based Realtime Algorithm Configuration
Master's Thesis, January 9, 2023
Supervisors: Dr. Theodor Lettmann and Dr. Viktor Bengs
Advisor: M.Sc. Jasmin Brandt

Intelligent Systems and Machine Learning (ISML)
Heinz Nixdorf Institute
Department of Computer Science
Paderborn University
Pohlweg 51
33098 Paderborn

Abstract

This work explored different parameter configuration selection strategies for the state-of-the-art online Algorithm Configuration algorithm: *Contextual Preselection under Plackett-Luce assumption* (CPPL). We aim to study the influence of different Multi-armed Bandit algorithms as parameter configuration selection strategies on the performance of the CPPL algorithm. To achieve this, we experimented with a variant of the Multi-armed Bandits called Multi-Dueling Bandits paired with contextual information. Our results show that a better parameter configuration selection strategy can improve the performance of the CPPL algorithm in terms of runtime and quality of the parameter configuration.

Acknowledgement

The course of this thesis is a particular phase in my life's journey. I wish to thank those who helped me during my work and holistically influenced my life. I humbly bow my head with reverence and dedicate this work to my grandparents and my parents, who have supported me in every step of this journey. Without their blessings and grace, this work would not have been completed.

I wish to thank my supervisor, Dr. Theodor Lettmann, Intelligent Systems and Machine Learning group at Paderborn University, for providing me the opportunity to initiate my master's thesis. Having him as a supervisor motivated me to conduct high-quality research. I am sincerely grateful to Dr. Viktor Bengs, Chair of Artificial Intelligence and Machine Learning at Ludwig-Maximilians-Universität München, who guided me patiently throughout this process. His valuable suggestions led to the development of new ideas used in this work.

I convey my enormous thanks to my thesis advisor, Ms. Jasmin Brandt, a Ph.D. student in Intelligent Systems and Machine Learning group at Paderborn University. Her endorsement through this period is beyond words. Apart from being a rare blend of scientific caliber, she is a wonderful person full of life. Her constant encouragement, valuable guidance, and outstanding suggestions have contributed to the successful completion of this study and thesis. She is an exceptionally kind person, and her generosity, compassion, and care for me are unparalleled. I am deeply indebted to her.

I furthermore want to thank Mr. Dimitri Weiß at Bielefeld University for his cooperation during the early stages of my thesis work. I am very grateful for his timely support. I also want to thank my friend Varun Golani, who has been a constant source of goodwill, good company, and support.

Finally, I want to thank the Paderborn Center for Parallel Computation (PC^2) for providing access to the NOCTUA 1 and NOCTUA 2 clusters and the IRB (Informatik Rechnerbetrieb) at Paderborn University for providing me with a suitable virtual machine to work on.

Contents

1 Introduction

In many applications, computationally challenging problems arise, and the ability to solve them efficiently is crucial for practical and economic reasons. Examples include scheduling, resource allocation, production planning, and sophisticated simulations. Applications of complex computational problems can be found in industry and academia, such as constraint satisfaction, vehicle routing, boolean satisfiability problem (SAT), and finding suitable hyper-parameters for a machine learning model. Typically, high-performance algorithms are used to solve these types of problems, with parameters that can be changed to control important aspects of their behavior, including the quality of the output and the run-time required to solve problem instances. Selecting suitable parameter configurations is an intricate and challenging process to achieve optimal results. Manually performing this tedious task occasionally is the norm in many cases. Problem-solving algorithms can be configured manually to meet the requirements of specific cases, such as SAT problems, which require the algorithm to determine whether a set of truth assignments exists to satisfy a given Boolean formula. However, this is time-consuming and prone to errors. Finding the best parameter configuration manually is difficult, if not impossible, because of the many combinations of parameters and the increased probability of making an error. Finding the ideal set of parameters often takes longer than expected and requires a lot of experimentation. In several contexts, it is imperative that a user can automate the task of configuring parameters to achieve a better result. Algorithm Configuration (AC) raises this problem as the automated search for configurations tuning in an algorithm. The AC problem involves finding the best set of parameters for a particular algorithm in order to get the best results on a high-computation task. Research on AC is ongoing and essential because the optimal parameter configuration is vital for achieving good performance on complex tasks. More information about this concept is discussed in Section 2.1.

To speed up the process of automating the search for suitable parameter configurations, many AC algorithms were developed based on the automatic task as a foundation. Schede et al. [56] provide an overview of different variants of both algorithm configuration problems and algorithm configurators. Their survey classified the AC problems into online and offline variants. Offline variants are similar to

batch learning in machine learning. The model is trained from a batch of problem instances and predicts the parameter configurations that perform best on average over all training problem instances. In an online variation, the problem instances come individually in a sequential manner (online manner) instead of a batch. The learning agent learns from the training problem instances, predicts suitable parameter configurations through a feedback mechanism (explained in further chapters), and changes its behavior in prediction based on that. This type of learning falls under the realm of reinforcement learning. This thesis will focus on the online variant of AC.

As part of learning the parameters in realtime, the configurator has to use a reinforcement learning algorithm that decides what parameter suits best based on the feedback it receives in terms of rewards. Furthermore, to choose the correct parameter configuration that will result in the best rewards, there might also be an exploration-exploitation dilemma, a fundamental problem in decision-making. The exploration-exploitation dilemma occurs when an agent must choose between exploring different options to gather more information or exploiting its current knowledge to maximize its reward. This dilemma arises because exploring new options carries the risk of potentially lower rewards in the short term, while exploiting current knowledge may lead to missed opportunities for long-term gain. In order to maximize its reward over time, an agent needs to find the right balance between exploration and exploitation. Multi-armed Bandit (MAB) problem is a research field that deals with this exploration-exploitation trade-off. As a classical reinforcement learning problem, MAB involves making a series of decisions to maximize a reward that is observed as feedback after each chosen action over time. To maximize the observed reward, the MAB problem requires an agent to choose among various actions, or "arms" (a term referred from the analogy of slot machines in casinos), with varying probabilities of yielding rewards. In other words, the agent "pulls" one "arm" at a time and observes a numerical reward that tells about the quality of the arm. The arm that generates the highest rewards on average is defined as the best arm. Several algorithms have been developed to solve the MAB problem (for example, $\epsilon-$greedy, Upper Confidence Bound (UCB) [7], Thompson Sampling (TS) [59] etc.). By balancing exploration and exploitation, these algorithms aim to maximize the rewards over time. Many of these algorithms follow a similar approach, like confidence bounds (introduced in UCB) or probability matching (in TS) to tackle the exploration-exploitation dilemma. We will discuss these approaches in more detail in Chapter 3. Choice-dependent real-world problems can use MABs to model various real-world situations, such as choosing which ads to display to users on a website to maximize clicks, which song to display so that the user adds it to

their playlist, or which medicine to prescribe to a patient which suits their body, etc. A detailed introduction on MAB problem is deferred to Section 2.2.

Contextual bandits are a variant of MAB that incorporate context, or additional information, into the decision-making process. For example, in online advertisements, the agent may access information about the user's age, location, and browsing history. This additional information helps the agent recommend advertisements on which the user will most likely click. This allows the user to choose the best advertisement based on their preferences. By taking additional information into account, as shown in the above example, contextual bandits can improve the accuracy of their predictions and ultimately lead to better performance. Multi-dueling bandits are also a variant of MAB, where rather than selecting a single action out of many, the agent chooses a subset of actions that include the optimal action as a group member. This variant can be seen in many realtime applications like recommender systems which can be seen in many online audio and video platforms like Spotify, Apple Music, Amazon Audible, etc. (for audio) and Netflix, YouTube, etc. (for video). In such systems, the user is presented with more than two alternatives and must make a distinct choice between them based on their preferences. In this way, the agent has a better chance of choosing one of the most promising candidates based on the actions that the user can choose from. Bengs and Hüllermeier [8] called this problem "Preselection Bandit (Pre-Bandit)" problem, details on which can be found in Section 2.4. Suppose these recommendation systems are coupled with additional information about the user, such as genre preference, in the case of a music platform like Apple Music. In that case, the system will have a better chance to recommend the songs, based on this information, which will give a high reward from the feedback by the user in the form of a "like". The Pre-Bandit problem can be extended to the contextual Pre-Bandit as shown by Mesaoudi-Paul et al. [43] in their online AC algorithm called *Contextual Preselection under Plackett-Luce assumption* (CPPL). A detailed introduction on CPPL algorithm is deferred to Section 2.5. This thesis focuses on studying the influence of different bandit algorithms with contextual information on the performance of the CPPL algorithm.

1.1 Background and Motivation

As introduced above, AC is the problem of finding optimum parameter configurations automatically for a solver to get the best performance on a highly computationally

challenging problem. It is a problem that attempts to find the best parameter configurations of a solver on multiple instances. When automated AC methods [56, Section 4 and 5, Section 7 to 10] is used, considerable time can be saved, and potentially better results can be achieved than if the parameters are manually configured. Using AC, algorithm designers and users do not have to spend as much time searching for high-quality parameters as they once did since it automatically identifies and recommends parameters with high quality. Additionally, automated AC methods can be developed to provide high-quality parameters for the specific problem instance being solved. This also includes instances that have never been seen before or were not even anticipated by the algorithm designer. The configuration of algorithms plays an essential role in machine learning because it improves the performance of algorithms across a wide range of applications. It is also crucial in developing machine learning methods, where choosing the suitable algorithm and its parameters can significantly affect the system's accuracy and efficiency. According to the survey by Schede et al. [56], most AC research has traditionally been focused on offline approaches, i.e., train-once approaches. These are based on a static set of instances representing a specific task (e.g. SAT problem) for which good parameter settings are found and after which the parameters are put into practice. Ignoring the fact that a representative set of instances may not be available for a given problem at the time of algorithm design, train-once methods suffer from changes or drift of problem instances. Furthermore, there may be a lack of time for repeated offline training.

These issues have been addressed by the online or Realtime Algorithm Configuration (RAC) methods, such as those used by *Realtime Algorithm Configuration through Tournaments* (ReACT) [25] and *Realtime Algorithm Configuration through Tournament Rankings* (ReACTR) [24], which modify the parameter configurations during run-time as more instances are introduced sequentially. A detailed introduction on RAC is deferred in Section 2.1.3. Mesaoudi-Paul et al. [44] introduced CPPL, an algorithm of the online variant of AC, where they used the Pre-Bandit [8] problem as the base of the CPPL algorithm for the parameter selection strategy. Through their research, Mesaoudi-Paul et al. [43] showed that the Pre-Bandit problem could be easily extended to the contextual Pre-Bandit. The basic idea of CPPL is that the algorithm chooses a set of parameters from a pool of parameters to run the solver on a problem instance based on contextual information of the problem instance. More details on this are covered in Section 2.5.

Algorithm Configuration using the Pre-Bandit problem involves two agents: a learner that preselects a subset of possible parameter configurations (arms) based on a piece of contextual information and a selector that chooses the final configuration from

this subset. This selection strategy is based on the Pre-Bandit problem, which is similar to the *battling bandits* by Saha and Gopalan [52]. The battling bandits are related to the *Preference-Based Multi-Armed Bandits* (PB-MAB) problem, also known as *dueling bandits* [66, 65], which is an extension of the standard stochastic MAB setting [12, 35] in case of preference-based feedback. It is important to emphasize that in this type of feedback, the online learner compares arms qualitatively, i.e., ranking or comparing arms in terms of pairwise comparisons. More details follow in Section 2.3. However, battling bandits do not use contextual information in general while selecting a subset of arms.

CPPL uses the contextual information combined with the selection strategy from the Pre-Bandit to select a subset of configurations (arms) to solve problem instances of a specific task. Our motivation forms in substituting this selection strategy with other bandit algorithms coupled with contextual information. There is a relevant amount of literature for subset selection strategies which can be found in [12, Section 6.5], termed as Multi-dueling bandits. But significantly less to no use of the context information in Multi-dueling bandits can be found. Therefore, in this thesis, we will use Multi-dueling bandit algorithms combined with context information to form new selection strategies and show a comparison between them through relevant comparative measures.

1.2 Related Work

A typical example of an RAC method is ReACT, a method that stores a set of parameterizations in parallel and runs them in parallel on instances that can be solved in realtime, developed by Fitzgerald et al. [25]. The ReACT method creates a tournament for each incoming problem instance and runs the parameterizations in parallel on it, termed as “races”. In the event that a parameterization does not “win” enough races against the other parameterizations, it is replaced with a random parameterization. The second method is ReACTR [24], which is an extension of the ReACT method. The main difference is that ReACTR uses a ranking system called “ TrueSkill” [28] as a ranking technique for the configurations to enhance its predecessor, ReACT. New parameterizations are generated using a genetic crossover mechanism [5]. ReACT and ReACTR address the issue of RAC directly rather than periodically doing a new offline configuration run. At run-time, they configure the solver for new instances as they arrive to ensure a high-quality solution. Mesaoudi-

Paul et al. [44] propose the CPPL algorithm to recommend parameter configurations for sequentially arriving problem instances automatically. A solver may run multiple parameterizations simultaneously for an incoming problem instance. Once one of the parameterizations has found a solution, the parallel-solving process terminates similar to ReACT and ReACTR. Detailed information is deferred to Section 2.5.

As a method that can be used for algorithm selection, bandit approaches have gained a reputation for success over the years. In Wang and Tropper [63], a mathematical model is presented for the adaptive time warp control problem in a single agent setting as a MAB problem. The work of Brochu et al. [10] on portfolio management and adaptation of acquisition functions in the context of Bayesian optimization with Gaussian processes is presented in the form of a MAB problem. In Phillips et al. [45], the problem of scheduling searches with different heuristics is treated as a MAB problem, and the solution is formulated by using dynamic TS. The main feature of these MAB approaches is their use of quantitative feedback in the form of absolute numerical rewards (e.g., the run-time of a solver or the accuracy of a learning method). CPPL, however, uses a variant of the MAB problem that considers preference, in which feedback is purely qualitative. Essentially, it receives feedback on which configuration did best on a given problem instance among a subset of configurations that were also tested. A survey by Busa-Fekete et al. [12] shows the recent development in this field which is extended to different variants.

Auer [7] introduced the contextual bandit problem, which is a variant of the MAB problem. In this problem, the expected reward for an arm is derived from the inner product of a feature vector and a weight vector. The feature vector describes the characteristics of an arm in a particular context; the weight vector is unknown but assumed to be universal among all arms. They propose an algorithm called LINREL with sub-linear regret. In order to solve the standard context-free bandit problem, Langford and Zhang [34] designed a modification of the famous $\epsilon-$greedy algorithm. It divides exploration and exploitation rounds into epochs. Other research literature on contextual MAB problem includes [39, 3, 1]. Lu et al. [39] use an UCB [7] method for their contextual bandit algorithm LINUCB. Dudıék et al. [18] introduced the contextual variant of the dueling bandits where the learner observes context information iteratively and observes the comparison as rewards of the arms chosen from this information. Later, another variant of this came into existence where instead of choosing a pair of arms, a set of arms of size $k \geq 2$ is chosen. This variant is named Multi-dueling bandits in the survey paper by Busa-Fekete et al. [12]. In CPPL, the authors considered the Plackett-Luce (PL) model [46, 41] for the utility model. In which the idea is to use a parameterized probability on the set of all

rankings over a finite set of arms, i.e., the probability of arms to be chosen in each step is proportional to their weight or "strength". Higher-weighted arms tend to rank higher. Details on PL model are deferred to Section 2.4.3.

Saha and Gopalan [52] examine regret minimization under various subset choice models in their battling bandits setting, where the goal of the learner is to identify the best arm from its selected multi-set of at least two arms and play it as often as possible. In this setting, the learner is given two main tasks, find the subset which contains the best arm and then exploit it by balancing through exploration-exploitation criteria. As part of their later research, Saha and Gopalan [54] analyzed the sample complexity of the battling bandit problem in terms of both winner feedback and full-ranking feedback. The winner's feedback is given in terms of the winning arm from the selected subset. Full-ranking feedback gives a total ranking on the arms in the selected subset. Saha and Gopalan [51] also studied the ranking problem under the PL model in the *probably approximately correct* (PAC) setting [19], where the learner selects exactly k items and receives either winner feedback or an ordered list of at most k items as feedback. In PAC setting, the learner is allowed to return a solution that is only approximately optional, reducing the sample complexity of the learner, i.e., the learner is quicker. Saha and Gopalan [53] analyzed the regret minimization problem in the context of the Multinomial Logit (MNL) model, considering two scenarios: a) the learner gets an ordered top-k ranked feedback from the selected subset of at most k arms, b) the learner receives a feedback in the form of the total ranking after selecting a fixed-size subset of k items. However, the CPPL algorithm considers a fixed-size subset of k items.

1.3 Goals

As we will see in Section 2.5, CPPL uses parameter selection strategies by considering it as a MAB problem. However, it only uses a single MAB strategy which is the UCB method with contextual information. This strategy is closely related to the contextual Pre-Bandit and is a mixture of contextual and Multi-dueling bandit variants of the MAB problem. However, algorithms in such variant combinations are very rare. There are several MAB algorithms in the literature aside from the UCB strategy (which is used in CPPL). Therefore, the goal of this thesis is to study the influence of these different MAB algorithm combinations from contextual and Multi-dueling bandits as a substitute for the UCB strategy by creating an experimental framework

of the CPPL algorithm in PYTHON. This includes extending the CPPL code (cf. Appendix A.2) to integrate new arm (parameter configuration) selection techniques and using the idea from algorithms implemented in the package *duelpy* [1] for creating those strategies.

1.4 Thesis Structure

Chapter 2

This chapter details the fundamentals of the topics used in this thesis. The chapter is divided into sections where each section will give a detailed overview of the following topics: Algorithm Configuration (Section 2.1), Multi-armed Bandit (Section 2.2), Dueling bandits (Section 2.3), Preselection Bandit (Section 2.4), CPPL (Section 2.5).

Chapter 3

This chapter gives the working details and pseudo-code of the pre-existing arm strategies and some newly established strategies during the thesis period. The newly established algorithms will combine existing multi-dueling bandit algorithms with contextual information.

Chapter 4

This chapter describes the simulation setup, the hardships in the setup, and the details of the dataset used for our experiments.

Chapter 5

The results of the experimentation with different arm selection strategies will be illustrated and discussed in this chapter.

Chapter 6

The conclusion of the experiments and a discussion on future works will be examined in this chapter.

[1] `https://gitlab.com/duelpy/duelpy`

2 Fundamentals

> *All compromise is based on give and take, but there can be no give and take on fundamentals. Any compromise on mere fundamentals is a surrender. For it is all given and not taken.*
>
> — **Mahatma Gandhi**
> about the importance of fundamentals

This chapter will discuss the fundamentals of the topics used in this thesis.

2.1 Algorithm Configuration

After a brief introduction in Chapter 1, let us look into more details of AC problem. The fundamental notations in this section are followed from the survey paper by Schede et al. [56].

2.1.1 Introduction

As already mentioned in Section 1.1, Algorithm Configuration is the problem of finding an algorithm or solver's optimal parameter configurations automatically. Over the past 20 years, significant research on the AC problem has resulted in a diverse range of approaches and problem variants. Many of these approaches can adapt to the specific characteristics of the instances they are solving, resulting in highly effective parameter configurations tailored to those instances. This includes providing high-quality parameters for instances that have yet to be encountered before or even anticipated by the algorithm's designer.

The working of an offline version of AC is illustrated in Figure 2.1. The configurator

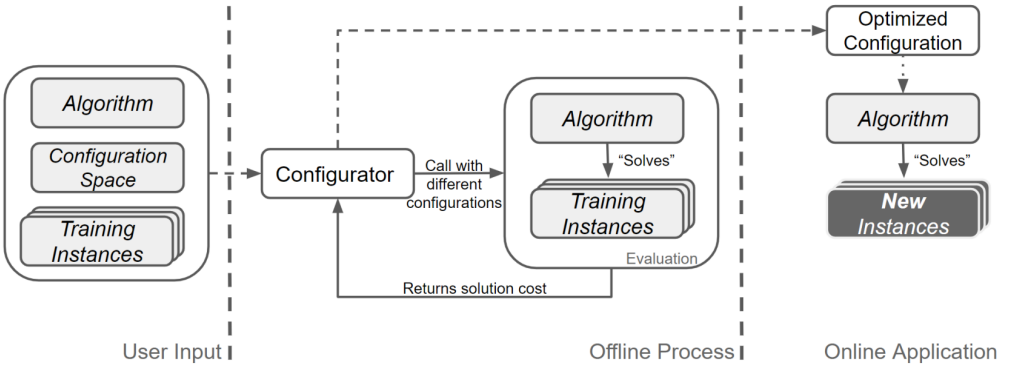

Fig. 2.1.: Algorithm Configuration Visualized [38]

tunes the parameters on some batches of training instances. It does so by calling the algorithm with different configurations to solve the training instances and then returning the solution cost, e.g., run-time, solution quality, etc., as feedback. Once the training is done, the selected configuration from the configurator is applied to new instances (test instances).

2.1.2 Basic Notations and Setting

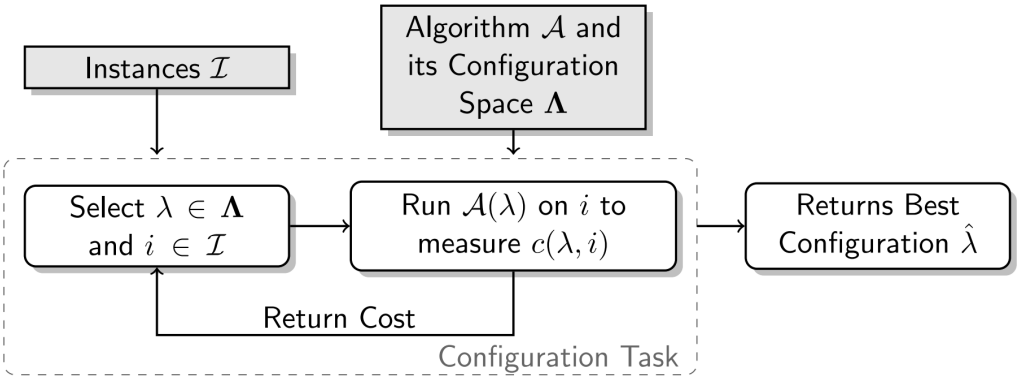

Fig. 2.2.: Configuration task in Algorithm Configuration [38]

From Figure 2.2, we have a set of parameters called configuration space Λ and some problem instance space $\mathcal{I}$. Over this instance space $\mathcal{I}$, a probability distribution $\mathcal{P}$ is defined. In the "Configuration Task", the configurator selects some parameters $\lambda \in \Lambda$. It runs the algorithm $\mathcal{A}(\lambda)$ on these parameters for a problem instance $i \in \mathcal{I}$ and measure its solution cost $c(\lambda, i)$. A cost function, $c : \Lambda \times \mathcal{I} \to \mathbb{R}$, represents the cost of running a given problem instance with a given configuration. This cost is

given as feedback to the configurator to adjust the selection of parameters for the next instance. The goal of the configurator is to find an optimal configuration $\lambda^* \in \Lambda$ where,

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \int_{\mathcal{I}} c(\lambda, i) d\mathcal{P}(i). \tag{2.1}$$

The distribution of $\mathcal{P}$ over $\mathcal{I}$ is not known in practice, so we need to solve a proxy problem for λ^*. For that, we are provided with sampled training instances $\mathcal{I}_{train} \subseteq \mathcal{I}$ and an aggregation function m. The aggregation function is often an average, such as the arithmetic mean, which is calculated by running $\mathcal{A}$ with a particular configuration on a group of instances and determining their costs. The next step is to identify the configuration that minimizes the total costs for all the training instances by aggregating them, i.e.,

$$\widehat{\lambda} \in \arg\min_{\lambda \in \Lambda} m\left(c, \mathcal{I}_{\text{train}}, \lambda\right). \tag{2.2}$$

An illustration of offline AC is shown in Figure 2.3. However, this thesis will deal

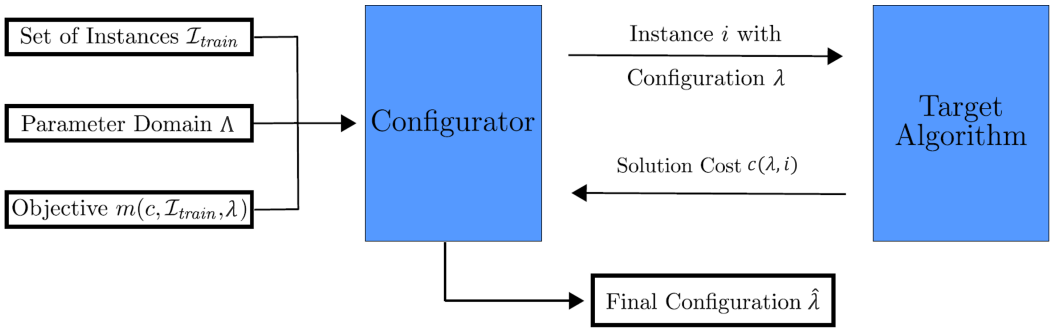

Fig. 2.3.: Illustration of offline Algorithm Configuration [56]

with the online version of the AC problem.

2.1.3 Realtime Algorithm Configuration

Instead of having a set of training instances $\mathcal{I}_{train}$, the instances will be provided in an online manner, i.e., the instances $i \in \mathcal{I}$ arrive sequentially. The main difference between offline AC and RAC is the availability of the instances. Let $I \subseteq \mathcal{I}$ be a problem instance set from the problem instance space $\mathcal{I}$, and we want to find a good parameter setting λ from a configuration space Λ. The cost function $c : \Lambda \times \mathcal{I} \to \mathbb{R}$ from the space of cost functions $\mathcal{C}$ quantifies the cost of running a problem instance

with a given configuration. The goal is to find a parameterization λ that minimizes the sum of costs across all problem instances, i.e., $\sum_{i \in \mathcal{I}} c(\lambda, i)$, which represents the average performance of λ across the set of problem instances. RAC considers the problem instances to be solved in a sequence, which is a realistic scenario for companies that have to solve routing or order processing problems on a daily basis and may need to adapt to changes in the problem instance space $\mathcal{I}$. As already mentioned in Section 1.2, CPPL algorithm is a state-of-the-art algorithm in RAC which uses a racing principle as shown in Figure 2.4.

