Parametric and Nonparametric Learners for Adaptive Experiments Under Interference

Shaikh Tanvir Hossain

A Dissertation Submitted in
the Partial Fulfillment of the Requirements for the Doctoral Degree (Dr. rer. pol.)
from the Faculty of Business and Economics at the
Technical University of Dortmund.

Supervisor, 1st Reviewer and Examiner	**2nd Reviewer and Examiner**	**3rd Reviewer and Examiner**
Professor Dr. Carsten Jentsch	Professor Dr. Philip Jung	Professor Dr. Alexander Marx
Chair of Business and Social Statistics Department of Statistics	Professorship Macroeconomics Department of Economics	Chair of Causality Department of Statistics

fakultät statistik

wiwi Department of Business and Economics

technische universität dortmund

DORTMUND, GERMANY
2026

.... to,
my teachers *for their guidance.*

Date of Defense

18th May, 2026

Preface & Acknowledgements

First and foremost, I would like to express my deepest thanks and gratitude to Professor Carsten Jentsch, my supervisor, mentor, teacher, and a guardian-like figure for me during my time in Germany. He took this immature student from Bangladesh and helped turn him into a researcher. Whatever correct knowledge and passion I have when it comes to statistics, econometrics, and research, I owe to him, and all mistakes are mine. Many times, unknowingly, I lost track, but he gave me time to recover and always had hope. He always encouraged me to go deep into any topic. If I am brutally honest, I think I do not deserve such a great supervisor. Really, if he had not been there, possibly I would not have been able to start, let alone finish, any research in the first place.

I would also like to express my sincere gratitude to my teachers from my undergraduate years, especially Professor Dr. Wasiqur Rahman Khan, Professor and Chair, Department of Economics and Social Sciences, BRAC University, and the late Professor Dr. M. Ataharul Islam, who taught me statistics and gave me the passion to study Statistics shortly after my undergraduate studies. Their teaching, encouragement, and intellectual influence played an important role in shaping my early interest in statistics and research.

Three people, particularly my mother, father, and my beloved wife, stood by me at all times, and truly all the time. Their sacrifices are so great that I do not think I can ever repay them with anything. My mother constantly prayed for me and supported me with everything that she could. My father was by my side financially and gave me mental strength, and with all his resources he helped me. And then my wife, who possibly gave all her good times for me. She was always there, stood firmly by my side, saw me working days, nights, weekends, and holidays, but never complained. Being also a great student herself, and better than me in mathematics, coding, and anything I can think of, she always encouraged me and helped me understand different things deeply. Many times I had some primitive ideas, but after discussing them with her, I eventually found what I was looking for. She always believed in me, even when I lost hope in myself. My entire student life owes a lot to her. During the final stage of this dissertation, she also critically read almost my full writing and found many typos, inconsistencies, and other mistakes, which I corrected.

I am also deeply grateful to my extended family, especially my in-laws, Professor Dr. Faridul Alam and Professor Dr. Selina Parvin, Sagar bhaiya, Jitu bhabi, and my two wonderful nieces Safwana and Safeeya (the new comer!). Their kindness, affection, encouragement, and support have meant a great deal to me throughout this journey.

I would also like to thank my colleagues, friends, and students both at the University of Mannheim and TU Dortmund, particularly Lena, Hanno, Julia, Tilemann, Robin, Jonas, and Simon, who were my close colleagues during my time there. They always helped me in different ways. Thanks to Chris, Dr. Christopher Walsh, now at the University of Konstanz, from whom I learned a lot. Then Ms. Bettina Hillsmann; I do not know how many times I bothered her, but she was always very helpful and kind. Thanks to Dr. Thorsten Ziebach for his positive attitude. I am also thankful to my students from the Statistical Theory, Asymptotics Part, Causality, and Julia courses, particularly Fabian Blunk, Constantine, and Lars, who always asked interesting questions and taught me different things. Then the hiwis, Anviet, Ina-Marie, and others, and finally everyone from the Chair of Business and Social Statistics at TU Dortmund.

I thank all the brothers and friends who helped me in different ways during this time in Germany and Bangladesh. My younger brother supported me whenever I needed help with fixing laptops or any other technical matters. Brothers and friends from Mannheim

and Dortmund, in particular, Ashiq bhai, Paul Schaefer, Salih Abi, both of them, Shahenshah, Enes, Onur, Erdem, Ilias, Mustafa, Ismail, both of them, brother Aatef, Mamum bhai, Taher bhai, and Talha, always stayed as brothers and helped me during my stay in Germany. Life in a foreign land is not easy, but they made it easier for me. Two of my close friends, Ahmed Ridoy and Ashfaq, came forward in difficult times during the final stage of this dissertation. I am really thankful to them.

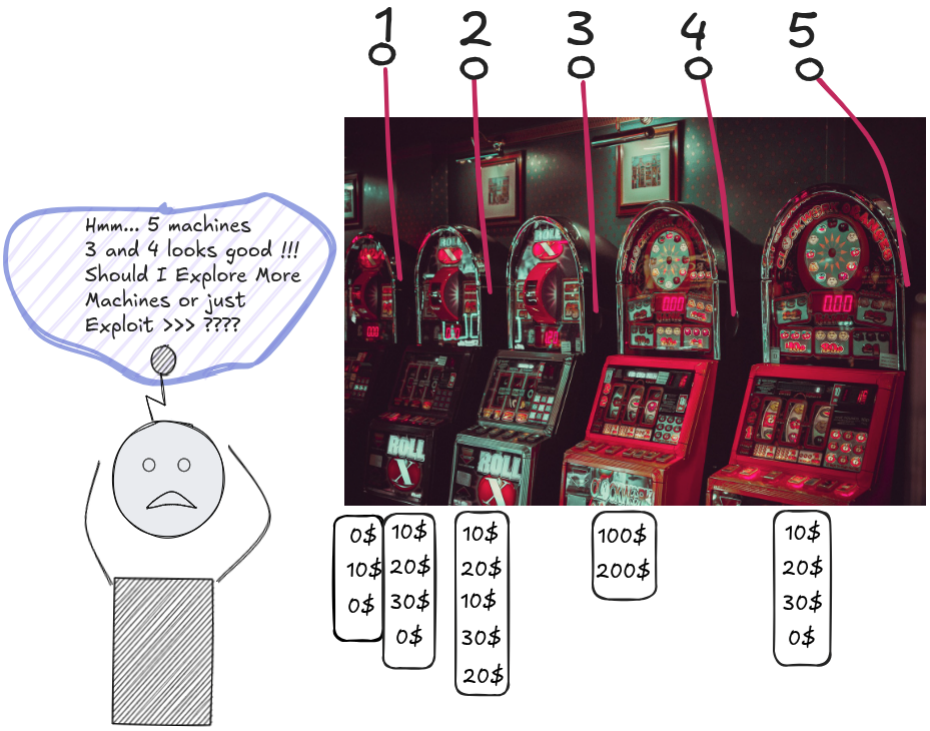
During my PhD phase, at some point, I did not have much progress and did not know how to proceed. For a change, I started teaching at East West University, Bangladesh. Honestly, this changed many things and gave me new hope. I understood the value of research and also found some new perspectives for my work. I would like to thank the senior faculty members, colleagues, and students at EWU. In particular, Professor Syed Abul Basher, now an independent researcher, who really cared and constantly pushed me to finish; he was a true mentor and a very inspiring person. My teacher, Dr. Tanvir Hussain, now at ICIMOD, who told me, The best PhD is a done PhD, so finish it, Tanvir! My colleagues, Muhammad Nazmul Khan, now a PhD candidate at UCR, Md. Abdus Sattar Moin, Senior Lecturer, and Dr. Md. Abdullah Al Mashrur, Assistant Professor, also deserve my thanks. I often discussed different ideas with them, which was always very helpful for understanding different concepts. Thanks also to the three chairpersons during my time at EWU, Dr. Biva Arani Mallik, Dr. Muntasir Chaudhury, and currently Dr. Iftekharul Huq. They always gave me a place to teach, even though I think I am a suboptimal candidate. I thank them for inviting me to present my ideas at their brown-bag seminars, where many interesting questions came up and further helped me refine my work. Finally, I met some wonderful students at EWU, who always inspired me. I learned a lot from them, particularly all the TAs and students from the ECO 104 and ECO 204 courses. Undoubtedly, they are some of the brightest students in Bangladesh, and I think they are far better than I was at their age.

Just a few words at the end. In the vast ocean of knowledge, I know this contribution is little, but I end with a prayer and hope that this dissertation is just the beginning of amazing things to come. This journey became very long. I made many mistakes, at some point was completely shattered and lost, had to learn things the hard way, but then regained the energy that I had once as a student. Now, when I look back, I would not trade this time for anything. This taught me how to be persistent, how to navigate, how to do research, and more importantly, how to get up and walk when you fall with absolutely no hope. There are many unsolved questions I would like to answer now, many ideas I would like to explore, and definitely many things to learn. Who knows, maybe someday I will come up with some lasting contributions. I pray that this work gets accepted, that this passion stays alive, and that I keep learning and contributing to the fields of statistics, econometrics, economics, and science in general as much as I can. Finally, I am truly thankful to God for giving me strength. All praise belongs to Him, and peace be upon His prophets. All mistakes are mine.

February, 2026
Shaikh Tanvir Hossain

TABLE OF CONTENTS

Appendices



Multi Armed Bandit Problem: Five slot machines with five arms, and a confused learner! The learner is unaware which machine or arm gives the highest rewards. She tries and tries, but needs to decide whether she stops playing now and exploits the machine which gave the highest reward or she explores more and more to find out better machines .... She will have to carefully *balance exploration and exploitation*, which is difficult!

INTRODUCTION

The multi armed bandit (MAB) problem has a rich literature starting from the two pioneering works: Thompson (1933) and Lai and Robbins (1985). The problem is simple, an experimenter or a learner has a set of possible treatments or actions (also called arms in the MAB literature) and faces the units sequentially, i.e., arriving one at a time. The goal is to assign optimal treatments as many times as possible over the horizon T. While the multi-armed bandit problem has been widely studied in the statistics, machine learning and economics literature, most of the works assume the reward of one unit is independent of treatments assigned to others. This assumption is violated when we have *interference* between units. Interference means that the treatment assigned to unit i may affect the outcome of unit j and vice versa. In this dissertation we address this gap by, proposing a detailed framework for the stochastic bandit under interference, extending parametric and nonparametric algorithms that learn different optimal vectors under interference, and finally evaluating the performance through numerical experiments, motivated by some real world cases. Here is a short summary of the next four chapters:

Chapter 1 gives a short but somewhat rigorous overview of the stochastic bandit problem and Thompson sampling algorithm. The technical details are borrowed from two excellent manuscripts, 1) Lattimore and Szepesvári (2020) and 2) Slivkins (2022). However, the presentation is our own, and we build the setup almost from scratch using the potential outcome notation, which is generally not common in the MAB literature. We define the classic stochastic bandit model, policy, regret, and then, go to an extended discussion of the Thompson sampling algorithm (Thompson, 1933). We give two concrete examples under Gaussian and Bernoulli rewards, and shows how the algorithm can be applied. Finally, we also provide a detailed Bayesian regret analysis under Gaussian rewards. At the end of this chapter, we also give a short discussion of *Explore Then Commit* (ETC), *Upper Confidence Bound* (UCB) algorithms, which are two other popular algorithms in the stochastic bandit literature. This chapter sets the stage for the next three chapters in the dissertation, where many notations and definitions are extended versions of the ones defined here.

Chapter 2, titled as *"Thompson Sampling in the Presence of Interference"* is the first contribution of this dissertation. The interference literature widely uses potential outcome framework, so thanks to the details regarding potential rewards from Chapter 1, which we use as building blocks for this chapter. So far, very few works have considered the multi-armed bandit problem under interference, notable are, Jia, Frazier, and Kallus (2025), Agarwal, Agarwal, Masoero, and Whitehouse (2024) and Gleich, Laber, and Volfovsky (2025). However, none of these works clearly defined the setup with all its components. In this chapter, we first carefully define the stochastic bandit problem under batched interference, where a batch of units arrives together, and there is a possible interference among them. We define the stochastic potential rewards, stochastic bandit model under interference, the notion of clustered interference, and different notions of optimal arm vectors and regrets under this

setting. We use a motivating example of policing on crime reduction to illustrate all the definitions and the proposed setup. We then we propose four parametric extensions of the Thompson sampling, which we call *"learners"*. The four algorithms vary based on the targets and assumptions they make, they are: i.i.d. learner, global one arm learner, clustered one arm learner and clustered multi learner. We clearly state the assumptions, complete algorithms and give concrete examples for each of them under Gaussian and Bernoulli rewards. We then provide Bayesian regret analysis for the global one arm learner under Gaussian rewards. Finally, we provide a detailed numerical experiment section, where we evaluate the performance of all four algorithms under different interference structures and parameter setups.

Chapter 3, titled as *"Nonparametric Learners for Stochastic Bandit Under Interference"* is the second contribution of this dissertation. Although the algorithms proposed in Chapter 2 can be applied quite flexibly in different interference settings, but one key limitations is, they are under the parametric assumptions, i.e., the reward distributions are assumed to follow some known class of parametric families and the priors are correctly applied. In practice, this brings two issues, first, we many not know the reward distributions, and second, even if we know the class of the distributions, the prior - posterior relationships are not always as straightforward as Gaussian and Bernoulli cases, where the posteriors come as neat conjugate forms. Motivated by these limitations, we develop nonparametric alternatives based on resampling ideas. In particular, we build on nonparametric Thompson sampling, proposed in (Riou & Honda, 2020) and subsampling duelling algorithm, proposed in (Baudry, Kaufmann, & Maillard, 2020). Both of these algorithms are designed for the classical stochastic bandit setup without interference and have strong theoretical guarantees. We extend these two algorithms to the interference setup proposed in Chapter 2. Although NPTS and SDA have fundamentally different designs, and there are no parameter updates like Thompson sampling, we adapt their designs under the same type of learner framework that we developed in Chapter 2. For each algorithm, we propose four interference variants that can be applied under different interference scenarios, these are nonparametric i.i.d. learners, global one arm learners, clustered one arm learners, and clustered multi arm learners. This yields eight nonparametric learners in total. We present these methods using the same potential reward formulation that we developed in Chapter 2, clarify their connections and differences, and finally evaluate their performance through some numerical experiments under different interference settings.

Chapter 4, titled *"Learning Optimal Price Under Interference,"* is the third contribution of this dissertation. This is a more application oriented chapter, and primarily motivated from the experience of street vendors or hawkers in Bangladesh. Learning optimal prices using bandit algorithms is not a novel idea per-se, for example, there is an early work of Kleinberg and Leighton (2003). However, we consider here, the problem of learning optimal price under interference where the purchase decision of one buyer is influenced by the

prices offered to other buyers she is connected to. The seller's task is to maximize her total revenue over a horizon T by learning the optimal price through sequential interactions with buyers. In this chapter, we first describe the seller's problem in this setting and then explain how interference arises. Finally, we apply both the parametric and nonparametric learners that we developed in Chapter 2 and Chapter 3 to learn the optimal price under interference through some numerical experiments. The experiments are carefully designed to mimic the hawker-buyer interactions in a city where buyers are connected through a social network. We believe this chapter opens some new doors to connect bandit algorithms, pricing problems and interference, which can be helpful in practical applications.

The dissertation use some pictures from Unsplash.com, a popular free image repository and all pictures are used under the Unsplash license. The slot machine picture at the beginning is by Carl Raw (hv4jH7B_U8Y), the hawker picture in Figure 1.1 is by Alexis Rodriguez (TAMc4ILMMkM), and the farmer receiving the call picture in Figure 1.2 is by Tarun Anand Giri (4JyhqhWyKb8). Thanks to all of them, for making these images freely available.

The rest of the pictures or images are generated either by Python, LaTeX TikZ package, and Obsidian Excalidraw plugin. The numerical experiments are done using Python 3. The code for the numerical experiments are saved in a private repository, which can be made available upon request.

Chapter 1.

STOCHASTIC BANDITS AND THOMPSON SAMPLING

(Unpublished)

In this chapter we present a short overview of the *stochastic multi-armed bandit* problem with some algorithms. We will define the stochastic bandit model using the potential outcome framework often known as Neyman-Rubin potential outcome (PO) framework (Rubin, 1974, 1976, 1980). Then we present a classic algorithm called *Thompson sampling (TS)* (Thompson, 1933) along with the Bayesian regret analysis. The technical details are borrowed mostly from the two excellent manuscripts, Lattimore and Szepesvári (2020) and Slivkins (2022), however the presentation is our own, and we slightly modified to fit the style of this thesis.

1.1 Introduction

Our setup is a sequential treatment assignment or adaptive experimentation setup. Here we have an experimenter or learner who would like to learn the best treatment from a finite set of treatments through sequential experimentation. We start with two concrete examples to motivate the sequential treatment assignment problem.

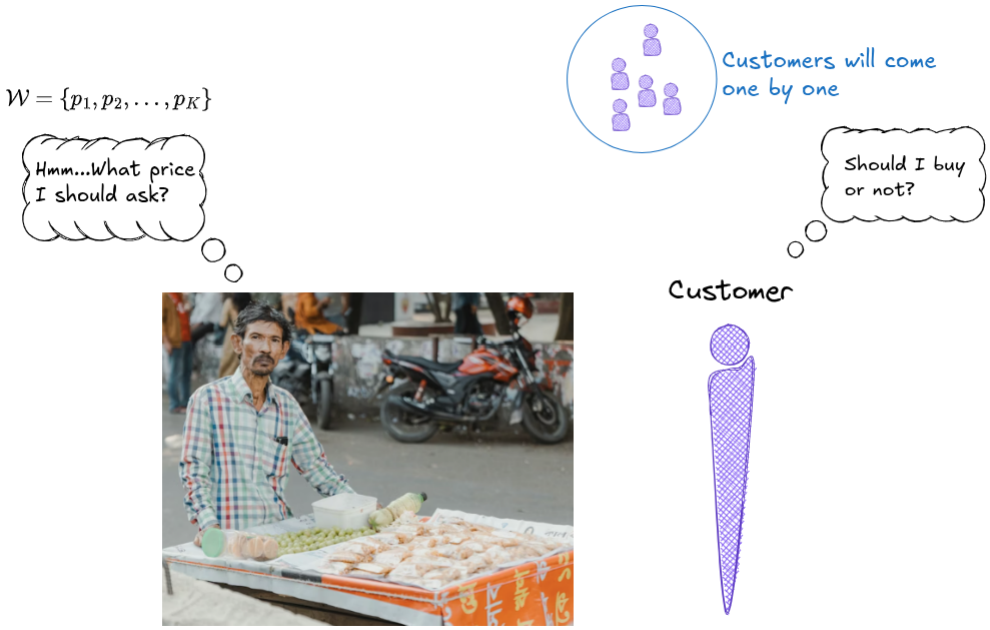


Figure 1.1: Dynamic Pricing: A hawker trying to learn the optimal price through sequential experimentation.

Figure 1.2: Agricultural Advisory Service Example: A public agricultural service company trying to learn optimal enrollment strategy through sequential experimentation through telephone calls.

Example 1.1 (Dynamic Pricing Problem)**.** Suppose a hawker (or mobile street vendor) in Bangladesh wakes up every morning and goes to different locations across in a city to sell his products. He has a new product, although he collected some data about the potential price but is unaware about the perfect price that maximizes profit. He faces a fundamental trade-off: if the price is too low, he loses potential revenue from customers who are willing to pay more, and if the price is too high, customers may lose interest, resulting in lost sales. One solution is to learn the optimal price adaptively. The hawker starts with a set of K candidate prices $\mathcal{W} = \{p_1, p_2, \ldots, p_K\}$. At each time period $t = 1, 2, \ldots, T$, a new customer arrives, and the hawker offers a price $W_t \in \mathcal{W}$. The customer then decides whether to purchase the product or not. If the customer makes a purchase, the hawker earns revenue $Y_t = W_t \times X_t$, where $X_t = 1$ indicates a successful purchase and $X_t = 0$ indicates a refusal. The hawker's objective is to maximize the cumulative revenue $\sum_{t=1}^{T} Y_t$. Essentially, this means selecting a sequence of prices $W_1, W_2, \ldots, W_T$ such that the total revenue $\sum_{t=1}^{T} Y_t$ is maximized. To succeed, the hawker must strike a balance between exploring different prices to learn customer demand and exploiting the best price found so far to maximize his earnings. This example is motivated by the dynamic pricing literature; see Kleinberg and Leighton (2003), Besbes and Zeevi (2009), and Den Boer (2015) for more details.