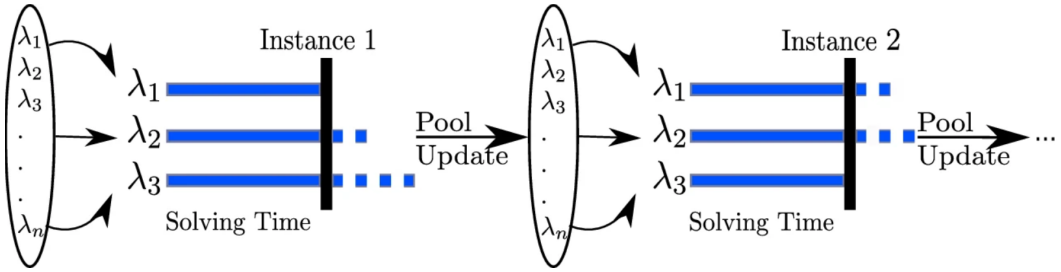

Fig. 2.4.: Illustration of racing principle in RAC [44, Figure 1]

Given a pool of parameterizations $\Lambda = \{\lambda_1, ..., \lambda_n\}$, the algorithm must select parameterizations of a limited number k according to the provided parallel computing resources. After running the parameterizations on a sampled instance, the algorithm observes which finishes first. This form of feedback is called "winner feedback" [43] and is described in Section 2.5.4. The other parameterizations are immediately stopped, and the pool is updated with new parameters for the next instance [44, Section 3.1]. At some point, the algorithm selects those parameter configurations which give the best performance in terms of execution time, i.e., an optimal configuration is one for which the run time or solving time is the lowest. Based on a history set h_t of the preceding instances, the goal of the algorithm is to find the optimal configuration $\hat{\lambda} \in \arg\min_{\lambda \in \Lambda} \mathbb{E}[c(\lambda, i_t) | h_t]$, where $t \in \{1, ..., T\}$, for a final time horizon T [56].

2.2 Multi-Armed Bandits

In the following sections, we will dive deeper into the concept of Multi-armed Bandit. The basic notations of the MAB problem follow from the book by Lattimore and Szepesvári [35].

2.2.1 Introduction

The study of Multi-armed Bandit has been ongoing for almost a century. It has now developed into a well-established field with many researchers who publish hundreds of articles annually. In recent years, MAB algorithms have also been adopted for practical use in industry. They have particularly been very useful in choice-dependent real-world problems to model various situations, such as choosing which ads to display to users on a website to maximize clicks, which song to display so that the user adds it to their playlist, or which medicine to prescribe to a patient which suits their system, etc. It would be helpful to simplify the problem by using a classical dilemma to help us understand MAB's base problem. Consider a two-armed slot machine is a casino in which you have already pulled each lever five times, resulting in the following payoffs (in euros):

Round	1	2	3	4	5	6	7	8	9	10
LEFT	10		0	0		10				0
RIGHT		0			0		0	10	0	

The slot machine's left arm appears to perform better than the right arm, with an average pay-off of €4 compared to an average pay-off of €2 for the right arm. If you have 10 more trials or "pulls" to make, you must decide how to use them. One option is to continue pulling the left arm since it has been more successful so far. Another option is to try the right arm a few more times, in case its lower pay-off was due to bad luck, and it may improve with additional trials. This dilemma illustrates the main challenge in bandit problems: determining the optimal balance between exploration (trying new options) and exploitation (using the currently most successful option). In order to solve bandit problems, it is essential to find the right balance between these two strategies.

Another simple example of this dilemma can be the restaurant choice problem shown in Figure 2.5. Imagine that your favorite restaurant is located just around the corner from your home. If you go there every day, you know exactly what to expect and can be confident in the quality of the food, but you also miss out on the opportunity to try new restaurants that might be even better. On the other hand, if you try new restaurants all the time, you may occasionally encounter unpleasant food. Similarly, online advisors must decide how to balance their use of the most successful, known ads with the potential benefits of trying new ads that may be even more successful.

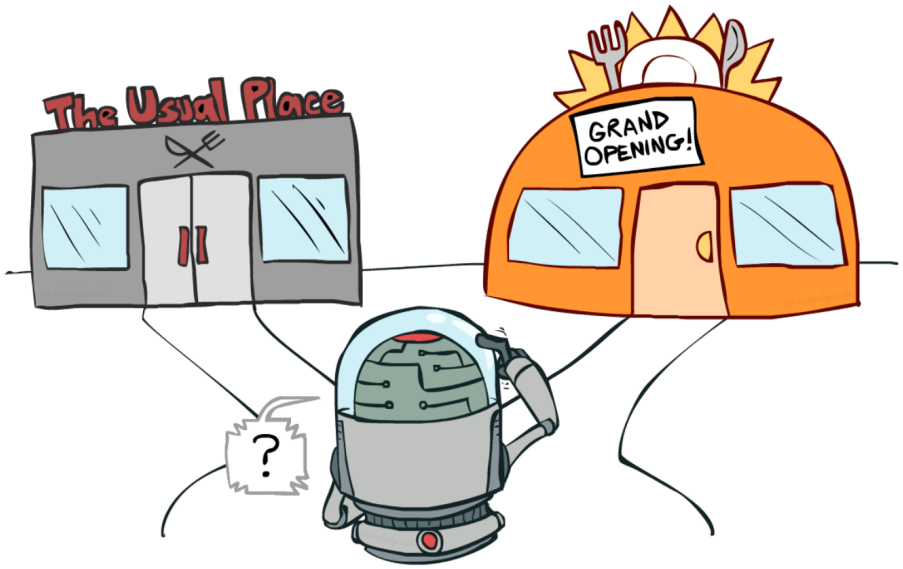

Fig. 2.5.: Exploration-Exploitation dilemma in restaurant choice problem [32]

2.2.2 Basic Problem Notations and Setting

A learner and environment play a sequential game in a bandit problem. It is played over $T \in \mathbb{T}$ rounds, where $\mathbb{T} := [T] = \{1, ..., T\}$, also known as **time horizon**. In the context of bandit problems, the learner must repeatedly make selections from a given set of alternatives in an online manner. These alternatives, or "arms", can be thought of as similar to the levers on a slot machine in a casino that can be "pulled". At each time step $t \in \mathbb{T}$, the learner first chooses an arm a_t from a given set A, and then receives a reward $\boldsymbol{r}_t \in \mathbb{R}$, revealed by the environment, based on an unknown probability distribution P_{a_t} with an expected value μ_{a_t} that reflects the arm's quality. The learner's goal is to identify the arm $a^* \in A$ with the highest expected value μ_{a^*}.

When making selections, the learner cannot predict the future and must base their choices on the history of selections and rewards, denoted as $\mathcal{H}_{t-1} = (a_1, \boldsymbol{r}_1, ..., a_{t-1}, \boldsymbol{r}_{t-1})$. The learner adopts a policy mapping from history to actions to interact with the environment. The learner's goal is often to maximize the cumulative reward over all rounds, which is the sum of all rewards received: $\sum_{t=1}^{T} \boldsymbol{r}_t$. One of the main challenges in bandit problems is that the learner does not have complete knowledge of the environment.

2.2.3 Learning Objectives

The learner's performance is evaluated based on the amount of regret it produces relative to a given policy $\pi : \mathcal{H} \rightarrow A$. Regret of a learner with respect to a policy π (not necessarily the policy that the learner is following) is the difference between the total expected reward that would have been obtained by following policy π for T rounds and the reward the learner would achieve when it pulls the optimal arm in each time step till T rounds. The regret relative to a set of policies Π (also referred to as the "competitor class" in [35]) is the maximum regret that would be incurred by any policy in the set Π. Alternatively, regret can be thought of as the difference between the learner's performance and the best arm (assuming there is no drift in the environment, but this is the case here since we have a fixed mean value μ_{a_i}). The regret depends on the specific environment, and environments with high regret are those in which the learner performs poorly. The ideal scenario is for the regret to be small for all environments. The worst-case regret is the maximum regret that could be incurred across all possible environments.

2.2.4 Applications

Let us look at a brief formalization of application examples in which MAB gives prominent solutions.

In the context of ad placement, each round corresponds to a user visiting a website, and the set of actions A is the set of available ads. This problem can be treated as a standard MAB problem, where in each round, the policy chooses an ad $a_t \in A$, and the reward is 1 if the user clicks on the ad, and 0 otherwise. This approach may be suitable for specialized websites where all the ads are likely relevant to the users. To be able to target ads effectively, however, companies such as Amazon need to consider the interests and behaviors of individuals to ensure their ads are relevant to those individuals. This additional information can be incorporated into the decision-making process using the user's characteristics as "context". For instance, a group of users might be clustered into smaller groups, and a different bandit algorithm might be implemented for each group.

Another application of bandit algorithms is in recommendation systems [33], which can be found on platforms such as Netflix, YouTube, etc. For example, Netflix must decide which movies to prominently display on the user's "Browse" page

while browsing the movie library. As with ad placement, users arrive at the page sequentially, and the reward can be determined based on whether a user watches a movie and whether it will be positively rated. There are, however, several challenges to take into account in the context of this decision-making process. Netflix displays a long list of movies, so the set of possible actions is extensive and is one of the many challenges that the learner faces while making a decision. However, it is essential to note that this is not an offline problem; the learner can influence the data by deciding which movies to recommend to each user. Suppose a particular movie or television show, e.g., The Imitation Game, is never recommended to users. In that case, only a few users will watch it, and there will be limited data available about its performance.

2.3 Dueling Bandits

In this section, we will delve into the details of the dueling bandits problem. The fundamentals and notations of dueling bandits or PB-MAB follow from the literature survey by Busa-Fekete et al. [12].

2.3.1 Introduction

It is a good idea to consider extending the MAB framework to include preference-based feedback, where the learner can qualitatively evaluate different options. This area of research, also known as PB-MAB or dueling bandits [66, 65], has recently gained popularity and has been explored in various papers. While this term was initially coined for a specific set of assumptions and models [58], it is now commonly used to refer to a range of similar settings. The dueling bandits problem comes to the surface when we are given a choice to choose two alternatives in parallel.

As a real-world choice-dependent example, GIFGIF [30], a popular crowd-sourcing website, utilizes pairwise preference-based feedback to label emotions for GIF images. This process involves presenting the user with two GIFs and asking them to choose the one that best represents a specific emotion. In this case, an image is considered the best option, which is believed to effectively convey the desired emotion more effectively than the other candidate GIFs in the set.

2.3.2 Basic Problem Notations

There is a set of arms, denoted by $A = \{a_1, ..., a_n\}$, and the learner can compare any pair of arms a_i and a_j as actions. The set of index pairs (i, j) such that $1 \leq i \leq j \leq n$ corresponds to the action space. The learner receives feedback based on an unknown probabilistic process described by a preference relation $\mathbf{Q} = [q_{i,j}]_{1 \leq i,j \leq n} \in [0, 1]^{n \times n}$, which specifies the probability of observing a preference for a_i over a_j in direct comparison, or

$$\mathbf{P}(a_i \succ a_j) = q_{i,j}. \tag{2.3}$$

These probabilities, represented by $q_{i,j}$, follow a Bernoulli distribution and are assumed to be stationary and independent across actions and iterations. This means that no matter what the outcome of previous iterations is, the outcome of any action comparison set (i, j) is determined by $q_{i,j}$. For all $i, j \in [n] := \{1, ..., n\}$, the preference relation $\mathbf{Q}$ is reciprocal, meaning that $q_{i,j} = 1 - q_{j,i}$.

In some cases, the comparison may result in a tie. To handle these cases, Busa-Fekete et al. [12] suggests assigning "half a point" to both arms, which is equivalent to flipping a coin to determine the winner, thus effectively reducing the problem to one with binomial outcomes. An arm a_i is said to beat arm a_j if the probability of winning a pairwise comparison, $q_{i,j}$, is greater than $\frac{1}{2}$. As $q_{i,j}$ approaches $\frac{1}{2}$, it becomes increasingly difficult to determine which arm is better based on a finite sample from $\mathbf{P}(a_i \succ a_j)$. When $q_{i,j} = \frac{1}{2}$, it is impossible to determine which arm is superior in a finite number of pairwise comparisons. Therefore, it is useful to consider

$$\Delta_{i,j} = q_{i,j} - \frac{1}{2} \tag{2.4}$$

as a measure of the difficulty of a PB-MAB task, regardless of the learner's specific goal. This quantity is known as "calibrated pairwise preference probabilities" [12].

2.3.3 Learning Protocol

As mentioned in Section 2.2, the decision-making process occurs in discrete steps either over a finite time horizon $\mathbb{T} := [T] = \{1, ..., T\}$ or over an infinite time horizon $\mathbb{T} := \mathbb{N}$. In each iteration $t \in [T]$, the learner selects a pair of indices $1 \leq i(t) \leq j(t) \leq n$ and observes the outcome of comparing the corresponding arms $a_{i(t)}$ and $a_{j(t)}$, with the probability of $a_{i(t)}$ winning, i.e., $a_{i(t)} \succ a_{j(t)}$ being $q_{i(t),j(t)}$ and the probability of $a_{j(t)}$ winning, i.e., $a_{j(t)} \succ a_{i(t)}$ being $q_{j(t),i(t)}$. On the basis of

finite sample sets, it is possible to estimate pairwise probabilities $q_{i,j}$. Let $n^t_{i,j}$ be the number of times the learner has compared arms a_i and a_j in the first t iterations, and let $w^t_{i,j}$ and $w^t_{j,i}$ be the number of times that a_i and a_j have won, respectively. Then, the proportion of wins for a_i against a_j till t is

$$\widehat{q}^t_{i,j} = \frac{w^t_{i,j}}{n^t_{i,j}} = \frac{w^t_{i,j}}{w^t_{i,j} + w^t_{j,i}}.$$

With the assumption that all the samples are independent and identically distributed (i.i.d.), $\widehat{q}^t_{i,j}$ is a reasonable estimate of the pairwise winning probability given by eq. (2.3). However, this estimate may be biased due to the fact that $n^t_{i,j}$ depends on the learner's choices, which are themselves influenced by the data. With Hoeffding's bound [29], a technique commonly used in the literature on bandits, one can calculate a high probability confidence interval $c^t_{i,j}$ for $q_{i,j}$. The confidence interval for an arm starts relatively high and gets smaller when the pull frequency (number of wins) is relatively high compared to the overall number of pulls we have done. Confidence intervals may take different forms but generally have the form

$$\left[\widehat{q}^t_{i,j} - c^t_{i,j},\ \widehat{q}^t_{i,j} + c^t_{i,j}\right].$$

If $\widehat{q}^t_{i,j} - c^t_{i,j} > \frac{1}{2}$, then a_i beats a_j with high probability. Conversely, if $\widehat{q}^t_{i,j} + c^t_{i,j} > \frac{1}{2}$, then a_i is beaten by a_j with high probability.

2.3.4 Learning Tasks

The basic learning task in the dueling bandit setting is to find the best arm. The best arm is the one that is preferred over all other arms in the set A. Specifically, $a_{i^*} \in A$ is the best arm if $\Delta_{i^*,j} > 0\ \forall j \in [n] \setminus i^*$. This definition corresponds to the natural concept of a **Condorcet winner**. Condorcet winners are not always present for all preference relations $\mathbf{Q}$. This is because of patterns in preference relations, which are standard in real-world problems (but, of course, this depends on the problem being regarded). Therefore, it should not be assumed that the Condorcet winners always exist.

Another goal for the learner is to find an entire ranking of the arms rather than just a single optimal arm. A natural approach is to rank the arms such that arm a_i is ranked higher than arm a_j if $\Delta_{i,j} > 0$, i.e., if a_i beats a_j. Due to preferential cycles in $\mathbf{Q}$, such a ranking may not exist, as with Condorcet's winner. In some

situations, it is only necessary to identify or order the top k items in a ranking rather than the entire ranking. These problems are known as the top-k ranking and top-k identification, respectively. The latter is typically easier to solve because it requires less information compared to the former, but there are cases where both tasks have the same level of difficulty.

In some cases, finding a close approximation to the optimal target in a given application is sufficient. However, it can be challenging to define approximation errors in preference-based settings without numerical rewards, compared to classical MAB settings. Despite this, there have been efforts to define appropriate replacements [20, 22, 21] for many preference-based targets. One such replacement is to consider an arm as ϵ-preferable to another arm if the difference between their calibrated pairwise preference probabilities in eq. (2.4) is at least $-\epsilon$, where $\epsilon \in \left(0, \frac{1}{2}\right)$. In other words, if $\Delta_{i,j} \geq -\epsilon$, a_i is $\epsilon-$prefereable to arm a_j. Using this definition, an ϵ-optimal arm can be defined as an arm that is ϵ-preferable to all other arms.

In any case, learning involves estimating the probability that one arm is preferred over another, represented by the pairwise preference matrix $\mathbf{Q}$. However, the ultimate goal is usually not to know $\mathbf{Q}$ but to identify the best arm or a ranking of all arms. The preference matrix may not accurately reflect this target if it is not well-defined. Therefore, the probabilities $q_{i,j}$ should be consistent enough to allow the learner to approximate and eventually identify the target. However, it is not always the case that the $q_{i,j}$ lead to a natural ordering of the arms or a Condorcet winner. Alternative methods for defining the target may be necessary, such as identifying an optimal arm, ranking the arms, or defining different concepts of optimal arms.

A ranking procedure, denoted by $\mathcal{R}$, takes the preference matrix $\mathbf{Q}$ and produces a complete preorder relation $\succ^{\mathcal{R}}$ on the arms. Another variant of the PB-MAB problem is to make the target predict the relation $\succ^{\mathcal{R}}$. The ranking procedure $\mathcal{R}$, which must be given as part of the problem specification, establishes the connection between $\mathbf{Q}$ and $\succ$. $\mathcal{R}$ is a function that takes in a preference matrix and returns a complete preorder on the set of arms. The output of this function, applied to a preference matrix $\mathbf{Q}$, is denoted as $\succ^{\mathcal{R}}$.

A **Copeland winner** is defined as follows. In a Copeland ranking, $a_i \succ^{CO} a_j$ iff $d_i > d_j$, where $d_i := |\{k \in [n] | q_{i,k} > \frac{1}{2}\}|$ is the Copeland score of a_i, where $|\{\cdot\}|$ represents the cardinality of the set. In simple words, an arm a_i is preferred over a_j whenever a_i beats more arms than a_j. A Copeland set is defined as $CP(\mathbf{Q}) = \{i \in [n] | i \in \text{argmax}_{j \in [n]} d_j\}$. In simpler terms, a set of arms in $[n]$ with the highest

Copeland score is called a Copeland set $CP(\mathbf{Q})$. For a preorder defined by $\succ^{CO}$, the Copeland set includes the arms that rank highest in $\succ^{CO}$. Each arm in the Copeland set, which beats the greatest number of other arms, is referred to as a Copeland winner, though it can be defeated by other arms, unlike a Condorcet winner.

A **Borda winner** is defined as follows. Borda ranking or sum of expectations (SE) [12] is similar to Copeland ranking. The major difference is that $a_i \succ^{SE} a_j$ iff for some $q_i := \frac{1}{n-1} \sum_{k \neq i} q_{i,k}$ and $q_j := \frac{1}{n-1} \sum_{k \neq j} q_{j,k}$,

$$q_i > q_j.$$

The Borda winner, like the Copeland winner, is an arm that ranks highest according to the preorder on the alternatives induced by $\succ^{SE}$. However, like the Copeland winner, the Borda winner may still be beaten by another arm but are suitable and commonly used winner concepts for the case where a Condorcet winner does not exist.

2.4 Preselection Bandits

This section will delve into the details of the Pre-Bandit problem. The fundamental notations for Pre-Bandits follow from the research paper by Bengs and Hüllermeier [8].

2.4.1 Introduction

The Pre-Bandit problem is related to preference-based settings, particularly the variant of battling bandits by Saha and Gopalan [52]. The basic idea behind the Pre-Bandit is that, instead of taking a pair of choices from a pool and then seeing which one has the higher preference, like in dueling bandits, the learner selects a subset of choices from the pool of size possibly greater than two.

Let us show the importance of this method with an example of a car dealership problem illustrated in Figure 2.6. A car dealership had a large variety of cars to choose from, but they needed help finding the perfect one for their customers. They would often present a single car to the customer and hope they would like

it, but this often led to disappointment and frustration for both parties. One day, a customer came in looking for a reliable, spacious car with good fuel efficiency. The salesperson knew she had several options that fit these criteria. However, she was still determining which one the customer would prefer. Instead of presenting just one car, she decided to try something new. She presented the customer with four cars. Two cars were among those they thought would fit the bill, and the other two were to see if the customer changed his mind. The salesperson asked them to choose the one they liked better. The customer was able to test drive all the cars and ultimately decided on the one that felt the most comfortable and met almost all of their needs. The dealership was thrilled with this approach as it allowed them to find the perfect car for their customer and satisfied both parties. From then on, the dealership always presented a selection of cars to their customers and asked for their preferences before making a final decision.

Fig. 2.6.: Car dealership problem example using preselection bandits

In this example, to choose the most suitable car, the salesperson changed their approach in two ways. Firstly, instead of sampling a single or a set of two cars from the pool, they sampled a subset of them. Secondly, they considered the contextual

criteria of the customer into account while showing them the cars to choose from. This strategy can also be useful in the context of RAC. Instead of choosing all the parameterizations or configurations of an algorithm to see which gives the best performance, we can select a subset and check among them.

2.4.2 Basic Notations and Setting

The Preselection Bandit problem involves a learner that selects a subset of options (called "arms") for a selector (e.g., a user or the environment) to choose from. The learner's goal in this scenario is to choose subsets of arms that will ultimately lead to highly preferred choices determined by the selector's preferences. The selector may not choose the best arm in this setting due to limited information or other constraints. The learner can learn about the selector's preferences through the choices made.

The Pre-Bandit problem has practical applications in fields such as information retrieval, used by search engines, and online advertising, where recommended advertisements can be considered a preselection. In both cases, the learner is responsible for selecting a subset of arms for the selector to choose from, eventually leading to highly preferred choices determined by the selector's preferences. The selector's decision-making process can include randomness, modeled using the PL model [40, 46]. The learner's performance is measured using a notion of regret based on the utilities of the chosen subsets.

In this setting, the arms are identified by a set $[n]$ containing $n \in \mathbb{N}$ (arbitrary but fixed) arms, which can be represented by their indices $\{1, ..., n\}$ for simplicity. The authors consider a total preference order, where the i^{th} arm is preferred over the j^{th} arm if $i \succ j$. The set of all subsets of $[n]$ with size k is denoted as $\mathbb{A}_k \in \mathbb{A}$, where $\mathbb{A} \in 2^{[n]}$ is the action space (of the learner) which corresponds to a family of subsets of $[n]$. Note that $\mathbb{A}$ is not required to be the set of all possible pairs of arms in $[n]$. The full action space is the union of all the sets of subsets of $[n]$ with sizes ranging from 1 to n, and is denoted as $\mathbb{A}_{full} := \bigcup_{k=1}^{n} \mathbb{A}_k$. A bijective mapping $\mathbf{r} : [n] \to [n]$ assigns ranks to the arms, with $\mathbf{r}(k)$ being the rank of the k^{th} arm. The inverse function $\mathbf{r}^{-1}$ determines the ordering induced by $\mathbf{r}$, with $\mathbf{r}^{-1}(k)$ being the index of the arm in the k^{th} position. If $S \subseteq [n]$, then a bijection $\mathbf{r}_S : S \to [|S|]$ that assigns ranks to the elements of the subset $S \in \mathbb{A}$, is called a full or partial ranking on the

subset S, and the inverse $\mathbf{r}_S^{-1}$ is the ordering induced by $\mathbf{r}$ on S. The action space $\mathbb{A}$ corresponds to the family of subsets of

The learner makes a series of decisions over a fixed time horizon, represented by $\mathbb{T} := [T] = \{1, ..., T\}$. At each time step $t \in \mathbb{T}$, the learner proposes a subset $S_t \in \mathbb{A}$ of the available arms, based on its observations up to t. The learner then observes the choice of one arm, represented by i_t, from the subset S_t made by the selector. The goal of the learner is to minimize the **cumulative regret**, represented by

$$R_T = \sum_{t=1}^{T} r(S_t), \tag{2.5}$$

over the time period T by selecting subsets S_t through a suitable regret function $r : \mathbb{A} \to \mathbb{R}_+$. $\mathbb{A}$ is analyzed for two possible characteristics that can be attributed to it:

1. **Restricted Preselection**: $\mathbb{A} = \mathbb{A}_k$, i.e., a subset can only consist of exactly k many arms as a preselection, where $k > 1 \in \mathbb{Z}$.

2. **Flexible Preselection**: $\mathbb{A} = \mathbb{A}_{full}$, i.e., any subset of $[n]$ can be a preselection given that it is non-empty.

For the sake of our problem mixture with AC, we will focus on the restricted preselection variant and not on the flexible preselection variant. As we will see in Section 2.5.6, we are using the restricted preselection because we have a fixed number of available kernels on which we can run the configurations in parallel.

It is not a good idea to estimate a pairwise preference matrix from the pairwise relation $\mathbf{Q}$, unlike in dueling bandits (cf. Section 2.3.2). In the Preselection Bandits, we cannot compare two arms directly, but only a subset of arms of size k. Therefore, we cannot estimate the probability that the arm i wins against arm j, but only the probability that the arm i wins in the subset S_t with $i \in S_t$ and $k-1$ other arms in S_t. So for each arm, we have to estimate the probability to win in all $\binom{n-1}{k-1}$ subsets in which it could be contained, and this binomial coefficient could be infeasible high for a huge amount of arms n. An exception could be if we have the Plackett-Luce assumption [46, 41] because then the probability if arm i wins in subset S_t, is not entirely random, but depending on the underlying latent utility.