///

Example 1.2 (Agricultural Advisory Services)**.** A public agricultural service company provides free cell-phone based advisory services to small landowner farmers. It enrolls different farmers in their service program, then calls them or sends them text messages about agricultural advice, such as bad weather forecasts, optimal planting times, pest control techniques, or even market prices for crops. The first task is to enroll as many farmers as possible into their service program so that the farmers receive this service in the future. The company faces a challenge: not all farmers are eager to enroll when contacted, and it is unclear in which method of contacting is most effective to get them enrolled. To address this challenge, the company decides to learn optimal enrollment strategy through sequential experimentation. It starts with a candidate set of K enrollment strategies, $\mathcal{W} = \{1, 2, \ldots, K\}$, e.g., calling at 10:00 a.m. and sending a reminder SMS one hour before, or calling at 6:30 p.m. with no reminder SMS, etc. At each time period $t = 1, 2, \ldots, T$, a new farmer is selected from a list of phone numbers, an enrollment strategy $W_t \in \mathcal{W}$ is being tested. If the farmer in the strategy (or call) patiently stays during the complete call, it is considered a successful enrollment. Let $Y_t = 1$ means the farmer at time t stays or gets enrolled and $Y_t = 0$ otherwise. In the enrollment call, the company asks / discusses different questions or issues e.g., farmer's identity, benefits of future messages, how to do registration, etc. Although the enrollment does not guarantee that farmers will act on future advice, but definitely it is a necessary first step for the service to have any impact. The organization's objective is to maximize cumulative enrollment $\sum_{t=1}^{T} Y_t$. Essentially this means picking a sequence of strategies $W_1, W_2, \ldots, W_T$ such that $\sum_{t=1}^{T} Y_t$ is maximized. This example is motivated from Kasy and Sautmann (2021) where the authors apply their adaptive experimentation technique for agricultural advisory services in India.

///

Both examples illustrate a common structure. We have a finite set of actions $\mathcal{W}$. There is a learner or experimenter who at time t picks a single action $W_t \in \mathcal{W}$, then the learner observes an outcome or reward Y_t associated with the chosen action W_t. For example, the learner chooses a price and observes revenue, or the learner picks a strategy and observes the enrollment. What's important is that, the learner only observes the reward Y_t associated with the chosen action W_t, and is unaware of what would have happened if she chose some other action $k \neq W_t$. We can formalize this idea using the potential outcome (PO) framework (Rubin, 1974, 1976, 1980). In particular, according to PO framework, for each time t, there is a set of potential outcomes or rewards $\{Y_t(k) : k \in \mathcal{W}\}$, where $Y_t(k)$ is the potential reward if action k is chosen at time t. Then the learner can only observe $Y_t = Y_t(W_t)$ and is unaware what would happen if she chose some other action $k \neq W_t$. Now after taking the action, the learner observes the reward, updates her knowledge re-

garding the rewards, prepares for the next unit and picks another action at $t+1$. This loop continues until time T. If the learner picks T actions for a horizon T, then the learner's goal is to maximize cumulative rewards $\sum_{t=1}^{T} Y_t$.

This problem has many names, adaptive experimentation, sequential policy learning and perhaps the most common is multi-armed bandit (MAB). MAB problem is quite common for the recommender systems in the computer science and machine learning literature, see for example Chapelle and Li (2011) and Vermorel and Mohri (2005) for some empirical applications. It has been extensively studied starting from two pioneering works - Thompson (1933) and Lai and Robbins (1985). The name *bandit* came from the slot machines in the rigged casinos. The idea is, in a casino when a player gives a coin and pulls an arm, then either gets some money (which is reward) or lose the coin, so it is perceived that the machine acts like a bandit, and probably will take only coins without giving much in return. If there are more than one slot machines, then the player's task to decide which slot machines to play so that she can win, then we have multiple bandits and hence the name multi-armed bandit (MAB).

Note that in the examples that we gave, only one unit arrives at each time t, and then the learner assigns a treatment to that single unit, observes reward and then next unit arrives. This is known as *online learning* as opposed to offline learning (or supervised learning), where in the latter the learner typically observes a batch or a fixed dataset $\{X_i, Y_i\}_{i=1}^{N}$ all at once and learns. By contrast, in online learning the data arrive sequentially and learner also learns sequentially.

Now that we have discussed the basic multi-armed bandit problem, the next task is to think about what kind of structure or modeling assumption we would like to have on the space of actions and rewards. Broadly speaking we can think of two classes of bandit problems, the *stochastic bandits* and the *adversarial bandits*. In the stochastic bandit case, we assume that for each action $k \in \mathcal{W}$, there is a fixed but unknown distribution of rewards from which the rewards are generated. So if we denote the reward distribution for action k as F_k, then at each time t, when the learner picks action $W_t = k$, the reward is generated following $Y_t \sim \mathrm{F}_k$. The distribution F_k, for any $k \in \mathcal{W}$ are unknown to the learner, however they are fixed over time. On the other hand, in the adversarial bandit case, there is no assumption of fixed distributions, instead the rewards can be generated from any arbitrary mechanism, even sometimes it can be adversarial. In this chapter and also in this thesis, we will focus only on the stochastic bandit problems and algorithms. In the next section we will formally define the stochastic bandit model, the task of the learner, and some important concepts like policy, regret and exploration-exploitation trade-off, which are crucial to understand the stochastic bandit algorithms.

1.2 Definitions and Protocols

We already gave practical examples and the basic multi-armed bandit problem in the last section. In this section we formalize the stochastic bandit model using the potential outcome framework. Then we define some important concepts like policy, goal and regret of the learner.

1.2.1 Stochastic Bandit Model

We start with some notations. Let $\mathcal{W}$ be a finite set of actions*, and $\mathscr{F}_\mu = \left\{ \mathrm{F}_{\mu(k)} : k \in \mathcal{W} \right\}$ be a set of probability distributions of potential rewards $Y_t(k)$ indexed by the parameter set $\mu := \{\mu(k) : k \in \mathcal{W}\}$. We assume the potential rewards are independent across both actions and time. This means, at any time t, $Y_t(k) \perp Y_t(l)$ for all $l \neq k$. Also the distributions $\mathrm{F}_{\mu(k)}$, for any $k \in \mathcal{W}$, are fixed over time and rewards are independently drawn from these distributions across time, i.e., $\{Y_t(k)\}_{t \geq 1}$ will form an identically and independently distributed (i.i.d.) sequence from $\mathrm{F}_{\mu(k)}$ for any $k \in \mathcal{W}$. For simplicity, we consider a parametric family indexed only by the mean, and this is why we write, $Y_t(k) \sim \mathrm{F}_{\mu(k)}$ where $\mu(k) = \mathbb{E}[Y_t(k)]$. Below we define the stochastic bandit model with the interaction protocol between a learner and an environment.

Definition 1.3 (Stochastic Bandit Model). *A stochastic bandit model is a repeated interaction between a learner and an environment where*

for $t = 1, 2, \ldots, T$ **do**

> *the learner selects an action* $W_t \in \mathcal{W}$
>
> *the environment generates potential outcomes* $Y_t(k) \sim \mathrm{F}_{\mu(k)}$ *for all* $k \in \mathcal{W}$
>
> *the learner observes only the realized outcome* $Y_t = Y_t(W_t)$*, not the counterfactual outcomes* $Y_t(k)$ *for* $k \neq W_t$

Interaction Protocol 1: *Stochastic Bandit Interaction*

We have the following information structure,

- ***Known or Observable to the Learner:*** *The learner knows the set of arms* $\mathcal{W}$*, time horizon* T *and observes all actions and outcomes till time* T*, i.e.,* $\{(W_t, Y_t)\}_{t=1}^T$
- ***Known to the Environment:*** *The environment knows the set of counterfactual distributions* $\mathscr{F}_\mu = \left\{ \mathrm{F}_{\mu(k)} : k \in \mathcal{W} \right\}$*, and observes what the learner observes.*

In the bandit literature, the outcome or reward generating process is explained with an entity or idea called an *environment*. In simple words, an environment knows the underlying distribution of the potential rewards and is able to generate rewards. So in our notation, the environment knows $\mathscr{F}_\mu$. Sometimes with some abuse of terminologies, $\mathscr{F}_\mu$ is itself

* In the machine learning literature, actions are often called *arms* and *rewards*, this is from the slot machine analogy, while in econometrics often we use *treatments* and *outcomes*.

called an environment. The learner on the other hand is completely unaware of reward mechanism and learns adaptively which action is best. We give some remarks regarding the definition below.

Remarks 1.4 (Remarks On Definition)**.**

- In our case we defined a stochastic bandit model with all the elements above in the definition, but in the literature sometimes the environment or F is also called a *stochastic bandit*. Or a for a fixed set of parameters, called an *instance* of a stochastic bandit.
- We are indexing the set of potential reward distributions only with mean $\mu(\cdot)$. For a parametric class, in principle, we should index with a general parameter vector, but for algorithms we explain this chapter indexing distributions only with mean is enough.

///

We give some parametric examples of stochastic bandits below.

Example 1.5 (Stochastic Bandit with Bernoulli or Gaussian Rewards)**.** Suppose we have two actions or arms, so action set $\mathcal{W} = \{1, 2\}$. Assume for each arm $k \in \mathcal{W}$ we have Bernoulli potential reward or outcome variable, i.e., $Y_t(k) \sim \text{Bernoulli}(p_k)$. So the set of potential rewards follow Bernoulli distributions with parameters p_1 and p_2. The environment is $\mathscr{F}_\mu = \{\text{Bernoulli}(p_1), \text{Bernoulli}(p_2)\}$, the mean rewards are $\mu(1) = p_1$ and $\mu(2) = p_2$. The environment knows both p_1 and p_2, but the learner doesn't know either of them. At time t, if the learner picks arm 1, she only observes a random outcome (1 or 0) from $\text{Bernoulli}(p_1)$, and is unaware of what would have happened if she had picked arm 2.

///

Example 1.6 (Stochastic Bandit with Gaussian Rewards)**.** If we assume the rewards are continuous rather than binary, then maybe the rewards are Gaussian distributions, i.e., $Y_t(1) \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $Y_t(2) \sim \mathcal{N}(\mu_2, \sigma_2^2)$, where μ_1 and μ_2 are the means and σ_1^2 and σ_2^2 are the variances of the two Gaussian distributions, respectively. Then $\mathscr{F}_\mu = \{\mathcal{N}(\mu_1, \sigma_1^2),\ \mathcal{N}(\mu_2, \sigma_2^2)\}$, the mean rewards are $\mu(1) = \mu_1$ and $\mu(2) = \mu_2$. If the learner picks arm 1 at time t, then she will observe a random outcome from $\mathcal{N}(\mu_1, \sigma_1^2)$, and is unaware of what would have happened if she picked arm 2.

///

Next, we write some details regarding three important concepts: policy function, goal and regret of the learner.

1.2.2 Policy, Goal and Regret of the Learner

At period t, before picking the arm, the learner first looks at the history of all past arm selections and associated rewards until time $t-1$, and then she chooses arm at t. The policy function formalizes this idea. A *policy function* at time t denoted with π_t is a mapping from the set of all possible histories up to time $t-1$ to the set of arms, i.e.,

$$\pi_t : (\mathcal{W} \times \mathcal{Y})^{t-1} \to \mathcal{W}, \tag{1.1}$$

and we call the sequence of policy functions $\pi = \{\pi_t\}_{t=1}^{T}$ *a policy for the learner* or simply *policy*. This means for a random sequence $(W_1, Y_1),\ (W_2, Y_2),\ \ldots, (W_{t-1}, Y_{t-1})$, we will have $\pi_t(W_1, Y_1,\ W_2, Y_2,\ \ldots, W_{t-1}, Y_{t-1}) \in \mathcal{W}$. Sometimes the policy is a mapping from history to *a distribution of arms* rather than to the set of arm $\mathcal{W}$ i.e., $\pi_t : (\mathcal{W} \times \mathcal{Y})^{t-1} \to \mathcal{P}(\mathcal{W})$, where $\mathcal{P}(\mathcal{W})$ is the set of all probability distributions over the arm set $\mathcal{W}$. In this case they are often called *randomized policies*, whereas the former is known as *deterministic policies*. In Example 1.5, a randomized policy at time t will give a probability distribution for two arms $\mathcal{W} = \{1, 2\}$ based on the history until time $t-1$, and then the learner will sample an arm from this distribution, however in the deterministic policy case, the policy will give a single arm from $\mathcal{W}$ based on the history until time $t-1$. So a policy function simply specifies how the learner will play. In the bandit literature these policy functions are actually *learning algorithms*. Although when it comes to stochastic bandit problems there are many learning algorithms, we will discuss a very well known algorithm called Thompson sampling (TS) (Thompson, 1933) in the next section and give outlines of some other algorithms at the end of this chapter, in Section 1.5.

In theory, a learner can be interested in different goals or objectives, but the most common one in the bandit setting is maximizing cumulative rewards in expectation. Writing the problem in our notation, the goal of the learner can be written as follows,

$$\text{Learner's Problem: } \underset{\pi}{\text{maximize}}\ \mathbb{E}\left[\sum_{t=1}^{T} \mu(W_t)\right] \tag{1.2}$$

So for any algorithm, we need to devise a policy π that selects a sequence of arms $W_1, W_2, \ldots, W_T$ such that it maximizes the total expected reward across all nodes in the batch over the horizon T.

Now in the bandit setting we have an equivalent way to write this maximization problem, which is known as minimizing *regret*. Essentially this is a performance metric to evaluate how the learner's algorithm is performing. Put it simply, the regret essentially measures how much the learner is losing by not knowing the best action in advance. Here is the formal definition.

Definition 1.7 (PSEUDO OR RANDOM REGRET AND EXPECTED REGRET)**.** *For a policy* $\pi = \{\pi_1, \ldots, \pi_T\}$ *and for an environment* $\mathscr{F}_\mu$*, the* random regret *also known as* pseudo regret *is defined as,*

$$\mathcal{R}_T(\mathscr{F}_\mu, \pi) := \sum_{t=1}^{T}(\mu^* - \mu(W_t)) = T\mu^* - \sum_{t=1}^{T}\mu(W_t) \tag{1.3}$$

and the expected regret *or simply* regret *is defined as* $\mathbb{E}(\mathcal{R}_T)$*, where expectation is taken over the randomness of the policy or arm selection.*

The idea of regret can be better understood if we relate it to the goal of the learner. Recall the learner's goal is to pick a sequence of arms $W_1, W_2, \ldots, W_T \in \mathcal{W}$, such that the cumulative reward $\sum_{t=1}^{T} Y_t$ is maximized. Since the rewards are random quantities, if we take expectation, then the learner's objective becomes maximizing expected cumulative reward $\sum_{t=1}^{T}\mathbb{E}[Y_t]$. Using the relation $Y_t = Y_t(W_t)$ we get $\sum_{t=1}^{T}\mathbb{E}[Y_t(W_t)] = \sum_{t=1}^{T}\mu(W_t)$, where $\mu(k) = \mathbb{E}[Y_t(k)]$ for any $k \in \mathcal{W}$. So in expectation, the learner's goal is to maximize $\sum_{t=1}^{T}\mu(W_t)$. Now since, $k^* = \arg\max_{k\in\mathcal{W}}\mu(k)$ is the best arm with mean reward $\mu^* = \max_{k\in\mathcal{W}}\mu(k)$, so the maximum possible reward for horizon T is $\sum_{t=1}^{T}\mu(k^*) = \sum_{t=1}^{T}\mu* = T\mu*$. Now in Regret minimization, rather than maximizing $\sum_{t=1}^{T}\mu(W_t)$ we minimize $\sum_{t=1}^{T}(\mu^* - \mu(W_t))$, i.e., we just flip the maximization problem into a minimization problem where loss is measured against the best possible expected cumulative reward μ^*. We can also let $r_t = \mu^* - \mu(W_t)$ be the instantaneous regret at any particular time t, then cumulative regret up to time T can also be written as, $\mathcal{R}_T = \sum_{t=1}^{T} r_t$.

In the definition of pseudo regret $\mathcal{R}_T$, the sequence of arms $W_1, W_2, \ldots, W_T$ are random quantities since they are generated from the policy π, however there are no randomness in the observed rewards Y_t since we already took expectation in the definition. The expected regret $\mathbb{E}(\mathcal{R}_T)$ takes expectation over the randomness of the arm selections from the policy π. Both the pseudo regret and the expected regret depends on the three things, the environment $\mathscr{F}_\mu$, the policy π and the horizon T. So changing any of the three will change the regret.

In the bandit literature, one of the most important questions is *what kind of regret is expected from a good policy or algorithm?* The short answer is sublinear, meaning $\lim_{T\to\infty}\mathbb{E}(\mathcal{R}_T)/T = 0$, or equivalently $\mathbb{E}(\mathcal{R}_T) = o(T)$. We want cumulative regret to grow slower than the horizon T, such that over time we accumulate less and less regret per period. We illustrate this idea with the following example. Suppose we have three arms $\mathcal{W} = \{1, 2, 3\}$ where rewards follow Normal distributions, i.e., $Y_t(k) \sim \mathcal{N}(\mu(k), 1)$ for $k = 1, 2, 3$ with $\mu(1) = 0.10$, $\mu(2) = 0.50$, and $\mu(3) = 0.20$. The best arm is $k^* = 2$ with mean reward $\mu^* = 0.50$. The table Table 1.1 shows one possible realization of arm plays over horizon $T = 8$.

Notice that although the learner initially explores different arms that give some instanta-

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$
W_t	1	3	1	2	1	2	2	2
$Y_t(W_t)$	0.05	0.30	0.10	0.52	0.47	0.63	0.41	0.58
$\mu(W_t)$	0.10	0.20	0.10	0.50	0.10	0.50	0.50	0.50
μ^*	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
$r_t = \mu^* - \mu(W_t)$	0.40	0.30	0.40	0	0.40	0	0	0
$\mathcal{R}_t = \sum_{s=1}^{t} r_s$	0.40	0.70	1.10	1.10	1.50	1.50	1.50	1.50

Table 1.1: One possible realization of arm plays W_t, observed rewards $Y_t(W_t)$, mean rewards $\mu(W_t)$, instantaneous regrets r_t, and pseudo regrets $\mathcal{R}_t$ for horizon $T = 8$. The best arm is $k^* = 2$ with mean reward $\mu^* = 0.50$. Notice how the learner switches to Arm 2 in later periods, causing R_t to stabilize.

neous regret r_t, eventually it learns the best arm and plays it frequently. After some time, the instantaneous regret r_t is often zero. So even though cumulative regret $\mathcal{R}_T = \sum_{t=1}^{T} r_t$ grows initially, it grows slowly relative to T. This is the idea of *sublinear regret*. A good learner achieves this sublinear behavior as quickly as possible. A policy or algorithm that achieves sublinear regret is called a *consistent policy*.

The next question we ask is - *how do we compare different sublinear algorithms?* The idea is to look at how fast cumulative regret grows with the horizon T. In other words, we seek the smallest $\alpha < 1$ such that $\mathbb{E}(\mathcal{R}_T) = O(T^\alpha)$. Of course, we need $\alpha < 1$, but the smaller α is, the better the algorithm performs in terms of regret. For example, some policies achieve $\mathbb{E}(\mathcal{R}_T) = O(\sqrt{T})$, others $\mathbb{E}(\mathcal{R}_T) = O\left(T^{2/3}\right)$, and the best achieve logarithmic regret $\mathbb{E}(\mathcal{R}) = O(\log T)$. Algorithms with $\sqrt{T}$ regret are preferred to those with $T^{2/3}$ regret, and algorithms with logarithmic regret $\log T$ are preferred to those with $\sqrt{T}$ regret.

If we ask what is the best we can achieve, an early and influential answer was given by Lai and Robbins (1985). They showed that for any fixed class of parametric distributions (e.g., single-parameter exponential families), the asymptotic lower bound for the regret of any consistent policy is:

$$\liminf_{T\to\infty} \frac{\mathbb{E}(\mathcal{R}_T)}{\log T} \geq \sum_{k:\Delta(k)>0} \frac{\Delta(k)}{D_{\mathrm{kl}}\left(\mathrm{F}_{\mu(k)} \| \mathrm{F}_{\mu^*}\right)}, \tag{1.4}$$

where $\Delta(k) = \mu^* - \mu(k)$ is the suboptimality gap (the gap between the mean of the best arm and arm k), and $D_{\mathrm{kl}}\left(\mathrm{F}_{\mu(k)} \| \mathrm{F}_{\mu^*}\right)$ denotes the Kullback-Leibler divergence between the reward distribution of arm k and the optimal arm. For example, in a Bernoulli bandit where $\mu(k)$ is the success probability of arm k and μ^* is the success probability of the optimal arm, we have:

$$D_{kl}\left(\mathrm{F}_{\mu(k)} \| \mathrm{F}_{\mu^*}\right) = \mathbb{E}_{Y(k)\sim \mathrm{F}_{\mu(k)}}\left[\log\left(\frac{f_k(y)}{f_{k^*}(y)}\right)\right]$$
$$= \mu(k)\log\left(\frac{\mu(k)}{\mu^*}\right) + (1-\mu(k))\log\left(\frac{1-\mu(k)}{1-\mu^*}\right) \tag{1.5}$$

where f_k and f_{k^*} are the probability mass functions of the Bernoulli distributions for arm k and the optimal arm k^* respectively, i.e., $f_k(y) = (\mu(k))^y(1-\mu(k))^{1-y}$ for $y \in \{0,1\}$.

Essentially, KL divergence measures how different the two distributions are. Note that in the Lai-Robbins lower bound, the KL divergence appears in the denominator. This implies that the closer the suboptimal arm's distribution is to the optimal arm's distribution (i.e., smaller KL divergence), the harder it is for the learner to distinguish between them. Consequently, the learner needs more samples from the suboptimal arm to confidently identify it as suboptimal, leading to higher regret. So in principle, our goal is to devise algorithms that can achieve this lower bound, i.e., that can achieve regret such that

$$\mathbb{E}\left(\mathcal{R}_T\right) \leq \sum_{k:\Delta(k)>0} \frac{\Delta(k)}{D_{\mathrm{kl}}\left(\mathrm{F}_{\mu(k)} \| \mathrm{F}_{\mu^*}\right)} \log T + o(\log T) \tag{1.6}$$

where in $o(\log T)$ we may have some additional terms that grow slower than $\log T$. Such algorithms are called *asymptotically optimal algorithms*.

Next, we show that regret can be decomposed into suboptimality gaps and number of times suboptimal arms are played. Sub-optimality gaps depends on the environment, so the learner has no control over it, however the number of times suboptimal arms are played depends on the policy or algorithm of the learner. So this tells us that if we devise algorithms that minimize the number of times suboptimal arms are played, then we can minimize regret.

Lemma 1 (Regret Decomposition). *The regret $\mathcal{R}_T$ can be decomposed as*

$$\mathcal{R}_T = \sum_{k\in\mathcal{W}} \Delta(k)\cdot n_T(k), \tag{1.7}$$

where $\Delta(k) = \mu^* - \mu(k)$ is the suboptimality gap for arm k, and $n_T(k) = \sum_{t=1}^{T} \mathbf{1}\{W_t = k\}$ is the number of times arm k has been played till time T.

Proof. We start with the definition,

$$\mathcal{R}_T = \sum_{t=1}^{T} \left(\mu^* - \mu(W_t)\right)$$

$$
\begin{aligned}
&= \sum_{t=1}^{T} \sum_{k \in \mathcal{W}} \mathbf{1}\{W_t = k\} \left(\mu^* - \mu(k)\right) \\
&= \sum_{k \in \mathcal{W}} \sum_{t=1}^{T} \mathbf{1}\{W_t = k\} \left(\mu^* - \mu(k)\right) \\
&= \sum_{k \in \mathcal{W}} n_T(k) \left(\mu^* - \mu(k)\right) \\
&= \sum_{k \in \mathcal{W}} \Delta(k) \cdot n_T(k). \qquad (1.8)
\end{aligned}
$$

This is the random or pseudo regret where not rewards but the arm selections are random due to policy, and then taking expectation in both sides gives the expected regret, or simply regret

$$
\mathbb{E}(\mathcal{R}_T) = \sum_{k \in \mathcal{W}} \Delta(k) \cdot \mathbb{E}\left(n_T(k)\right), \qquad (1.9)
$$

■

Here Δ_k depends on the environment F, so a learner should adopt a strategy that minimizes $n_T(k)$.

We end this section with a brief discussion of exploration and exploitation, this is possibly the most important feature of any bandit algorithm. Although we will demonstrate this balance in action in the next section when we discuss Thompson Sampling algorithm, but nevertheless here we explain the idea in words.

Recall in Table 1.1 we showed that a good learner initially plays different arms, but once she becomes more certain about the best arm, she plays it more frequently. The first behavior is what we call exploration, in short - *trying different arms and observing their rewards*. The second behavior is actually exploitation - *once the learner is sufficiently confident about which arm is best, she exploits this knowledge by playing that arm more frequently*.

It's obvious that a good algorithm must balance these two aspects carefully, too much or too little of these aspects might lead to a bad performance. First note that, without exploration, a learner cannot discover the best arm. However, the issue is, excessive or misdirected exploration also increases the cumulative regret. This is because every pull of a suboptimal arm brings a loss of $\Delta(k) = \mu^* - \mu(k)$. On the other hand, if we go for pure exploitation since we get some good performance of an arm during the early phase, this may lead to premature convergence to a suboptimal arm. When designing a bandit algorithm a learner needs to balance this trade-off with great care, and always keep in mind that - we need to explore enough so that we can identify the best arm with high confidence, but also we need to start exploiting our knowledge so that we can minimize the regret along the way.

Algorithm 1: Thompson Sampling Algorithm (General Case)

Input: $\mathcal{W}$, T, $\widetilde{\theta}(k)\ \forall k \in \mathcal{W}$
Initialize: $\widehat{\theta}_0(k) = \widetilde{\theta}(k)\ \forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  sample $\mu'_t(k) \sim Q^{\widehat{\theta}_{t-1}(k)}\ \forall k \in \mathcal{W}$
  pick $W_t = \arg\max_{k \in \mathcal{W}} \mu'_t(k)$
  observe $Y_t = Y_t(W_t)$
  update $\widehat{\theta}_t(W_t)$
Output: $W_t, Y_t, \forall\ t \in [T]$

1.3 Thompson Sampling Algorithm

Now we introduce and discuss the Thompson sampling (TS) algorithm for the stochastic bandit problem. It is a well known algorithm for it's simplicity, empirical performance and strong theoretical guarantees. It was first proposed by Thompson (1933), but gained renewed interest time to time in various settings. At the core, Thompson sampling is a Bayesian algorithm that maintains prior-posterior distributions over the parameters (e.g., means) of rewards for the arms that are updated over each observation of rewards and at every t, and then samples from these distributions to make decisions. In this chapter we give a somewhat detailed explanation of the basic Thompson sampling and in the next chapter we extend this classic algorithm to the interference setting.

We start with introducing some notations. We recall, $\mathscr{F}_\mu = \{\mathrm{F}_{\mu(k)} : k \in \mathcal{W}\}$ be the set of reward distributions, where mean for arm k is denoted with $\mu(k) = \mathbb{E}[Y_t(k)]$. Thompson sampling is a Bayesian algorithm, and it requires specifying a *prior distribution* for each arm, that summarizes our initial belief about that arm's reward before observing any data, this is what we call a *"prior"*. In our case, we let $Q^{\theta(k)}$ to be the prior distribution of the mean $\mu(k)$, where $\theta(k)$ are the parameters of the prior distribution, i.e., this is the *prior parameters* of arm k. The idea of the Thompson sampling is, we start with some initialization of the prior parameters for each arm, observe rewards over time, and then update the *posterior parameters* based on the observed rewards. We outline the general Thompson sampling procedure Algorithm 1, here we do not specify the exact forms of the reward distributions or the updates for the prior/posterior distributions, since this is a general procedure.

The algorithm has four simple steps, *sampling*, *selection*, *arm pull* and *update*. First at time $t = 0$, the learner starts with a time horizon T, an arm set $\mathcal{W}$ and some fixed values $\widetilde{\theta}(k)$, $\forall k \in \mathcal{W}$, as initialization values for the prior. If $\widehat{\theta}_t(k)$ denotes the posterior parameters of arm k after time t, then with the initialization values, the learner sets $\widehat{\theta}_0(k) = \widetilde{\theta}(k)$ for each arm $k \in \mathcal{W}$.

Then at each period t, $\forall k \in \mathcal{W}$, the learner generates a random mean $\mu'_t(k)$ from the current prior distribution $Q^{\widehat{\theta}_{t-1}(k)}$. Then from these sampled means, she selects the arm W_t that has the highest sampled mean. After that she pulls the selected arm W_t and observes the reward $Y_t = Y_t(W_t)$. The learner does not observe rewards from the other arms. And then she finally updates the prior parameters $\widehat{\theta}_{t-1}(W_t)$ to the posterior parameters $\widehat{\theta}_t(W_t)$ based on the observed reward Y_t. So these new posterior parameters will be used as prior parameters for the next time step for the selected arm. The prior parameters for all other arms remain unchanged i.e., $\widehat{\theta}_t(k) = \widehat{\theta}_{t-1}(k)$ for all $k \neq W_t$. This process is repeated until the time horizon T is reached.

We give a concrete example using Gaussian rewards with Gaussian priors. Let $Y_t(k) \sim \mathcal{N}(\mu(k), \sigma^2)$, and assume the priors are also Gaussian, i.e., $\mu(k) \sim \mathcal{N}(\widehat{\mu}_t(k), \widehat{\sigma}_t^2(k))$ (this is called conjugate prior, we will see some details later), the key steps of the algorithm can be summarized in the following Figure 1.3,

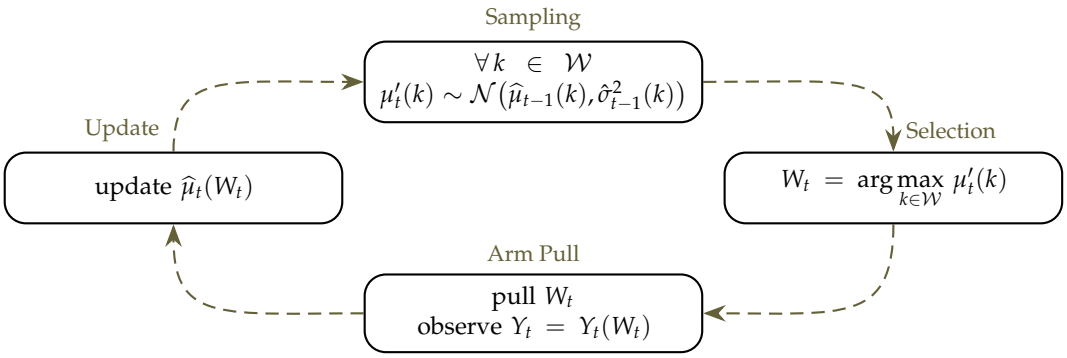


Figure 1.3: Thompson Sampling steps: Sampling, Selection, Arm Pull, and Update.

The specific form of the update depends on the reward distributions and the prior assumption. We cannot always obtain analytical forms for the updates; however, sometimes it is possible to derive neat analytical results. In particular, when we have conjugate priors -meaning the priors and derived posteriors belong to the same family of distributions and closed form parameter updates are available. We discuss the Thompson Sampling procedure with the conjugate priors and updates for Bernoulli and Gaussian rewards in the next sections. In both cases, the updates can be expressed in terms of two key quantities: $S_t(k)$, the cumulative reward from arm k, and $n_t(k)$, the number of times arm k has been played up to time t:

$$S_t(k) = \sum_{s=1}^{t} \mathbb{1}\{W_s = k\} Y_s \tag{1.10}$$

$$n_t(k) = \sum_{s=1}^{t} \mathbb{1}\{W_s = k\} \tag{1.11}$$

Algorithm 2: Thompson Sampling (Bernoulli Rewards)

Input: $\mathcal{W}$, T, $\widetilde{\alpha}(k)$, $\widetilde{\beta}(k)$ $\forall k \in \mathcal{W}$
Initialize: $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k)$, $\widehat{\beta}_0(k) = \widetilde{\beta}(k)$, $S_0(k) = 0$ *and* $n_0(k) = 0$ $\forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
    sample $p'_t(k) \sim \text{Beta}(\widehat{\alpha}_{t-1}(k), \widehat{\beta}_{t-1}(k))$ $\forall k \in \mathcal{W}$
    pick $W_t = \arg\max_{k \in \mathcal{W}} p'_t(k)$
    observe $Y_t = Y_t(W_t)$
    update
        $S_t(W_t) = S_{t-1}(W_t) + Y_t$ and $n_t(W_t) = n_{t-1}(W_t) + 1$
        $\widehat{\alpha}_t(W_t) = \widetilde{\alpha}(W_t) + S_t(W_t)$ and $\widehat{\beta}_t(W_t) = \widetilde{\beta}(W_t) + n_t(W_t) - S_t(W_t)$
Output: $W_t, Y_t, \forall\, t \in [T]$

While running the algorithm, these can be calculated recursively as,

$$\begin{aligned} S_t(W_t) &= S_{t-1}(W_t) + Y_t \\ n_t(W_t) &= n_{t-1}(W_t) + 1 \end{aligned} \tag{1.12}$$

In the following sections for Bernoulli and Gaussian rewards, we write the parameter updates in terms of cumulative rewards $S_t(\cdot)$ and number of plays $n_t(\cdot)$.

1.3.1 *Bernoulli Rewards with Beta Priors*

One of the most well known examples of Thompson sampling is assuming Bernoulli rewards with Beta Priors, that is for all arm $k \in \mathcal{W}$, we assume the rewards follow $Y_t(k) \sim \text{Bernoulli}(p(k))$ where $\mu(k) = \mathbb{E}[Y_t(k)] = p(k)$ is the success probability for arm k, and $p(k)$ follows Beta distribution, i.e.,

$$p(k) \sim \text{Beta}\left(\alpha(k), \beta(k)\right) \tag{1.13}$$

where $\alpha(k)$ and $\beta(k)$ are parameters of the prior distribution. Since we sample the means and update the parameters at each time t, we denote the parameters at time t with $\widehat{\alpha}_t(k)$ and $\widehat{\beta}_t(k)$ and the sampled mean at time t with $p'_t(k)$. The Thompson sampling procedure for Bernoulli rewards is summarized in Algorithm 2.

At $t = 0$, we set $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k)$ and $\widehat{\beta}_0(k) = \widetilde{\beta}(k)$. The parameter updates are coming from the conjugate prior property of the Beta distribution. In particular, using the Bayes' rule, one can show that, if the prior follows $\text{Beta}(\alpha, \beta)$, i.e., $p(k) \sim \text{Beta}(\alpha, \beta)$, then after observing n Bernoulli trials with s successes and $n - s$ failures, the posterior distribution follows $\text{Beta}(\alpha + s, \beta + n - s)$ (the proof is given in Section A1.1). Therefore, in our setup,

after observing rewards up to time t, the posterior parameters for the reward of arm k can be written as,

$$\begin{aligned}\widehat{\alpha}_t(k) &= \widetilde{\alpha}(k) + \sum_{s=1}^{t} \mathbb{1}\{W_s = k\} Y_s = \widetilde{\alpha}(k) + S_t(k) \text{ and} \\ \widehat{\beta}_t(k) &= \widetilde{\beta}(k) + \sum_{s=1}^{t} \mathbb{1}\{W_s = k\}(1 - Y_s) = \widetilde{\beta}(k) + n_t(k) - S_t(k)\end{aligned} \tag{1.14}$$

Note that for Bernoulli Thompson sampling, it is possible to perform a recursive update of the posterior parameters directly. In that case, there is no need to keep track of $S_t(k)$ and $n_t(k)$, and the update becomes $\widehat{\alpha}_t(W_t) = \widehat{\alpha}_{t-1}(W_t) + Y_t$ and $\widehat{\beta}_t(W_t) = \widehat{\beta}_{t-1}(W_t) + 1 - Y_t$. However, since we generally need $S_t(k)$ and $n_t(k)$ for other purposes in this work in general, we keep track of these cumulative statistics and use them to update the posterior parameters.

For Bernoulli rewards, in practice we often assume $\widetilde{\alpha}(k) = 1$ and $\widetilde{\beta}(k) = 1$ for all $k \in \mathcal{W}$, meaning we start with $\text{Beta}(1, 1)$, which is equivalent to the uniform distribution over $[0, 1]$. With this assumption, the update formulas simplify to:

$$\begin{aligned}\widehat{\alpha}_t(k) &= 1 + S_t(k) \\ \widehat{\beta}_t(k) &= 1 + n_t(k) - S_t(k)\end{aligned} \tag{1.15}$$

To demonstrate typical behavior of the Thompson Sampling algorithm with Bernoulli rewards, we provide a graphical illustration in Figure 1.4. Here $p(1) = 0.6$ and $p(2) = 0.2$. This means Arm 1 is the optimal arm. We start with a uniform prior, $\text{Beta}(1, 1)$, for both arms. As the algorithm progresses, it plays Arm 1 most of the time, collecting more samples from it. As a result, the posterior for Arm 1 concentrates around its true mean with decreasing variance. Meanwhile, Arm 2's posterior also concentrates around its true mean, but with higher variance since it is played less frequently. So eventually we will see that the optimal arm is played more often as the algorithm progresses.

We can derive the mean and variance of the sampled mean $p'_{t+1}(k)$. Since we assume $p'_{t+1}(k) \sim \text{Beta}(\widehat{\alpha}_t(k), \widehat{\beta}_t(k))$, from the properties of the Beta distribution, we have,

$$\begin{aligned}\mathbb{E}[p'_{t+1}(k)] &= \frac{\widehat{\alpha}_t(k)}{\widehat{\alpha}_t(k) + \widehat{\beta}_t(k)} \\ &= \frac{\widetilde{\alpha}(k) + S_t(k)}{\widetilde{\alpha}(k) + \widetilde{\beta}(k) + n_t(k)}\end{aligned} \tag{1.16}$$

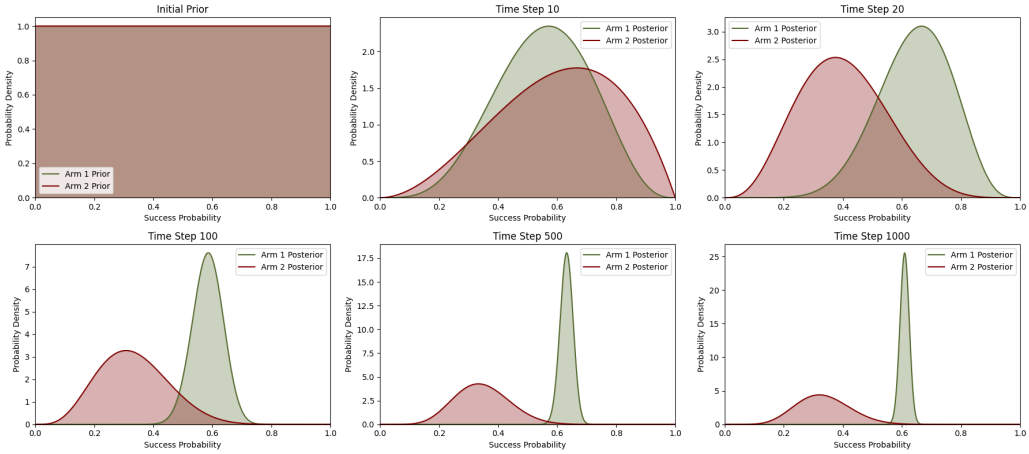


Figure 1.4: Thompson sampling with Bernoulli rewards and Beta priors in action. Environment parameters are $p(1) = 0.6$ and $p(2) = 0.2$. We start from Uniform prior, Beta$(1,1)$, for both arms, and as the algorithm progresses, the posterior distribution for the best arm (Arm 1) concentrates around its true mean with decreasing variance.

$$\begin{aligned}\mathbb{V}[p'_{t+1}(k)] &= \frac{\widehat{\alpha}_t(k)\widehat{\beta}_t(k)}{(\widehat{\alpha}_t(k)+\widehat{\beta}_t(k))^2(\widehat{\alpha}_t(k)+\widehat{\beta}_t(k)+1)} \\ &= \frac{(\widetilde{\alpha}(k)+S_t(k))\left(\widetilde{\beta}(k)+n_t(k)-S_t(k)\right)}{\left(\widetilde{\alpha}(k)+\widetilde{\beta}(k)+n_t(k)\right)^2\left(\widetilde{\alpha}(k)+\widetilde{\beta}(k)+n_t(k)+1\right)}\end{aligned} \tag{1.17}$$

Where we wrote the posterior parameters with initial values, since these can be written as $\widehat{\alpha}_t(k) = \widetilde{\alpha}(k) + S_t(k), \widehat{\beta}_t(k) = \widetilde{\beta}(k) + n_t(k) - S_t(k)$. Setting the initial values $\widetilde{\alpha}(k) = \widetilde{\beta}(k) = 1$, i.e., assuming uniform prior, the mean and variance simplify to,

$$\begin{aligned}\mathbb{E}[p'_{t+1}(k)] &= \frac{1+S_t(k)}{2+n_t(k)} \\ \mathbb{V}[p'_{t+1}(k)] &= \frac{(1+S_t(k))\,(1+n_t(k)-S_t(k))}{(2+n_t(k))^2\,(3+n_t(k))}\end{aligned} \tag{1.18}$$

This gives some interesting insights. First note that, as we play arm k more often, the sampled $p'_{t+1}(k)$ will be close to it's expected value $\mathbb{E}[p'_{t+1}(k)]$ since the variance $\mathbb{V}[p'_{t+1}(k)]$ decreases with $n_t(k)$. Moreover, as $n_t(k)$ increases, $\mathbb{E}\left[p'_{t+1}(k)\right] = \frac{1+S_t(k)}{2+n_t(k)} \approx \quad \frac{S_t(k)}{n_t(k)} = \overline{p}_t(k)$, where $\overline{p}_t(k)$ is the empirical mean of the observed rewards for arm k up to time t. So this means when arm is played long enough, i.e., after good amount of exploitation, sampled p'_{t+1} will get close to the empirical mean $\overline{p}_t(k)$. And since by the Law of Large Numbers, $\overline{p}_t(k) \to p(k)$, our sampled p'_{t+1} will be close to the true success probability $p(k)$.