2.4.3 The Plackett-Luce Model

The notations in this subsection follow from [43]. The PL model [46, 41] is a probability model that describes the likelihood of different rankings over a set of n arms based on their weights or latent utilities represented by the parameter $\boldsymbol{v} = (v_1, ..., v_n)^\top \in \mathbb{R}_+^n$. According to the PL model, the probability mass function is defined as follows:

$$\mathbb{P}(\mathbf{r} \mid \boldsymbol{v}) = \prod_{i=1}^{n} \frac{v_{\mathbf{r}^{-1}(i)}}{\sum_{j=i}^{n} v_{\mathbf{r}^{-1}(j)}},$$

where $\mathbf{r}$ is a ranking and $\mathbf{r}^{-1}(i)$ is the index of the arm on position i. The probability mass function is highest for the ranking that orders the arms according to their parameter values in descending order.

The probability of a marginal, i.e., a partial ranking on a subset of arms, represented by $\mathbf{r}_S$, can be calculated by summing the probabilities of all the possible linear extensions of $\mathbf{r}_S$ given the vector of utility values $\boldsymbol{v}$. This probability can also be expressed as the product of the ratios of the values at each position in the ranking to the sum of the values of all the items ranked below it. The inverse ranking, represented by $\mathbf{r}_S^{-1}$, is the ordering determined by the partial ranking $\mathbf{r}_S$. Formally, the probability of a partial ranking $\mathbf{r}_S$ on a subset $S \subset [n]$ is given as:

$$\mathbb{P}(\mathbf{r}_S \mid \boldsymbol{v}) = \sum_{\mathbf{r} \in E(\mathbf{r}_S)} \mathbb{P}(\mathbf{r} \mid \boldsymbol{v}) = \prod_{i=1}^{|S|} \frac{v_{\mathbf{r}_S^{-1}(i)}}{\sum_{j=i}^{|S|} v_{\mathbf{r}_S^{-1}(j)}}. \tag{2.6}$$

This marginal probability is a very useful property of PL model and can be used to get the probability that alternative $k \in S$ gets ranked highest by

$$\mathbb{P}(\mathbf{r}_S(k) = 1 \mid \boldsymbol{v}) = \frac{v_k}{\sum_{i \in S} v_i}. \tag{2.7}$$

2.5 Contextual Preselection under Plackett-Luce Assumption

In this section, we will dive deep into the fundamentals of CPPL algorithm. The fundamental notations for CPPL are from the research paper by Mesaoudi-Paul et al. [43] and Mesaoudi-Paul et al. [44].

2.5.1 Introduction

As discussed in Section 2.1, AC helps algorithm designers identify and recommend high-quality parameters automatically. RAC (cf. Section 2.1.3) setting is a realistic one in which instance set, I, is considered to be sequential in nature, i.e., the instances arrive sequentially instead of a batch.

As new instances arrive, ReACT [25], ReACTR [24], and CPPL [44] select a high-quality parameter configuration of the solver in realtime. ReACTR is an RAC approach and an extension to ReACT that uses the racing principle (cf. Figure 2.4) to evaluate multiple parameter configurations in parallel on multiple CPU cores and then uses a ranking mechanism called TrueSkill [27] to select the best configuration for a given instance. It also uses TrueSkill to decide which configurations to replace with new ones and how to generate new configurations using a genetic crossover mechanism.

CPPL is a realtime algorithm configuration algorithm that uses a different approach for ranking, selection, and generation while maintaining the racing principle and parallel execution of ReACTR. CPPL connects the online contextual Preselection Bandit setting to the Realtime Algorithm Configuration problem. The algorithm solves the realtime algorithm configuration tasks. Additionally, it uses a novel genetic engineering technique [5] to generate new configurations.

2.5.2 Algorithm Configuration with Bandits

The goal of the learner in a bandit problem, as described in Section 2.2.2, is to maximize the sum of rewards or minimize the total regret. The regret is the expected difference between the optimal sum of rewards and the actual sum of collected rewards. In Algorithm Configuration, the configurations can be considered as arms, and running an algorithm (or a "solver" of a problem instance) on a configuration corresponds to "pulling" an arm. The objective is to minimize the runtime of the recently seen problem instance that is represented through contextual information. Most MAB approaches for algorithm selection use numerical rewards, such as runtime or accuracy, as feedback. In contrast, the approach proposed by Mesaoudi-Paul et al. [44] uses qualitative feedback in the form of preferences, which is similar to the dueling bandits (cf. Section 2.3.2) and consecutively the preselection bandits (cf. Section 2.4.2).

2.5.3 Plackett-Luce Model with Feature and Contextual Information for Winner Feedback

As already defined in Section 2.4.3, the notations used by Mesaoudi-Paul et al. [43] for the contextual PL model are a little different, but the overall idea is the same. To incorporate context information $\boldsymbol{x}_i \in \mathbb{R}^d$ about the i^{th} arm, the **constant latent utility** v_i, also referred to as "estimated skill parameters" in [44], can be replaced by a log-linear function of the arm's context, as proposed in [16, 55]:

$$v_i = v_i(\mathbf{X}) = \exp\left(\theta^\top \boldsymbol{x}_i\right), \quad i \in \{1, \ldots, n\}, \tag{2.8}$$

where, θ is an unknown weight parameter and the corresponding feature vectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ are summarized in a matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$. A PL model that incorporates context information is then defined as follows:

$$\mathbb{P}(\mathbf{r} \mid \theta, \mathbf{X}) = \prod_{i=1}^{n} \frac{v_{\mathbf{r}^{-1}(i)}(\mathbf{X})}{v_{\mathbf{r}^{-1}(i)}(\mathbf{X}) + \ldots + v_{\mathbf{r}^{-1}(n)}(\mathbf{X})} = \prod_{i=1}^{n} \frac{\exp\left(\theta^\top \boldsymbol{x}_{\mathbf{r}^{-1}(j)}\right)}{\sum_{j=i}^{n} \exp\left(\theta^\top \boldsymbol{x}_{\mathbf{r}^{-1}(j)}\right)}. \tag{2.9}$$

The probability for an arm $k \in S$ to get the top rank among other arms in S is analogous to eq. (2.7) and is denoted by

$$P_\theta(k \mid S, \mathbf{X}) = \frac{\exp\left(\theta^\top \boldsymbol{x}_k\right)}{\sum_{j \in S} \exp\left(\theta^\top \boldsymbol{x}_j\right)}. \tag{2.10}$$

For an observation $(k, S, \mathbf{X})$, the corresponding log-likelihood function of θ would be

$$\mathcal{L}(\theta \mid k, S, \mathbf{X}) = \theta^\top \boldsymbol{x}_k - \log\left(\sum_{j \in S} \exp\left(\theta^\top \boldsymbol{x}_j\right)\right). \tag{2.11}$$

Schäfer and Hüllermeier [55] demonstrated the concavity of eq. (2.10) and eq. (2.11).

2.5.4 Contexual Preselection Bandits Problem

In the following, we formally define the contextual Preselection Bandits. The online contextual Preselection Bandits with winner feedback, as introduced in [43], is a sequential online decision problem, in which, in each time step $t \in \{1, ..., T\}$, a

learner is presented a context $\mathbf{X}_t = (\boldsymbol{x}_{t,1}, ..., \boldsymbol{x}_{t,n})$. For each time step t and arm i, $\boldsymbol{x}_{t,i}$ is a vector of length d that represents the features of the context in which an arm must be chosen and the properties of the arm itself. These features can be obtained by applying a joint feature map to the context and the arm. After the context has been revealed, the learner selects a subset $S_t \subset [n]$ of k arms and receives feedback on this subset of arms. This feedback can take one of two forms:

1. A ranking of the arms in the subset also called the partial ranking feedback scenario in the PL model, where the context information is $\mathbf{X}_t$ and a fixed but unknown weight parameter is $\theta^* \in \mathbb{R}^d$, or

2. A top-ranked arm in the subset, which is generated (assumed) by eq. (2.10), also called to the winner feedback scenario in the PL model, where the context information is $\mathbf{X}_t$, and a fixed but unknown weight parameter is $\theta^* \in \mathbb{R}^d$.

The aim of the learner in each time step t is to choose a subset of k arms S_t that will result in the lowest overall regret, meaning it should contain the arm best suited for the current context $\mathbf{X_t}$. For a given time step t, the best arm i^* is defined as the arm with the highest latent utility, given by eq. (2.8), according to the PL model, i.e.,

$$i^*(t) = \underset{i \in [n]}{\operatorname{argmax}}\, v_i^* \left(\mathbf{X}_t\right) = \underset{i \in [n]}{\operatorname{argmax}} \exp\left(\boldsymbol{x}_i^\top \theta^*\right). \tag{2.12}$$

Here, the parameter θ^* is a fixed but unknown weight parameter.

It should be noted that this definition is in line with other potential definitions of the best arm. For example, the Borda and the Copeland winners (cf. Section 2.3.4) for context-dependent Borda and Copeland scores, respectively. The regret for a learner selecting a subset S_t at time t is calculated as the difference between the latent utility of the best arm $i^*(t)$ and the latent utility of the best arm included in S_t, normalized by the utility of the best arm shown in eq. (2.13),

$$r(S_t) = \frac{v_{i^*(t)}^* \left(\mathbf{X}_t\right) - \max_{j \in S_t} v_j^* \left(\mathbf{X}_t\right)}{v_{i^*(t)}^* \left(\mathbf{X}_t\right)}. \tag{2.13}$$

The cumulative regret for a series of selections (S_t) during a time horizon T is the sum of the individual regrets at each time step given as

$$R_T = \sum_{t=1}^{T} r(S_t) = T - \sum_{t=1}^{T} \frac{\max_{j \in S_t} v_j^* \left(\mathbf{X}_t\right)}{v_{i^*(t)}^* \left(\mathbf{X}_t\right)}. \tag{2.14}$$

A regret of zero occurs if the best arm is included in the selected subset, in a manner similar to the concept of weak regret in the problem of dueling bandits [15], where no regret occurs when the best arm is included in the duel.

2.5.5 Learning Tasks

One of the key tasks for the CPPL algorithm is to estimate the unknown but fixed weight parameter θ^*, which determines the underlying contextual PL model of the winner feedback scenario. Because of the parametric nature of the PL model, the algorithm minimizes the loss of the log-likelihood eq. (2.11) by using optimization methods such as the stochastic gradient descent (SGD) algorithm as used in [43] and subsequently in [44] to estimate θ^*. Mesaoudi-Paul et al. [43] has used the Polyak-Ruppert averaged SGD [49, 47] defined by

$$\bar{\theta}_t = \frac{1}{t}\sum_{i=1}^{t}\hat{\theta}_i, \quad \text{with} \quad \hat{\theta}_i = \hat{\theta}_{i-1} - \gamma_t \nabla l\left(\hat{\theta}_{i-1}; Z_i\right), \tag{2.15}$$

where $l(\cdot;\cdot)$ is some loss function, Z_i is the single observation in time step i, and γ_t is the learning rate. As a loss function, they have used the negative log-likelihood.

At time step t the observation consists of a triple set comprising context information $\mathbf{X_t}$, selected subset by the learner $S_t \subset [n]$ of size k and a feedback by the environment $Y_t \sim P_\theta(\cdot|S_t, \mathbf{X_t})$. This triple set $(Y_t, S_t, \mathbf{X_t})$ is being used by SGD to estimate θ^*. Under the contextualized PL model eq. (2.10) with winner feedback, $Y_t = k_t \in S_t$ is the top-ranked armed among the subset S_t. However, a learner typically selects S_t based on the subsets selected up to time step t, so the observations are not independent. It is also possible that contexts $\mathbf{X_t}$ are not necessarily i.i.d. [43].

Despite this, the authors use the asymptotic results for the i.i.d. case as an approximation. More specifically, for deriving confidence bounds on the contextualized utility parameters, they use the above results for the Polyak-Ruppert averaged SGD method in eq. (2.15) as follows. For some $\alpha \in (\frac{1}{2}, 1)$ and $\gamma_1 > 0$, the Polyak-Ruppert averaged SGD estimate for θ^* follows:

$$\begin{aligned}
\bar{\theta}_t &= (t-1)\bar{\theta}_{t-1}/t + \hat{\theta}_t/t, \\
\hat{\theta}_t &= \hat{\theta}_{t-1} + \gamma_1 t^{-o} \nabla \mathcal{L}\left(\hat{\theta}_{t-1} \mid Y_t, S_t, \mathbf{X}_t\right).
\end{aligned}$$

The estimation for the true unknown contextualized utility parameter of an arm at a given time step is then obtained by applying the updated estimate for θ^* to the contextualized PL model. Formally, for an arm $i \in [n]$ at time step t, the estimation for the true unknown contextualized utility parameter, $v_i^*(\mathbf{X_t}) = \exp\left(\boldsymbol{x}_{t,i}^\top \theta^*\right)$ is given by

$$\hat{v}_{t,i} = \exp\left(\boldsymbol{x}_{t,i}^\top \bar{\theta}_t\right). \tag{2.16}$$

2.5.6 Realtime Algorithm Configuration as Contextual Preselection

In CPPL algorithm [44, Algorithm 1], the RAC problem is modeled as an online contextual preselection problem, in which each possible configuration of algorithm parameters is treated as a separate "arm" or option. The framework proceeds in discrete time step $t = 1, 2, ...$ and initializes a random parameter vector $\hat{\theta}_0$. Each step starts with observing a new problem instance $i \in I$, where I is the instance set (cf. Section 2.1.3), followed by building a joint feature map of $f(i)$ and $f(\lambda_j), j \in [n]$, where $f(\cdot)$ denote the features of instance i and respectively configuration λ_j. The joint feature map used by Mesaoudi-Paul et al. [44] is a polynomial of the second degree, which consist of all polynomial combinations of the features with degree less than or equal to 2, i.e., it is defined for two vectors $\boldsymbol{x} \in \mathbb{R}^r$ and $\boldsymbol{y} \in \mathbb{R}^c$ as:

$$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \left(1, x_1, \cdots, x_r, y_1, \cdots, y_c, x_1^2, \cdots, x_r^2, y_1^2, \cdots, y_c^2, x_1y_1, x_1y_2, \cdots, x_ry_c\right). \tag{2.17}$$

By incorporating the current estimate of θ, the joint feature map of the parameterization, and the problem features, the latent utilities from eq. (2.8) are determined.

A procedure called *ARM_SELECTION* determines the k most optimistic parameterizations within the pool of candidates based on the estimated skill parameter and the confidence bounds. Different *ARM_SELECTION* strategies will be examined in Chapter 3. Once S_t is chosen, Y_t is revealed according to the considered winner feedback scenario. After observing the winner feedback, the estimated parameter will be updated using the Polyak-Ruppert averaged SGD method based on the log-likelihood function gradient for the observed triple. At the end of each time step, a regret is calculated as described in Section 2.5.4 and a cumulative regret using eq. (2.14) during time horizon T.

In our experiments in Chapter 5, we will use this cumulative regret as the baseline to compare the newly established algorithms from Section 3.2. Usually, for an infinite

configuration space Λ, regret cannot be easily computed since it is not trivial to find an optimal configuration. However, since we are only considering scenarios with a finite number of configurations in the experiments, we can find an optimal one and compute the regret as the suboptimality of our choices. Algorithm 1 shows a pseudo-code of the framework.

Algorithm 1: CPPL Framework

Input: n parameterizations $(\lambda_1, \lambda_2, ..., \lambda_n) \subset \Lambda$, subset size k, time horizon T (since we are in an online scenario, usually the total number of problem instances in the set I are treated as the time horizon), $\alpha \in \left(\frac{1}{2}, 1\right)$, $\gamma_1 > 0$, ω

Initialize $\hat{\theta}_0$ randomly
$\bar{\theta}_0 = \hat{\theta}_0$
for $t = 1, 2, ..., T$ **do**
    Observe the context vectors $\mathbf{X}_t = (\boldsymbol{x}_{t,1}..., \boldsymbol{x}_{t,n})$
    Compute $\hat{v}_{t,i} = \exp\left(\boldsymbol{x}_{t,i}^\top \bar{\theta}_t\right) \ \forall i \in [n]$
    Choose S_t from ARM_SELECTION$(\hat{v}_t, c_t, k)$
    Observe Y_t according to the winner feedback scenario
    Update $\bar{\theta}_t$ by $\bar{\theta}_t = (t-1)\bar{\theta}_{t-1}/t + \hat{\theta}_t/t$ with

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \gamma_1 t^{-\alpha} \nabla \mathcal{L}\left(\hat{\theta}_{t-1} \mid Y_t, S_t, \mathbf{X}_t\right)$$

The number k corresponds to the number of CPU cores available on the machine in general terms. However, for our experiments, we will choose fixed values of k. The solver attempts to solve the problem instances using k different parameter configurations, and the configuration that can solve the problem first is the winner. This feedback is then used to update the estimate for θ using the SGD scheme. Any parameterizations that perform poorly are discarded from the pool of candidates. After discarding certain parameterizations in the previous step, new ones are generated using a genetic approach. An illustration of the algorithm can be seen in Figure 2.7.

The illustration in Figure 2.7 corresponds to Mesaoudi-Paul et al. [44, Algorithm 1], which has some differences to Algorithm 1. We are not randomly creating and discarding the parameters in our working framework. As we will see in Chapter 4, our setup has a fixed amount of parameters from the parameter space Λ for simulation. The reason is that the results for comparison by the framework in Figure 2.7 are given in terms of average runtime on an instance with CPPL. However, this would not make sense when comparing the framework to different Pre-Bandit strategies for arm selection.

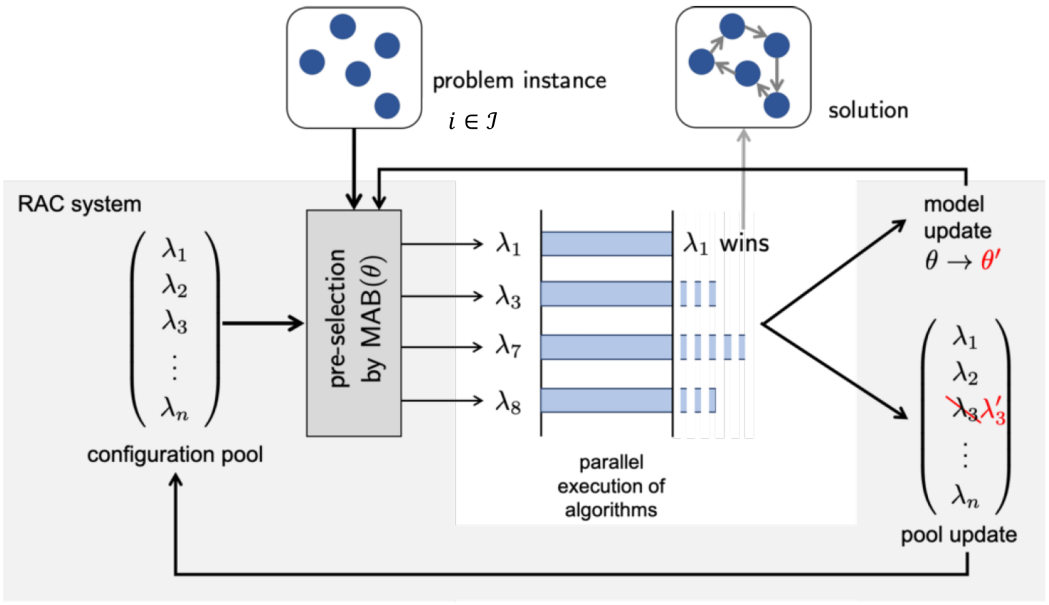

Fig. 2.7.: Illustration of CPPL when instances arrive and are solved sequentially [56]

That is because, in the above framework, we would not be able to calculate the regret of the algorithm since the framework is terminated once a set of parameterizations has solved the instance. The regret does not work because the best existing configuration is unknown. We cannot find it in finite time because there are infinitely many configurations because of the infinite configuration space. To avoid this, we will only focus on preselecting the arms, and for checking the regret of the selected subset, we will observe the already computed run times for the instance solver. We will discuss this in detail in Chapter 4.

3 Preselection Bandit Algorithms with Contextual Information

In this chapter, we will look at some of the existing preselection algorithms and the candidate algorithms which can be integrated with the CPPL framework. As discussed in Section 2.5.6, Pre-Bandits can be used to find a subset of k most optimistic arms from a pool of n different arms. We will be focusing on the winner feedback mechanism of the contextualized Pre-Bandit problem as stated in Section 2.5.4 for the implementation of the candidate arm selection strategies and comparing the cumulative regrets results in Chapter 5. Although there are very few Pre-Bandit algorithms that use contextual information, the basis of comparing multiple arms is similar to that of dueling bandits (cf. Section 2.3). Therefore, our goal is to use the winner-feedback (cf. Section 2.5.4) based multi dueling bandit algorithms [12, Section 6], which is a variant of dueling bandits, with contextual information in the CPPL framework to see if we can get better results than the baseline algorithm, i.e., Upper Confidence Bound algorithm in the CPPL framework.

Algorithm 1 in Section 2.5.6 shows the CPPL framework. We will add only those parts that can be substituted to select the subset S_t (line 7) in Algorithm 1. As you will see in the later parts of Section 3.2, some algorithms have additional input parameters. We will add those inputs as part of the algorithm's inputs. This means that the procedure *ARM_SELECTION* can have different input parameters according to individual strategies. A few algorithms will not follow the general updating rule (line 9) of Algorithm 1. In that case, we will write the whole framework depending on that strategy.

In the following Section 3.1, we will have a look at the already existing Pre-Bandit algorithm with contextual information by Mesaoudi-Paul et al. [43]. In Section 3.2, we will discuss some candidate algorithms implemented with contextual information to integrate with the CPPL framework.

3.1 Existing Algorithms

In this section, we will have a detailed look at the already existing contextual Pre-Bandit algorithm used in the CPPL algorithm for *ARM_SELECTION* strategy.

3.1.1 Upper Confidence bound

In addition to estimating the unknown parameter, the online learning task in the CPPL framework also involves addressing the exploration-exploitation problem. To do this, Mesaoudi-Paul et al. [43] adopt an approach similar to the UCB algorithm by using confidence bounds on the contextualized utility parameters in eq. (2.8) to choose the subset with the highest upper confidence bounds. Since the learner uses a history of the selected subsets so far to determine the following selection S_t (cf. Section 2.5.4), the observation is not independent. Additionally, the contexts $\mathbf{X}_t$ are also not necessarily i.i.d. [43]. However, the authors guaranteed that for a confidence bound $c_{t,i}$ and using the estimates in eq. (2.16), the following holds with high probability for the i.i.d. case,

$$|\hat{v}_{t,i} - v_i^*(\mathbf{X_t})| \leq c_{t,i}. \tag{3.1}$$

They also concluded that in the case of i.i.d., for any arm $i \in [n]$, an asymptotic upper confidence bound for $v_i^*(\mathbf{X_t})$ is defined as

$$\hat{v}_{t,i} + c_{t,i}, \tag{3.2}$$

provided t is sufficiently large. For our work, we will call this amalgamation as *skill vector*. Note that the *skill vector* does not necessarily be the same for all the strategies. It is just a joint representation of the entities which are used for the decision-making process by the learner. We will not be focusing on the detailed calculation for the confidence bound $c_{t,i}$ here since it is out of the scope of the thesis. A more detailed version for calculating the confidence bound $c_{t,i}$ can be found in [43, pg. 7]. Algorithm 2 shows how UCB is used as an *ARM_SELECTION* strategy in the CPPL framework.

The subset $S_t \subset [n]$ of k number of arms with the highest upper confidence bounds on the latent utility are selected. The mean observed rewards $(\hat{v}_{t,i})$ or estimated skill parameters for an arm $i \in [n]$ and the confidence bound $(c_{t,i})$ have a significant role

Algorithm 2: ARM_SELECTION: UCB

Input: Estimated skill parameter $\hat{v}_t$, confidence bound c_t, subset size k

1 Choose S_t as:

$$\underset{S_t \subseteq [n], |S_t|=k}{\operatorname{argmax}} \sum_{i \in S_t} (\hat{v}_{t,i} + c_{t,i})$$

2 **return** S_t

in the exploitation and exploration factor respectively of the algorithm. The arms with high observed rewards have a high value of $\hat{v}$ and thus a relatively high value in the *skill vector* and are pulled or selected from the set of arms $[n]$ more often, hence, exploitation. On the other hand, $c_{t,i}$ are the confidence bounds, which get smaller the more often an arm is pulled. So an arm that was pulled only rarely has a high value of c_t and thus again a high value in the *skill vector* and is pulled with high probability in the subsequent iterations, hence exploration. After the arm is pulled, the confidence bound shrinks a bit because we get another observation for the pulled arm and are more confident about the estimated $\hat{v}_{t,i}$.

3.2 Newly Established Algorithms

In this section, we will have a detailed look at the contemporary contextual preselection algorithms which can be used in the CPPL algorithm for *ARM_SELECTION* strategy.

3.2.1 Introduction

Mesaoudi-Paul et al. [44] uses the approach of the UCB algorithm [7] for solving the exploration-exploitation trade-off in getting the optimal set of parameters. Many variants of the UCB algorithm and several other well-established concepts, such as the probability matching heuristic like Thompson Sampling (TS) [59] can be seen in various literature over the years. There are so many bandit variants that are somehow related. There is a vast amount of literature where the dueling set is extended to more than two competitors. Still, different terminologies are used for such settings, for example, multi-dueling bandits [11, 57], combinatorial bandits [13, 35], battling bandits [52], choice bandits [2], etc. For the setting of online contextual

preselection, we have chosen the regret minimization dueling bandit algorithms with the PL model for our CPPL framework. Some algorithms are not originally meant to be used with contextualized information but are only for comparing two arms. Nevertheless, we have modified them to use the contextual information and output a subset $S_t \subset [n]$ of k number of arms with high *skill vector*. Note that the *skill vector* does not need to be the same as eq. (3.2) for all the new strategies. We will see in the following subsections how different strategies use the *skill vector* to choose the arms for S_t in the CPPL framework.