Algorithm 3: Thompson Sampling (Gaussian Rewards, Known Variance)

Input: $\mathcal{W}$, T, $\sigma^2(k)$, $\widetilde{\sigma}^2(k)$, $\widetilde{\mu}(k)$ $\forall k \in \mathcal{W}$
Initialize: $\widehat{\sigma}_0^2(k) = \widetilde{\sigma}^2(k)$, $\widehat{\mu}_0(k) = \widetilde{\mu}(k)$, $S_0(k) = 0$, $n_0(k) = 0$ $\forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  sample $\mu_t'(k) \sim \mathcal{N}\left(\widehat{\mu}_{t-1}(k), \widehat{\sigma}_{t-1}^2(k)\right)$ $\forall k \in \mathcal{W}$
  pick $W_t = \arg\max_{k \in \mathcal{W}} \mu_t'(k)$
  observe $Y_t = Y_t(W_t)$
  update
    $S_t(W_t) = S_{t-1}(W_t) + Y_t$ and $n_t(W_t) = n_{t-1}(W_t) + 1$
    $\widehat{\sigma}_t^2(W_t) = \left(\frac{n_t(W_t)}{\sigma^2(W_t)} + \frac{1}{\widetilde{\sigma}^2(W_t)}\right)^{-1}$
    $\widehat{\mu}_t(W_t) = \left(\frac{S_t(W_t)}{\sigma^2(W_t)} + \frac{\widetilde{\mu}(W_t)}{\widetilde{\sigma}^2(W_t)}\right)\widehat{\sigma}_t^2(W_t)$
Output: $W_t, Y_t, \forall\, t \in [T]$

1.3.2 Gaussian Rewards with Gaussian Priors

Now we look at another well-known case of Thompson sampling that is assuming Gaussian rewards with known variance. In particular, we assume for all $k \in \mathcal{W}, Y_t(k) \sim \mathcal{N}\left(\mu(k), \sigma^2(k)\right)$, where the means $\mu(k) = \mathbb{E}\left[Y_t(k)\right]$ are *unknown* to the learner but the variances $\mathbb{V}\left[Y_t(k)\right] = \sigma^2(k)$ are *known*. Now for the unknown means, in Bayesian setting we assume Gaussian priors. We denote $\widehat{\mu}_t(k)$ and $\widehat{\sigma}_t^2(k)$ as the parameters of the posterior distribution updated at time t, and sampling is done with $\mu_t'(k) \sim \mathcal{N}\left(\widehat{\mu}_{t-1}(k), \widehat{\sigma}_{t-1}^2(k)\right)$ at time t. Note that we only place a prior on the mean, as the variance is assumed to be known to the learner. At $t = 0$, we set $\widehat{\mu}_0(k) = \widetilde{\mu}(k)$ and $\widehat{\sigma}_0^2(k) = \widetilde{\sigma}^2(k)$, where $\widetilde{\mu}(k)$ and $\widetilde{\sigma}^2(k)$ are the initial prior parameters.

It can be shown that if the prior is Gaussian, then the posterior will also be Gaussian[†] Using the Bayes' rule, after time t, the posterior parameters are updated as:

$$\begin{aligned} \widehat{\sigma}_t^2(k) &= \left(\frac{n_t(k)}{\sigma^2(k)} + \frac{1}{\widetilde{\sigma}^2(k)}\right)^{-1} \\ \widehat{\mu}_t(k) &= \left(\frac{S_t(k)}{\sigma^2(k)} + \frac{\widetilde{\mu}(k)}{\widetilde{\sigma}^2(k)}\right)\widehat{\sigma}_t^2(k) \end{aligned} \tag{1.19}$$

where $S_t(k)$ and $n_t(k)$ are the cumulative rewards and cumulative counts of playing arm k up to time t. Thompson Sampling algorithm for Gaussian rewards with known variance follows the same structure as the Bernoulli case, only the posterior parameter updates are now different. The algorithm is outlined in Algorithm 3.

Suppose $\sigma^2(k) = 1, \widetilde{\mu}(k) = 0$, and $\widetilde{\sigma}^2(k) = 1$ for all $k \in \mathcal{W}$, meaning the reward distribu-

† This is another conjugate prior result, similar to the Beta-Bernoulli case, proof is given in Appendix A1.

tions for all arms have unit variance and the prior distributions of the means are standard normal. In this case, the posterior parameters simplify to:

$$\begin{aligned} \widehat{\sigma}_t^2(k) &= \frac{1}{n_t(k)+1} \\ \widehat{\mu}_t(k) &= \frac{S_t(k)}{n_t(k)+1} \end{aligned} \tag{1.20}$$

So if $n_t(k)$ is large, the posterior mean $\widehat{\mu}_t(k)$ will be close to the empirical mean $\frac{S_t(k)}{n_t(k)}$, and the variance of the posterior distribution will be very small. And here also we see the same phenomenon as in the Bernoulli case - as we play an arm more often, our uncertainty about its mean decreases, and our estimate of the mean converges to the empirical mean of the observed rewards and ultimately to the true mean $\mu(k)$.

Figure 1.5 shows the posterior distribution for Gaussian rewards at different time steps of the Thompson Sampling algorithm. Here again Arm 1 is the optimal arm but the now the rewards are Gaussian. The figure shows that we start from assuming same Gaussian prior, $\mathcal{N}(0,1)$, for both arms. As the algorithm progresses, it plays Arm 1 most of the time, collecting more samples from it. As a result, the posterior for Arm 1 concentrates around its true mean with decreasing variance. Meanwhile, Arm 2's posterior also concentrates, but with higher variance since it is played less frequently. So eventually we will see that the optimal arm is played more often as the algorithm progresses.

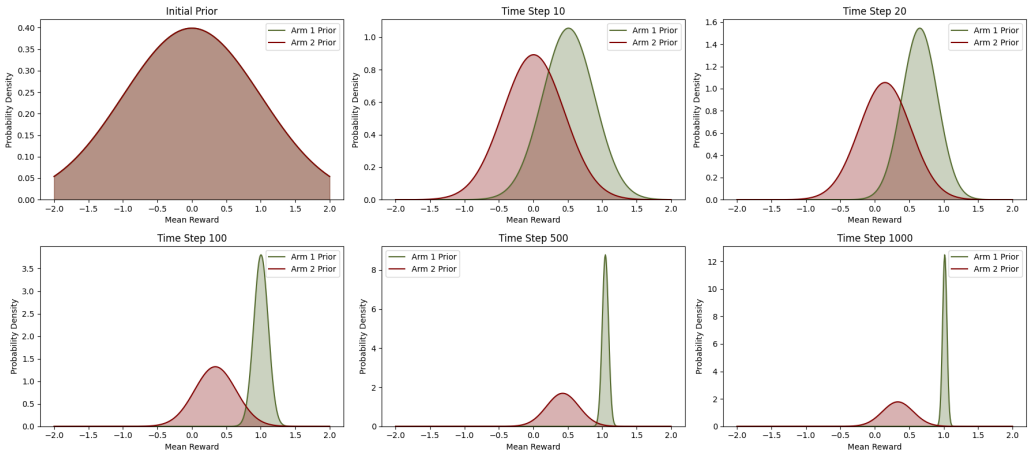


Figure 1.5: Thompson sampling with Gaussian rewards and Gaussian priors in action. Environment parameters are $\mu(1) = 1$ and $\mu(2) = 0.5$ with $\sigma^2(1) = \sigma^2(2) = 1$. We start from Gaussian prior, $\mathcal{N}(0,1)$, for both arms, and as the algorithm progresses, the posterior distribution for the best arm (Arm 1) concentrates around its true mean with decreasing variance.

1.4 Bayesian Bandit and Regret of Thompson Sampling

In this section, we give a brief overview of the regret analysis of Thompson sampling. We follow the Bayesian approach and give the Bayesian regret bound as opposed to the frequentist bound. Since Thompson sampling is a Bayesian algorithm, it's possible to analyze the regret both in the frequentist and in the Bayesian setting. The Bayesian setting has been primarily developed by Russo and Van Roy (2014), Russo and Roy (2016) and also see Russo, Roy, Kazerouni, Osband, and Wen (2020). They exploit a crucial property of Thompson sampling, known as *probability matching*, that states, *conditional on the history, the posterior distribution of the optimal arm k^* coincides with the distribution of the arm W_t drawn by Thompson sampling*. This idea leads to the key result of the Bayesian Regret, and we reproduced the proof for the Gaussian case below.

On the frequentist side, the key contributions are from Agrawal and Goyal (2012), Kaufmann, Korda, and Munos (2012) and Agrawal and Goyal (2013), also see the work Agrawal and Goyal (2017) which has all the key results from the earlier works. After the regret analysis we give some short remarks related to the frequentist regret analysis.

Before the Bayesian regret analysis, first we need to define the Bayesian bandit environment, the interaction protocol and also the Bayesian regret. The notion of Bayesian regret is similar to the frequentist regret defined in Definition 1.7, but the key difference is here the regret is averaged over all possible instances of the bandit environment according to the prior distribution. We give the formal definition below.

1.4.1 Bayesian Bandit Environment and Bayesian Regret

As we defined in Definition 1.3, the environment in the stochastic bandit setting is a fixed set of reward distributions $\mathscr{F}_\mu = \left\{\mathrm{F}_{\mu(k)} : k \in \mathcal{W}\right\}$, where the parameter set $\mu := \{\mu(k) : k \in \mathcal{W}\}$ is fixed but unknown to the learner. In the Bayesian bandit environment, the crucial difference is - *although the learner may know the class of reward distributions (e.g., Bernoulli or Gaussian), but the parameters are not fixed objects like in the frequentist setting, rather the parameter depends on a randomly drawn instance from a prior distribution*.

The Bayesian bandit environment needs three elements, $(\mathcal{E}, Q, \mathscr{F}_\mathcal{B})$, where $\mathcal{E}$ is the parameter space (set of all possible parameter vectors), Q is the prior distribution over $\mathcal{E}$ and $\mathscr{F}_\mathcal{B}$ is the collection of all set of reward distributions for different parameter vectors or we also call instances. Here each instance $\nu \in \mathcal{E}$ is a parameter vector for a specific bandit environment. For a given instance $\nu \in \mathcal{E}$, if we let $\mathscr{F}_\nu := \{\mathrm{F}_{\nu(k)} : k \in \mathcal{W}\}$ be the set of distributions for all arms, where $\mathrm{F}_{\nu(k)}$ is the marginal reward distribution for arm k, then we can write $\mathscr{F}_\mathcal{B} := \{\mathscr{F}_\nu : \nu \in \mathcal{E}\}$. Often $\mathscr{F}_\mathcal{B}$ is simply called the Bayesian bandit environment. We give a concrete example of Bernoulli rewards with a discrete $\mathcal{E}$.

Example 1.8 (Discrete Prior). Let $(\mathcal{E}, Q, \mathscr{F}_{\mathcal{B}})$ be a 2-armed Bayesian Bernoulli bandit with $\mathcal{E} = \{\nu_1, \nu_2, \nu_3\}$, where $\nu_1 = (0.7, 0.3)$, $\nu_2 = (0.5, 0.8)$ and $\nu_3 = (0.9, 0.2)$. This means there are three possible instances of the bandit environment in this setting. For example, if ν_2 is drawn, then the reward distributions for arm 1 and arm 2 are $\mathrm{F}_{\nu_2(1)} = \text{Bernoulli}(0.5)$ and $\mathrm{F}_{\nu_2(2)} = \text{Bernoulli}(0.8)$ respectively. The prior Q means the probability distribution over these three instances. For example, we may have $Q(\nu_1) = 0.4, Q(\nu_2) = 0.4, Q(\nu_3) = 0.2$. This means there are 40% chance that instance ν_1 is drawn, 40% chance for instance ν_2 and 20% chance for instance ν_3. So this is the probability distribution over the instance space $\mathcal{E}$. Now if ν_2 is drawn then we can write the marginal distributions as $\mathrm{F}_{\nu_2(1)} = \text{Bernoulli}(0.5)$ and $\mathrm{F}_{\nu_2(2)} = \text{Bernoulli}(0.8)$, and the $\mathscr{F}_{\nu_2} = \{\mathrm{F}_{\nu_2(1)}, \mathrm{F}_{\nu_2(2)}\} = \{\text{Bernoulli}(0.5), \text{Bernoulli}(0.8)\}$. Similarly we can write the marginal distributions for other instances. With this the Bayesian bandit environment can be written as, $\mathscr{F}_{\mathcal{B}} = \{\mathscr{F}_{\nu_j} : j \in \{1, 2, 3\}\}$, which is the set of all possible pairs of marginal distributions for all instances in this setting, which is the set of all possible pairs of marginal distributions for all instances in this setting.

///

Here is the formal definition of the Bayesian bandit environment and Bayesian bandit model.

Definition 1.9 (BAYESIAN BANDIT MODEL). *A Bayesian bandit environment is defined by a tuple $(\mathcal{E}, Q, \mathscr{F}_\mathcal{B})$ where $\mathcal{E}$ is the parameter space (set of all possible parameter vectors), Q is the prior distribution over $\mathcal{E}$ and $\mathscr{F}_\mathcal{B} := \{\mathscr{F}_\nu : \nu \in \mathcal{E}\}$ is the collection of all set of reward distributions for all instances, where for each instance $\nu \in \mathcal{E}$, we have $\mathscr{F}_\nu := \{\mathrm{F}_{\nu(k)} : k \in \mathcal{W}\}$ as the set of all reward distributions for all arms, and $\mathrm{F}_{\nu(k)}$ is the marginal reward distribution for arm k. A Bayesian bandit model is a repeated interaction between a learner and a Bayesian bandit environment where,*

for $t = 0$ **do**
  a bandit instance $\nu \sim Q$ is drawn for the learner from the parameter space $\mathcal{E}$ according to prior Q
for $t = 1, 2, \ldots, T$ **do**
  the learner selects an action $W_t \in \mathcal{W}$
  the environment generates potential outcomes $Y_t(k) \sim \mathrm{F}_{\nu(k)}$ for all $k \in \mathcal{W}$, where ν is a fixed instance drawn at $t = 0$
  the learner observes only the realized outcome $Y_t = Y_t(W_t)$, not the counterfactual outcomes $Y_t(k)$ for $k \neq W_t$.

Interaction Protocol 2: *Bayesian Bandit Interaction*

We have the following information structure,

- ***Known or Observable to the Learner:*** *The learner knows the set of arms $\mathcal{W}$, time horizon T, parameter space $\mathcal{E}$, prior Q and class of distributions for $\mathrm{F}_\nu(k)$, for all $k \in \mathcal{W}$ and all $\nu \in \mathcal{E}$, but does not know which ν appeared at time $t = 0$. She observes all actions and outcomes till time T, i.e., $\{(W_t, Y_t)\}_{t=1}^T$.*
- ***Known to the Environment:*** *The environment knows the complete set of counterfactual distributions $\mathscr{F}_\mathcal{B} = \{\mathscr{F}_\nu : \nu \in \mathcal{E}\}$, with $\mathscr{F}_\nu := \{\mathrm{F}_{\nu(k)} : k \in \mathcal{W}\}$ for each $\nu \in \mathcal{E}$ and observes what the learner observes.*

Below we give another example with a continuous prior for the Bernoulli rewards.

Example 1.10 (Continuous Prior). Although three instances in Example 1.8 is a very simple and easy to understand, in practice we may have continuous prior distributions over the parameter space. This is common in the Bernoulli bandit case, where the unknown success probabilities are often assigned Beta priors. For this we still have $(\mathcal{E}, Q, \mathscr{F}_\mathcal{B})$ as a 2-armed Bayesian Bernoulli bandit, but now because of the Beta distribution, we have $\mathcal{E} = [0, 1]^2$, i.e., continuous parameter space, infinite number of possible instances. Here we assume the prior Q is a product of independent Beta priors over the two arms. In particular, we can let prior for arm k as $Q_k = \mathrm{Beta}(\alpha_k, \beta_k)$ for parameters $\alpha_k, \beta_k > 0$. Therefore, the prior over the instance space $\mathcal{E}$ can be written as a joint distribution Q. Now if we think about a bandit instance $\nu = (\mu(1), \mu(2))$, then this is randomly drawn as $\nu \sim Q$. Again, for each $\nu \in \mathcal{E}$, we can write the marginal for each $k \in \mathcal{W}$ as $\mathrm{F}_{\nu(k)} = \mathrm{Bernoulli}(\mu(k))$, and the set of all marginal distributions for this instance

as $\mathscr{F}_\nu = \{F_{\nu(1)}, F_{\nu(2)}\} = \{\text{Bernoulli}(\mu(1)), \text{Bernoulli}(\mu(2))\}$. And the Bayesian bandit environment can be written as, $\mathscr{F}_\mathcal{B} = \{\mathscr{F}_\nu : \nu \in \mathcal{E}\}$, where $\nu \in [0,1]^2$.

///

With the definition and examples of Bayesian bandit and protocol, now we define the Bayesian regret in Definition 1.11.

Definition 1.11 (BAYESIAN REGRET)**.** *Consider a Bayesian bandit environment $(\mathcal{E}, Q, \mathscr{F}_\mathcal{B})$, for a policy π interacting with this environment over time horizon T, the Bayesian regret after horizon T is defined as*

$$\mathcal{BR}_T(\pi, \mathcal{E}, Q, \mathscr{F}_\mathcal{B}) = \mathbb{E}_{\nu\sim Q,\pi}\left[\sum_{t=1}^{T}(\mu_\nu^* - \mu_\nu(W_t))\right] \tag{1.21}$$

where $\mu_\nu^* = \max_{k\in\mathcal{W}} \mu_\nu(k)$ is the optimal mean reward for the instance ν drawn from prior Q, and W_t is the arm selected by policy π at time t. The expectation is taken over both the randomness in the bandit instance ν drawn from prior Q and the randomness in the policy π.

Bayesian regret can be understood similarly as the expected regret $\mathbb{R}(\mathcal{R}_T)$ we defined in Definition 1.7, the only difference is that we first define the expected regret for a fixed instance ν and then we average out the randomness of the bandit instances in $\mathcal{E}$ according to the prior Q. So for a fixed instance ν, the expected regret can be defined as,

$$\mathbb{E}_\pi^\nu\left[\sum_{t=1}^{T}(\mu_\nu^* - \mu_\nu(W_t))\right] \tag{1.22}$$

At this stage the Expectation is taken over the randomness of the policy π for a fixed instance ν. Now we take the expectation of this regret over the randomness of the bandit instances in $\mathcal{E}$, this gives the Bayesian Regret.

$$\mathcal{BR}_T(\pi, \mathcal{E}, Q, \mathscr{F}_\mathcal{B}) = \mathbb{E}_{\nu\sim Q,\pi}\left[\sum_{t=1}^{T}(\mu_\nu^* - \mu_\nu(W_t))\right] = \mathbb{E}_{\nu\sim Q}\left[\mathbb{E}_\pi^\nu\left[\sum_{t=1}^{T}(\mu_\nu^* - \mu_\nu(W_t))\right]\right] \tag{1.23}$$

So once $(\mathcal{E}, Q, \mathscr{F}_\mathcal{B})$ is fixed, the Bayesian regret is a deterministic quantity, where we average over the randomness instances, the policy, and the reward noise.

1.4.2 Regret Analysis of Thompson Sampling

We now give the detailed regret analysis of Thompson sampling algorithm for Gaussian rewards in the Bayesian setting. This follows primarily from the works of Russo and Van Roy (2014) and Lattimore and Szepesvári (2020). In particular Lattimore and Szepesvári (2020)

give the proof for sub-Gaussian random variables where the means are bounded in $[0,1]$, we adapt the proof in the Gaussian setup. Now, before proving the theorem first we prove the following lemma which states that, in the bandit setup for all arms, the empirical means concentrate around the true means with high probability.

Lemma 2 (CONCENTRATION OF EMPIRICAL MEANS). *Consider a K-armed Bayesian bandit environment with horizon T. Conditional on a fixed instance ν, for any arm k, assume the rewards are Gaussian with mean $\mu_\nu(k)$ and variance σ^2, i.e., $Y_t(k) \sim \mathcal{N}\left(\mu_\nu(k), \sigma^2\right)$. For a fixed $\delta > 0$, assuming $n_{t-1}(k) \geq 1 \; \forall k \in \mathcal{W}$, define the clean event as,*

$$E(\delta) = \left\{ \forall t \in [T], \forall k \in \mathcal{W} : \left|\overline{Y}_{t-1}(k) - \mu_\nu(k)\right| < \sigma \sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} \right\} \tag{1.24}$$

where, $\overline{Y}_{t-1}(k)$ is the empirical mean of arm k up to time $t-1$, and $n_{t-1}(k)$ is the number of times arm k has been played up to time $t-1$. Then conditional ν, it is possible to show,

$$\mathbb{P}\left(E^c(\delta)\right) \leq 2TK\delta \tag{1.25}$$

Proof. Fix an instance ν, then for an arm $k \in \mathcal{W}$ the clean event can be written as:

$$E(\delta, k) = \bigcap_{t=1}^{T} \left\{ \left|\overline{Y}_{t-1}(k) - \mu(k)\right| < \sigma \sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} \right\} \tag{1.26}$$

The complement of this event is then,

$$E^c(\delta, k) = \bigcup_{t=1}^{T} \left\{ \left|\overline{Y}_{t-1}(k) - \mu_\nu(k)\right| \geq \sigma \sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} \right\} \tag{1.27}$$

Looking at the event, it should be clear that there is an issue if we would like to apply the tail bound for a sequence of i.i.d. Gaussian variables in this setting‡. In particular, the term $n_{t-1}(k)$ is random and selected by the algorithm. To tackle this issue we use here the idea of pre-generated reward tapes following Slivkins (2022). The idea is to assume that for each arm k, we have a pre-generated i.i.d. rewards $Y^1(k), Y^2(k), \ldots, Y^T(k)$ from $\mathcal{N}\left(\mu_\nu(k), \sigma^2\right)$. Then we define $\overline{y}^m(k) = \frac{1}{m}\sum_{j=1}^{m} Y^j(k)$ as the empirical mean of the 1st m rewards from the tape of arm k. Now we can apply the concentration bound given in eq. (1.28) to $\overline{y}^m(k)$ for a fixed m since it does not depend on the history.

‡ Let $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} \mathcal{N}(\mu, \sigma^2)$. For a fixed $\delta > 0$, the Gaussian tail bound for the sample mean is given as:

$$\mathbb{P}\left(|\overline{X}_n - \mu| \geq \sigma \sqrt{\frac{2\log(1/\delta)}{n}} \right) \leq 2\,\delta \tag{1.28}$$

.
The proof is given in Section A2.2

$$\mathbb{P}\left(|\overline{y}^m(k) - \mu_v(k)| \geq \sigma\sqrt{\frac{2\log(1/\delta)}{m}}\right) \leq 2\delta \tag{1.29}$$

Now at time t, there exists $m \in \{1, \ldots, T\}$ such that $n_{t-1}(k) = m$, and we will have $\overline{Y}_{t-1}(k) = \overline{y}^m(k)$, this gives,

$$\begin{aligned} E^c(\delta, k) &= \bigcup_{t=1}^{T} \left\{ \left|\overline{Y}_{t-1}(k) - \mu_v(k)\right| \geq \sigma\sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} \right\} \\ &\subseteq \bigcup_{m=1}^{T} \left\{ |\overline{y}^m(k) - \mu_v(k)| \geq \sigma\sqrt{\frac{2\log(1/\delta)}{m}} \right\} \end{aligned} \tag{1.30}$$

Applying the union bound we get,

$$\begin{aligned} \mathbb{P}\left(E^c(\delta, k)\right) &\leq \sum_{m=1}^{T} \mathbb{P}\left(|\overline{y}_m(k) - \mu_v(k)| \geq \sigma\sqrt{\frac{2\log(1/\delta)}{m}}\right) \\ &\leq \sum_{m=1}^{T} 2\delta = 2T\delta \end{aligned} \tag{1.31}$$

Finally applying union bound over all arms $k \in \mathcal{W}$ we get,

$$\mathbb{P}\left(E^c(\delta)\right) = \mathbb{P}\left(\bigcup_{k\in\mathcal{W}} E^c(\delta, k)\right) \leq \sum_{k\in\mathcal{W}} \mathbb{P}\left(E^c(\delta, k)\right) \leq \sum_{k\in\mathcal{W}} 2T\delta = 2TK\delta \tag{1.32}$$

■

This completes the lemma. The event we defined in eq. (1.24) is sometimes called *clean event* or *good event* in formally (Slivkins, 2022). The name came because on this event, the empirical means concentrate around the true means for all arm k in all time periods. In the proof we separately consider the regret on this clean event and its complement. We already proved that the probability of the complement is small, so the regret on this event will be small as well. And on the clean event, we can use the concentration property to bound the regret. Now we are ready to prove the main theorem regarding the Bayesian regret of Thompson Sampling for Gaussian rewards.

Theorem 1.12 (Bayesian Regret of Thompson Sampling for Gaussian Rewards)**.** *Consider a K-armed Bayesian bandit environment with horizon $T \geq 2$. Under Gaussian reward assumption, for an instance $\nu \sim Q$, for all $k \in \mathcal{W}$, we have $Y_t(k) \sim \mathcal{N}\left(\mu_\nu(k), \sigma^2\right)$ wtih $\mu_\nu(k) \in [-\widetilde{B}, \widetilde{B}]$. Assuming $n_{t-1}(k) \geq 1, \forall k \in \mathcal{W}$ (i.e., all arms have been pulled at least once), the Bayesian regret of Thompson sampling algorithm outlined in Algorithm 1 satisfies:*

$$\mathcal{BR}_T \leq 8\sigma\sqrt{KT\log T} + 8K\widetilde{B} \tag{1.33}$$

Proof. We start with the definition of Bayesian regret,

$$\begin{aligned}\mathcal{BR}_T = \mathbb{E}_{\pi,\nu\sim Q}\left[\sum_{t=1}^{T}(\mu_\nu(k^*) - \mu_\nu(W_t))\right] &= \sum_{t=1}^{T}\mathbb{E}_{\pi,\nu\sim Q}\left[(\mu_\nu(k^*) - \mu_\nu(W_t))\right] \\ &= \sum_{t=1}^{T}\mathbb{E}_{\pi,\nu\sim Q}\left[\mathbb{E}\left[\mu_\nu(k^*) - \mu_\nu(W_t) \mid \mathcal{F}_{t-1}\right]\right] \end{aligned} \tag{1.34}$$

where $\mathcal{F}_{t-1} = \sigma(W_1, Y_1, \ldots, W_{t-1}, Y_{t-1})$ is the sigma algebra generated by the history up-to time $t-1$. Now let's define the upper confidence bound $U_t(k)$ as,

$$U_t(k) := \text{clip}_{[-\widetilde{B},\widetilde{B}]}\left(\overline{Y}_{t-1}(k) + \sigma\sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}}\right) \tag{1.35}$$

where the clipping function ensures that the upper confidence bound stays within the range $[-\widetilde{B}, \widetilde{B}]$[§]. Now from eq. (1.34), we can write,

$$\begin{aligned}\mathbb{E}[\mu_\nu(k^*) - \mu_\nu(W_t) \mid \mathcal{F}_{t-1}] &= \mathbb{E}[\mu_\nu(k^*) - U_t(W_t) + U_t(W_t) - \mu_\nu(W_t) \mid \mathcal{F}_{t-1}] \\ &= \mathbb{E}[\mu_\nu(k^*) - U_t(k^*) \mid \mathcal{F}_{t-1}] + \mathbb{E}[U_t(W_t) - \mu_\nu(W_t) \mid \mathcal{F}_{t-1}]\end{aligned} \tag{1.36}$$

where we used

$$\mathbb{P}(k^* = k \mid \mathcal{F}_{t-1}) = \mathbb{P}(W_t = k \mid \mathcal{F}_{t-1}) \quad \forall k \in \mathcal{W} \tag{1.37}$$

This is the probability matching property of Thompson sampling, which says, given $\mathcal{F}_{t-1}$, the true optimal arm k^* and the sampled arm W_t have the same conditional distribution, in particular we have $\mathbb{E}[U_t(W_t) \mid \mathcal{F}_{t-1}] = \mathbb{E}\left[U_t(k^*) \mid \mathcal{F}_{t-1}\right]$. Now define,

$$A = \sum_{t=1}^{T}(\mu_\nu(k^*) - U_t(k^*)), \quad B = \sum_{t=1}^{T}(U_t(W_t) - \mu_\nu(W_t)), \tag{1.38}$$

§ This is following the proof given in Lattimore and Szepesvári (2020).

and because of the linearity of expectation and probability matching property in eq. (1.37) we can write the Bayesian regret as,

$$\mathcal{BR}_T = \mathbb{E}_{\pi,\nu\sim Q}[A+B] = \mathbb{E}_{\pi,\nu\sim Q}[A] + \mathbb{E}_{\pi,\nu\sim Q}[B] \tag{1.39}$$

Next we bound A and B separately by considering two cases a) when the clean event $E(\delta)$ holds and b) when its complement $E(\delta)^c$ holds.

Case a) On Event E:

First we bound A. Note that, for all t and k, the concentration inequality holds here, i.e.,

$$\left|\overline{Y}_{t-1}(k) - \mu_\nu(k)\right| < \sigma\sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} \tag{1.40}$$

Since $U_t(k)$ is an upper confidence bound, on event E, for any k, we have

$$\mu_\nu(k^*) - U_t(k^*) \leq 0 \Rightarrow A \leq 0 \tag{1.41}$$

Now for B, for each t and k,

$$\begin{aligned}
&\overline{Y}_{t-1}(k) - \mu_\nu(k) < \sigma\sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} \\
\rightsquigarrow\ &\overline{Y}_{t-1}(k) + \sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} - \mu_\nu(k) < 2\sigma\sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}} \\
\rightsquigarrow\ &U_t(k) - \mu_\nu(k) < 2\sigma\sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}}
\end{aligned}$$

Now summing over time gives,

$$\begin{aligned}
B &= \sum_{t=1}^{T} \left(U_t(W_t) - \mu_\nu(W_t)\right) \\
&= \sum_{t=1}^{T} \sum_{k\in\mathcal{W}} \mathbb{1}\{W_t = k\} \left(U_t(k) - \mu_\nu(k)\right) \\
&< \sum_{t=1}^{T} \sum_{k\in\mathcal{W}} \mathbb{1}\{W_t = k\} \sigma\sqrt{\frac{8\log(1/\delta)}{n_{t-1}(k)}} \\
&= \sigma\sqrt{8\log(1/\delta)} \sum_{k\in\mathcal{W}} \sum_{t=1}^{T} \mathbb{1}\{W_t = k\} \frac{1}{\sqrt{n_{t-1}(k)}} \\
&\leq \sigma\sqrt{8\log(1/\delta)} \sum_{k\in\mathcal{W}} \int_0^{n_T(k)} \frac{1}{\sqrt{s}}\, ds
\end{aligned}$$

$$
\begin{aligned}
&= \sigma\sqrt{8\log(1/\delta)} \sum_{k\in\mathcal{W}} \left[2\sqrt{s}\right]_0^{n_T(k)} \\
&= \sigma\sqrt{8\log(1/\delta)} \sum_{k\in\mathcal{W}} 2\sqrt{n_T(k)} \\
&= \sigma\sqrt{32\log(1/\delta)} \sum_{k\in\mathcal{W}} \sqrt{n_T(k)} \\
&\leq \sigma\sqrt{32\log(1/\delta)} \sqrt{\left(\sum_{k\in\mathcal{W}} 1\right)\left(\sum_{k\in\mathcal{W}} n_T(k)\right)} \\
&= \sigma\sqrt{32\log(1/\delta)}\sqrt{KT} \\
&= \sigma\sqrt{32KT\log(1/\delta)}.
\end{aligned}
$$

where in the last step we used Cauchy-Schwarz inequality, in particular,

$$
\begin{aligned}
&\left(\sum_{k=1}^{K} \sqrt{n_T(k)}\right)^2 \leq \left(\sum_{k=1}^{K} 1\right)\left(\sum_{k=1}^{K} n_T(k)\right) = KT \\
&\Rightarrow \sum_{k=1}^{K} \sqrt{n_T(k)} \leq \sqrt{KT}
\end{aligned}
$$

So on event $E(\delta)$, we have

$$
A + B \leq \sigma\sqrt{32KT\log(1/\delta)} \tag{1.42}
$$

Case b) On Event $E(\delta)^c$:

On $E(\delta)^c$, we use a worst-case bound. Since $\mu_\nu(k) \in [-\widetilde{B}, \widetilde{B}]$ and $U_t(k) \in [-\widetilde{B}, \widetilde{B}]$ (due to clipping), for each t:

$$
\begin{aligned}
\mu_\nu(k^*) - U_t(k^*) &\in [-2\widetilde{B}, 2\widetilde{B}], \\
U_t(W_t) - \mu_\nu(W_t) &\in [-2\widetilde{B}, 2\widetilde{B}]
\end{aligned}
$$

So for each t

$$
\mu_\nu(k^*) - U_t(k^*) + U_t(W_t) - \mu_\nu(W_t) \in [-4\widetilde{B}, 4\widetilde{B}] \tag{1.43}
$$

Thus on event $E(\delta)^c$:

$$
A + B = \sum_{t=1}^{T} (\mu_\nu(k^*) - U_t(k^*) + U_t(W_t) - \mu_\nu(W_t)) \leq 4T\widetilde{B} \tag{1.44}
$$

Since $\mathbb{P}(E(\delta)^c) \leq 2TK\delta$, the contribution from E^c is;

$$\begin{aligned}\mathbb{E}\left[(A+B)\cdot \mathbb{1}_{E(\delta)^c}\right] &\leq 4T\widetilde{B}\cdot \mathbb{P}\left(E(\delta)^c\right)\\ &\leq 4T\widetilde{B}\cdot 2TK\delta\\ &= 8T^2K\widetilde{B}\delta\end{aligned}$$

Combining both cases, we have;

$$\begin{aligned}\mathcal{BR}_T &= \mathbb{E}[A+B]\\ &= \mathbb{E}[(A+B)\mid E]\mathbb{P}(E) + \mathbb{E}[(A+B)\mid E^c]\mathbb{P}(E^c)\\ &\leq \sigma\sqrt{32KT\log(1/\delta)} + 8T^2K\widetilde{B}\delta\\ &= \sigma\sqrt{32KT\log(1/\delta)} + 8T^2K\widetilde{B}\delta\end{aligned}$$

If we plug $\delta = \frac{1}{T^2}$, then

$$\begin{aligned}\mathcal{BR}_T &\leq \sigma\sqrt{64KT\log(T)} + 8K\widetilde{B}\\ &= 8\sigma\sqrt{KT\log(T)} + 8K\widetilde{B}\end{aligned} \tag{1.45}$$

■

Remarks 1.13 (Regarding Some Assumptions in the Proof)**.**

- We assumed $n_{t-1}(k) \geq 1$ for brevity, otherwise the UCB bound needs to be separately defined for $n_{t-1}(k) = 0$ and $n_{t-1}(k) \geq 1$ and then we need to separately track $n_{t-1}(k) = 0$. This won't change the order of the regret, and only the additional constant term in the regret will change.
- The high-probability bound for the clean event is first derived conditional on a fixed instance ν. Since the upper bound $2TK\delta$ is same for any ν, we can average over the prior distribution Q and obtain the same bound in the Bayesian setting.

///

The Bayesian regret analysis presented above is not the tightest possible bound for Thompson sampling, in particular, this is not the Lai-Robbins lower bound (Lai & Robbins, 1985) (see eq. (1.4)). And Thompson sampling can actually achieve better performance than this bound claims. However, this is shown in the frequentist, or instance-dependent setting as opposed to the Bayesian setting considered where we took the average over the all instances. We briefly summarize the key results here in the following remark.

Remarks 1.14 (Frequentist Regret of Thompson Sampling)**.** In the frequentist setting, Agrawal and Goyal (2012) provided the first finite-time analysis of Thompson sampling for Bernoulli bandits. For two arms with gap defined as Δ, they showed

$$\mathbb{E}[\mathcal{R}_T] = O\left(\frac{\log T}{\Delta} + \frac{1}{\Delta^3}\right)$$

For K arms, the bound is similar. This result actually gives $\log T$ regret but still not optimal in terms of gap dependence. The work by Kaufmann et al. (2012) came shortly after this and they gave the first asymptotically optimal finite-time analysis of Bernoulli Thompson sampling with Beta priors. They proved that the expected regret can be bounded as

$$\mathbb{E}[\mathcal{R}_T] \leq (1+\varepsilon) \sum_{k:\Delta(k)>0} \frac{\Delta(k)\log T}{D_{\mathrm{kl}}(\mathrm{F}_{\mu(k)} \| \mathrm{F}_{\mu^*})} + C(\varepsilon, \mu)$$

Now this matches the Lai-Robbins lower bound for Bernoulli bandits. Here $D_{\mathrm{kl}}(\mathrm{F}_{\mu(k)} \| \mathrm{F}_{\mu^*})$ is the KL divergence between Bernoulli distributions with parameters $\mu(k)$ and μ^* (see eq. (1.5)). Finally building on these results, Agrawal and Goyal (2013) introduced a martingale-based analysis that achieved two important result, a) for bounded rewards, they recovered an optimal-order problem-dependent bound

$$\mathbb{E}[\mathcal{R}_T] \leq (1+\varepsilon) \sum_{k:\Delta(k)>0} \frac{\Delta(k)\log T}{D_{\mathrm{kl}}(\mathrm{F}_{\mu(k)} \| \mathrm{F}_{\mu^*})} + O\left(\frac{K}{\varepsilon^2}\right)$$

and b) they proved the first near-optimal distribution-independent (also called problem-independent or minimax) guarantee

$$\mathbb{E}[\mathcal{R}_T] = O(\sqrt{KT\log T})$$

for Thompson Sampling with general priors. These results were gathered and organized in the journal version in Agrawal and Goyal (2017). This work also presents a unified treatment of the Beta-Bernoulli case, Gaussian priors case and also the bounded reward case. Note that what we get in as minimax or worst-case regret in the frequentist setting is similar to what we get in the Bayesian setting, however the key difference is that in the Bayesian setting we took average over all instances, while in the frequentist setting the worst-case regret is defined as the maximum regret over all possible instances.

///

Algorithm 4: Explore Then Commit (ETC)

Input: $\mathcal{W}$, m, T
for $t = 1, 2, \ldots, T$ **do**
  if $t \leq mK$ **then**
    $W_t = (t \mod K) + 1$ // play each arm m times
  else
    $W_t = \arg\max_k \widehat{\mu}_{mK}(k)$ // exploit the arm with max mean
  observe Y_t
Output: W_t and Y_t $\forall t = 1, 2, \ldots, T$

1.5 Other Algorithms for Stochastic Bandits

We end this section with a brief mention of two popular algorithms for stochastic bandits which are not covered with details here, but we refer the reader to Lattimore and Szepesvári (2020) and Slivkins (2022) for more details.

1.5.1 Explore Then Commit (ETC)

First algorithm we discuss is known as *Explore-then-Commit (in short ETC)* outlined in Algorithm 4. It is a very simple algorithm and one can say it is an obvious way to proceed to know about the best arm. The algorithm has two stages, in the first stage the learner will play all the arms for equal number of times, in the algorithm this is denoted with m and then observe the rewards. This means if $|\mathcal{W}| = K$, then the first stage runs for mK times. At the end of this stage, the learner then computes the sample means of the rewards of all arms at that time point, denoted with $\widehat{\mu}_{mK}(k)$. Then in the second stage starting from $mK + 1$, the learner will commit to playing the arm which has the highest sample mean and play this arm in all the remaining rounds.

Explore then commit algorithm has a very clear separation of exploration and exploitation phases. The exploration phase is the first mK rounds where the learner plays all arms equal number of times, and the exploitation phase is the rest of the rounds where the learner plays the arm with the highest sample mean.

1.5.2 Upper Confidence Bound (UCB)

Now we explain the Upper Confidence Bound (UCB) algorithm. The algorithm is outlined in Algorithm 5. The UCB is possibly one of the most well known algorithms in the stochastic bandit setting. The algorithm is very simple and intuitive. The idea is to play the arm with the highest upper confidence bound, or we say *"become optimistic in face of uncertainty"*. In particular, at each time t, for each arm k, the algorithm computes an upper confidence bound $\text{UCB}_{t-1}(k)$ based on the rewards observed from arm k up to time $t-1$.

Algorithm 5: Upper Confidence Bound (UCB)

Input: $\mathcal{W}$, T, $\delta > 0$

Initialize: $\widehat{\mu}_0(k) = 0$, $n_0(k) = 0$, $S_0(k) = 0 \; \forall k \in \mathcal{W}$

for $t = 1, 2, \ldots, T$ **do**

compute $\text{UCB}_{t-1}(k) = \begin{cases} \infty, & \text{if } n_{t-1}(k) = 0 \\ \widehat{\mu}_{t-1}(k) + \sqrt{\frac{2\log(1/\delta)}{n_{t-1}(k)}}, & \text{otherwise} \end{cases}, \qquad \forall k \in \mathcal{W}$

select $W_t = \text{argmax}_{k \in \mathcal{W}} \text{UCB}_{t-1}(k)$

observe reward $Y_t(W_t)$

update

$$S_t(W_t) = S_{t-1}(W_t) + Y_t(W_t)$$
$$n_t(W_t) = n_{t-1}(W_t) + 1$$
$$\widehat{\mu}_t(W_t) = \frac{S_t(W_t)}{n_t(W_t)}$$

Output: $W_t, Y_t \; \forall t \in [T]$

Then the algorithm selects the arm with the highest upper confidence bound. After observing the reward, the algorithm updates the cumulative rewards and counts and then also the sample mean, but only for the played arm. For the other arms, these values remain the same as in time $t-1$. Note that if an arm has not been played yet, i.e., $n_{t-1}(k) = 0$, then the upper confidence bound is set to infinity to ensure that each arm is played at least once. In practice, we first initialize the algorithm by playing each arm once, and then start the UCB selection from time $K+1$.

Now, how do we get the upper confidence bound? The idea is to use concentration inequalities. In particular, using the Gaussian concentration inequality[¶] that we have already used in the proof of Thompson sampling (see eq. (A2.6)) we get,

$$\mathbb{P}\left(\mid \widehat{\mu}_t(k) - \mu(k) \mid \leq \sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{n}} \right) \geq 1 - 2\delta$$

[¶] UCB is generally written for the sub-Gaussian random variables. A random variable Y is called sub-Gaussian with parameter σ^2, denoted with $Y \sim s\mathcal{G}(\sigma^2)$, if for all $\lambda > 0$, we have

$$\mathbb{E}\left[e^{\lambda(Y - \mathbb{E}[Y])}\right] \leq e^{\frac{\lambda^2\sigma^2}{2}}$$

Many well known distributions are sub-Gaussian, for example, Gaussian distribution with variance σ^2, Bernoulli distribution with parameter p (here $\sigma^2 = 1/4$), bounded distributions in $[a, b]$ (here $\sigma^2 = (b-a)^2/4$) etc. For more details on sub-Gaussian random variables we refer the reader to Lattimore and Szepesvári (2020).

where δ is the error probability. This helps to define the upper confidence bound of the mean $\mu(k)$ after time $t-1$ as

$$\text{UCB}_{t-1}(k) = \widehat{\mu}_{t-1}(a) + \sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{n_{t-1}(k)}}$$

With some early ideas, the original UCB has been formalized in the work of Auer, Cesa-Bianchi, and Fischer (2002). This is one of the seminal works in the stochastic bandit literature. In the original work, the authors proposed to set $\delta = \frac{1}{t}$. This is because as we play the arm more often with time, we want to be more confident about the mean estimate, thus we decrease the error probability δ with t.