3.2.2 Contextualized Linear Stochastic Transitivity Imitator

Contextualized Linear Stochastic Transitivity Imitator or CoLSTIM algorithm was introduced by Bengs et al. [9] aimed for regret minimization task in dueling bandits problem with context information. Since we are working in an online setting with a regret minimization setting, this algorithm fits to be a suitable contender for us.

CoLSTIM algorithm [9, Algorithm 1] initially uses the feedback process, which is determined by *Contextualized Linear Stochastic Transitivity* (CoLST) utility model which will be described in the following. CoLSTIM is a computationally efficient algorithm that makes its choices based on imitating the feedback process using perturbed context-dependent utility estimates of the CoLST model [9]. Although CoLSTIM is designed initially to work in the CoLST setting, we will modify the algorithm for our CPPL framework (cf. Algorithm 1 in Section 2.5.6). The linear stochastic transitivity (LST) model is a type of probability model that is used to predict the outcome of a comparison between two choices from a set of n choices. The model has two parameters: An n-dimensional parameter $\boldsymbol{u} = (u_1, ..., u_n)^\top \in \mathbb{R}^n$, where each component represents the (latent) utility of a choice alternative, and a comparison function, represented by a cumulative distribution function called $F : \mathbb{R} \to [0, 1]$, which maps real numbers to a range between 0 and 1 and is symmetrical. We will not be discussing the LST model in depth since it is not in the scope of the thesis but a detailed version can be found in [9, Section 2].

The working of the CoLSTIM algorithm is quite similar to the UCB algorithm (cf. Algorithm 2 in Section 3.1.1) in the CPPL framework. The learner's task is to include the best arm (with the highest latent context-dependent utility) in the duel. This changes to include the arm with the highest latent context-dependent utility in our framework's chosen subset S_t. At the core of the learning task, the learner has

to estimate the unknown weight parameter θ^*, which determines the underlying CoLST model of the feedback mechanism. Similar to the estimation of θ^* in the CPPL algorithm, which uses SGD for minimizing the negative log-likelihood function, CoLSTIM uses the maximum likelihood estimate (MLE) to derive the estimation of the θ^*. Bengs et al. [9] argued that the computation of the MLE involves a computationally expensive operation, as the entire history is used for this estimation step. This issue was shared by most of the algorithms for logistic bandits [23, 37, 62] and stochastic contextual dueling bandits [50]. This issue can be rectified by using SGD to optimize the log-likelihood function in an online manner like in CPPL algorithm. The authors also stated that although their theoretical guarantees do not hold for this SGD variant, the difference in the regret from MLE and SGD from their experiments [9, Section E] are negligible. As far as the cumulative elapsed time for making a decision is concerned, the SGD offers an advantage over the MLE [9]. CoLSTIM's arm selection process has low computational costs because the two maximization problems that need to be solved are simple.

Unlike UCB which uses the context vector for getting the *skill vector*, CoLSTIM uses contrast vectors, $z_{t,i,j} = \boldsymbol{x}_{t,i} - \boldsymbol{x}_{t,j}$. This in turn gives a $d \times \binom{n}{2}$ dimensional contrast matrix

$$\boldsymbol{Z}_t = (z_{t,1,2} z_{t,1,3} \dots z_{t,1,n} z_{t,2,3} \dots z_{t,n-1,n})$$

in each time step t. The learner then selects a pair of arms $(i_t, j_t) \in [n]$ after observing the contrast information from $\boldsymbol{Z}_t$. For our setting to choose the subset S_t, we will divide the algorithm into two versions:

- CoLSTIM-Context: In this version of the CoLSTIM algorithm, the subset is chosen by observing the context information from $\mathbf{X}_t$.

- CoLSTIM-Contrast: In this version of the CoLSTIM algorithm, the subset is chosen by observing the contrast information from $\boldsymbol{Z}_t$.

In both versions, the calculation of the confidence bounds c_t and SGD on the estimates will be similar to UCB in CPPL by Mesaoudi-Paul et al. [43]. We will now show the pseudo-code of both variants. The inputs of all the new preselection algorithms are similar to Algorithm 2 since we are in an online CPPL framework. Note that it is implied that all the inputs from the framework are also used in the following algorithms, even though we are not explicitly mentioning them in the pseudo-code.

CoLSTIM-Context

We have modified the CoLSTIM algorithm [9, Algorithm 1] for our online CPPL framework. This version of CoLSTIM selects a selection of k arms $S_t \subset [n]$ using the contextual information. Algorithm 3 shows the pseudo-code of this modified version of CoLSTIM algorithm with context information.

Algorithm 3: ARM_SELECTION: CoLSTIM-Context

Input: Estimated skill parameter $\hat{v}_t$, confidence bound c_t, subset size k, confidence threshold $C_{thresh} > 0$, perturbation distribution G with cumulative distribution function F

1 Sample $\tilde{\epsilon}_{t,i} \sim G \; \forall i \in [n]$

`// Sample perturbation variable`

2 $\epsilon_{t,i} = \min\left(C_{thresh}, \max\left(-C_{thresh}, \tilde{\epsilon}_{t,i}\right)\right) \forall i \in [n]$

`// Update trimmed perturbation variable`

3 Choose S_t as:

$$\underset{S_t \subseteq [n], |S_t| = k}{\operatorname{argmax}} \sum_{i \in S_t} \left(\hat{v}_{t,i} + \epsilon_{t,i} c_{t,i}\right)$$

4 **return** S_t

The strategy starts with sampling a perturbation variable $\tilde{\epsilon}_{t,i}$ from the perturbation distribution G. It then updates the trimmed perturbation variable $\epsilon_{t,i}$ using the confidence threshold C_{thresh}. The *skill vector* now consists of the estimated contextualized skill parameter and a confidence bound with a perturbed noise. The best arm in the subset S_t has the largest perturbed context-dependent utilities as it is most likely to win any duel according to the imitated CoLST model [9].

Similar to UCB strategy (cf. Algorithm 2 in Section 3.1.1, the estimated skill parameters $(\hat{v}_{t,i})$ and the confidence bound term $(\epsilon_{t,i} c_{t,i})$ have a significant role in the exploitation and exploration factor respectively in the strategy. The arms with high observed rewards have a high value of $\hat{v}_{t,i}$ and thus a relatively high value in the *skill vector* and are pulled or selected from the set of arms $[n]$ more often, hence, exploitation. On the other hand, $c_{t,i}$ coupled with the trimmed perturbation variable $\epsilon_{t,i}$ are the confidence bounds, which get smaller the more often an arm is pulled. So an arm that was pulled only rarely has a high value in the confidence bounds and thus again a high value in the *skill vector* and is pulled with high probability in the subsequent iterations, hence exploration. After the arm is pulled, the confidence bound shrinks a bit because we get another observation for the pulled arm and are more confident about the estimated $\hat{v}_{t,i}$.

In this version, both the estimated contextualized skill parameter and the confidence bounds were calculated using the context information $\mathbf{X}_t$. This changes for the second version of the algorithm in the next section.

CoLSTIM-Contrast

In this version of the CoLSTIM algorithm, the subset S_t is chosen using the contrast information. Algorithm 4 shows the pseudo-code for the modified version of CoLSTIM with a contrast matrix.

Algorithm 4: ARM_SELECTION: CoLSTIM-Contrast

Input: Estimated skill parameter $\hat{v}_t$, confidence bound c_t, subset size k, arms n, confidence threshold $C_{thresh} > 0$, perturbation distribution G with cumulative distribution function F, confidence width constant $c_1 > 0$

1 Sample $\tilde{\epsilon}_{t,i} \sim G \; \forall i \in [n]$

`// Sample perturbation variable`

2 $\epsilon_{t,i} = \min\left(C_{thresh}, \max\left(-C_{thresh}, \tilde{\epsilon}_{t,i}\right)\right) \forall i \in [n]$

`// Update trimmed perturbation variable`

3 Choose i_t as:

$$\underset{i \in [n]}{\operatorname{argmax}} \left(\hat{v}_{t,i} + \epsilon_{t,i} c_{t,i}\right)$$

4 **for** $k = 1, 2, ..., n$ **do**

5 $\quad z_{t,k,i_t} = \boldsymbol{x}_{t,k} - \boldsymbol{x}_{t,i_t}$ `// Create the contrast matrix with respect to best arm` i_t `in` S_t

6 Calculate confidence bound $c_{t,i}$ by replacing $\boldsymbol{x}_{t,i}$ with contrast vectors z_{t,i,i_t} in [43, Page 7]

7 Choose the rest of the candidate set J_t as:

$$\underset{J_t \subseteq [n], |J_t| = k}{\operatorname{argmax}} \sum_{i \in J_t} \left(z_{t,i,i_t}^{\top} \bar{\theta} + c_1 c_{t,i}\right)$$

8 **if** $i_t \in J_t$ **then**

9 $\quad S_t = J_t$

10 **else**

11 $\quad$ Remove the least ranked candidate from J_t

12 $\quad S_t = J_t \cup i_t$

13 **return** S_t

All the initial steps (lines 1-2) are the same as in Algorithm 3. The "best arm" i_t of the subset is chosen from the context information. As explained earlier, the *skill vector* for arm i_t has the greatest perturbed dependent utilities and is most likely to

win. The candidate set J_t is chosen as the arms that pose the greatest competition to the i_t, meaning the arms that have the highest chances of winning the duel against i_t based on the current upper confidence width (represented by $c_1 c_{t,i}$). This method of selecting the candidate set is standard in non-contextual dueling bandits and is designed to minimize average regret [12]. Hence, we modified it to sample a candidate set in the contextual setting.

3.2.3 Thompson Sampling

Thompson Sampling (TS) method is one of the earliest heuristics used for MAB problems. It is a stochastic algorithm that uses the probability matching strategy for choosing actions from a probability distribution of rewards over the choices and updates this distribution based on the reward received [14]. The uncertainty about which arm is the best (measured by the entropy of the distribution) determines the amount of exploration the algorithm performs.

In TS, a prior distribution is assumed for the reward probabilities of each arm. At each time step, an arm is chosen based on its posterior probability of being the optimal choice. As a simple example, suppose that a TS algorithm is trying to choose between two actions, A and B, and that it currently estimates that the probability of action A yielding a higher reward is 60% and the probability of action B yielding a higher reward is 40%. In this case, the probability matching strategy would involve choosing action A with a probability of 0.6 and action B with a probability of 0.4. Probability matching is often used in TS because it allows the algorithm to explore different options and gather more information about their relative rewards. This can be particularly useful when the reward probabilities are uncertain or when the number of available actions is large.

Since TS is also a regret minimization algorithm, we have chosen this algorithm as a contender for the arm selection strategy in our framework. For our online setting, Double-Thompson-Sampling (DTS) algorithm by Wu and Liu [64], a variant of the TS algorithm, was an excellent contender for our arm selection strategy. However, since it uses a preference matrix, which is not possible in our framework (cf. Section 2.4.2), we had to reject this algorithm. In the end, we used two variants of the TS algorithm; the first one is from Sui et al. [57] for Bernoulli bandits and the second one from Agrawal and Goyal [3] for contextual bandits.

Thompson Sampling for Bernoulli Bandits

TS for Bernoulli bandits [57, Algorithm 1] uses the historical number of wins (W_i) and losses (F_i) of an arm $i \in [n]$. This algorithm is further used as a subroutine in the *Independent Self Sparring* (ISS) [57, Algorithm 2], which is also a contender for the arm selection strategy and will be discussed in Section 3.2.4. Sui et al. [57] did not consider the contextual setting of the arms, but we will modify this to include the contextual information. The rewards $\boldsymbol{r}_t$, at time step t, for the k arms in S_t from this strategy are observed as 1 (win), and the rest of the arms in S_t have an observed reward of 0 (loss) in the regarded time step t.

The sampling process of Beta-Bernoulli Thompson Sampling is a two-step process:

- For each arm i, sample $\hat{\theta}_i \sim Beta\,(W_i + 1, F_i + 1)$.
- Choose the arm with maximal $\hat{\theta}_i$.

Note that the contextual information is missing, and only arm i is chosen, which has a maximal $\hat{\theta}_i$. To add contextual information $\mathbf{X}_t$ with $\hat{\theta}$, the *skill vector* now build using only the estimated skill parameter

$$\hat{v}_{t,i} = \boldsymbol{x}_{t,i}^{\top} \hat{\theta}_i.$$

A pseudo-code for the Beta-Bernoulli Thompson Sampling with contextual information strategy is presented in Algorithm 5. We have presented a complete pseudo-code instead of the *ARM_SELECTION* procedure because it is different from our CPPL framework in terms of updating the estimation of weights $\hat{\theta}$ (cf. Algorithm 1).

This strategy is different from our CPPL framework in the following sense: For estimating the unknown weights, TS does not use SGD. It uses a Beta prior for estimating the average utility of each arm and distributes the rewards according to the latent mean utility $\hat{\theta}_i$ of arm i. The learning process for the TS is to learn the Beta distribution. TS approach relies on a probabilistic representation of the wins and losses of the arms to drive the exploration process. In practice, computing the probability of each action being the best according to all the probability distributions is difficult. However, one advantage of the TS method is that it does not require explicit computation of these probabilities. Instead, based on the current probability distributions, an estimate of the expected reward can be drawn for each action, and the action with the highest estimated reward can be chosen. This simple process

Algorithm 5: Thompson Sampling for Bernoulli Bandits with Contextual Information

Input: n arms , subset size k, time horizon T (since we are in an online scenario, usually the total number of problem instances in the set I are treated as the time horizon)

Set $W_i = 0, F_i = 0 \; \forall i \in [n]$
for $t = 1, 2, ..., T$ **do**
  Observe the context vectors $\mathbf{X}_t = (x_{t,1} ... x_{t,n})$
  for $j = 1, ..., n$ **do**
    Sample $\hat{\theta}_j$ from $Beta\left(W_j + 1, F_j + 1\right)$
    $\hat{v}_{t,j} = \exp\left(\boldsymbol{x}_{t,j}^{\top} \hat{\theta}_j\right)$
  Choose S_t as:

$$\underset{S_t \subseteq [n], |S_t| = k, i \in S_t}{\operatorname{argmax}} \left(\hat{v}_{t,i}\right)$$

  Observe the rewards $\boldsymbol{r}_t$ of S_t according to the winner feedback scenario
  Update $W_i \leftarrow W_i + \boldsymbol{r}_t$, $F_i \leftarrow F_i + (1 - \boldsymbol{r}_t)$

is equivalent to directly sampling an action based on its probability of having the highest expected reward. Once the distribution is learned, the arm with the highest latent mean utility $\hat{\theta}_i$ is chosen. This follows the exploitation part.

Thompson Sampling for Conxtual Bandits with Linear Payoffs

Agrawal and Goyal [3] designed a generalization of TS algorithm for the stochastic contextual MAB problem with linear payoffs. They provide a sub-linear regret bound with high probability in the contextual setting. However, similar to the TS sub-routine by Sui et al. [57], they also sample a single arm instead of a subset of size k. This section will show how we modify this to return a subset.

Note that, similar to Algorithm 5, this strategy is also different from the CPPL framework in updating the estimated weights $\hat{\theta}$; therefore, we will give a complete pseudo-code instead of the *ARM_SELECTION* procedure. Similar to Algorithm 5, the learner uses the contextual features along with the features of the problem instances and rewards of the arms played from the subset S_t in the past to choose the arms to contain in the current round. The learner's goal is to gather enough information about the relationship between feature vectors and rewards so that it can predict, with some level of certainty, which arm is likely to provide the best rewards based on the feature vectors. The learner competes with a group of predictors, each of

which takes in the feature vectors and predicts which arm will give the best reward. If the learner can perform almost as well as the best predictor in hindsight (i.e., has low regret), then it is said to have successfully competed with that group.

As a consequence of contextual bandits being set up with linear payoff functions, the learner competes on the feature vectors with a class of all linear predictors. Each predictor is defined by a d-dimensional parameter $\bar{\theta} \in \mathbb{R}^d$. The predictor ranks the arms according to the dot product of the feature vector for each arm and weight parameter given by $\boldsymbol{x}_i^\top \bar{\theta}$. Agrawal and Goyal [3] considered the stochastic contextual bandit problem under linear realizability assumption. In other words, they assume that there is an underlying parameter $\theta \in \mathbb{R}^d$ such that the expected reward for each arm i, given context $\boldsymbol{x}_i$ is $\boldsymbol{x}_i^\top \theta$. The learner aims to learn the underlying parameter corresponding to θ under this realizability assumption. As a standard assumption, realizability is assumed in the existing literature on contextual Multi-armed Bandit, e.g., [7, 23, 17].

This setting is very close to our online contextual setting. Therefore, we will modify and use it in our contextual Pre-Bandit setting. The Thompson Sampling for Contextual bandits algorithm proposed by Agrawal and Goyal [3, Algorithm 1] uses a Gaussian likelihood function and Gaussian prior. A modified version to sample a subset S_t of size k can be seen in Algorithm 6.

In this version, the "likelihood" of the rewards $\boldsymbol{r}_{t,i}$ at time step t with context $\boldsymbol{x}_{t,i}$ and parameter θ is given by a probability density function of Gaussian distribution $\mathcal{N}\left(\boldsymbol{x}_{t,i}^\top \theta, v^2\right)$, where $v = R\sqrt{\frac{24}{\epsilon} d \ln\left(\frac{1}{\delta}\right)}$ with $\epsilon \in (0,1)$ and a constant $R \geq 0$. At every time step t, they generate a sample $\tilde{\theta}_t$ of d-dimension from the multivariate Gaussian distribution $\mathcal{N}\left(\hat{\theta}_t, v^2 B_t^{-1}\right)$ (line 4) and choose the arms in S_t that maximizes $\hat{v}_{t,j} = \exp\left(\boldsymbol{x}_{t,j}^\top \tilde{\theta}_{t,j}\right)$ (line 7). Because the sampling is from a multivariate Gaussian distribution, the authors concluded that their algorithm was efficient, regardless of how many arms are involved (or how infinite). The efficiency of the algorithm depends if $\underset{S_t \subseteq [n], |S_t| = k, i \in S_t}{\operatorname{argmax}} (\hat{v}_{t,i})$ is efficiently solvable. To illustrate this, the authors considered a case where the set of arms at time t can be expressed as a d-dimensional convex set (each vector is a context vector and therefore corresponds to an arm). In this case, the problem of selecting S_t can be solved efficiently.

Algorithm 6: Thompson Sampling for Contextual bandits

Input: n arms , subset size k, time horizon T (since we are in an online scenario, usually the total number of problem instances from I are treated as the time horizon),

Set $B = I_d, \hat{\theta} = 0_d, f = 0_d$
for $t = 1, 2, ..., T$ **do**
  Observe the context vectors $\mathbf{X}_t = (x_{t,1}...x_{t,n})$
  Sample $\tilde{\theta}_t$ from $\mathcal{N}\left(\hat{\theta}, v^2 B^{-1}\right)$
  for $j = 1, ..., n$ **do**
    $\hat{v}_{t,j} = \exp\left(\boldsymbol{x}_{t,j}^\top \tilde{\theta}_{t,j}\right)$
  Choose S_t as:

$$\underset{S_t \subseteq [n], |S_t| = k, i \in S_t}{\operatorname{argmax}} (\hat{v}_{t,i})$$

  Observe the rewards $\boldsymbol{r}_t$ of S_t according to the winner feedback scenario
  for $i \in S_t$ **do**
    Update $B = B + \boldsymbol{x}_{t,i}^\top \boldsymbol{x}_{t,i}$
    $f = f + \boldsymbol{x}_{t,i} \boldsymbol{r}_t$
  $\hat{\theta} = B^{-1} f$

3.2.4 Independent Self Sparring

Sui et al. [57] proposed an algorithm for selecting subsets of arms from a pool of arms in a way that minimizes cumulative regret. The algorithm aims to minimize the number of times sub-optimal arms are selected while still allowing for exploration by selecting each arm a small number of times. The authors consider two scenarios: a finite number of arms with a total order and an infinite number of arms modeled using a Gaussian process. They were inspired by the work of Ailon et al. [4] and proposed the SELFSPARRING framework, which uses a stochastic MAB algorithm as a subroutine to sample the set of k arms to duel. In this framework, k MAB algorithms control the selection of each arm in the set S_t, effectively reducing the conventional dueling bandit problem to a series of MAB algorithms competing against each other. We focused on the *Independent Self Sparring* algorithm [57, Algorithm 3] for our framework, which uses a stochastic Beta-Bernoulli Thompson Sampling algorithm. Since they are not for contextual bandits problem, we will modify it to use the contextual information $\mathbf{X}_t$ in selecting S_t.

In this section, we will split the Independent Self Sparring into two variants depending on the MAB algorithm used in its sub-routines. We will use Thompson Sampling for Bernoulli bandits with context information from Algorithm 5 and Thompson

Sampling for contextual bandits from Algorithm 6 as two different sub-routine variants. We will make use of [57, Theorem 1] which guarantees that IndSelfSparring [57, Algorithm 3] converges to an optimal arm as running time $t \to \infty$. Thus, as $t \to \infty$, the Beta distributions for each arm i are convergent to only choosing the optimal arm. Note that since this algorithm uses Thompson Sampling, the update rule for the estimates will also be different from the CPPL framework for this strategy. Therefore, we are presenting a complete pseudo code for both variants.

Independent Self Sparring for Bernoulli Bandits with Contextual Information

This variant is the same as the IndSelfSparring [57, Algorithm 3] algorithm, which uses the Thompson Sampling for Bernoulli Bandits as the sub-routine to sample the estimate of the weight parameter $\hat{\theta}_i$. We have modified the sub-routine to comply with the contextual information in Algorithm 5. We will use the same to modify the IndSelfSparring [57, Algorithm 3] to work with contextual information. Algorithm 7 shows the complete pseudo code for this strategy.

Algorithm 7: IndSelfSparring with Thompson Sampling for Bernoulli Bandits with Contextual Information

Input: n arms , subset size k, time horizon T (since we are in an online scenario, usually the total number of problem instances from I are treated as the time horizon), η the learning rate

1 Set $W_i = 0, F_i = 0 \; \forall i \in [n]$
2 **for** $t = 1, 2, ..., T$ **do**
3 Observe the context vectors $\mathbf{X}_t = (x_{t,1} ... x_{t,n})$
4 **for** $j = 1, ..., n$ **do**
5 Sample $\hat{\theta}_j$ from $Beta\left(W_j + 1, F_j + 1\right)$
6 $\hat{v}_{t,j} = \exp\left(\boldsymbol{x}_{t,j}^\top \hat{\theta}_j\right)$
7 Choose S_t as:

$$\underset{S_t \subseteq [n], |S_t| = k, i \in S_t}{\operatorname{argmax}} \left(\hat{v}_{t,i}\right)$$

8 Observe the pairwise feedback matrix $R = \{r_{ij} \in \{0, 1, \emptyset\}\}_{k \times k}$ of S_t according to the winner feedback scenario
9 **for** $i, j \in S_t$ **do**
10 **if** $r_{ij} \neq \emptyset$ **then**
11 Update $W_i \leftarrow W_i + \eta \cdot r_{ij}$, $F_i \leftarrow F_i + \eta(1 - r_{ij})$

The algorithm uses the stochastic Beta-Bernoulli Thompson Sampling to sample the estimates $\hat{\theta}$ (line 5) for each arm and create the *skill vector* with contextual

information (line 6). The subset is selected based on the top k arms from the *skill vector* (line 7). The rewards in the pairwise feedback matrix are updated with respect to the best arm in S_t (line 8). This is according to our winner feedback scenario in the CPPL framework. The best arm in S_t will get the reward of 1 (win), and the others will get 0 (lose). The posteriors are updated using a learning rate η (lines 9-11). This method averages the performance of each arm over all subsets it is contained in. This is a questionable approach for a small sample set because an arm can be the winner of a subset. Although the quality of the arm is not really good, the competitors are even worse. For a large sample set, the average should approximate the real quality of the arm.

Independent Self Sparring for Contextual Bandits

This variant uses the Thompson Sampling for Contextual Bandits from Algorithm 6 as the sub-routine to sample the estimate of the weight parameter $\hat{\theta}_i$. We have modified the sub-routine to comply with the selection of the subset S_t in Algorithm 6. We will use the same to modify the INDSELFSPARRING [57, Algorithm 3]. Algorithm 8 shows the pseudo-code of the algorithm.

Algorithm 8: INDSELFSPARRING with Thompson Sampling for Contextual Bandits

Input: n arms , subset size k, time horizon T (since we are in an online scenario, usually the total number of problem instances from I are treated as the time horizon), η the learning rate

1 Set $B = I_d, \hat{\theta} = 0_d, f = 0_d$
2 **for** $t = 1, 2, ..., T$ **do**
3 Observe the context vectors $\mathbf{X}_t = (x_{t,1}...x_{t,n})$
4 Sample $\tilde{\theta}_t$ from $\mathcal{N}\left(\hat{\theta}, v^2 B^{-1}\right)$
5 **for** $j = 1, ..., n$ **do**
6 $\hat{v}_{t,j} = \exp\left(\boldsymbol{x}_{t,j}^\top \tilde{\theta}_{t,j}\right)$
7 Choose S_t as:

$$\underset{S_t \subseteq [n], |S_t| = k, i \in S_t}{\operatorname{argmax}} (\hat{v}_{t,i})$$

8 Observe the pairwise feedback matrix $R = \{r_{ij} \in \{0, 1, \emptyset\}\}_{k \times k}$ of S_t according to the winner feedback scenario
9 **for** $i \in S_t$ **do**
10 Update $B = B + \eta \cdot \boldsymbol{x}_{t,i}^\top \boldsymbol{x}_{t,i}$
11 $f = f + \eta \cdot \boldsymbol{x}_{t,i} \boldsymbol{r}_t$
12 $\hat{\theta} = B^{-1} f$

At each time step t, a sample $\tilde{\theta}t$ of d-dimension is generated from a multivariate Gaussian distribution $\mathcal{N}\left(\hat{\theta}t, v^2 B_t^{-1}\right)$ (line 4), and the arms in S_t are chosen that maximize $\hat{v}t, j = \exp\left(\boldsymbol{x}t, j^\top \tilde{\theta}_{t,j}\right)$ (line 7). The rewards in the pairwise feedback matrix R are updated with respect to the best arm in S_t (line 8), according to the winner feedback scenario in the CPPL framework (cf. Section 2.5.4). The best arm in S_t receives a reward of 1 (win), and the other arms receive a reward of 0 (lose). The posteriors are updated using a learning rate η, the contextual information, and the reward from the feedback for each arm in the selected subset (lines 9-11). The estimated parameter $\hat{\theta}$ is then updated based on the updated posteriors (line 12).