1.6 Conclusion

The stochastic multi armed bandit is a very well studied problem in the multi armed bandit literature. In this chapter we tried to present a short but to somewhat deeper summary to the basic problem, discussed how this is connected to the potential outcome or potential reward framework, defined the key concepts like regret and then present one of the most popular algorithms, Thompson sampling with its Bayesian regret analysis for Gaussian rewards. The reason we went with more details with the Thompson sampling and not for other algorithms since in the next chapter we exclusively focus on Thompson sampling and extend it to the setting with interference, i.e., when arm pulled for one unit, can affect the rewards of other units. We briefly mentioned two popular algorithms at the end, Explore-then-Commit (ETC) and Upper Confidence Bound (UCB). There are many other algorithms for stochastic bandits and we refer the reader to Lattimore and Szepesvári (2020) and Slivkins (2022) for more details on different algorithms and their analyses.

Chapter 2.

Thompson Sampling in the Presence of Interference

(Unpublished)

In this article we explore the problem of *sequential treatment allocation*, also more widely known as *multi-armed bandit problem (MAB)* in the presence of interference. In a *sequential treatment allocation* problem a learner or a policy maker assigns treatments sequentially rather than in a batch. This is a well studied problem in the field of **Reinforcement Learning** (Sutton & Barto, 2018). The setup has many names, *multi armed bandit*, *sequential decision making* or *sequential treatment allocation* and also *adaptive experiments*. The word *adaptive or sequential* comes from the fact that the learner or the policy maker doesn't have access to all the units in the study at once, rather she receives the units *sequentially or online* and observes the outcomes *sequentially*. The goal of the learner is to assign the treatments such that it maximizes the cumulative *rewards* (or equivalently minimizes cumulative regret) over the horizon T. The MAB problem has a long history starting from the pioneering work of Thompson (1933) and then Robbins (1952). For a comprehensive survey we refer the reader to some excellent monographs, namely Lattimore and Szepesvári (2020), **?** and also Bubeck, Cesa-Bianchi, and Lugosi (2013) Recently some important contributions are also appearing in the field of economics and econometrics, e.g., see the works of Kasy and Sautmann (2021) and Kock, Preinerstorfer, and Veliyev (2023).

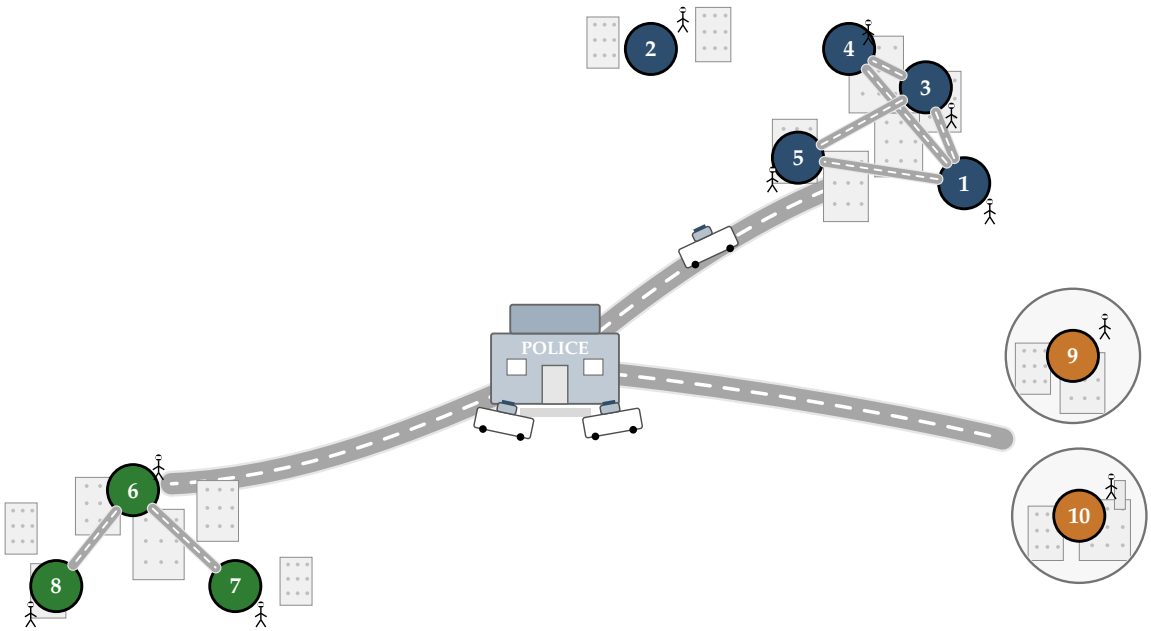


Figure 2.1: Central Police is trying to learn the optimal policing strategy to reduce crime in different neighborhoods of a city.

Recently there has been some interest in the field of causal inference and treatment effect estimation with the presence of *interference*, for example see Hudgens and Halloran (2008), Aronow and Samii (2017), Forastiere, Airoldi, and Mealli (2021) and Leung (2022) for some methodological works on interference. Dealing with interference is hard since interference makes the units dependent on each other. In particular, we say *unit i and j interfere with each other when the treatment assigned to unit j affects the outcome of unit i* (Cox, 1958). This is in violation of the well known *stable unit treatment value assumption* or SUTVA proposed in (Rubin, 1974, 1976, 1980) where the potential outcome of unit i only depends on treatment assigned to unit i, also see Imbens and Rubin (2015) for more details on SUTVA. In many causal inference or treatment effect estimation setup, the presence of interference has to be carefully dealt with, for example Sobel (2006) considered a study of housing mobility, where units in the poor areas are financed to relocate to better neighborhoods. The author argued - ignoring interference can lead to biased estimates and wrong conclusions, since units moving to different areas may influence the outcomes of all the units in the neighboring areas, not just for the units who move. In different studies, Blattman, Green, Ortega, and Tobón (2021) and Verbitsky-Savitz and Raudenbush (2012) investigated the effects of policing on crime, here also effect of interference is expected, since increasing policing in some areas may also displace the crime to the neighboring areas*.

The classical multi armed bandit setting almost always implicitly assume SUTVA or the *No Interference* assumption, i.e., the treatment of unit i at time t, denoted by $W_{i,t}$ only affects the outcome of unit i at time t. In this case the potential outcome is denoted by $Y_{i,t}(W_{i,t})$. Note that, this notation explicitly avoids interference. Although the interference in the causal inference literature has already advanced in many old settings both in experimental (Hudgens & Halloran, 2008; Leung, 2022) and in the observational studies (Forastiere et al., 2021; Papadogeorgou, Mealli, & Zigler, 2018; Lee et al., 2024), it is rather unclear how to deal with interference in adaptive experimentation settings like multi-armed bandits, where interference is definitely present in many practical scenarios. For example, a seller could be trying to learn the optimal price of a new product. She is aware that different prices might be beneficial since different customers have different willingness to pay. However, if the customers are socially connected, then a price offered to customer i may affect the purchasing decision of customer j. For example if they live in a same area and offered different prices then they might perceive the seller unfair seller, and question the product's quality. Another example, suppose a government agency is trying to promote an agricultural consulting service among rural farmers. It has different persuasion strategies, definitely different persuasion strategies may work for different farmers. However,

* We refer the reader to some works which we think provides important contribution in the interference literature; Sobel (2006); Hudgens and Halloran (2008); Tchetgen and VanderWeele (2012); Ugander, Karrer, Backstrom, and Kleinberg (2013); Cai, Janvry, and Sadoulet (2015); Eckles, Karrer, and Ugander (2017); Aronow and Samii (2017), Jagadeesan, Pillai, and Volfovsky (2017); Chin (2019); Forastiere et al. (2021), Lee, Zeng, and Hudgens (2024); Leung (2022),Leung (2023)

the agency must realize that, if the farmer receives the call, she may discuss it with her neighboring farmer j, and farmer j's decision to enroll may be affected by farmer i's decision or the strategy used for farmer i. And it is possible to give other examples where the experimenter or policymaker needs to carefully think about interference while designing adaptive experiments.

Our contributions are as follows. We develop a formal framework for stochastic multi-armed bandits with interference using the potential outcomes. In this setup the interference structure is modeled using the *stochastic counterfactuals*, where each counterfactual reward for a treatment combination is modeled as a random variable as opposed to fixed number (Robins & Greenland, 1989, 2000; VanderWeele & Robins, 2012; Pearl, 2009; Chin, 2019). In the special case, with no interference setup this becomes the regular multi-armed bandit setup but a batched version. We then propose three algorithms based on Thompson sampling, namely: Global One or $G-1$ learner, Clustered One or $C-1$ learner and Clustered Multi or $C-M$ learner. These learners or algorithms bring different ways to balance the exploration-exploitation trade-off in the presence of different interference structures. The $G-1$ learner finds the global single best action. The $C-1$ learner finds the same as $G-1$, however it explores and exploits based on the clustered interference structure. Finally the $C-M$ learner finds the best action in each cluster separately. We derive the posterior updates for all the three learners and provide the regret analysis for the $G-1$ learner. Finally, we perform some Monte Carlo experiments with different interference setups and compare the performance of three learners with the baseline Thompson sampling algorithm that ignores interference altogether, which we call IID learner.

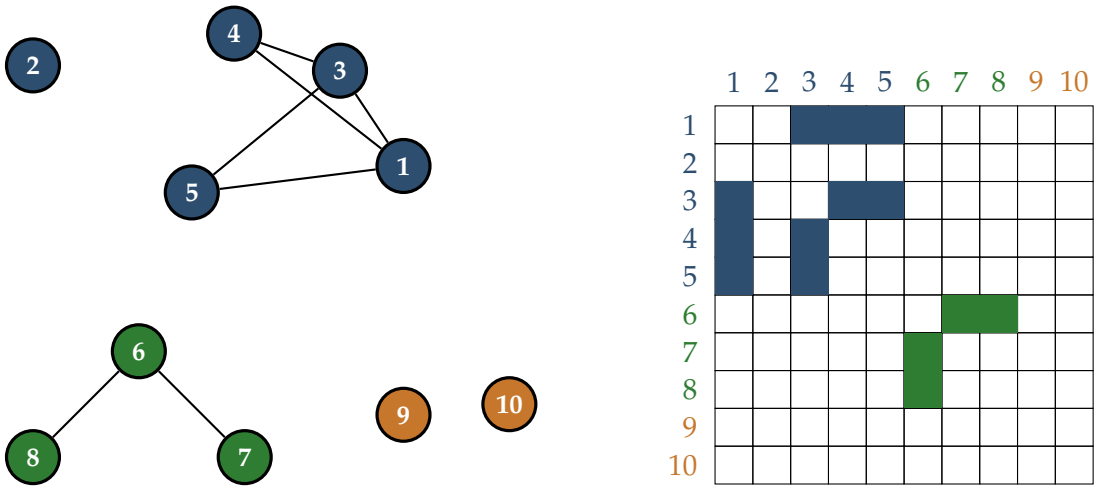
Bandit algorithms with interference is a relatively new area of research, with recent works going in different directions like adversarial bandits (Jia et al., 2025), contextual bandits, essentially regression based (Agarwal et al., 2024; Xu, Lu, & Song, 2024; Faruk & Zheleva, 2023) and also UCB-style algorithms (Su, Lu, & Song, 2019; Zhang & Wang, 2024). Our contribution is to bring classic Thompson sampling (Thompson, 1933) into this setting, and propose simpler and more direct extensions of Thompson sampling that are specifically designed to handle the combinatorial complexity of the action space. Perhaps the closest to our work is the work of Gleich et al. (2025). They also use Thompson sampling for networks with interference; however, their approach is quite different from ours. In particular, they assume, the adjacency matrix is known to the learner, and there is a degree based linear model of rewards, and then they learn the parameters using Thompson sampling based procedure. Whereas, our approach is more straightforward, focusing on simplifying the action space directly through structural assumptions about interference. The algorithms we propose don't require the knowledge of the adjacency matrix, and we believe computationally more scalable and very simple to implement.

This chapter is organized as follows. In Section 2.1, we present a motivating example of

policing strategies for crime reduction, which we will use as a running example to define our setup. In Section 2.2 we present the novel setup of stochastic potential rewards and stochastic bandit model with interference, and define the regret in this setting. In Section 2.3 we propose the three Thompson sampling based algorithms to deal with interference. In Section 2.4, we define the Bayesian bandit environment, Bayesian regret and finally give the Bayesian $G-1$ regret analysis for the $G-1$ learner in Gaussian rewards case. In Section 2.5, we present some Monte-Carlo experiments to compare the performance of the proposed algorithms under different interference setups. Finally, in Section 2.6, we conclude and discuss some future directions.

2.1 Motivating Example : Policing Strategies on Crime

Let's start with a motivating example of policing strategies on crime reduction. Suppose we have the following network of 10 nodes in three clusters as shown in Figure 2.2.



(a) Network representation. **(b)** Adjacency matrix.

Figure 2.2: The network and adjacency matrix for the motivating example. The network consisting of 10 nodes in three distinct clusters, represented by three colors: There is no inter-cluster connectivity; edges exist only within clusters.

Following the work of Blattman et al. (2021), we can contextualize this network in a **policing on crime** scenario. Suppose the central police department in a city faces the challenge of reducing crime across various neighborhoods in a city. In this story, we can assume the nodes represent small neighborhoods in this city and the edges represent good transportation between them. Since the city is large, we can think about different clusters, in particular in the network we have three clusters, cluster 1 - blue nodes (1–5), cluster 2 - green nodes (6–8), and cluster 3 - orange nodes (9–10). Each cluster may represent a separate large area, like districts within a city. So the story goes like this, the central police

Arm / Action k	**Strategy** k	**Direct Effect** (β_k)	**Spillover Effect** (δ_k)
1	Intense Internal	-5.2% | $\downarrow\downarrow\downarrow$	6.2% | $\uparrow\uparrow\uparrow$
2	Moderate Internal	-3.0% | $\downarrow\downarrow$	2.3% | $\uparrow\uparrow$
3	Strong Border	-1.0% | $\downarrow$	0.0

Table 2.1: Policing strategies with their direct and spillover effects. The arrows indicate the magnitude and direction of the effects when strategy k is implemented in neighborhood i. For example, if strategy 1 is implemented in neighborhood i, then neighborhood i experiences strongest decrease in crime (due to direct effect), however at the same time neighboring area $j \neq i$ also experience a strongest increase in crime (due to spillover effect).

is responsible for all of these neighborhoods, and currently thinking about different **policing strategies**; *1) Intense Internal*, *2) Moderate Internal*, and *3) Strong Border*. We denote these policing strategies as the three actions, $\mathcal{W} = \{1, 2, 3\}$ or strategies or arms. These three strategies differ in terms of *direct* and *interference or spillover* effects. We explain these strategies in detail below.

Intense Internal strategy: If the police department implements this strategy in neighborhood i, then this means the police focuses on intense patrolling inside i, uses significant resources inside the neighborhood and as a result border becomes weak, allowing criminals to escape to neighboring areas more easily. So this strategy aims to aggressively reduce crime within the neighborhood i, but simultaneously brings strong negative spillover effects to neighboring areas $j \neq i$.

Moderate Internal strategy: In this strategy, the police maintains a balanced approach, go for a moderate presence inside neighborhood i while also keeping a watchful eye on the borders. This strategy aims to strike a balance between internal policing and border control.

Strong Border strategy: This is the last option, which focuses heavily on securing the borders of neighborhood i. The police recognizes that criminals may attempt to escape to neighboring areas, so they prioritize strong border control measures. However, this emphasis on border security may lead to a moderate or relatively weak internal policing presence within neighborhood i, since resources are diverted to border control. Consequently, while the direct effect of this strategy is the lowest in terms of reducing crime within neighborhood i, it yields the zero spillover effects for neighboring areas.

At the beginning of each week t, the central police decides a set of policing strategies $W_{1,t}, W_{2,t}, \ldots, W_{10,t}$ for all 10 neighborhoods, where for all i, $W_{i,t} \in \mathcal{W}$. After a week of implementing these strategies, the police observes the change in crime in the neighborhoods,

denoted with $Y_{1,t}, Y_{2,t}, \ldots, Y_{10,t}$, learns which strategies are effective in each neighborhood, and uses this knowledge to implement strategies at $t+1$. The goal of the police department is to learn an optimal policing strategy combination that minimizes the overall crime rate across all neighborhoods for time periods $t = 1, 2, \ldots, T$, taking into account both direct effects and spillover effects due to interference.

A simple interference model that we can use in this setting is given in eq. (2.1), known as the *linear-in-means* model (Manski, 1993). According to this model, at time t, if the police department implements a strategy vector $\boldsymbol{W}_t = (W_{1,t}, W_{2,t}, \ldots, W_{10,t})$ across all neighborhoods then we can write

$$\begin{aligned} Y_{i,t} = \alpha_i &+ \beta_1 \cdot \mathbb{1}\{W_{i,t} = 1\} + \beta_2 \cdot \mathbb{1}\{W_{i,t} = 2\} + \beta_3 \cdot \mathbb{1}\{W_{i,t} = 3\} \\ &+ \delta_1 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 1\} + \delta_2 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 2\} + \delta_3 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 3\} + \epsilon_{i,t} \end{aligned} \tag{2.1}$$

Where $Y_{i,t}$ is the percentage change of crime in neighborhood i after period t, α_i is the baseline percentage change in crime in the absence of any policy intervention, A_{ij} denotes the entries of row-normalized adjacency matrix denoted with $\mathbf{A}$. The entries here are fraction of connections and the adjacency matrix in its binary form is shown in Figure 2.2†. We are assuming there is no self loop, so $A_{ii} = 0$. Then the term $\sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = k\}$ is the fraction of neighborhoods of i that are assigned strategy k at time t. The coefficient β_k is the direct effect of implementing strategy k on unit i, δ_k is the spillover effect of strategy k that comes from the connected neighborhoods applying strategy k, $\epsilon_{i,t}$ is an error term capturing unobserved factors affecting the percentage change in crime in neighborhood i at time t, and we assume $\epsilon_{i,t}$ are i.i.d. across i and t with mean zero and variance σ^2.

We give an example of some values for direct effect β_k and δ_k to reflect the characteristics of three policing strategies that we have explained in Table 2.1. To interpret these parameters: $\beta_1 = -5.2$ indicates that, if the Intense Internal strategy is implemented in neighborhood i, then it reduces crime in neighborhood i by 5.2%. However, at the same time if for any of the neighboring areas $j \neq i$, the police department also implements the Intense Internal strategy, then from j, the neighborhood i experiences a 6.2% increase in crime due to displacement effects as criminals would escape to neighboring areas for weak border control. So it has high direct effect but also high negative spillover effect‡.

† If we denote the adjacency matrix shown in Figure 2.2 with $\mathbf{G}$, where $G_{ij} = 1$ if there is an edge between node i and j, and 0 otherwise. Let the degree of node i be $d_i = \sum_{j=1}^{N} G_{ij}$, then the row-normalized adjacency matrix $\mathbf{A}$ is defined with $A_{ij} = \frac{G_{ij}}{d_i}$ if $d_i > 0$, and $A_{ij} = 0$ if $d_i = 0$.

‡ In this case positive spillover means increasing reward or reducing crime and negative spillover means reducing reward or increasing crime for unit i

Similarly, for the moderate internal strategy ($\beta_2 = -3.0$, $\delta_2 = 2.3$) gives a moderate 3% crime reduction, and it has smaller negative spillovers of 2.3%. Finally, the Strong Border strategy ($\beta_3 = -1.0, \delta_3 = 0$) has the weakest direct effect (1% reduction) but prevents crime displacement entirely with zero spillover through stronger border presence.

If the police department implements the strategy vector $\boldsymbol{W}_t$ at time t, then it can only observe one possible realities of outcome and doesn't observe what would happen at the same time if it had implemented a different strategy vector $\boldsymbol{W}_t' \neq \boldsymbol{W}_t$. This is the key idea of potential rewards in this setting. Next, we formalize this setup in detail.

2.2 Definitions and Protocols in the Interference Setting

In this section, we define the setup of stochastic multi-armed bandit model with interference. Recently Agarwal et al. (2024) and Gleich et al. (2025), uses similar setup for stochastic bandits with interference, however we carefully state the full details of the model and definitions, which the authors of the other work didn't provide[§]. We consider a fixed batch $\mathcal{I}$ of N nodes, where a learner interacts with the batch repeatedly over the horizon T, assigns a treatment vector to all the nodes at each time t, observe the rewards generated from the environment at the end of t, updates her strategy and adaptively finds the optimal arm vector for the entire batch within the horizon T.

We start with the definition of the stochastic potential rewards with interference, we then define the stochastic bandit model in this setting, and finally specify learner's goal, policy and different notions of regret under different optimal scenarios. In each section, after defining some components, we also connect the definitions with the motivating example that we gave in Section 2.1.

2.2.1 Stochastic Potential Rewards with Interference

Let $\mathcal{I} = \{1, 2, \ldots, N\}$ be a fixed set of N nodes in the batch and $\mathcal{W}$ be the set of arms or actions (or treatments) available to the learner[¶]. Often we can index the arms as $1, 2, \ldots, K$ and can write, $\mathcal{W} = [K] = \{1, 2, \ldots, K\}$. The set of all possible arm combinations or arm vectors for N nodes can be written as $\mathcal{W}^N$, where for K arms we have $|\mathcal{W}^N| = K^N$, i.e., there are K^N possible arm vectors of length N. These are all possible arm combinations or arm vectors for all N nodes in the batch $\mathcal{I}$. In this setting there are several possible distributions, we now define them carefully. For any fixed arm vector $\boldsymbol{k} = (k_1, k_2, \ldots, k_N) \in \mathcal{W}^N$, let $\mathrm{F}_{\boldsymbol{k}}$ denote a joint probability distribution of potential reward vector $\boldsymbol{Y}_t(\boldsymbol{k}) = (Y_{1,t}(\boldsymbol{k}), Y_{2,t}(\boldsymbol{k}), \ldots, Y_{N,t}(\boldsymbol{k}))' \in \mathbb{R}^N$, i.e., $\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathrm{F}_{\boldsymbol{k}}$. Here, $Y_{i,t}(\boldsymbol{k})$ is

§ Our work has been independently developed and after developing the initial framework we came to know about these works.

¶ We also call arms, the terminology came from slot machine analogy. In this work we use the words, treatments, arms or actions interchangeably.

the potential reward for node i at time t when the arm vector $\boldsymbol{k}$ is assigned to all nodes in the batch. The joint distribution $\mathrm{F}_{\boldsymbol{k}}$ captures the dependence structure among the potential rewards of all nodes under arm vector $\boldsymbol{k}$. The marginal distributions for node i in this case is denoted with $\mathrm{F}_{i,\boldsymbol{k}}$, i.e., $Y_{i,t}(\boldsymbol{k}) \sim \mathrm{F}_{i,\boldsymbol{k}}$, where we can let $\mu_i(\boldsymbol{k}) = \mathbb{E}\left[Y_{i,t}(\boldsymbol{k})\right]$ and $\sigma_i^2(\boldsymbol{k}) := \mathbb{V}\left(Y_{i,t}(\boldsymbol{k})\right)$ be the marginal mean and variance of the potential reward $Y_{i,t}(\boldsymbol{k})$. Also, for the joint distribution $\mathrm{F}_{\boldsymbol{k}}$, we let $\boldsymbol{\mu}(\boldsymbol{k}) = (\mu_1(\boldsymbol{k}), \mu_2(\boldsymbol{k}), \ldots, \mu_N(\boldsymbol{k}))'$ and $\boldsymbol{\Omega}(\boldsymbol{k}) := \mathbb{C}\mathrm{ov}\left(\boldsymbol{Y}_t(\boldsymbol{k})\right)$ to be the mean vector and variance-covariance matrix of the potential reward vector $\boldsymbol{Y}_t(\boldsymbol{k})$ under arm vector $\boldsymbol{k}$. Note that, the potential reward $Y_{i,t}(\boldsymbol{k})$ for node i at time t depends on the entire arm vector $\boldsymbol{k}$, not just the arm assigned to node i, this captures the interference or spillover effects at the distributional level (we discuss this in detail below). The mean function $\mu_i(\boldsymbol{k})$ maps from the space of all arm vectors to the range of mean rewards $M \subset \mathbb{R}$, i.e., $\mu_i : \mathcal{W}^N \to \mathcal{M} \subset \mathbb{R}$.

All of the distribution so far is for a fixed arm vector $\boldsymbol{k}$. However, in this setting we have K^N possible arm vectors in the set $\mathcal{W}^N$. So for each arm vector $\boldsymbol{k} \in \mathcal{W}^N$, there is a corresponding joint distribution $\mathrm{F}_{\boldsymbol{k}}$ for the potential reward vector $\boldsymbol{Y}_t(\boldsymbol{k})$. Therefore, we have a set of joint distributions for all possible arm vectors, denoted with $\mathscr{F} := \{\mathrm{F}_{\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{W}^N\}$. If we fix node i, then the set of all marginal distributions for node i for all possible arm vectors is denoted by $\mathscr{F}_i := \{\mathrm{F}_{i,\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{W}^N\}$. Following is the formal definition of the stochastic potential rewards with interference.

Definition 2.1 (Stochastic Potential Rewards with Interference). *For a batch of nodes $\mathcal{I} = \{1, 2, \ldots, N\}$, action set $\mathcal{W}$, define the set of all stochastic potential reward distributions with interference with $\mathscr{F} := \{\mathrm{F}_{\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{W}^N\}$, where $\mathcal{W}^N$ is the set of all possible arm vectors of length N. At time t, when arm vector $\boldsymbol{W}_t \in \mathcal{W}^N$ is selected, only one random draw of the potential reward vector $\boldsymbol{Y}_t(\boldsymbol{W}_t) = (Y_{1,t}(\boldsymbol{W}_t), Y_{2,t}(\boldsymbol{W}_t), \ldots, Y_{N,t}(\boldsymbol{W}_t))' \sim \mathrm{F}_{\boldsymbol{W}_t}$ is observed, the potential rewards under any other arm vectors $\boldsymbol{k} \neq \boldsymbol{W}_t$ are unobserved. The potential reward vectors $\{\boldsymbol{Y}_t(\boldsymbol{k})\}_{t=1}^T$ are independently and identically distributed across time periods, i.e., $\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathrm{F}_{\boldsymbol{k}}$ for all t.*

We give a concrete example to illustrate the idea of stochastic potential rewards with interference using the running example in Section 2.1.

Example 2.2 (Example of Potential Rewards with Policing on Crime Reduction).

Let's explain the idea of stochastic potential rewards with the running example of policing strategies on crime reduction, that we discussed in Section 2.1. We recall, we have a batch of neighborhoods (or nodes) $\mathcal{I} = \{1, 2, \ldots, 10\}$ and three policing strategies (or arms or actions or treatments) $\mathcal{W} = \{1, 2, 3\}$ shown in Table 2.1. This gives the set $\mathcal{W}^{10} = \{1, 2, 3\}^{10}$, which contains $|\mathcal{W}^N| = 3^{10} = 59,049$ different arm combinations or vectors for the whole batch. For any arm vector $\boldsymbol{k} \in \mathcal{W}^{10}$, the potential reward vector for the whole batch at time t is, $\boldsymbol{Y}_t(\boldsymbol{k}) = (Y_{1,t}(\boldsymbol{k}), Y_{2,t}(\boldsymbol{k}), \ldots, Y_{10,t}(\boldsymbol{k}))'$, where $Y_{i,t}(\boldsymbol{k})$ is the

potential percentage change of crime in neighborhood i at time t under arm vector $\boldsymbol{k}$. In this case the set of all joint distributions of the potential reward vectors for all possible arm vectors is $\mathscr{F} = \{\mathrm{F}_{\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{W}^{10}\}$, which is actually a set of 59,049 joint distributions.

At each time t, the police department selects an arm vector $\boldsymbol{W}_t \in \mathcal{W}^{10}$ and implements the chosen vector of strategies across all neighborhoods. For example, if at time t the police decides to implement $\boldsymbol{W}_t = (1, 2, 1, 3, 2, 1, 2, 3, 1, 2)$, meaning neighborhood 1 gets Intense Internal strategy, neighborhood 2 gets Moderate Internal strategy, neighborhood 4 gets Strong Border strategy, and so on. Then this chosen vector $\boldsymbol{W}_t$ induces a joint potential reward distribution $\mathrm{F}_{\boldsymbol{W}_t}$ for the entire batch. The police department then observes $\boldsymbol{Y}_t(\boldsymbol{W}_t) = (Y_{1,t}(\boldsymbol{W}_t), Y_{2,t}(\boldsymbol{W}_t), \ldots, Y_{10,t}(\boldsymbol{W}_t))' \sim \mathrm{F}_{\boldsymbol{W}_t}$. This is a random draw from the joint distribution $\mathrm{F}_{\boldsymbol{W}_t}$, capturing the percentage change in crime in all neighborhoods under the selected policing strategy vector $\boldsymbol{W}_t$. The potential rewards under any other arm vectors $\boldsymbol{k} \neq \boldsymbol{W}_t$ are unobserved.

If we would like to be more specific regarding the class of distributions, we can assume for any $\boldsymbol{k} \in \mathcal{W}^N$ the joint distribution $\mathrm{F}_{\boldsymbol{k}}$ is a multivariate Gaussian distribution with the mean vector $\mu(\boldsymbol{k}) = (\mu_1(\boldsymbol{k}), \mu_2(\boldsymbol{k}), \ldots, \mu_{10}(\boldsymbol{k}))'$ and variance-covariance matrix $\boldsymbol{\Omega}(\boldsymbol{k})$. Then the potential reward vector at time t under arm vector $\boldsymbol{k}$ follows the joint distribution,

$$\boldsymbol{Y}_t(\boldsymbol{k}) = (Y_{1,t}(\boldsymbol{k}), Y_{2,t}(\boldsymbol{k}), \ldots, Y_{10,t}(\boldsymbol{k}))' \sim \mathcal{N}(\mu(\boldsymbol{k}), \boldsymbol{\Omega}(\boldsymbol{k})) \tag{2.2}$$

This is only for one arm vector $\boldsymbol{k}$, and we have 59,049 such joint distributions for all possible arm vectors in $\mathcal{W}^{10}$. Also in this case the marginal distribution of $Y_{i,t}(\boldsymbol{k})$ is univariate Gaussian: $Y_{i,t}(\boldsymbol{k}) \sim \mathcal{N}(\mu_i(\boldsymbol{k}), \sigma_i^2(\boldsymbol{k}))$, where $\sigma_i^2(\boldsymbol{k})$ is the i-th diagonal element of $\boldsymbol{\Omega}(\boldsymbol{k})$. Here $\mu_i(\boldsymbol{k})$ captures the expected percentage change in crime in neighborhood i under arm vector $\boldsymbol{k}$, accounting for both direct and spillover effects from other neighborhoods' strategies, and $\sigma_i^2(\boldsymbol{k})$ captures the uncertainty in this outcome and this also depends on the full arm vector $\boldsymbol{k}$ due to interference effects.

We already specified the reward $Y_{i,t}$ in eq. (2.1). So following this, here the mean function $\mu_i(\cdot)$ for neighborhood i under random arm vector $\boldsymbol{W}_t = (W_{1,t}, W_{2,t}, \ldots, W_{10,t})$ can be written as follows,

$$\begin{aligned}\mu_i(\boldsymbol{W}_t) = {} & \alpha_i + \beta_1 \cdot \mathbb{1}\{W_{i,t} = 1\} + \beta_2 \cdot \mathbb{1}\{W_{i,t} = 2\} + \beta_3 \cdot \mathbb{1}\{W_{i,t} = 3\} \\ & + \delta_1 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 1\} + \delta_2 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 2\} + \delta_3 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 3\}\end{aligned} \tag{2.3}$$

If the interference or spillover only influences the mean function, but not the variance-covariance structure, then we can assume for any arm vector $\boldsymbol{k} \in \mathcal{W}^{10}$, the variance-covariance matrix is diagonal, i.e., $\boldsymbol{\Omega}(\boldsymbol{k}) = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_N^2)$, where σ_i^2 is fixed for all arm vectors $\boldsymbol{k}$. This means that the uncertainty in each node's reward is not influenced by interference, only the mean is affected. This assumption may be reasonable in some applications. In this case for an arm vector $\boldsymbol{k} \in \mathcal{W}^{10}$, the full model can be specified as

$$Y_{i,t}(\boldsymbol{k}) = \mu_i(\boldsymbol{k}) + \epsilon_{i,t}, \quad \epsilon_{i,t} \sim \mathcal{N}(0, \sigma_i^2) \quad \text{and } \epsilon_{i,t} \perp \epsilon_{j,t} \text{ for } i \neq j \quad \forall t$$

///

There is an important difference between our stochastic potential reward (or outcome) setup and the deterministic potential outcomes, primarily developed without interference by Rubin (1974, 1976, 1980) also see Imbens and Rubin (2015), and then with interference in Sobel (2006); Hudgens and Halloran (2008); Tchetgen and VanderWeele (2012) and Forastiere et al. (2021). In Rubin's framework, for an arm set $\mathcal{W} = [K]$, each unit i has K fixed potential outcomes $\{y_i(k)\}_{k \in \mathcal{W}}$. For a fixed unit, these outcomes are deterministic and maybe determined by unit's observable or unobservable characteristics. Then randomness in outcome Y_i enters in two stages, first through sampling from the population and then treatment assignment W_i. So we write the observed outcome $Y_i = Y_i(W_i)$. The same idea has been extended to interference, in this case for K arms and N units, for each i we have now K^N fixed potential outcomes $\{y_i(\boldsymbol{k})\}_{\boldsymbol{k} \in \mathcal{W}^N}$. Then the observed outcome is $Y_i = Y_i(\boldsymbol{W})$, where $\boldsymbol{W}$ is the random arm vector of treatment assignments to all N units. Again in this setup Y_i has two levels of randomness, first from sampling and second from arm vector $\boldsymbol{W}$. In both cases, with or without interference, what's important is, once we fix unit i, the potential outcomes are fixed or deterministic.

In the stochastic framework, we have a third level of randomness and that is at the unit level, which can be explained through the intrinsic randomness of the units (i.e., in repeated experiments same unit may react differently) and this is discussed in Robins and Greenland (1989, 2000); VanderWeele and Robins (2012); Pearl (2009). Without interference for each unit i, in this setup, in population, we have a distribution of potential outcomes $\{\text{F}_{i,k}\}_{k \in \mathcal{W}}$ and the randomness in observed outcome Y_i can come from three sources. First two are same as the deterministic case, first from sampling, and second from treatment assignment W_i. The third source however is, from the unit level randomness $\text{F}_{i,k}$, and we observe one realization from this distribution when treatment $k \in \mathcal{W}$ is assigned to unit i, i.e., $Y_i(k) \sim \text{F}_{i,k}$. So in this setup even if the same unit i is assigned the same treatment k multiple times, the observed outcome $Y_i(k)$ may differ each time due to the intrinsic randomness captured by $\text{F}_{i,k}$. This is very different from the deterministic potential outcome setup, where for a fixed unit i and treatment k, the potential outcome $y_i(k)$ is fixed or de-

terministic. We borrow the same idea in the interference setting in bandits. Here for each unit i we have a set of K^N potential outcome distributions $\{\mathrm{F}_{i,\boldsymbol{k}}\}_{\boldsymbol{k}\in\mathcal{W}^N}$, where $\boldsymbol{k}$ is the arm vector for all N units. Then the observed outcome $Y_i = Y_i(\boldsymbol{W}_i)$, where for fixed unit i and arm vector $\boldsymbol{k}$, unit level randomness is explained via $\mathrm{F}_{i,\boldsymbol{k}}$.

So far, the interference is assumed to hold across all nodes in the batch $\mathcal{I}$, this is often called the global interference. However, in many applications, including our running example in Section 2.1, interference is present locally, where the effects are limited to certain groups based on some common characteristics, e.g., geographical proximity, social connections, or other relevant factors. In particular, in our running example, in Figure 2.2, although we defined interference with a complete adjacency matrix $\mathbf{A}$, but in fact the interference exists only in clusters, note that edges exist only within clusters. This is known as *clustered interference* structure (Forastiere et al., 2021). Roughly, this means the spillover or interference exists only among nodes within the same clusters. What benefit does it bring? Think about the case where we have no knowledge of the network or adjacency matrix. In this case, for each node i, if we would like to know which arm is best, we need to consider $3^{10} = 59,049$ combinations. But now with clustered interference, even if we don't know the network, and only know that cluster 1 has only 5 nodes, then for any node in this cluster we only need to consider only $3^5 = 243$ combinations. For cluster 2, we need to consider $3^3 = 27$ combinations, and for cluster 3 only $3^2 = 9$ combinations. Which gives in total $3^5 + 3^3 + 3^2 = 243 + 27 + 9 = 279$ combinations. This is significantly smaller. At the distributional level, we can define the clustered interference structure for stochastic potential rewards with interference as follows.

Definition 2.3 (CLUSTERED INTERFERENCE IN STOCHASTIC POTENTIAL REWARDS)**.** *Assume that the nodes in $\mathcal{I}$ can be partitioned into a fixed set of M disjoint clusters $\mathscr{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M\}$ such that $\bigcup_{m=1}^{M} \mathcal{C}_m = \mathcal{I}$ and $\mathcal{C}_m \cap \mathcal{C}_{m'} = \emptyset$ for all $m \neq m'$. Then the set of stochastic potential rewards with interference $\mathscr{F} := \{\mathrm{F}_{\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{W}^N\}$ is said to have clustered interference structure if for any node $i \in \mathcal{C}_m$,*

$$\mathrm{F}_{i,(\boldsymbol{k}_{\mathcal{C}_m}, \boldsymbol{k}_{-\mathcal{C}_m})} \overset{d}{=} \mathrm{F}_{i,(\boldsymbol{k}_{\mathcal{C}_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m})} \quad \text{for all } \boldsymbol{k}_{\mathcal{C}_m} \in \mathcal{W}^{N_m},\ \boldsymbol{k}_{-\mathcal{C}_m},\ \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m} \in \mathcal{W}^{N-N_m} \tag{2.4}$$

where $N_m = |\mathcal{C}_m|$ is the number of nodes in cluster $\mathcal{C}_m$, $\boldsymbol{k}_{\mathcal{C}_m}$ is any arm vector for nodes belong to cluster $\mathcal{C}_m$, $\boldsymbol{k}_{-\mathcal{C}_m}$ and $\widetilde{\boldsymbol{k}}_{-\mathcal{C}_m}$ any two different arm vectors for nodes not belong to cluster $\mathcal{C}_m$ and $\overset{d}{=}$ denotes equality in distribution.

This says that for any node i in cluster $\mathcal{C}_m$, its reward distribution is unaffected by the arms assigned to nodes outside the cluster $\mathcal{C}_m$. Only the arms assigned to nodes within the same cluster $\mathcal{C}_m$ can influence the reward distribution of node i. Clustered interference structure will play significant role in two of our learners we explain in Section 2.3. We give further remarks on Definition 2.1 and Definition 2.3 below.

Remarks 2.4 (Remarks on Stochastic Potential Rewards with Interference)**.**

i) Only Mean Level Interference: Although we defined stochastic potential rewards with interference in full generality, in this chapter we will primarily focus on the cases where the interference only affects the mean function, and not any other parts of the distribution. The reason is, our bandit algorithms are primarily designed to maximize reward means. It is possible to incorporate variance to get more risk-aware policies, but we leave that for future work. If the interference only affects the mean function, then for a fixed arm vector $\boldsymbol{k}$, the marginal distributions for any node i can be written as $\mathrm{F}_{i,\boldsymbol{k}} = \mathrm{F}_{\mu_i(\boldsymbol{k})}$, where $\mathrm{F}_{\mu_i(\boldsymbol{k})}$ could be a parametric distribution indexed by the mean for node i, $\mu_i(\boldsymbol{k})$. Then for any arm vector $\boldsymbol{k} \in \mathcal{W}^N$, we can write the joint distribution of all N units by $\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})}$, i.e., $\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})}$, where $\boldsymbol{\mu}(\boldsymbol{k}) = (\mu_1(\boldsymbol{k}), \mu_2(\boldsymbol{k}), \ldots, \mu_N(\boldsymbol{k}))'$ is the mean vector of the joint distribution.

In this case the set of all potential reward distributions for all possible arm vectors can be written as, $\mathscr{F} = \mathscr{F}_{\boldsymbol{\mu}} := \{\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$. Similarly, for any node i, the set of all marginal distributions can be written as $\mathscr{F}_{\mu_i} = \{\mathrm{F}_{\mu_i(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$, where the mean function is $\mu_i(\boldsymbol{k})$.

ii) Independent Arm Vectors, But Dependent Nodes: Another simplification that we will follow is, for any t, we assume

$$Y_{i,t}(\boldsymbol{k}) \perp Y_{i,t}(\widetilde{\boldsymbol{k}}) \quad \text{for all } \boldsymbol{k} \neq \widetilde{\boldsymbol{k}} \in \mathcal{W}^N \text{ and } \forall i \in \mathcal{I}$$

This means that there is no dependency between the two distributions $\mathrm{F}_{\mu_i(\boldsymbol{k})}$ and $\mathrm{F}_{\mu_i(\widetilde{\boldsymbol{k}})}$. This is the extension of the usual independent arm assumption in the classic stochastic bandit literature. Also Robins and Greenland (1989, 2000) argued that in general when we observe $\boldsymbol{k}$, we don't observe $\widetilde{\boldsymbol{k}}$, so it is perhaps natural to assume they are independent. However, for a fixed arm vector $\boldsymbol{k}$, the node level distributions denoted by $Y_{i,t}(\boldsymbol{k})$ and $Y_{j,t}(\boldsymbol{k})$, for all $i, j \in \mathcal{I}$ may be dependent since their means depend on the full arm vector $\boldsymbol{k}$, this is actually the idea of interference effects.

(iii) Clustered Interference Structure: If we assume we have only mean level interference, i.e., interference effects are captured only through the mean functions $\mu_i(\cdot)$, then in this case for all i, we have

$$\mathrm{F}_{\mu_i\left(\boldsymbol{k}_{\mathcal{C}_m}, \boldsymbol{k}_{-\mathcal{C}_m}\right)} \stackrel{d}{=} \mathrm{F}_{\mu_i\left(\boldsymbol{k}_{\mathcal{C}_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m}\right)}, \text{ for all } \boldsymbol{k}_{-\mathcal{C}_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m} \in \mathcal{W}^{N-N_m} \tag{2.5}$$

and in particular for any $\mathcal{C}_m \in \mathscr{C}$, the means satisfy,

$$\mu_i\left(\boldsymbol{k}_{\mathcal{C}_m}, \boldsymbol{k}_{-\mathcal{C}_m}\right) = \mu_i\left(\boldsymbol{k}_{\mathcal{C}_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m}\right) \text{ for all } \boldsymbol{k}_{-\mathcal{C}_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m} \in \mathcal{W}^{N-N_m} \tag{2.6}$$

Which means that for any node i in cluster $\mathcal{C}_m$, its mean reward only depends on the

arms assigned to nodes within the same cluster $\mathcal{C}_m$, and not on the arms assigned to nodes outside the cluster. In this case, we can write the mean function for node $i \in \mathcal{C}_m$ as

$$\mu_i(\boldsymbol{W}_t) := \alpha_i + \beta_1 \cdot \mathbb{1}\{W_{i,t} = 1\} + \beta_2 \cdot \mathbb{1}\{W_{i,t} = 2\} + \beta_3 \cdot \mathbb{1}\{W_{i,t} = 3\} + \delta_1 \sum_{j \in \mathcal{C}_m} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 1\} + \delta_2 \sum_{j \in \mathcal{C}_m} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 2\} + \delta_3 \sum_{j \in \mathcal{C}_m} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 3\}$$

We will assume this structure in some of the algorithms we propose in Section 2.3.