4 Experiment Setup

We have now come to our experimental setup for our framework. In this chapter, we will discuss the dataset on which we experimented with our algorithms, the initial challenges we faced while experimenting with the framework, and how it evolved from its initial version to the final version. We will also discuss a different version of the framework, which uses novel arm selection strategies for experiments. The experiments follow the evaluation setup of Mesaoudi-Paul et al. [44] and Mesaoudi-Paul et al. [43].

4.1 Initial CPPL Framework

The initial setup of the CPPL framework follows the RAC setup of the CPPL algorithm from Mesaoudi-Paul et al. [44, Algorithm 1]. In this setup, we were using CaDiCaL [48] and Glucose [6] solvers for the boolean satisfiability (SAT) problem, which require the algorithm to determine whether a set of truth assignments exists to satisfy a given Boolean formula. Our initial goal for the thesis was to expand this framework with other *ARM_SELECTION* procedure strategies like in Algorithm 1.

4.1.1 Functionality

To give more clarity on the functionality of this framework and how it was different from Algorithm 1 in Section 2.5.6, we will show the pseudo-code of this framework from Mesaoudi-Paul et al. [44, Algorithm 1].

The framework first initializes a pool of configurations of size n with random parameterizations $(\lambda_1, \lambda_2, ..., \lambda_n) \subset \Lambda$ (line 1). This configuration pool represents the set of arms $[n]$ where each parameterization represents an arm in our terminology. In each time step t, instead of observing the context information like in Algorithm 1 (cf. Section 2.5.6), the framework observes a problem instance $\boldsymbol{i} \in I$, where $I \subseteq \mathcal{I}$ is

Algorithm 9: Initial CPPL Framework [44, Algorithm 1]

Input: The number of arms n, subset size k, problem instance set I, $\alpha \in \left(\frac{1}{2}, 1\right)$, $\gamma_1 > 0$, ω, a feature extraction function f

1 Initialize n random parameterizations $[n] = \{\lambda_1, ..., \lambda_n\} \subset \Lambda$
2 Initialize $\hat{\theta}_0$ randomly
3 $\bar{\theta}_0 = \hat{\theta}_0$
4 **for** $t = 1, 2, ...$ **do**
5 Observe the problem instance $\boldsymbol{i} \in I$
6 **for** $j = 1, ..., n$ **do**
7 $\hat{v}_{t,j} = \exp\left(\boldsymbol{x}_{t,j}^\top \bar{\theta}\right)$, where $\boldsymbol{x}_{t,j} = \Phi(f(\lambda_j), f(\boldsymbol{i}))$ `// Create the contrast vectors using a joint feature map (cf. eq. (2.17) in Section 2.5.6)`
8 Choose S_t from ARM_SELECTION$(\hat{v}_t, c_t, k)$
9 Run the parameterizations of S_t to solve $\boldsymbol{i}$ and observe the winner parameterization $w_t \in S_t$ which terminates first.
10 Update $\bar{\theta}_t$ by $\bar{\theta}_t = (t-1)\bar{\theta}_{t-1}/t + \hat{\theta}_t/t$ with

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \gamma_1 t^{-\alpha} \nabla \mathcal{L}\left(\hat{\theta}_{t-1} \mid Y_t, S_t, \mathbf{X}_t\right)$$

11 Let $K \leftarrow \{\lambda_i \in [n] \mid \exists \lambda_j \neq \lambda_i \text{ s.t. } \hat{v}_{t,j} - c_{t,j} \geq \hat{v}_{t,i} + c_{t,i}\}$
12 $\Lambda \leftarrow \Lambda \backslash K$
13 Generate $|\Lambda| - |K|$ new parameterizations using the genetic approach as described in Mesaoudi-Paul et al. [44, Section 3.3]

a problem instance set from the problem instance space $\mathcal{I}$ (line 5). It then calculates the contextual vectors $x_{t,i}\ \forall i \in [n]$ using the joint feature map from eq. (2.17) where the input of the joint feature map are the features of the parameters $f(\lambda)$ and the features of the problem instance $f(\boldsymbol{i})$ (line 7). After getting a context vector from the joint feature map, the estimated contextualized utility parameter $\hat{v}_{t,j}\ \forall j \in [n]$ is calculated using eq. (2.16). A subset $S_t \subset [n]$ of k parameterizations is selected from *ARM_SELECTION* procedure strategy like in Algorithm 1 (line 8). Initially, the UCB strategy was adopted as the *ARM_SELECTION* procedure as described in Algorithm 2. The framework was built to run the k parameterizations in parallel to solve the given problem instance resulting in a winner feedback scenario (line 9). The parallelism of running the parametrizations falls on the number of CPU cores available in the machine, i.e., $k = \#\{\text{CPU cores available}\}$. Stochastic gradient descent is used to update the estimate of θ after obtaining the winner's feedback (line 10). A racing strategy [42] (cf. Figure 2.4) is used to discard poor-performing parameterizations (lines 11-12), as discussed in Section 2.1.3 and shown in Figure 2.7. In order to prune poor-performing parameterizations, the upper bound on their estimated

skill parameter $\hat{v}_{t,i}$ must be lower than the respective lower bound of another parameterization. In the end, the discarded parameterizations are replaced by generating new ones according to a genetic approach described in Mesaoudi-Paul et al. [44, Section 3, 3] (line 13).

4.1.2 Dataset

For the dataset of this setup, we followed the dataset setup from Mesaoudi-Paul et al. [44] with a small amount of modification in the number of problem instances. We used the two solvers CaDiCaL [48] and Glucose [6] for the SAT problem. For CaDiCaL, we chose 89 discrete and 66 categorical parameter types. Similarly, for Glucose, we chose 9 categorical, 9 continuous, and 16 discrete parameter types. Due to the lack of support for non-numerical variables in Principal Component Analysis (PCA), categorical parameters are not considered in this framework.

For the problem instances, we used the last 5000 problem instances from *SW-GCPsat4j*[1] folder [26]. 54 features were chosen for the SAT instances. A PCA procedure was applied to reduce the dimensionality and correlation of the features for the considered problem scenario because solvers and problem instances have high-dimension features. CPPL's performance is not significantly affected by changing the dimensionality of the PCA, as shown in the experiments by Mesaoudi-Paul et al. [44, Section 4].

4.2 Challenges with Initial Framework

This section will look at our challenges while working on the initial CPPL framework. Since this framework was constructed by Mesaoudi-Paul et al. [44], our job was to make it more systematic. The framework was written in PYTHON language with only methods and no classes. The initial code was poorly organized and needed some reorganization by introducing header and helper classes. The framework was designed in such a way that the only result which could be collected was the execution time by the solvers on each problem instance. Since the framework terminates solving all the other parameterizations (arms) once a winner arm is declared, there

[1] https://www.cs.ubc.ca/labs/algorithms/Projects/ParamILS/aaai07-experiments.html

was no possibility of finding a regret using eq. (2.14) because we are using a weak regret of the dueling bandits problem in the contextual version. Therefore, there will be no regret from this framework, and the only comparison criteria would have been the execution time on each problem instance. This comparison criterion was insufficient to consider the correctness of an *ARM_SELECTION* strategy. Therefore, a regret comparison criterion was necessary, and in this framework setup, it is impossible to calculate that. Hence we opted for a different framework setup where we already know the run times of each parameterization solved by the solver on problem instances. This new framework gives us a clear image of the arm selection strategies' correctness and execution times. In Section 4.3, we will describe the new framework and the details of the dataset used in this setup.

4.3 Final CPPL Framework

As already described in Section 4.2 the challenges and the need for a new framework, this section provides a detailed view of the dataset used in this "final" setup, the differences between the "final" and the "initial" setup (cf. Algorithm 9), the hardware used for the experimentation and some new *ARM_SELECTION* strategies for UCB (cf. Algorithm 2 in Section 3.1.1) and CoLSTIM (cf. Algorithms 3 and 4 in Section 3.2.2) in a version of this framework setup.

4.3.1 Dataset

Following the experimental dataset of Mesaoudi-Paul et al. [43], we use $n = 20$ variants of SAPS solver [61], and CPLEX [31] solver. We also use $n = 15$ from the previous SAPS and CPLEX solver variants for diversity in the results.

SAPS

The authors produced the SAPS variants through randomly chosen parameterizations [43, Table 1] as a pool of candidate algorithms. SAPS variant parameterizations are used from [43, Table 1], where each parameterization represents an arm, and each

parameterization represents a feature vector of length 4. For completeness, we have provided the parameters in Appendix A.1.

Similar to [43], we used the last 5000 problem instances from the sat_SWGCP folder of the AClib[2]. In this setup, each instance is described by a feature vector with a length of 28, rather than the length of 54 in the previous setup. These feature vectors were generated using the SATzilla software[3]. To calculate the required running time $\boldsymbol{R}_{i,\lambda}$ for each instance-parameterization combination (i, λ), the ubcsat framework [60] was used, i.e., the time the solver takes with parametrization to terminate on problem instance i.

Before running the algorithms, the features of the instances were preprocessed by normalizing them to a range of $[0, 1]$. Features that were highly correlated with each other or had low variance (less than 0.01) were discarded. A greedy approach was used for the final preprocessing step, in which the most highly correlated remaining features (if the correlation exceeded a predefined threshold of 0.95) were removed iteratively. The final result was a feature representation with a size of 7.

CLPEX

In CPLEX, we used a variant of Combinatorial Auction (CA) problem [36]. A CA is a type of auction where bids are placed on groups of goods rather than single items. These instances are encoded as Mixed Integer Programming (MIP) problem and solved using CPLEX optimizer. We used the arbitrary domain for the CA problem. For CPLEX, we initially took 1500 problem instances with 30 features and 114 parameterizations. We did a preprocessing step on the parameters and features of the solver and instance problems, respectively. Similar to the preprocessing process for SAPS, we remove the parameterization with low variance (lower than 0.15) and those highly correlated with others. For features, we removed those with very low variance (we took a variance threshold of 0.01, similar to that of SAPS) and a correlation threshold of (0.9). After this preprocessing step, we end up with a parameterization representation of size 24 and a feature representation of size 13.

[2] http://www.aclib.net
[3] http://www.cs.ubc.ca/labs/beta/Projects/SATzilla

4.3.2 Functionality and Differences

In the functionality of the "final" framework, our main focus will be on the pre-selection step in Figure 2.7. The pseudo-code of this framework can be seen in Algorithm 1 in Section 2.5.6. The framework uses parameterizations as input instead of randomly initializing them on the spot. We use a time horizon T, equal to the number of problem instances for each solver to solve. For this framework, we use a SAPS solver with a feature vector of length 4 instead of 155 and 34 for CaDiCaL and Glucose, respectively. Therefore $\Lambda_{SAPS} \subseteq \mathbb{R}^{[n\times 4]}$ is the parameter space for the SAPS solver. Similarly, for the MIP dataset, we chose a CPLEX solver with length 24 for the parameter feature vector; hence $\Lambda_{CPLEX} \subseteq \mathbb{R}^{[n\times 24]}$ is the parameter space for CPLEX. We calculate the context vectors for dimension d before starting the execution using the joint feature mapping eq. (2.17) of the features of configurations $f(\lambda)\ \forall \lambda \in \Lambda$ and features of each instance $f(\boldsymbol{i})\ \forall \boldsymbol{i} \in I$. Please note that the dimension d of context vectors is different for each solver. This results in a context matrix $\mathbf{X} \in \mathbb{R}^{[T\times n\times d]}$. In each time step t, we observe a context vector $\mathbf{X_t}$ and compute the estimated contextualized utility parameter $\hat{v}_{t,i}\ \forall i \in [n]$. After getting the subset S_t from the *ARM_SELECTION* strategy, to observe winner feedback, we will look at the run times $\boldsymbol{R}_{\boldsymbol{i},\lambda}$ for the parametrization $\lambda \in S_t$ of the solver on the problem instance $\boldsymbol{i}$ at time step t. The parameterization with the least run time is declared the winner of S_t. After getting the winner's feedback, the algorithm updates the estimate θ and calculates the regret. Since we have run times for all the parameterizations beforehand, we can calculate the weak regret for the contextual version of the dueling bandits using eq. (2.13).

With regret in the picture for comparison criteria, we can compare the different *ARM_SELECTION* strategies as discussed in **??**. We use the cumulative regret from eq. (2.14) of the arm selection strategy UCB (cf. Algorithm 2 in Section 3.1.1) in this framework as the baseline for comparing the quality of S_t from different strategies. We run our experiments for different subset sizes $k \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 16\}$. Since we are using a stochastic online setting, to ensure that no randomness affects the regret predominantly, we run our experiments 50 times. Since we are using $n = 20$ and $n = 15$, we will check the cumulative regrets for $k \leq n/2$ and $k > n/2$. For $n = 15$, we excluded $k = 16$ from the subset sizes since it would not be feasible.

4.3.3 Hardware Setup

For our experiments, we initially used NOCTUA 1 cluster from the Paderborn Center for Parallel Computation (PC^2). For our initial setup (cf. Section 4.1), since we were parallelly running the arms based on the number of CPUs available on the system, we opted for Intel Xenon Gold "Sky-lake" 6148 with 40 cores, but we restricted the subset size to $k = 16$ for $n = 20$. However, the computational time to process all the problem instances for both CaDiCaL and Glucose took more than a day for a single subset size. That is also one of the reasons why we switched from the initial setup to our final setup. We initially ran our experiments in the final setup on NOCTUA 1. However, we were running the experiments for 50 reps parallelly, and each CPU had only 40 cores. Therefore, there was a delay in the 10 jobs execution. We also faced resource allocation delays when running in NOCTUA 1. Therefore, we migrated to NOCTUA 2.

For the completion of our experiments further, we used NOCTUA 2 cluster from PC^2. We ran our experiments parallelly on AMD Milan 7763, 2.45 GHz CPU with 50 cores. To match the average run time comparison criteria of the initial CPPL framework, we also calculated the average total execution time of each experiment. We will see these results in Chapter 5.

4.3.4 New Version of ARM_SELECTION Strategies for UCB and CoLSTIM

Additionally, we also created another version of the “final” framework, especially for arm selection strategies, UCB (cf. Algorithm 2 in Section 3.1.1) and CoLSTIM (cf. Algorithms 3 and 4 in Section 3.2.2). In this version of the strategies, half of the arms are chosen through exploration and the other half through exploitation. This type of strategy is novel and has not been used in any literature. In the following, we have given the pseudo code for this version of strategies for UCB and CoLSTIM. The main idea is that if we have an even number of k, we select the first half of the subset with arms based on exploitation and the other arms with exploration. If we have an odd number of k, we select $\lfloor \frac{k-1}{2} \rfloor + 1$ arms from the exploitation factor and the rest from the exploration. We call this “separated exploration-exploitation” approach or “Explore-Exploit” in short. Algorithms 10 to 12 shows the pseudo code of UCB, CoLSTIM-Context, and CoLSTIM-Contrast strategies using the Explore-

Exploit approach, respectively. We will discuss the results of these strategies in Chapter 5 and compare them with the strategies mentioned in Section 3.1.1 and Section 3.2.2 in terms of cumulative regret and the average execution time.

Algorithm 10: ARM_SELECTION: UCB separated exploration-exploitation

Input: Estimated skill parameter $\hat{v}_t$, confidence bound c_t, subset size k

if $k\%2 == 0$ **then**

  Choose $S_t\_exploit$ as:

$$\underset{S_t\_exploit \subseteq [n], |S_t\_exploit| = \frac{k}{2}}{\operatorname{argmax}} (\hat{v}_{t,i})$$

  Choose $c_{t,i}$ only for the arms which are not already in $S_t\_exploit$

  Choose $S_t\_explore$ as:

$$\underset{S_t\_explore \subseteq [n], |S_t\_explore| = \frac{k}{2}}{\operatorname{argmax}} (c_{t,i})$$

else

  Choose $S_t\_exploit$ as:

$$\underset{S_t\_exploit \subseteq [n], |S_t\_exploit| = \lfloor \frac{k-1}{2} \rfloor + 1}{\operatorname{argmax}} (\hat{v}_{t,i})$$

  Choose $c_{t,i}$ only for the arms which are not already in $S_t\_exploit$

  Choose $S_t\_explore$ as:

$$\underset{S_t\_explore \subseteq [n], |S_t\_explore| = \lfloor \frac{k-1}{2} \rfloor}{\operatorname{argmax}} (c_{t,i})$$

$S_t = S_t\_exploit \cup S_t\_explore$

return S_t

Algorithm 11: ARM_SELECTION: CoLSTIM-Context separated exploration-exploitation

Input: Estimated skill parameter $\hat{v}_t$, confidence bound c_t, subset size k, confidence threshold $C_{thresh} > 0$, perturbation distribution G with cumulative distribution function F

Sample $\tilde{\epsilon}_{t,i} \sim G \; \forall i \in [n]$

`// Sample perturbation variable`

$\epsilon_{t,i} = \min\left(C_{thresh}, \max\left(-C_{thresh}, \tilde{\epsilon}_{t,i}\right)\right) \forall i \in [n]$

`// Update trimmed perturbation variable`

if $k\%2 == 0$ **then**

  Choose $S_t\_exploit$ as:

$$\underset{S_t\_exploit \subseteq [n], |S_t\_exploit| = \frac{k}{2}}{\operatorname{argmax}} \left(\hat{v}_{t,i}\right)$$

  Choose $c_{t,i}$ only for the arms which are not already in $S_t\_exploit$

  Choose $S_t\_explore$ as:

$$\underset{S_t\_explore \subseteq [n], |S_t\_explore| = \frac{k}{2}}{\operatorname{argmax}} \left(\epsilon_{t,i} c_{t,i}\right)$$

else

  Choose $S_t\_exploit$ as:

$$\underset{S_t\_exploit \subseteq [n], |S_t\_exploit| = \lfloor \frac{k-1}{2} \rfloor + 1}{\operatorname{argmax}} \left(\hat{v}_{t,i}\right)$$

  Choose $c_{t,i}$ only for the arms which are not already in $S_t\_exploit$

  Choose $S_t\_explore$ as:

$$\underset{S_t\_explore \subseteq [n], |S_t\_explore| = \lfloor \frac{k-1}{2} \rfloor}{\operatorname{argmax}} \left(\epsilon_{t,i} c_{t,i}\right)$$

$S_t = S_t\_exploit \cup S_t\_explore$

return S_t

Algorithm 12: ARM_SELECTION: CoLSTIM-Contrast separated exploration-exploitation

Input: Estimated skill parameter $\hat{v}_t$, confidence bound c_t, subset size k, arms n, confidence threshold $C_{thresh} > 0$, perturbation distribution G with cumulative distribution function F, confidence width constant $c_1 > 0$

Sample $\tilde{\epsilon}_{t,i} \sim G \; \forall i \in [n]$

`// Sample perturbation variable`

$\epsilon_{t,i} = \min\left(C_{thresh}, \max\left(-C_{thresh}, \tilde{\epsilon}_{t,i}\right)\right) \forall i \in [n]$

`// Update trimmed perturbation variable`

Choose i_t as:

$$\underset{i \in [n]}{\operatorname{argmax}} \left(\hat{v}_{t,i} + \epsilon_{t,i} c_{t,i}\right)$$

for $k = 1, 2, ..., n$ **do**

  $z_{t,k,i_t} = \boldsymbol{x}_{t,k} - \boldsymbol{x}_{t,i_t}$ `// Create the contrast matrix with respect to best arm` i_t `in` S_t

Calculate confidence bound $c_{t,i}$ by replacing $\boldsymbol{x}_{t,i}$ with contrast vectors z_{t,i,i_t} in [43, Page 7]

Choose the rest of the candidates set J_t as:

if $k\%2 == 0$ **then**

  Choose $J_t\_exploit$ as:

$$\underset{J_t\_exploit \subseteq [n], |J_t\_exploit| = \frac{k}{2}}{\operatorname{argmax}} \left(z_{t,i,i_t}^{\top} \bar{\theta}\right)$$

  Choose $c_{t,i}$ only for the arms which are not already in $J_t\_exploit$

  Choose $J_t\_explore$ as:

$$\underset{J_t\_explore \subseteq [n], |J_t\_explore| = \frac{k}{2}}{\operatorname{argmax}} \left(c_1 c_{t,i}\right)$$

else

  Choose $J_t\_exploit$ as:

$$\underset{J_t\_exploit \subseteq [n], |J_t\_exploit| = \lfloor \frac{k-1}{2} \rfloor + 1}{\operatorname{argmax}} \left(z_{t,i,i_t}^{\top} \bar{\theta}\right)$$

  Choose $c_{t,i}$ only for the arms which are not already in $J_t\_exploit$

  Choose $J_t\_explore$ as:

$$\underset{J_t\_explore \subseteq [n], |J_t\_explore| = \lfloor \frac{k-1}{2} \rfloor}{\operatorname{argmax}} \left(c_1 c_{t,i}\right)$$

$J_t = J_t\_exploit \cup J_t\_explore$

if $i_t \in J_t$ **then**

  $S_t = J_t$

else

  Remove the least ranked candidate from J_t

  $S_t = J_t \cup i_t$

return S_t

5 Results

In this chapter, we will present the results of our experimental setup from Chapter 4. The results of the experiments will provide insight into the comparisons of the performances of the *ARM_SELECTION* strategies discussed in Chapter 3. We will present the results clearly and concisely, providing necessary details and analysis to help the reader understand the implications of the findings.

All the results shown in this chapter are on the framework described in Section 4.3 and on the hardware setup described in Section 4.3.3. Please note that since the strategies have very big names, we have abbreviated them as follows:

- ARM_SELECTION: UCB (cf. Algorithm 2 in Section 3.1.1): "UCB"
- ARM_SELECTION: CoLSTIM-Context (cf. Algorithm 3 in Section 3.2.2): "CoLSTIM-Context"
- ARM_SELECTION: CoLSTIM-Contrast (cf. Algorithm 4 in Line 4): "CoLSTIM-Contrast"
- Thompson Sampling for Bernoulli Bandits with Contextual Information (cf. Algorithm 5 in Section 3.2.3): "TS"
- Thompson Sampling for Contextual bandits (cf. Algorithm 6 in Line 9): "TSC"
- IndSelfSparring with Thompson Sampling for Bernoulli Bandits with Contextual Information (cf. Algorithm 7 in Section 3.2.4): "ISS"
- IndSelfSparring with Thompson Sampling for Contextual Bandits (cf. Algorithm 8 in Line 11): "ISSC"
- ARM_SELECTION: UCB separated exploration-exploitation (cf. Algorithm 10 in Section 4.3.4): "UCB Explore-Exploit"

- ARM_SELECTION: CoLSTIM-Context separated exploration-exploitation (cf. Algorithm 11 in Section 4.3.4): “CoLSTIM-Context Explore-Exploit”

- ARM_SELECTION: CoLSTIM-Contrast separated exploration-exploitation (cf. Algorithm 12 in Section 4.3.4): “CoLSTIM-Context Explore-Exploit”

5.1 Simulation Result

We calculated the cumulative regrets of all the *ARM_SELECTION* strategies for $n \in \{15, 20\}$. We have also calculated the mean execution time required to solve all the problem instances for the datasets discussed in Section 4.3.1. This helps in getting an estimate of how fast the strategy will work if applied as the *ARM_SELECTION* strategy in the initial setup discussed in Algorithm 9 in Section 4.1. Please note that some results of cumulative regret for MIP dataset (cf. Section 4.3.1) are very hard to distinguish in the linear scale plots. Therefore, along with the linear scale plots, we also provide logarithmic scale plots of such results in the following sections. For completeness, we provide the logarithmic scale plots of all the strategies with different k in appendix A.3.

The experimental results are divided into four subsections. Section 5.1.1 shows the results of *ARM_SELECTION* strategies discussed in Chapter 3 for $n = 20$. Similarly, Section 5.1.2 shows the results of *ARM_SELECTION* strategies discussed in Chapter 3 for $n = 15$. Section 5.1.3 shows the results of *ARM_SELECTION* strategies discussed in Section 4.3.4 for $n = 20$ and $n = 15$.

5.1.1 $n = 20$ with SAPS and CPLEX solver

We will discuss the setup results where we took 20 variants of SAPS and CPLEX solvers. We executed the *ARM_SELECTION* strategies for different subset sizes k as mentioned in Section 4.3.2. Figure 5.1 describes the linear scale cumulative regret curves for all the strategies in Chapter 3. Since the linear scale results of MIP dataset for $k \in \{9, 10, 16\}$ are very hard to distinguish, Figure 5.2 shows the regret curves in the logarithmic scale.

Figure 5.3 shows the mean execution times for the same experimental setup. Since the strategies from Algorithms 5 to 8 in Sections 3.2.3 and 3.2.4, respectively, have very less mean execution times compared to Algorithms 2 to 4, Figure 5.4 shows the execution times of these strategies separately. Again, the mean execution times for Algorithms 5 and 7 are relatively negligible compared to Algorithms 6 and 8, therefore, Figure 5.5 shows them to compare which one is the fastest among them.

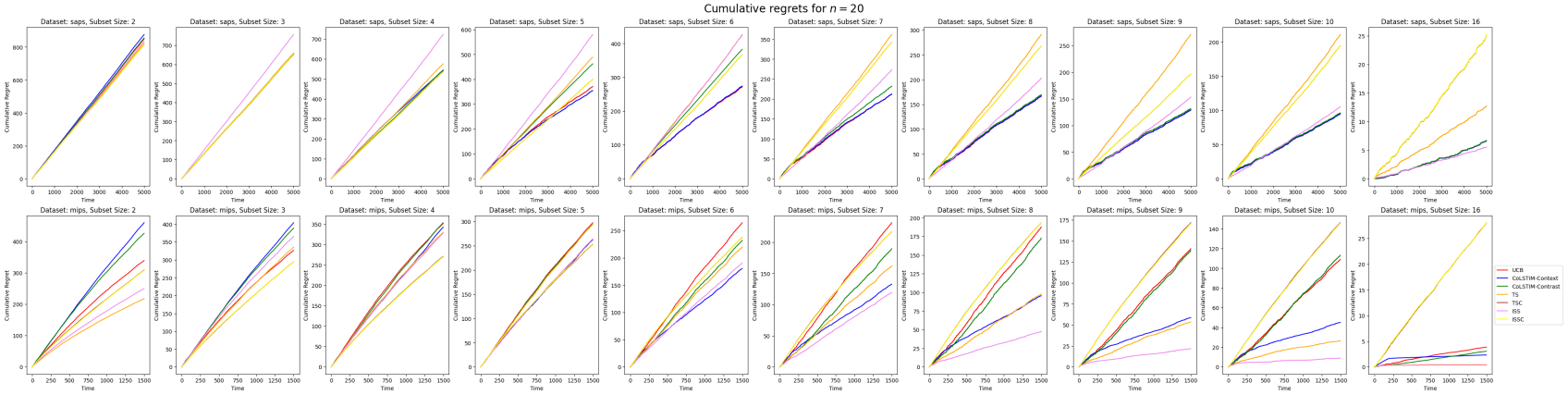

Fig. 5.1.: Cumulative regret for $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

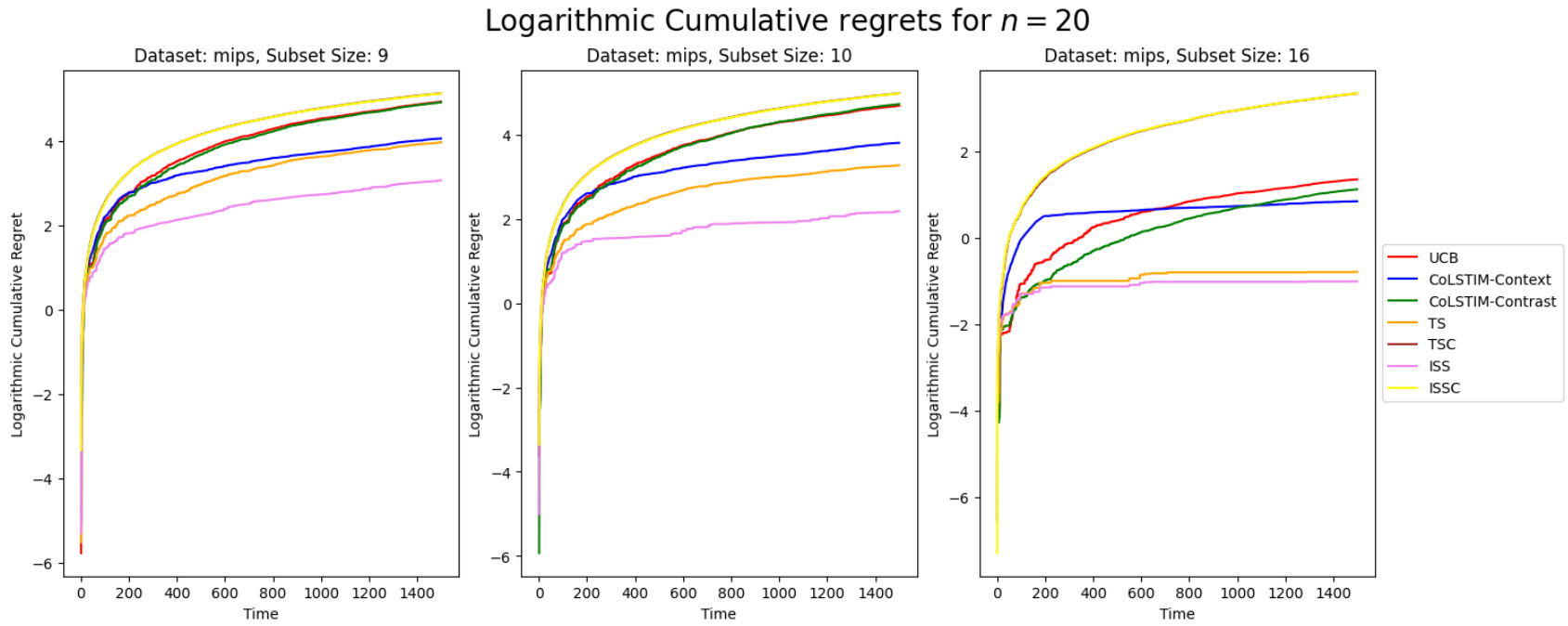

Fig. 5.2.: Logarithmic Cumulative regret for $n = 20$ with different subset sizes $(k \in \{9, 10, 16\})$ for MIP dataset on 50 repetition

Discussion and Analysis

Let us analyze the cumulative regret for the 20 arms setup. In the experiments on the SAPS solver for SAT problem from Figure 5.1, we can see that some of the strategies perform very close to the baseline strategy UCB (cf. Algorithm 2 in Section 3.1.1). CoLSTIM-Context (cf. Algorithm 3) and CoLSTIM-Contrast (cf. Algorithm 4) strategies, from Section 3.2.2, almost overlap the cumulative regrets of the baseline. The probability-matching strategies like Thompson Sampling (cf. Section 3.2.3) and Independent Self-Sparring (specifically IndSelfSparring for contextual bandits, cf. Algorithm 8) perform well for low subset sizes ($k \in \{2, 3, 4\}$). However, for a large subset size, they do not perform well in terms of cumulative regrets. IndSelfSparring with Thompson Sampling for Bernoulli Bandits with Contextual Information (cf. Algorithm 7) performs better than the baseline and the CoLSTIM strategies when $k > \frac{n}{2}$.

In the experiments on the CPLEX solver for MIP problem from Figure 5.1, we can see that most of the strategies perform much better than the baseline strategy. Please note that if $k = 2$, the setting changes to a dueling bandit setting with contextual information. In this setting, Thompson Sampling (abbreviated as TS in Figure 5.1) beats every other strategy. Almost every strategy performs better than the baseline for $k \in \{4, 5, 6, 7\}$. When we increase the subset size greater than 7, IndSelfSparring with Thompson Sampling for Bernoulli Bandits with Contextual Information (cf. Algorithm 7) has the lowest cumulative regret among the other strategies and suffers almost sublinear regret for $k \in \{9, 10, 16\}$ as shown in Figure 5.2.

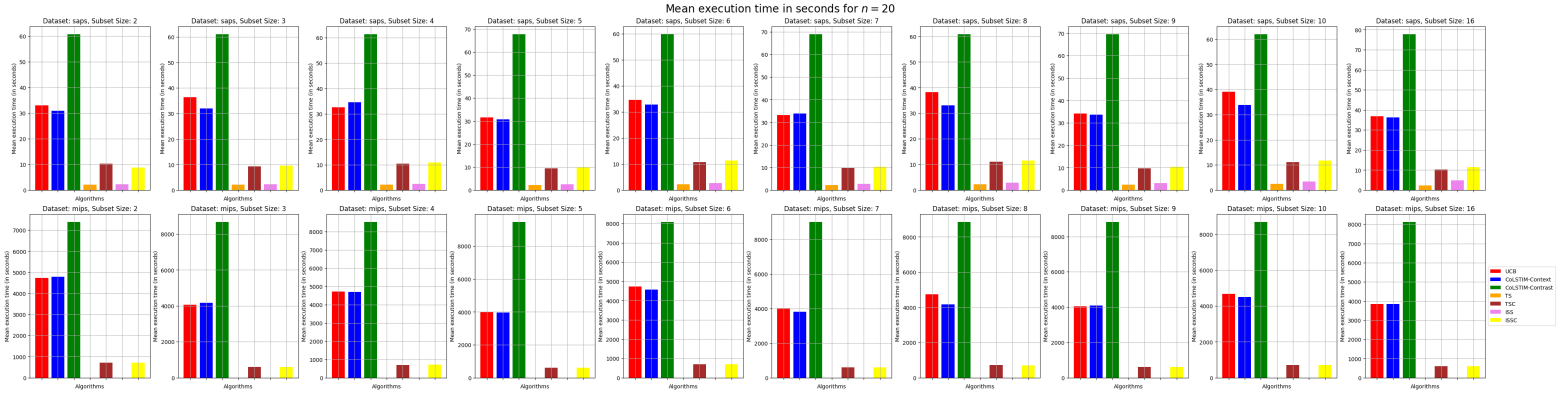

Fig. 5.3.: Mean execution times for $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

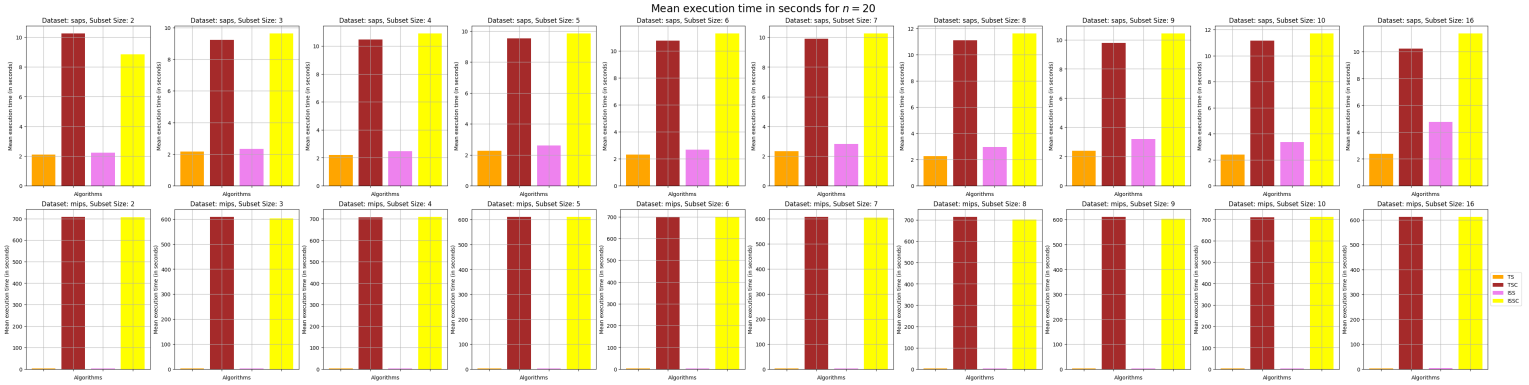

Fig. 5.4.: Mean execution times of Algorithms 5 to 8 for $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

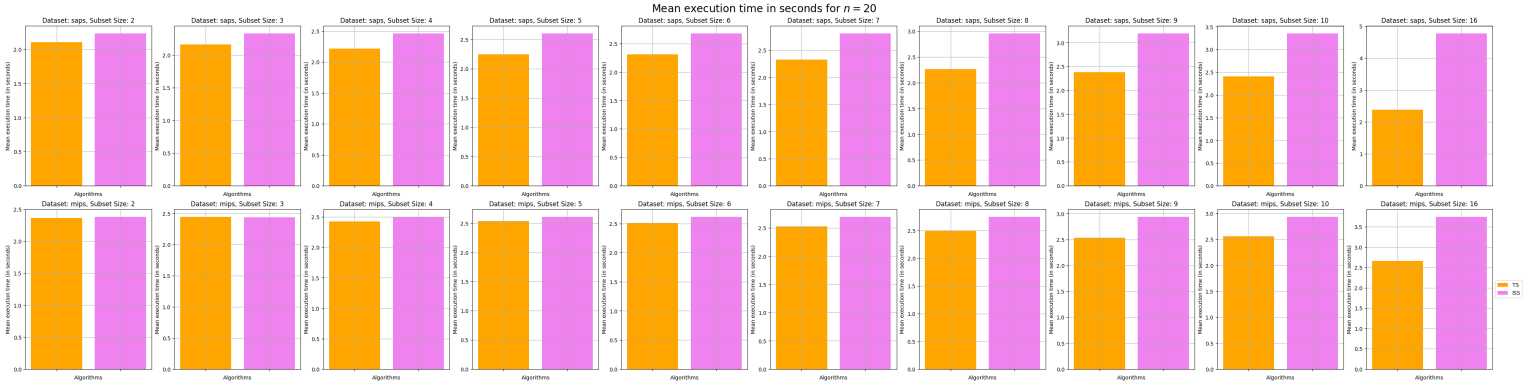

Fig. 5.5.: Mean execution times of Algorithms 5 and 7 for $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

Discussion and Analysis

For both SAPS and CPLEX solvers, *ARM_SELECTION* strategy CoLSTIM-Contrast (cf. Algorithm 4 in Section 3.2.2) has the highest mean execution times to solve all the problem instances. From Figure 5.3, we can see that the probability-matching strategies have the lowest run time. This is because they do not need to perform complex calculations to calculate confidence bounds but only need to learn the probability density functions. In the SAPS solver, for a small k, UCB (cf. Algorithm 2 in Section 3.1.1) has lower execution timings than the CoLSTIM strategies in Section 3.2.2. As we move closer to $k \leq \frac{n}{2}$, CoLSTIM-Context (cf. Algorithm 3) has lower execution times than the baseline strategy. From Figure 5.4, we can see that TSC (cf. Algorithm 6 in Section 3.2.3) and ISSC (cf. Algorithm 8 in Section 3.2.4) have comparable run times. These values fluctuate for different k, however, there is merely a difference of milliseconds in their run time. From Figure 5.5, TS (cf. Algorithm 5 in Section 3.2.3) has the lowest mean execution time for SAT and MIP datasets. There is only one case ($k = 3$) where the execution times of TS and ISS (cf. Algorithm 7 in Section 3.2.4) are the same for the MIP dataset. But in the others, TS is the fastest in both datasets.

5.1.2 $n = 15$ with SAPS and CPLEX solver

This section will discuss the setup results where we took 15 random variants of SAPS and CPLEX solvers from the previous variants. Similar to Section 5.1.1, we executed the *ARM_SELECTION* strategies for different subset sizes k. Note that, for a $n = 15$ setup, including $k = 16$ is not feasible. Therefore, we excluded it from our results for the 15 arm setup. Figure 5.6 describes the linear scale cumulative regret curves. Since the linear scale results of MIP dataset for $k \in \{8, 9, 10\}$ are very hard to distinguish, Figure 5.7 shows the regret curves in the logarithmic scale.

Figure 5.8 shows the mean execution times for the same experimental setup. Similar to Section 5.1.1, Figure 5.4 shows the execution times of strategies from Algorithms 5 to 8 in Sections 3.2.3 and 3.2.4, respectively, separately. Again, the mean execution times for Algorithms 5 and 7 are relatively negligible compared to Algorithms 6 and 8, therefore, Figure 5.10 shows them to compare which one is the fastest among them.

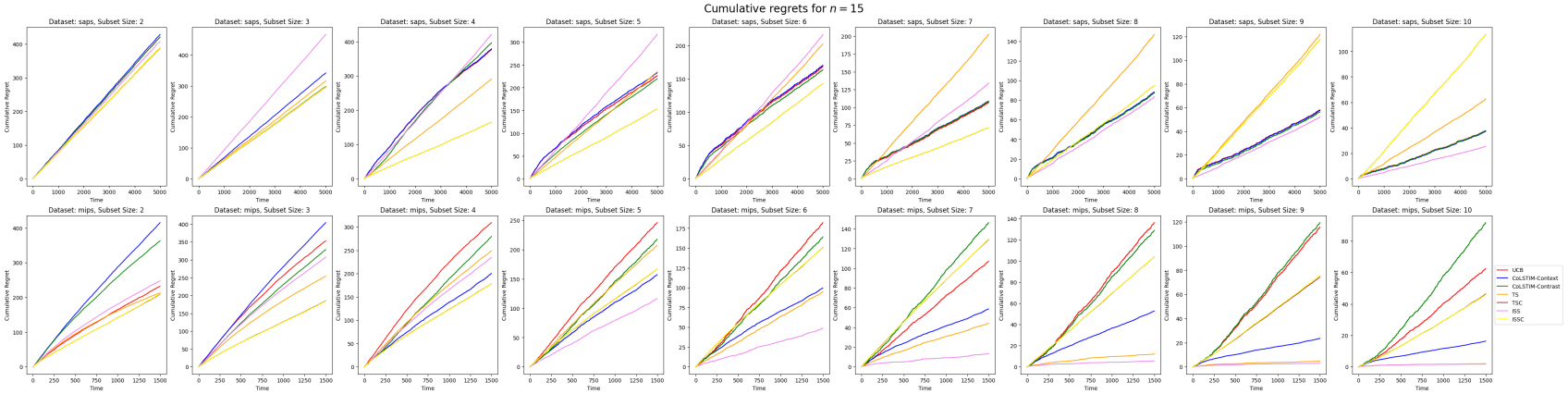

Fig. 5.6.: Cumulative regret for $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

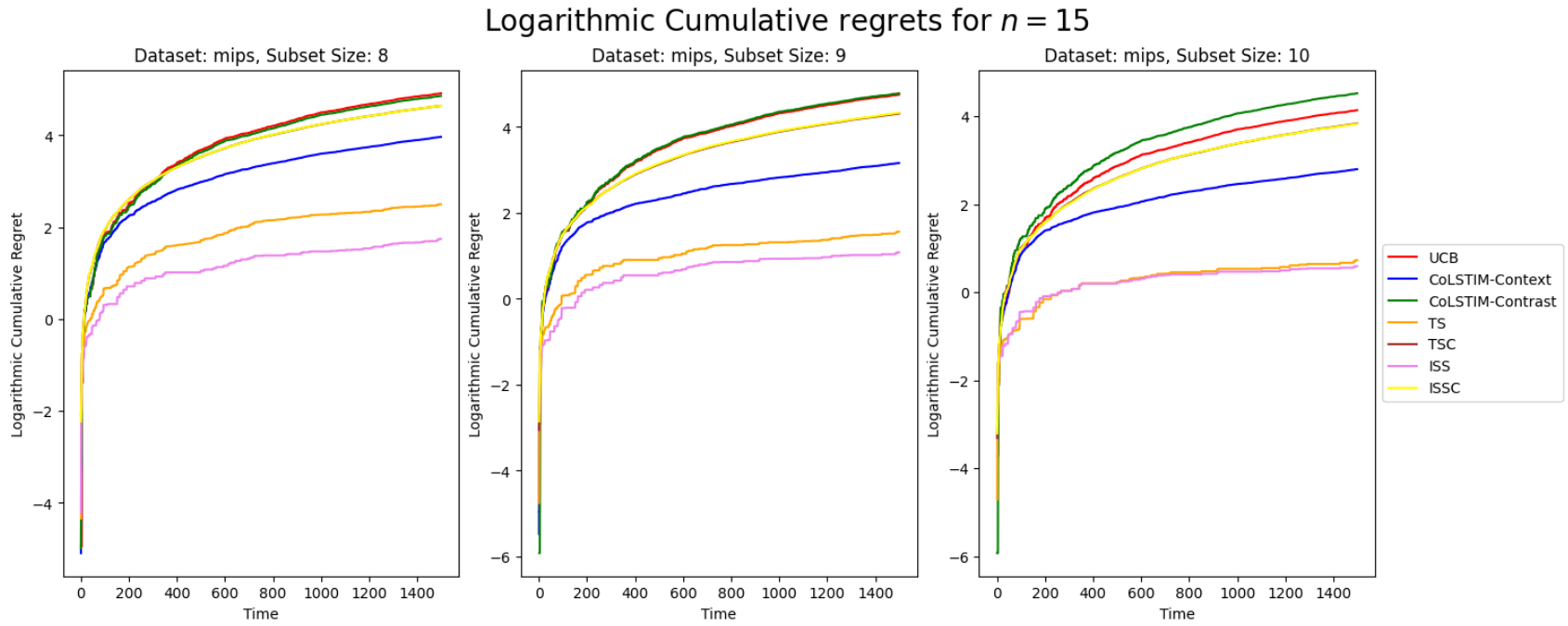

Fig. 5.7.: Logarithmic Cumulative regret for $n = 15$ with different subset sizes $(k \in \{8, 9, 10\})$ for MIP dataset on 50 repetition

Discussion and Analysis

In a 15 arm setup for SAPS solver, again, some arm strategies perform better than the baseline strategy for $k < \frac{n}{2}$. However, for $k > \frac{n}{2}$, only a few perform well. ISSC (cf. Algorithm 8 in Section 3.2.4) is a clear winner for $k < \frac{n}{2}$, however, ISS (cf. Algorithm 7 in Section 3.2.4) takes over the lead for $k > \frac{n}{2}$. CoLSTIM strategies (cf. Algorithms 3 and 4 in Section 3.2.2) again match the cumulative regrets similar to the baseline. TS (cf. Algorithm 5 in Section 3.2.3) have an unpredictable result performing worst in some cases and performing well than the baseline in others.

For the CPLEX solver, for $k \in \{2, 3, 4\}$, TSC (cf. Algorithm 6 in Section 3.2.3) and ISSC beat every other strategy and have the lowest cumulative regrets. Things change for the other values of k; strategy ISS beats every other strategy and has the least cumulative regret. It almost has a sub-linear regret for $k > \frac{n}{2}$. CoLSTIM-Context strategy (cf. Algorithm 3) also beats the baseline strategy for almost every value of k.

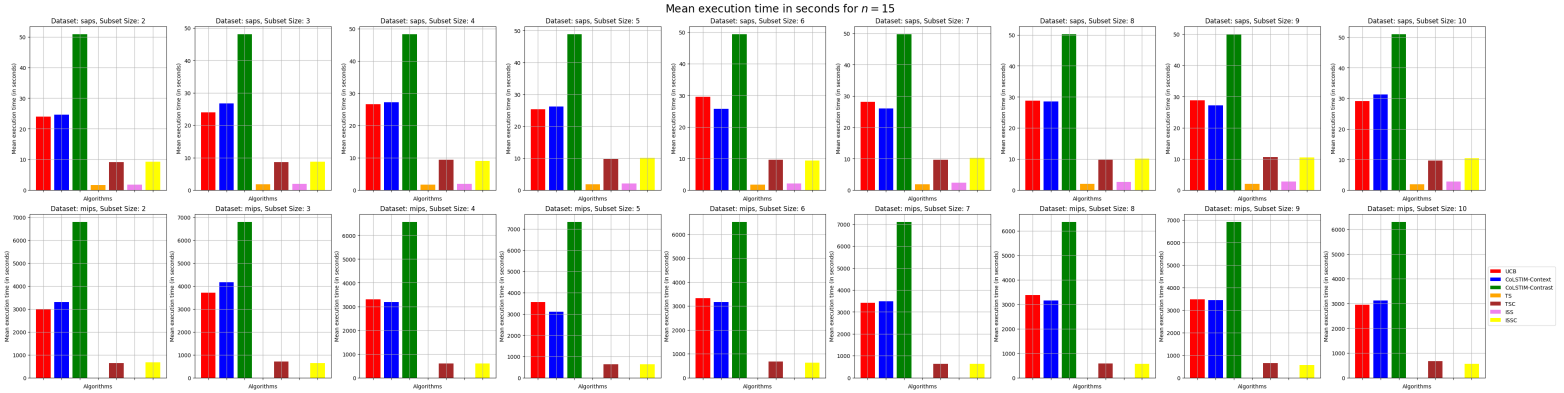

Fig. 5.8.: Mean execution times for $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

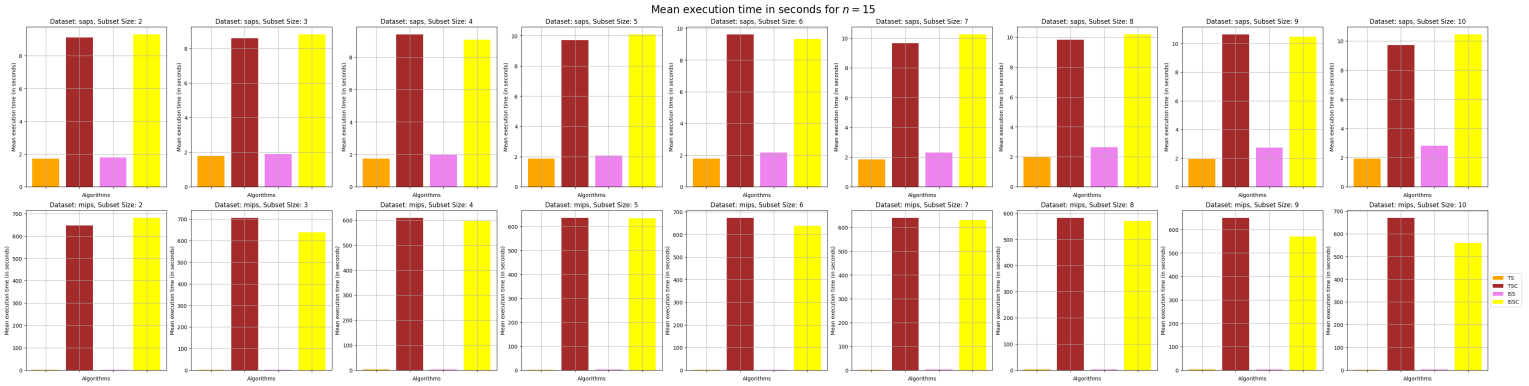

Fig. 5.9.: Mean execution times of Algorithms 5 to 8 for $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

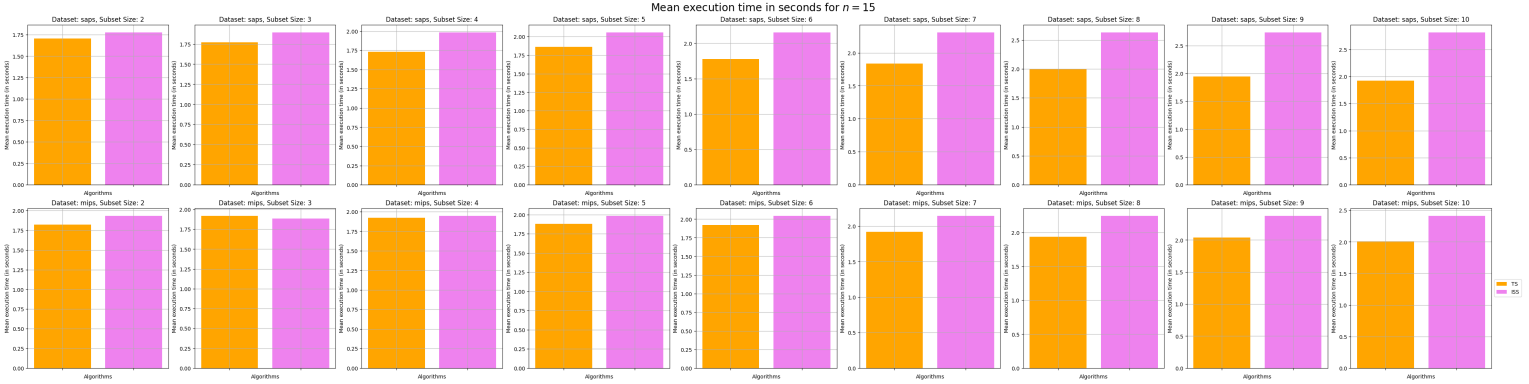

Fig. 5.10.: Mean execution times of Algorithms 5 and 7 for $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

Discussion and Analysis

The analysis of the mean execution times for the $n = 15$ setup is similar to the $n = 20$ setup. TS is the fastest arm strategy among the rest, and CoLSTIM-Contrast is the slowest.

5.1.3 Comparison of Results for New Version of Arm Selection Strategies for UCB and CoLSTIM

In this section, we will compare the results of different versions of *ARM_SELECTION* strategy for UCB and CoLSTIM inspired from Section 4.3.4. This section is then divided into two subsections for the different number of arms variation.

$n = 20$ with SAPS and CPLEX solver executed 50 times

This section shows the results of 20 variants of SAPS and CPLEX solvers. Similar to Section 5.1.1, we executed the *ARM_SELECTION* strategies for different subset sizes k.

Figure 5.11 describes the linear scale cumulative regret curves. Since the linear scale results of MIP dataset for $k \in \{16\}$ are very hard to distinguish, Figure 5.12 shows the regret curves in the logarithmic scale.

Figure 5.13 shows the mean execution times for the same experimental setup.

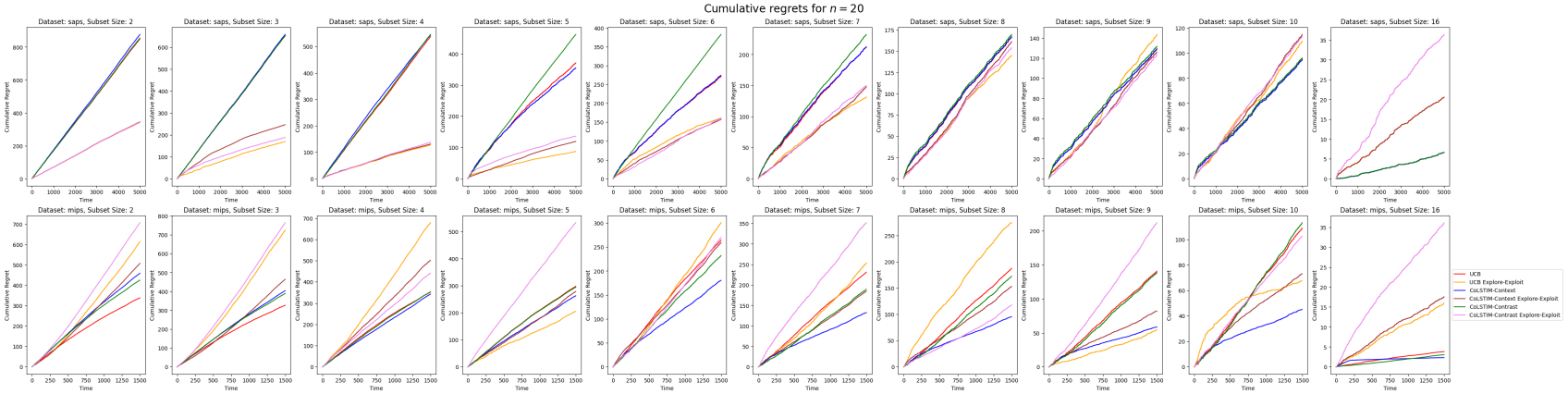

Fig. 5.11.: Cumulative regret for for Algorithm 10, Algorithm 11, Algorithm 12, $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

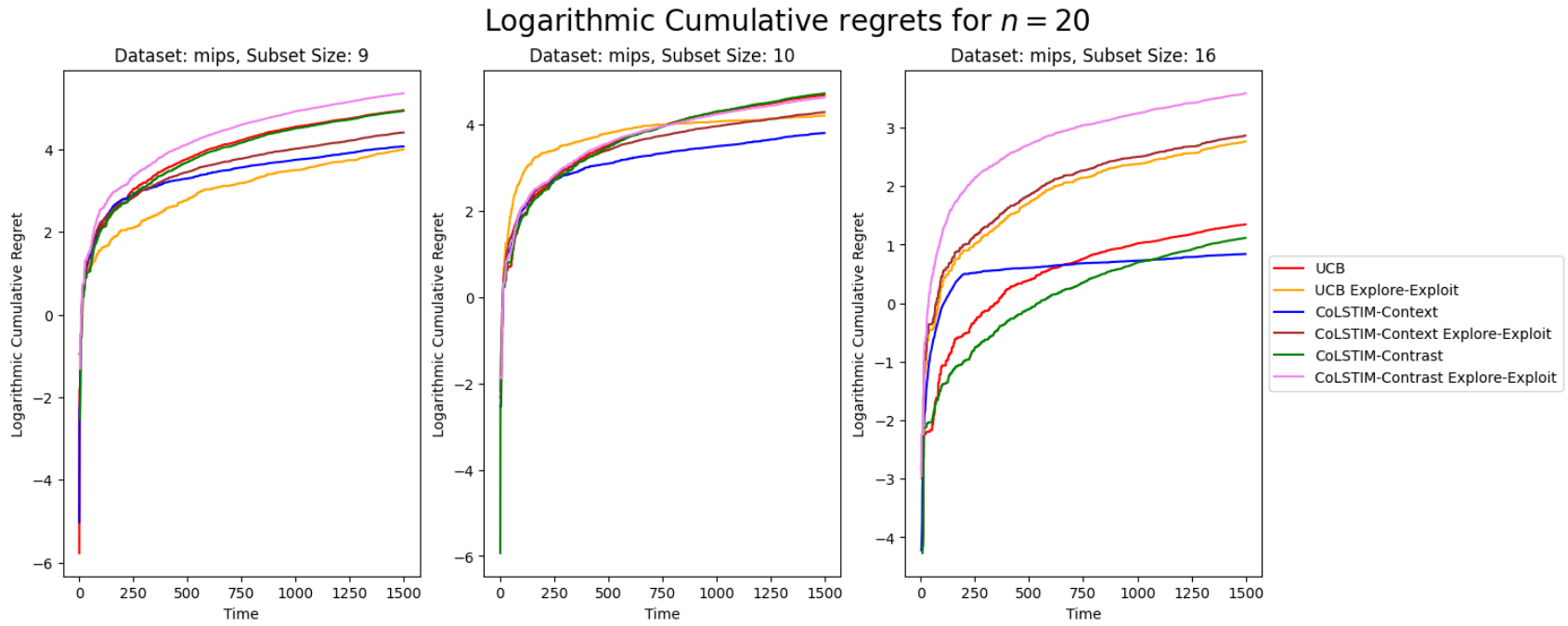

Fig. 5.12.: Logarithmic Cumulative regret for Algorithm 10, Algorithm 11, Algorithm 12, $n = 20$ with different subset sizes $(k \in \{9, 10, 16\})$ for MIP dataset on 50 repetition

Discussion and Analysis

We now analyze the cumulative regrets of Algorithms 2 to 4 in Sections 3.1.1 and 3.2.2, respectively and the novel strategies of separated exploration and exploitation from Algorithms 10 to 12 in Section 4.3.4 for $n = 20$ setup. The difference between the Explore-Exploit strategies from Section 4.3.4 and the strategies used by Algorithms 2 to 4 from Chapter 3 is that the former choose half of the arms in subset S_t using exploration and the other half through exploitation. From Figure 5.11, we can see that for the SAPS solver, the Explore-Exploit strategies perform comparatively well for $k < \frac{n}{2}$. UCB Explore-Exploit strategy from Algorithm 10 can be seen as a winner for many cases. However, for $k \geq \frac{n}{2}$, the Explore-Exploit strategies performs worse, particularly, CoLSTIM-Contrast Explore-Exploit (cf. Algorithm 12).

For CPLEX, however, the old strategies still prevail with the least cumulative regrets. For $k \in \{5, 9\}$, the UCB Explore-Exploit strategy works better than the others but fails miserably for $k \in \{2, 3, 4, 8\}$. CoLSTIM-Contrast Explore-Exploit was the least favorable of them in most cases.

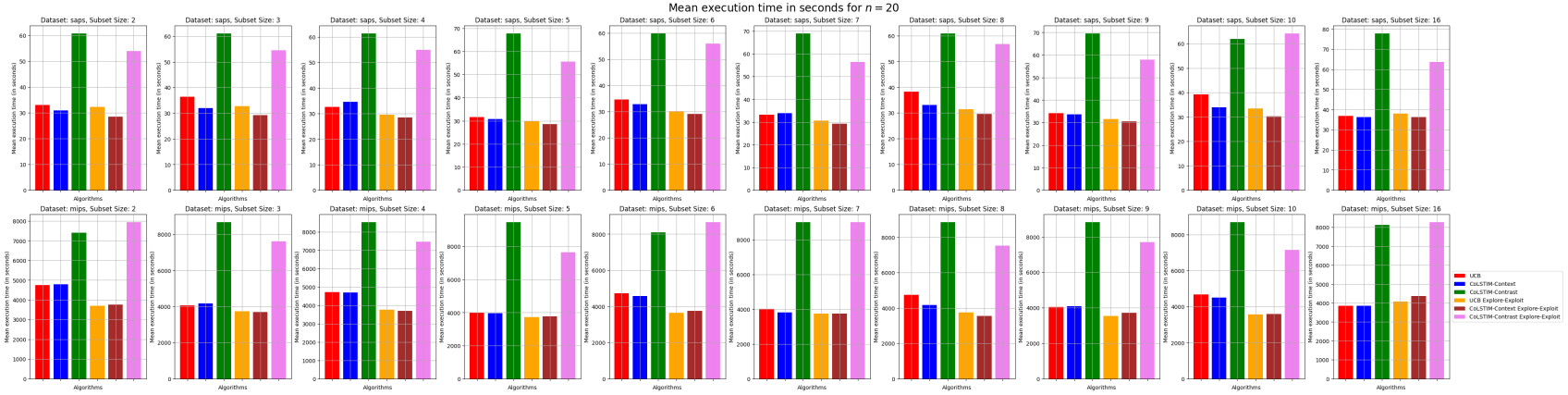

Fig. 5.13.: Mean execution times for Algorithm 10, Algorithm 11, Algorithm 12, $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

Discussion and Analysis

As for the mean execution timing comparison, Figure 5.13 shows that the strategies using the separated exploration-exploitation method (cf. Algorithms 10 to 12 in Section 4.3.4) are faster than the algorithms which do not use this method (cf. Algorithms 2 to 4 in Chapter 3). However, they are still slower than the probability-matching strategies. A significant difference can be seen in the execution timings of CoLSTIM-Contrast (cf. Algorithm 4 in Section 3.2.2). The timings of UCB Explore-Exploit (cf. Algorithm 10 in Section 3.1.1) and CoLSTIM-Context Explore-Exploit (cf. Algorithm 11) are still comparable and fluctuate for different subset sizes. Though their run times are still lower than the CoLSTIM-Contrast Explore-Exploit (cf. Algorithm 12).

$n = 15$ with SAPS and CPLEX solver executed 50 times

This section shows the setup results where we took 15 random variants of SAPS and CPLEX solvers from the previous variants. Similar to the previous setup of $n = 20$, we executed the *ARM_SELECTION* strategies for different subset sizes k. However, we excluded the case where $k = 16$.

Figure 5.14 describes the linear scale cumulative regret curves. Since the linear scale results of MIP dataset for $k \in \{8, 9, 10\}$ are very hard to distinguish, Figure 5.15 shows the regret curves in the logarithmic scale.

Figure 5.16 shows the mean execution times for the same experimental setup.

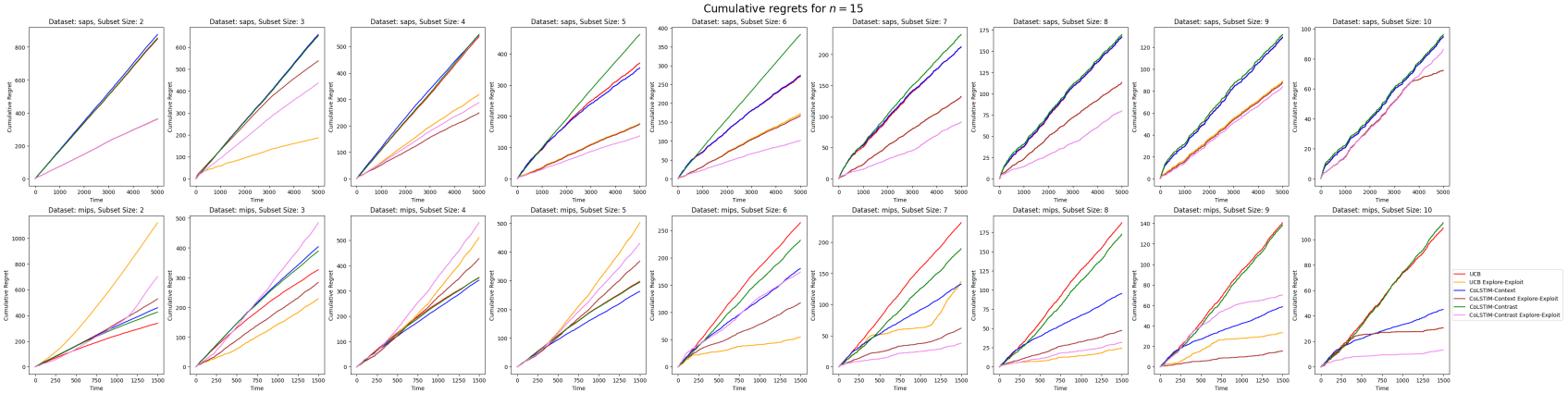

Fig. 5.14.: Cumulative regret for for Algorithm 10, Algorithm 11, Algorithm 12, $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

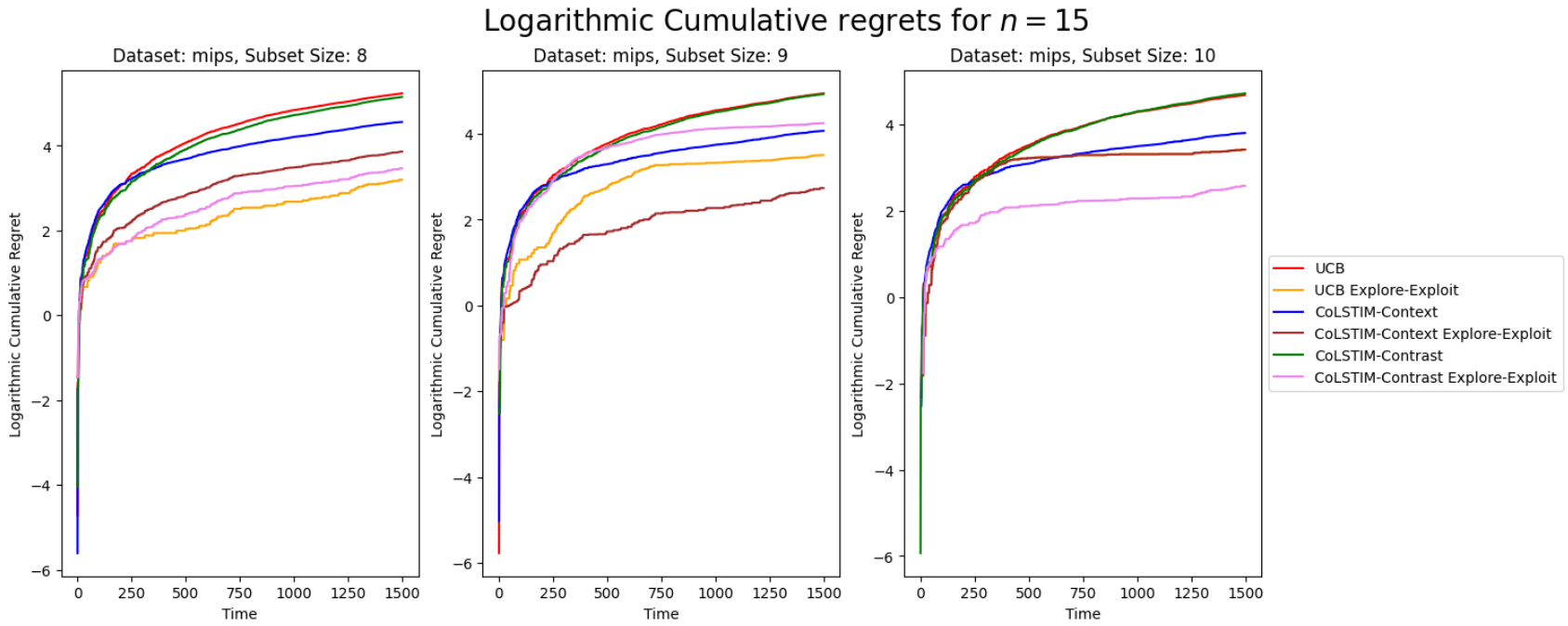

Fig. 5.15.: Logarithmic Cumulative regret for Algorithm 10, Algorithm 11, Algorithm 12, $n = 15$ with different subset sizes $(k \in \{9, 10, 16\})$ for MIP dataset on 50 repetition

Discussion and Analysis

For a $n = 15$ setup and using SAPS solver, CoLSTIM-Contrast Explore-Exploit (cf. Algorithm 12) works best in most cases, unlike in the $n = 20$ setup. The regrets of UCB Explore-Exploit (cf. Algorithm 10) and CoLSTIM-Context Explore-Exploit (cf. Algorithm 11) are almost similar for $k \in \{5, 6, 7, 8, 9, 10\}$. Although for $k = 3$, the UCB Explore-Exploit strategy works the best. Unlike the setup of $n = 20$, the separated Exploration-Exploitation strategies performed very well for all subset sizes.

For the CPLEX solver, UCB Explore-Exploit strategy works well for $k \in \{3, 6, 8\}$. For $k \in \{9, 10\}$, the CoLSTIM-Context Explore-Exploit and CoLSTIM-Contrast Explore-Exploit has a non-increasing logarithmic cumulative regret as seen in Figure 5.15. Not all Explore-Exploit strategies worked better for $k \in \{2, 4, 5\}$. A similar case is also seen for $k \in \{4, 5\}$.

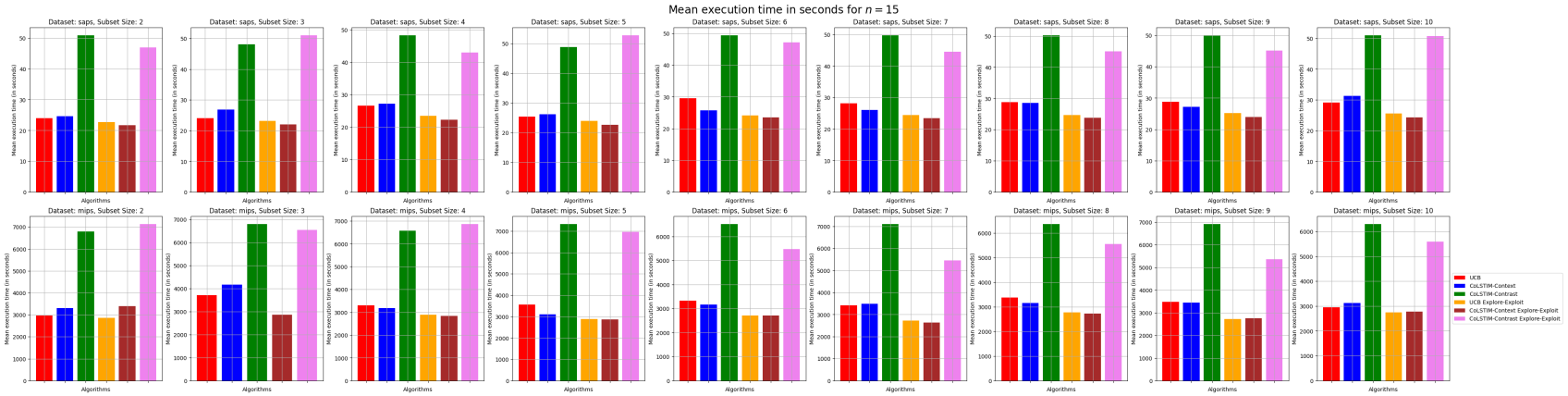

Fig. 5.16.: Mean execution times for Algorithm 10, Algorithm 11, Algorithm 12, $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

Discussion and Analysis

Similar to our analysis for 20 arms setting in Section 5.1.3, the mean execution times of the Explore-Exploit strategies are less than the strategies in Sections 3.1.1 and 3.2.2. The CoLSTIM-Contrast (cf. Algorithm 4) is still the slowest running strategy. Similar to $n = 20$, the execution timings of the UCB (cf. Algorithm 2) and CoLSTIM-Context (cf. Algorithm 3) have comparable running times and fluctuate in different subset sizes.

5.2 Conclusion

In the real-world problem of using the CPPL framework from Algorithm 9, the subset size usually equals the number of CPU cores available. Since most modern computers have at least 8 core CPUs, therefore, looking at the results for $k \geq 8$, the arm strategy IndSelfSparring with Thompson Sampling for Bernoulli Bandits with Contextual Information (cf. Algorithm 7), works best. This is determined by comparing the cumulative regrets for the CPLEX solver and the regret curves for the SAPS solver for both $n = 20$ and $n = 15$ setups. As for the performance related to the execution time, it also has the least execution times and is a strong contender for replacing the UCB strategy from the CPPL framework in Section 4.1.1. Thompson Sampling for Bernoulli Bandits with Contextual Information (cf. Algorithm 5) is also a possibility in terms of execution performance. However, it has mediocre regret curves for SAPS solver.

The Explore-Exploit arm selection strategies from Section 4.3.4 are also fine challengers for the *ARM_SELECTION* procedure (line 8) in Algorithm 9, especially for the CPLEX solver in the $n = 15$ setup. For SAPS solver and $k > 8$, the Explore-Exploit strategies are also good contenders.

6 Conclusion and Future work

In this chapter, we will summarize our work and findings and will discuss the implications of our work. The chapter also highlights our study's limitations and will suggest future research directions.

The CPPL algorithm is a state-of-the-art algorithm for Realtime Algorithm Configuration. It is faster and better than the other known algorithm configurators like ReACT and ReACTR. CPPL uses the concept of Preselection Bandits with contextual information to select the set of parameters from a pool of parameters for the solver to configure to solve the problem instances of a particular problem.

In this thesis, we conducted experiments with different parameter configuration (termed as arm) selection strategies, which we refer to as the *ARM_SELECTION* procedure in the CPPL framework (cf. Algorithm 1). We were trying to find a good alternative to the UCB (cf. Section 3.1.1) strategy for better results. For that, we used the variant of Multi-armed Bandit called the Multi-dueling bandits paired with contextual information.

Based on our experiments, we believe that the CPPL framework has the potential to perform even better in the future. By replacing the UCB arm selection procedure for selecting a subset of configurations with a faster and better arm selection strategy as shown in Chapter 5, the CPPL framework could be improved in many ways.

For the scope of our thesis, we were limited by the time to test further combinations of contextual and multi-dueling bandit algorithms to form a sub-optimal strategy. More algorithms that ensure not only a Condorcet winner but also Copeland and Borda winners (cf. Section 2.3.4) can also be explored to generate a better strategy for the *ARM_SELECTION* procedure.

The CPPL framework can also be experimented on different problems instead of just being confined to SAT and MIP problems. An experimental setup can also be done to check the IndSelfSparring with Thompson Sampling for Bernoulli Bandits with Contextual Information (cf. Algorithm 7) on CaDiCaL and Glucose solvers since it

shows promising results in terms of cumulative regrets for SAPS and CPLEX solvers (cf. Section 5.2). Since contextual information was lacking in other datasets we discovered, we were only restricted to the dataset mentioned in Section 4.3.1. A further experimental study on different datasets will also benefit this field.

Appendix

A

In this Chapter, we include additional materials used in the main chapters of this thesis.

A.1 SAPS Solver Parameter Configurations

α	ρ	ps	wp
1.54114	0.851212	0.739441	0.846641
1.85872	0.662701	0.532759	0.193693
1.48834	0.839351	−0.219562	0.973629
0.807838	0.787876	0.634953	0.987749
1.27937	0.78502	0.871624	0.769004
1.08792	0.0999449	0.624429	−0.271802
1.65351	0.828302	0.711434	0.952419
1.45122	0.515806	0.804639	−0.202731
1.6245	0.553748	0.464212	0.419496
1.40361	0.728122	0.743332	0.733345
1.92961	0.91072	0.995255	0.958284
1.40955	0.740847	−0.0788468	−0.367871
1.79481	0.82973	0.81912	0.318974
1.8575	0.654704	0.588427	0.466252
1.74699	0.699139	0.610026	0.96016
1.88793	0.464533	0.924795	0.727195
1.6724	0.830823	0.704286	0.791863
1.97717	0.941355	0.824751	0.935904
1.71295	−0.955414	0.988518	0.585664
0.859881	0.932354	−0.0785504	0.496108

Tab. A.1.: SAPS parameterizations (Adapted from [43, Table 1])

In this section, we will provide further details on the parameterizations of the SAPS solver (cf. Section 4.3.1), which we used in our experiments to help the reader better understand the information presented and reproduce the results. Table A.1 reports the parameterizations of the 20 SAT solver used in Section 4.3.1 for our experiments.

A.2 Source Code Repository

We also provide the source code links on GitHub here for the initial experimental setup from Section 4.1 and the final setups from Section 4.3.

- **Initial Setup:** `https://github.com/sshivam95/CPPL`
- **Final Setup:** `https://github.com/sshivam95/Multi-Dueling-Bandits`
- **Final Setup for new strategies:** `https://github.com/sshivam95/Multi-Dueling-Bandits/tree/Framework-v2`

A.3 Logarithmic Cumulative Regret Curves of SAT and MIP Dataset

As already shown and discussed in Chapter 5, we have only shown the logarithmic cumulative regrets of MIP dataset to compare the regrets better. However, for the sake of completeness, we will provide the logarithmic cumulative regret curves of all datasets with different k for $n \in \{15, 20\}$.

Figure A.1 and Figure A.2 shows the logarithmic cumulative regret of all the arm strategies for $n = 20$ and $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition respectively. Similarly, Figure A.3 and Figure A.4 shows the logarithmic cumulative regret of the Explore-Exploit arm strategies (cf. Algorithms 10 to 12 in Section 4.3.4) for $n = 20$ and $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition respectively.

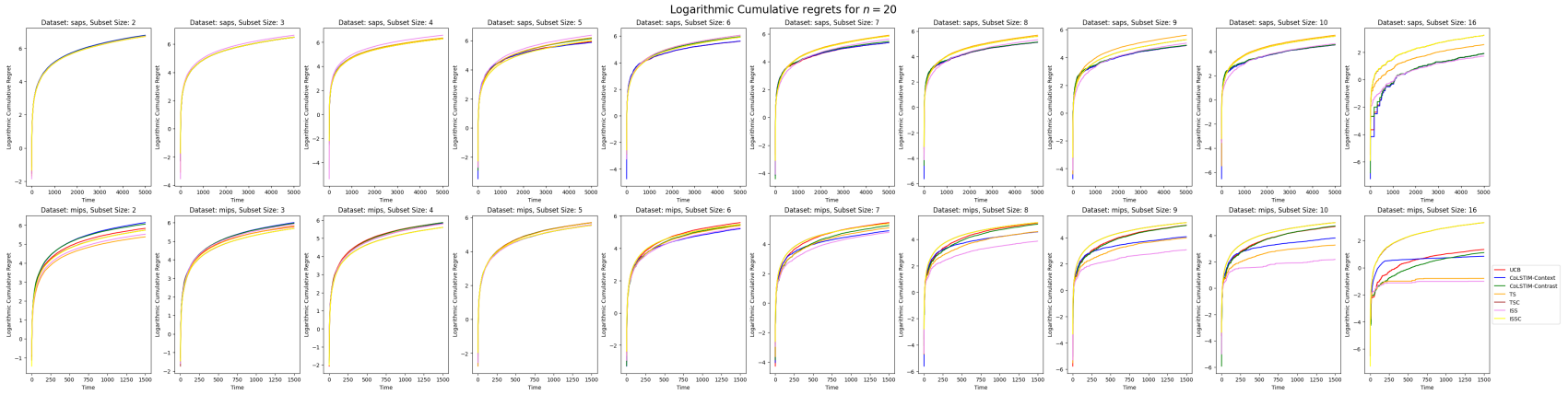

Fig. A.1.: Logarithmic Cumulative regret for $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

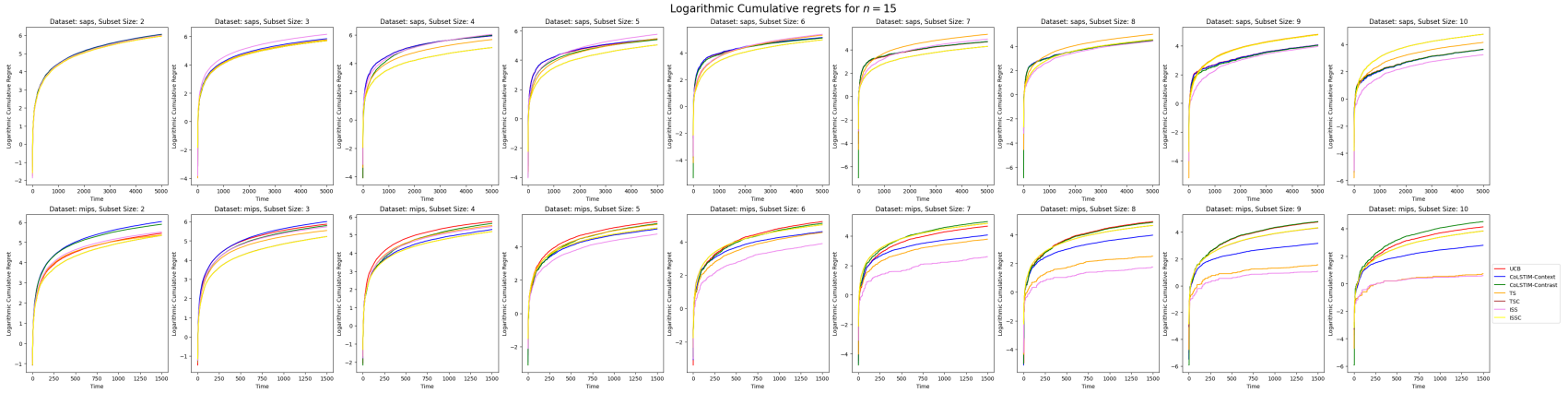

Fig. A.2.: Logarithmic Cumulative regret for $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

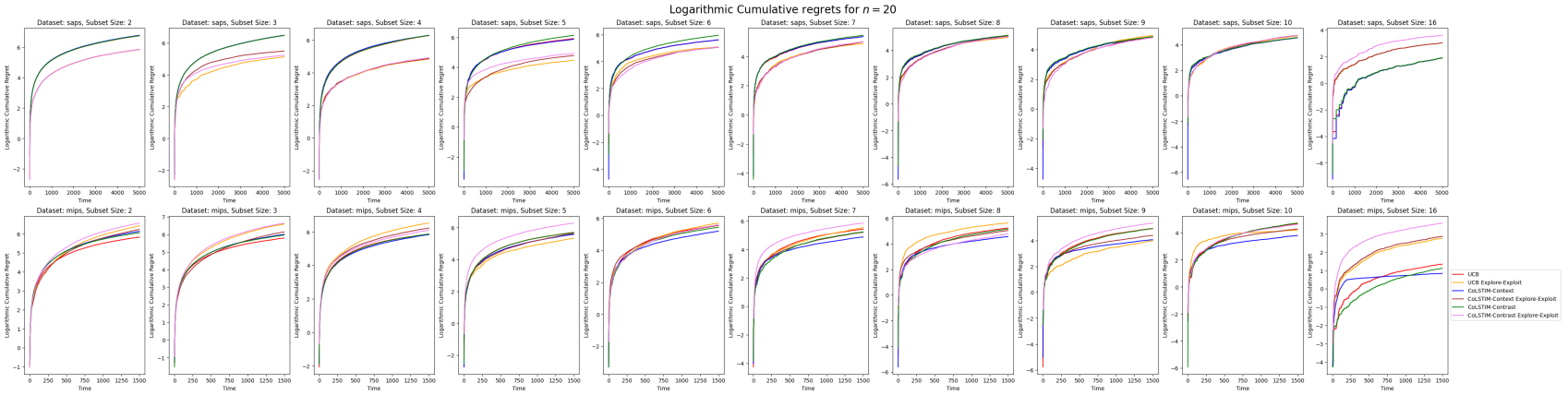

Fig. A.3.: Logarithmic Cumulative regret of Explore-Exploit strategies for $n = 20$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

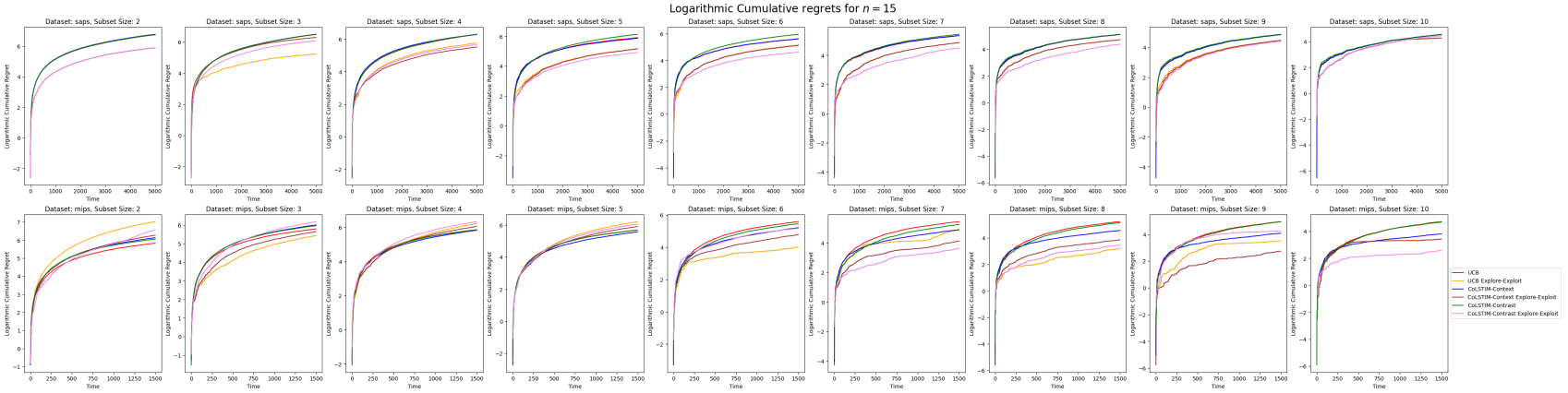

Fig. A.4.: Logarithmic Cumulative regret of Explore-Exploit strategies for $n = 15$ with different subset sizes (k) for SAPS solver and MIP dataset on 50 repetition

Bibliography

[1] Alekh Agarwal, Daniel Hsu, Satyen Kale, et al. "Taming the monster: A fast and simple algorithm for contextual bandits". In: *International Conference on Machine Learning*. PMLR. 2014, pp. 1638–1646 (cit. on p. 6).

[2] Arpit Agarwal, Nicholas Johnson, and Shivani Agarwal. "Choice bandits". In: *Advances in neural information processing systems* 33 (2020), pp. 18399–18410 (cit. on p. 35).

[3] Shipra Agrawal and Navin Goyal. "Thompson sampling for contextual bandits with linear payoffs". In: *International conference on machine learning*. PMLR. 2013, pp. 127–135 (cit. on pp. 6, 40, 42, 43).

[4] Nir Ailon, Zohar Karnin, and Thorsten Joachims. "Reducing dueling bandits to cardinal bandits". In: *International Conference on Machine Learning*. PMLR. 2014, pp. 856–864 (cit. on p. 44).

[5] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. "Model-based genetic algorithms for algorithm configuration". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015 (cit. on pp. 5, 25).

[6] Gilles Audemard and Laurent Simon. "Glucose and Syrup in the SAT'17". In: *Proceedings of SAT Competition* (2017), pp. 16–17 (cit. on pp. 49, 51).

[7] Peter Auer. "Using confidence bounds for exploitation-exploration trade-offs". In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422 (cit. on pp. 2, 6, 35, 43).

[8] Viktor Bengs and Eyke Hüllermeier. "Preselection bandits". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 778–787 (cit. on pp. 3, 4, 20).

[9] Viktor Bengs, Aadirupa Saha, and Eyke Hüllermeier. "Stochastic Contextual Dueling Bandits under Linear Stochastic Transitivity Models". In: *arXiv preprint arXiv:2202.04593* (2022) (cit. on pp. 36–38).

[10] Eric Brochu, Matthew W Hoffman, and Nando de Freitas. "Portfolio allocation for Bayesian optimization". In: *arXiv preprint arXiv:1009.5419* (2010) (cit. on p. 6).

[11] Brian Brost, Yevgeny Seldin, Ingemar J Cox, and Christina Lioma. "Multi-dueling bandits and their application to online ranker evaluation". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 2016, pp. 2161–2166 (cit. on p. 35).

[12] Robert Busa-Fekete, Eyke Hüllermeier, and Adil El Mesaoudi-Paul. "Preference-based online learning with dueling bandits: A survey". In: *arXiv e-prints* (2018), arXiv–1807 (cit. on pp. 5, 6, 16, 17, 20, 33, 40).

[13] Nicolo Cesa-Bianchi and Gábor Lugosi. “Combinatorial bandits”. In: *Journal of Computer and System Sciences* 78.5 (2012), pp. 1404–1422 (cit. on p. 35).

[14] Olivier Chapelle and Lihong Li. “An empirical evaluation of thompson sampling”. In: *Advances in neural information processing systems* 24 (2011) (cit. on p. 40).

[15] Bangrui Chen and Peter I Frazier. “Dueling bandits with weak regret”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 731–739 (cit. on p. 28).

[16] Weiwei Cheng, Krzysztof Dembczynski, and Eyke Hüllermeier. “Label ranking methods based on the Plackett-Luce model”. In: *ICML*. 2010 (cit. on p. 26).

[17] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. “Contextual bandits with linear payoff functions”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 208–214 (cit. on p. 43).

[18] Miroslav Dudıék, Katja Hofmann, Robert E Schapire, Aleksandrs Slivkins, and Masrour Zoghi. “Contextual dueling bandits”. In: *Conference on Learning Theory*. PMLR. 2015, pp. 563–587 (cit. on p. 6).

[19] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. “PAC bounds for multi-armed bandit and Markov decision processes”. In: *International Conference on Computational Learning Theory*. Springer. 2002, pp. 255–270 (cit. on p. 7).

[20] Moein Falahatgar, Yi Hao, Alon Orlitsky, Venkatadheeraj Pichapati, and Vaishakh Ravindrakumar. “Maxing and ranking with few assumptions”. In: *Advances in Neural Information Processing Systems* 30 (2017) (cit. on p. 19).

[21] Moein Falahatgar, Ayush Jain, Alon Orlitsky, Venkatadheeraj Pichapati, and Vaishakh Ravindrakumar. “The limits of maxing, ranking, and preference learning”. In: *International conference on machine learning*. PMLR. 2018, pp. 1427–1436 (cit. on p. 19).

[22] Moein Falahatgar, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. “Maximum selection and ranking under noisy comparisons”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1088–1096 (cit. on p. 19).

[23] Sarah Filippi, Olivier Cappe, Aurélien Garivier, and Csaba Szepesvári. “Parametric bandits: The generalized linear case”. In: *Advances in Neural Information Processing Systems* 23 (2010) (cit. on pp. 37, 43).

[24] Tadhg Fitzgerald, Yuri Malitsky, and Barry O’Sullivan. “Reactr: Realtime algorithm configuration through tournament rankings”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015 (cit. on pp. 4, 5, 25).

[25] Tadhg Fitzgerald, Yuri Malitsky, Barry O’Sullivan, and Kevin Tierney. “React: Realtime algorithm configuration through tournaments”. In: *Seventh Annual Symposium on Combinatorial Search*. 2014 (cit. on pp. 4, 5, 25).

[26] I. P. Gent, H. H. Hoos, P. Prosser, and T. Walsh. “Morphing: Combining Structure and Randomness”. In: 1999, pp. 654–660 (cit. on p. 51).

[27]Shengbo Guo, Scott Sanner, Thore Graepel, and Wray Buntine. “Score-based Bayesian skill learning”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2012, pp. 106–121 (cit. on p. 25).

[28]Ralf Herbrich, Tom Minka, and Thore Graepel. “TrueSkill™: a Bayesian skill rating system”. In: *Advances in neural information processing systems* 19 (2006) (cit. on p. 5).

[29]Wassily Hoeffding. “Probability inequalities for sums of bounded random variables”. In: *The collected works of Wassily Hoeffding*. Springer, 1994, pp. 409–426 (cit. on p. 18).

[30]`http://gifgif.media.mit.edu/` (cit. on p. 16).

[31]*IBM: CIBM ILOG CPLEX Optimization Studio: User’s MANUAL FOR CPLEX*. 2016 (cit. on p. 52).

[32]Dan Klein and Pieter Abbeel. 2012 (cit. on p. 14).

[33]Pushmeet Kohli, Mahyar Salek, and Greg Stoddard. “A fast bandit algorithm for recommendation to users with heterogenous tastes”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. 1. 2013, pp. 1135–1141 (cit. on p. 15).

[34]John Langford and Tong Zhang. “The epoch-greedy algorithm for contextual multi-armed bandits”. In: *Advances in neural information processing systems* 20.1 (2007), pp. 96–1 (cit. on p. 6).

[35]Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020 (cit. on pp. 5, 12, 15, 35).

[36]Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. “Towards a universal test suite for combinatorial auction algorithms”. In: *Proceedings of the 2nd ACM conference on Electronic commerce*. 2000, pp. 66–76 (cit. on p. 53).

[37]Lihong Li, Yu Lu, and Dengyong Zhou. “Provably optimal algorithms for generalized linear contextual bandits”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2071–2080 (cit. on p. 37).

[38]Marius Lindauer and Andre Biedenkapp. *Algorithm configuration - automl*. 2020 (cit. on p. 10).

[39]Tyler Lu, Dávid Pál, and Martin Pál. “Contextual multi-armed bandits”. In: *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 485–492 (cit. on p. 6).

[40]R Duncan Luce. “Individual choice behavior, a theoretical analysis”. In: *Bull. Amer. Math. Soc* 66.1960 (1960), pp. 259–260 (cit. on p. 22).

[41]Robert D Luce. *Individual Choice Behavior: A theoretical analysis, New York, NY: John Willey and Sons*. 1959 (cit. on pp. 6, 23, 24).

[42]Oded Maron and Andrew Moore. “Hoeffding races: Accelerating model selection search for classification and function approximation”. In: *Advances in neural information processing systems* 6 (1993) (cit. on p. 50).

[43] Adil El Mesaoudi-Paul, Viktor Bengs, and Eyke Hüllermeier. "Online preselection with context information under the Plackett-Luce model". In: *arXiv preprint arXiv:2002.04275* (2020) (cit. on pp. 3, 4, 12, 24, 26, 28, 33, 34, 37, 39, 49, 52, 53, 58, 81).

[44] El Mesaoudi-Paul, Dimitri Weiß, Viktor Bengs, Eyke Hüllermeier, Kevin Tierney, et al. "Pool-based realtime algorithm configuration: A preselection bandit approach". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2020, pp. 216–232 (cit. on pp. 4, 5, 12, 24–26, 28–30, 35, 49–51, 93).

[45] Mike Phillips, Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. "Efficient search with an ensemble of heuristics". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015 (cit. on p. 6).

[46] R. L. Plackett. "The Analysis of Permutations". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 24.2 (1975), pp. 193–202 (cit. on pp. 6, 22–24).

[47] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: *SIAM journal on control and optimization* 30.4 (1992), pp. 838–855 (cit. on p. 28).

[48] SEPARATE DECISION QUEUE. "CADICAL at the SAT Race 2019". In: *SAT RACE 2019* (2019), p. 8 (cit. on pp. 49, 51).

[49] David Ruppert. *Efficient estimations from a slowly convergent Robbins-Monro process*. Tech. rep. Cornell University Operations Research and Industrial Engineering, 1988 (cit. on p. 28).

[50] Aadirupa Saha. "Optimal Algorithms for Stochastic Contextual Preference Bandits". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 30050–30062 (cit. on p. 37).

[51] Aadirupa Saha and Aditya Gopalan. "Active ranking with subset-wise preferences". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 3312–3321 (cit. on p. 7).

[52] Aadirupa Saha and Aditya Gopalan. "Battle of Bandits." In: *UAI*. 2018, pp. 805–814 (cit. on pp. 5, 7, 20, 35).

[53] Aadirupa Saha and Aditya Gopalan. "Combinatorial bandits with relative feedback". In: *Advances in Neural Information Processing Systems* 32 (2019) (cit. on p. 7).

[54] Aadirupa Saha and Aditya Gopalan. "PAC battling bandits in the plackett-luce model". In: *Algorithmic Learning Theory*. PMLR. 2019, pp. 700–737 (cit. on p. 7).

[55] Dirk Schäfer and Eyke Hüllermeier. "Dyad ranking using Plackett–Luce models based on joint feature representations". In: *Machine Learning* 107.5 (2018), pp. 903–941 (cit. on p. 26).

[56] Elias Schede, Jasmin Brandt, Alexander Tornede, et al. "A Survey of Methods for Automated Algorithm Configuration". In: *arXiv preprint arXiv:2202.01651* (2022) (cit. on pp. 1, 4, 9, 11, 12, 31).

[57]Yanan Sui, Vincent Zhuang, Joel W Burdick, and Yisong Yue. “Multi-dueling bandits with dependent arms”. In: *arXiv preprint arXiv:1705.00253* (2017) (cit. on pp. 35, 40–42, 44–46).

[58]Yanan Sui, Masrour Zoghi, Katja Hofmann, and Yisong Yue. “Advancements in Dueling Bandits.” In: *IJCAI*. 2018, pp. 5502–5510 (cit. on p. 16).

[59]William R Thompson. “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples”. In: *Biometrika* 25.3-4 (1933), pp. 285–294 (cit. on pp. 2, 35).

[60]Dave AD Tompkins and Holger H Hoos. “UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT”. In: *International conference on theory and applications of satisfiability testing*. Springer. 2004, pp. 306–320 (cit. on p. 53).

[61]Dave AD Tompkins, Frank Hutter, and Holger H Hoos. “Scaling and probabilistic smoothing (saps)”. In: *SAT* (2004) (cit. on p. 52).

[62]Sharan Vaswani, Abbas Mehrabian, Audrey Durand, and Branislav Kveton. “Old dog learns new tricks: Randomized ucb for bandit problems”. In: *arXiv preprint arXiv:1910.04928* (2019) (cit. on p. 37).

[63]Jun Wang and Carl Tropper. “Optimizing time warp simulation with reinforcement learning techniques”. In: *2007 Winter Simulation Conference*. IEEE. 2007, pp. 577–584 (cit. on p. 6).

[64]Huasen Wu and Xin Liu. “Double thompson sampling for dueling bandits”. In: *Advances in neural information processing systems* 29 (2016) (cit. on p. 40).

[65]Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. “The k-armed dueling bandits problem”. In: *Journal of Computer and System Sciences* 78.5 (2012), pp. 1538–1556 (cit. on pp. 5, 16).

[66]Yisong Yue and Thorsten Joachims. “Interactively optimizing information retrieval systems as a dueling bandits problem”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 1201–1208 (cit. on pp. 5, 16).

List of Figures

List of Algorithms

Glossary

AC Algorithm Configuration. v, 1, 2, 3, 4, 8, 9, 10, 11, 23, 25, 91, 94

CA Combinatorial Auction. 53, 94

CoLST *Contextualized Linear Stochastic Transitivity*. 36, 37, 38, 94

CoLSTIM *Contextualized Linear Stochastic Transitivity Imitator*. 36, 37, 38, 39, 52, 55, 57, 58, 59, 60, 63, 65, 67, 69, 71, 73, 75, 77, 93, 94

CPPL *Contextual Preselection under Plackett-Luce assumption*. v, 3, 4, 5, 6, 7, 8, 12, 24, 25, 28, 29, 33, 34, 35, 36, 37, 38, 41, 42, 45, 46, 47, 49, 50, 51, 55, 77, 79, 93, 94

DTS Double-Thompson-Sampling. 40, 94

i.i.d. independent and identically distributed. 18, 28, 34, 94

ISS *Independent Self Sparring*. 41, 44, 59, 65, 67, 94

ISSC *Independent Self Sparring* for Contextual Bandits. 59, 65, 67, 94

LST linear stochastic transitivity. 36, 94

MAB Multi-armed Bandit. v, 2, 3, 5, 6, 7, 8, 12, 13, 15, 16, 19, 25, 40, 42, 43, 44, 79, 94

MIP Mixed Integer Programming. 53, 54, 60, 62, 63, 64, 65, 66, 68, 69, 70, 72, 73, 74, 76, 79, 82, 83, 84, 91, 92, 94

MLE maximum likelihood estimate. 37, 94

MNL Multinomial Logit. 7, 94

PAC *probably approximately correct*. 7, 94

PB-MAB *Preference-Based Multi-Armed Bandits*. 5, 16, 17, 19, 94

PCA Principal Component Analysis. 51, 94

PL Plackett-Luce. 6, 7, 22, 23, 24, 26, 27, 28, 29, 36, 94

Pre-Bandit Preselection Bandit. 3, 4, 5, 7, 8, 20, 22, 23, 25, 26, 30, 33, 34, 43, 79, 94

RAC Realtime Algorithm Configuration. 4, 5, 11, 12, 22, 25, 29, 49, 79, 91, 94

ReACT *Realtime Algorithm Configuration through Tournaments*. 4, 5, 6, 25, 79, 94

ReACTR *Realtime Algorithm Configuration through Tournament Rankings*. 4, 5, 6, 25, 79, 94

SAT boolean satisfiability. 49, 51, 63, 65, 79, 94

SGD stochastic gradient descent. 28, 30, 37, 41, 94

TS Thompson Sampling. 2, 6, 35, 40, 41, 42, 43, 44, 45, 46, 59, 63, 65, 67, 69, 77, 79, 93, 94

TSC Thompson Sampling for Contextual Bandits. 59, 65, 67, 94

UCB Upper Confidence Bound. 2, 6, 7, 33, 34, 35, 36, 37, 38, 52, 54, 55, 56, 59, 63, 65, 69, 71, 73, 75, 77, 79, 93, 94

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties. I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

All statements that have been adopted literally or analogously are marked as such.

Paderborn, January 9, 2023

Shivam Sharma