///

2.2.2 Stochastic Bandit Model with Interference

Now with the definition of the stochastic potential rewards with interference, we can define the stochastic bandit model with interference. The definition is similar to the stochastic bandit model without interference in Definition 1.3. However here we have interference or spillover effects, so the reward for each node depends on the full arm vector assigned to all nodes in the batch. The formal definition is given below. In the following we assume that the interference effects are captured through the mean functions only, so we work with $\mathscr{F}_{\boldsymbol{\mu}} = \{\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$.

Definition 2.5 (Stochastic Bandit Model With Interference)**.** *A stochastic bandit model with interference is a repeated interaction between a learner and an environment where for a batch $\mathcal{I}$ of N nodes,*

for $t = 1, 2, \ldots, T$ **do**

the learner selects an arm vector $\boldsymbol{W}_t \in \mathcal{W}^N$ for the batch

the environment generates potential reward vector $\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})}$ for all $\boldsymbol{k} \in \mathcal{W}^N$

the learner observes only the realized outcome vector $\boldsymbol{Y}_t = \boldsymbol{Y}_t(\boldsymbol{W}_t)$, and not any observations from the counterfactual distribution $\mathrm{F}_{\boldsymbol{\mu}(\widetilde{\boldsymbol{k}})}$ for $\widetilde{\boldsymbol{k}} \neq \boldsymbol{W}_t$.

Interaction Protocol 3: *Stochastic Bandit With Interference*

We have the following information structure,

- ***Known or Observable to the Learner:*** *The learner knows the set of arm vectors $\mathcal{W}^N$, time horizon T and observes all actions and outcome vectors till time T, i.e., $\{(\boldsymbol{W}_t, \boldsymbol{Y}_t)\}_{t=1}^T$*
- ***Known to the Environment:*** *The environment knows the set of all counterfactual joint distributions $\mathscr{F}_{\boldsymbol{\mu}} = \left\{\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\right\}$, and observes what the learner observes.*

In summary, at each time point t, the learner picks an arm vector $\boldsymbol{W}_t = (W_{1,t}, \ldots, W_{N,t})$ and assigns these arms to all nodes in the batch $\mathcal{I}$. The environment then generates rewards from $\mathscr{F}_{\boldsymbol{\mu}}$, which is unknown to the learner. The learner observes the reward vector $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{N,t})'$, where $\boldsymbol{Y}_t = \boldsymbol{Y}_t(\boldsymbol{W}_t) \sim \mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{W}_t)}$.

In the Interference setting when the learner assigns an arm vector to a batch, and learns regarding the outcomes, there are different approaches she can take. The most exhaustive approach is when she tries all possible arm vectors $\boldsymbol{k} \in \mathcal{W}^N$. But this creates an enormous arm vector space. In particular if $\mathcal{W} = [K]$, then this is of size $|\mathcal{W}^N| = K^N$, and this grows exponentially with batch size N. For example, just for three arms, $K = 3$ and 10 nodes in our running example in Section 2.1, full space contains $3^{10} = 59,049$ distinct arm vectors. This makes the problem of learning the optimal arm vector intractable even for moderate batch sizes.

However, in different situations, it is possible to restrict attention to some special arm vectors that could give us near optimal (or sometimes even optimal) performance while dramatically reducing the action space. We introduce two such restricted classes: **global one, in short** $G-1$ arm vectors, where all nodes receive the same treatment, and **Cluster One, in short** $C-1$ arm vectors, where treatments are uniform within clusters but may vary across clusters. We give the definitions below,

Definition 2.6 ($G-1$ & $C-1$ ARM VECTORS)**.** *Consider a batch $\mathcal{I} = \{1, 2, \ldots, N\}$ of nodes and action set $\mathcal{W}$, and the batch $\mathcal{I}$ admits a partition into M disjoint clusters $\mathscr{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M\}$ such that $\bigcup_{m=1}^{M} \mathcal{C}_m = \mathcal{I}$ and $\mathcal{C}_m \cap \mathcal{C}_{m'} = \emptyset$ for all $m \neq m'$, then we can define the following $G-1$ and $C-1$ arm vectors,*

- $G-1$ ***arm vector:*** *An arm vector $\boldsymbol{k} \in \mathcal{W}^N$ is called a* global one *or $G-1$* arm vector *if all nodes receive the same arm, i.e., there exists $k \in \mathcal{W}$ such that $\boldsymbol{k} = k \cdot \mathbf{1}_N = (k, k, \ldots, k)'$, where $\mathbf{1}_N$ denotes the N-dimensional vector of ones. For batch of size N, we will denote the $G-1$ arm vector corresponding to arm k as $\boldsymbol{k}_N$. The set of all $G-1$ arm vectors is denoted by $\mathcal{W}^{g1}$ with cardinality $|\mathcal{W}^{g1}| = |\mathcal{W}|$.*
- $C-1$ ***arm vector:*** *Let $N_m = |\mathcal{C}_m|$ be the size of cluster m, so $\sum_{m=1}^{M} N_m = N$. With respect to the clusters in $\mathscr{C}$, an arm vector $\boldsymbol{k} \in \mathcal{W}^N$ is called a* cluster one *or $C-1$* arm vector *if all nodes in the cluster $\mathcal{C}_m$ receive the same arm. Formally, if there exist arms $k^{(1)}, k^{(2)}, \ldots, k^{(M)} \in \mathcal{W}$ such that for all $i \in \mathcal{C}_m$, we have $k_i = k^{(m)}$ for any $m = 1, 2, \ldots, M$, then $C-1$ vector can be written as, $\boldsymbol{k}_{N,C-1} = (k^{(1)} \cdot \mathbf{1}_{N_1}, k^{(2)} \cdot \mathbf{1}_{N_2}, \ldots, k^{(M)} \cdot \mathbf{1}_{N_M})'$, where the clusters are ordered according to some fixed indexing. The set of all $C-1$ arm vectors is denoted by $\mathcal{W}^{c1}$ with cardinality $|\mathcal{W}^{c1}| = |\mathcal{W}|^M$.*

If we write $\boldsymbol{k}_{N_m}^{(m)} = k^{(m)} \cdot \mathbf{1}_{N_m}$, as a cluster specific uniform assignment, i.e., all nodes in cluster m receive same arm $k^{(m)} \in \mathcal{W}$, then we can write any $C-1$ arm vector as $\boldsymbol{k}_{N,C-1} = (\boldsymbol{k}_{N_1}^{(1)}, \boldsymbol{k}_{N_2}^{(2)}, \ldots, \boldsymbol{k}_{N_M}^{(M)})$. The vectors like $\boldsymbol{k}_{N_m}^{(m)}$ can be thought of as cluster-specific $G-1$ arm vectors of size N_m.

$C-1$ arm vectors can be quite useful. In particular, they dramatically reduce the size of the action space. If the batch size is N and there are K arms, then the full action space has size K^N, while the $G-1$ arm vector space has size K, and the $C-1$ arm vector space has

size K^M, where M is the number of clusters. When $M \ll N$, this is a significant reduction in complexity, making it feasible to learn optimal or near-optimal arm vectors within these restricted classes.

Below we give an example to illustrate the idea of $G-1$ and $C-1$ arm vectors using the crime reduction example from Section 2.1.

Example 2.7 ($G-1$ and $C-1$ Arm Vectors for Policing on Crime Reduction)**.** In the example we have $N = 10$ neighborhoods partitioned into $M = 3$ clusters. The clusters are $\mathcal{C}_1 = \{1,2,3,4,5\}$, $\mathcal{C}_2 = \{6,7,8\}$, and $\mathcal{C}_3 = \{9,10\}$, and we have $K = 3$ strategies. So this gives $|\mathcal{W}^{g1}| = 3$ arm vectors which are called $G-1$ arm vectors. Essentially these are the vectors where all neighborhoods receive the same arm / strategy, we write them as $\mathbf{1}_{10}, \mathbf{2}_{10}$, and $\mathbf{3}_{10}$, which means all neighborhoods get strategy 1, all neighborhoods get strategy 2, and all neighborhoods get strategy 3 respectively.

The $C-1$ **arm vectors** are the vectors where all neighborhoods in a cluster receive the same strategy and there are $|\mathcal{W}^{c1}| = 3^3 = 27$ such vectors. For example, one such $C-1$ arm vector is $\boldsymbol{k}_{N,C-1}$ is $(1,1,1,1,1,2,2,2,3,3)$, this means cluster 1 gets 1, cluster 2 gets 2, cluster 3 gets 3. Since clusters are ordered serially, in this case we have $\boldsymbol{k}_{N_1}^{(1)} = \mathbf{1}_5$ for cluster 1, $\boldsymbol{k}_{N_2}^{(2)} = \mathbf{2}_3$ for cluster 2, and $\boldsymbol{k}_{N_3}^{(3)} = \mathbf{3}_2$ for cluster 3. These are cluster specific uniform assignments or we can also say cluster-wise $G-1$ arm vector. If we write concatenate them as a vector, we get the $C-1$ vector as $\boldsymbol{k}_{N,C-1} = (\mathbf{1}_5\ ,\ \mathbf{2}_3\ ,\ \mathbf{3}_2)'$.

///

2.2.3 Policy, Goal and Regret in the Presence of Interference

Now, we need to define the goal of the learner, policy function and regret in this setting. Conceptually these are similar to the stochastic bandit setting without interference (see Section 1.2.2). However, the difference is now we have interference, i.e., the reward for each node $Y_{i,t}$ depends on the full arm vector $\boldsymbol{W}_t$ assigned to all nodes in the batch at time period t, so some adjustments are needed.

The policy function here specifies how at each time t, the learner selects an arm vector $\boldsymbol{W}_t$ based on the history of all observed rewards and arm vectors up to time $t-1$. Formally, we can write this function as, $\pi_t : \left(\mathcal{W}^N \times \mathcal{Y}^N\right)^{t-1} \to \mathcal{W}^N$, where $\mathcal{Y}$ is the outcome space for individual node rewards. This means the policy function takes a sequence of random pairs $(\boldsymbol{W}_1, \boldsymbol{Y}_1), (\boldsymbol{W}_2, \boldsymbol{Y}_2), \ldots, (\boldsymbol{W}_{t-1}, \boldsymbol{Y}_{t-1})$ as inputs and outputs an arm vector $\boldsymbol{W}_t \in \mathcal{W}^N$ to be played at time t. The learner can also play a randomized policy where $\pi_t : \left(\mathcal{W}^N \times \mathcal{Y}^N\right)^{t-1} \to \mathcal{P}\left(\mathcal{W}^N\right)$, selecting from a distribution over possible arm vectors. The complete policy for the learner will be denoted by $\pi := \{\pi_t\}_{t=1}^T$, which is the sequence

of policy functions across all-time points that determines the learner's strategy throughout the interaction.

The goal of the learner is to devise a policy π such that it maximizes cumulative batch sum of the expected reward on average (or in expectation), over the batch $\mathcal{I}$ and over the horizon T. Writing the problem in our notation, the goal can be written as follows,

$$\begin{aligned} \text{Learner's Problem: } \underset{\pi}{\text{maximize}}\ \mathbb{E}\left[\sum_{t=1}^{T}\sum_{i\in\mathcal{I}}\mu_i(\boldsymbol{W}_t)\right] \text{ or equivalently,} \\ \underset{\pi}{\text{maximize}}\ \mathbb{E}\left[\sum_{t=1}^{T}\mu_\Sigma(\boldsymbol{W}_t)\right], \end{aligned} \tag{2.7}$$

where we used a shorthand $\mu_\Sigma(\cdot) := \sum_{i\in\mathcal{I}}\mu_i(\cdot)$ to denote the batch sum of the expected rewards. So for any algorithm, we need to devise a policy π that selects a sequence of arm vectors $\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_T$ such that it maximizes the total expected reward across all nodes in the batch over the horizon T.

Now in the bandit setting we have an equivalent way to write this maximization problem, which is known as minimizing *regret*. In this case the definition of regret is similar to the Definition 1.7, which we defined without interference. But since now we have a batch we need to define the best arm vector such that it maximizes the batch sum of expected reward for all nodes in $\mathcal{I}$. The regret in this case is calculated against the best possible arm vector in the full action space $\mathcal{W}^N$ denoted by $\boldsymbol{k}^*$, where

$$\boldsymbol{k}^* = \underset{\boldsymbol{k}\in\mathcal{W}^N}{\operatorname{argmax}}\sum_{i\in\mathcal{I}}\mu_i(\boldsymbol{k}) = \underset{\boldsymbol{k}\in\mathcal{W}^N}{\operatorname{argmax}}\,\mu_\Sigma(\boldsymbol{k}) \tag{2.8}$$

In the following we define the standard regret under interference against this optimal arm vector $\boldsymbol{k}^*$. However, since in the interference setting the action space $\mathcal{W}^N$ is huge, finding this optimal arm vector $\boldsymbol{k}^*$ can become computationally very hard or intractable even for moderate batch sizes and number of arms. Moreover, sometimes our policy or algorithm will also restrict attention to special arm vectors like $G-1$ or $C-1$ arm vectors defined in Definition 2.6, so we can also define regrets based on these special arm vectors. All the regrets in the following are defined for the stochastic bandit model with interference defined in Definition 2.5.

Definition 2.8 (STANDARD REGRET UNDER INTERFERENCE)**.**
*Let $\boldsymbol{k}^* = \arg\max_{\boldsymbol{k}\in\mathcal{W}^N} \mu_\Sigma(\boldsymbol{k})$ be the optimal arm vector in $\mathcal{W}^N$. For a policy $\pi = \{\pi_t\}_{t=1}^T$ and an environment $\mathscr{F}_{\boldsymbol{\mu}}$, the **standard pseudo regret under interference** of the learner is defined as,*

$$\mathcal{R}_T(\mathscr{F}_{\boldsymbol{\mu}}, \pi) = \sum_{t=1}^{T} (\mu_\Sigma^* - \mu_\Sigma(\boldsymbol{W}_t)) = T \times \mu_\Sigma(\boldsymbol{k}^*) - \sum_{t=1}^{T} \mu_\Sigma(\boldsymbol{W}_t), \tag{2.9}$$

where $\mu_\Sigma^* := \mu_\Sigma(\boldsymbol{k}^*) = \max_{\boldsymbol{k}\in\mathcal{W}^N} \mu_\Sigma(\boldsymbol{k})$ is the maximum batch sum of expected reward over all arm vectors in $\mathcal{W}^N$. The regret is defined as $\mathbb{E}[\mathcal{R}_T(\mathscr{F}_{\boldsymbol{\mu}}, \pi)]$ where the expectation is over the randomness in the policy π.

Now for the other notions of regret based on $G-1$ and $C-1$ arm vectors, before defining regret, we first need to define the optimal $G-1$ and $C-1$ arm vectors as,

$$\boldsymbol{k}_N^* = \underset{\boldsymbol{k}_N\in\mathcal{W}^{g1}}{\operatorname{argmax}}\, \mu_\Sigma\left(\boldsymbol{k}_N\right) \tag{2.10}$$

$$\boldsymbol{k}_{N,C-1}^* = \underset{\boldsymbol{k}_{N,C-1}\in\mathcal{W}^{c1}}{\operatorname{argmax}}\, \mu_\Sigma\left(\boldsymbol{k}_{N,C-1}\right) \tag{2.11}$$

where $\boldsymbol{k}_N^*$ is the optimal $G-1$ arm vector such that it maximizes the batch sum of expected reward over all $G-1$ arm vectors in $\mathcal{W}^{g1}$. Similarly, $\boldsymbol{k}_{N,C-1}^*$ is the optimal $C-1$ arm vector such that it maximizes the batch sum of expected reward over all $C-1$ arm vectors in $\mathcal{W}^{c1}$. With these two arm vectors now we can define two more regrets, which are easier to optimize against since the action spaces are smaller.

Definition 2.9 ($G-1$ AND $C-1$ REGRET UNDER INTERFERENCE)**.**
*Let $\boldsymbol{k}_N^*$ and $\boldsymbol{k}_{N,C-1}^*$ be the optimal $G-1$ and $C-1$ arm vectors in the set $\mathcal{W}^{g1}$ and $\mathcal{W}^{c1}$ respectively. For a policy $\pi = \{\pi_t\}_{t=1}^T$ and an environment $\mathscr{F}_{\boldsymbol{\mu}}$, the $G-1$ and $C-1$ **pseudo regrets under interference** for a learner are defined as,*

$$\mathcal{R}_T^{g1}(\mathscr{F}_{\boldsymbol{\mu}}, \pi) = \sum_{t=1}^{T} \left(\mu_\Sigma\left(\boldsymbol{k}_N^*\right) - \mu_\Sigma(\boldsymbol{W}_t)\right) = T \times \mu_\Sigma\left(\boldsymbol{k}_N^*\right) - \sum_{t=1}^{T} \mu_\Sigma(\boldsymbol{W}_t) \tag{2.12}$$

$$\mathcal{R}_T^{c1}(\mathscr{F}_{\boldsymbol{\mu}}, \pi) = \sum_{t=1}^{T} \left(\mu_\Sigma\left(\boldsymbol{k}_{N,C-1}^*\right) - \mu_\Sigma(\boldsymbol{W}_t)\right) = T \times \mu_\Sigma\left(\boldsymbol{k}_{N,C-1}^*\right) - \sum_{t=1}^{T} \mu_\Sigma(\boldsymbol{W}_t) \tag{2.13}$$

where $\mu_\Sigma\left(\boldsymbol{k}_N^*\right) = \max_{\boldsymbol{k}\in\mathcal{W}^{g1}} \mu_\Sigma(\boldsymbol{k})$ and $\mu_\Sigma\left(\boldsymbol{k}_{N,C-1}^*\right) = \max_{\boldsymbol{k}\in\mathcal{W}^{c1}} \mu_\Sigma(\boldsymbol{k})$ are the maximum batch sum of expected reward over all $G-1$ arm vectors and over all $C-1$ arm vectors in $\mathcal{W}^{g1}$ and $\mathcal{W}^{c1}$ respectively. The $G-1$ and $C-1$ regrets are defined as $\mathbb{E}[\mathcal{R}_T^{g1}(\mathscr{F}_{\boldsymbol{\mu}}, \pi)]$ and $\mathbb{E}[\mathcal{R}_T^{c1}(\mathscr{F}_{\boldsymbol{\mu}}, \pi)]$ where the expectation is taken over the randomness in the policy π.

With our motivating example, we now give example of the optimal $G-1$ and $C-1$ arm vectors.

Example 2.10 (Optimal $G-1$ and $C-1$ Arm Vectors for Policing on Crime Reduction)**.** In our running example of crime reduction in Section 2.1, following the linear-in-means model eq. (2.1), with the parameters from the Table 2.1 and setting $\alpha_i = 0$, we find the global optimal arm $\boldsymbol{k}^* \in \mathcal{W}^N$ to be,

$$\boldsymbol{k}^* = \underset{\boldsymbol{k}\in\mathcal{W}^N}{\operatorname{argmax}}\, \mu_\Sigma(\boldsymbol{k}) = (3,1,3,2,2,3,1,1,1,1)',$$

Note that, we write this as an optimal arm vector that maximizes reward. Since $Y_{i,t}$ is a percentage change in crime, negative is better, so this is a minimization problem since police department wants to minimize crime. However, to keep the standard reward maximization structure in bandit literature, we let $-Y_{i,t}$ as the reward for node i at time t. This helps us to keep the reward maximization structure, that is standard in bandit literature. Then we can define $\mu_i(\boldsymbol{k}) = -\mathbb{E}[Y_{i,t}(\boldsymbol{k})]$ as the expected reward, which is now the expected percentage reduction in crime, so if we have $\mu_i(\boldsymbol{k}) > 0$, this means on average we have reduction in crime for neighborhood i when arm vector $\boldsymbol{k}$ is assigned. Now calculating mean gives $\mu^*_\Sigma = \mu_\Sigma(\boldsymbol{k}^*) = 25.74\%$.

This means if we could assign the arm vector $\boldsymbol{k}^*$ to the 10 neighborhoods simultaneously, then we can achieve percentage reduction in crime on an average by 25.74% and this is the maximum possible expected reduction we can get in this setup. We can also calculate the optimal $G-1$ and $C-1$ arm vectors along with their corresponding means. We find,

$$\begin{aligned}\boldsymbol{k}^*_N &= \underset{\boldsymbol{k}_N\in\mathcal{W}^{g1}}{\operatorname{argmax}}\, \mu_\Sigma\left(\boldsymbol{k}_N\right) = (2,2,2,2,2,2,2,2,2,2)' \\ &= \mathbf{2}_{10},\end{aligned}$$

and this leads to $\mu_\Sigma\left(\boldsymbol{k}^*_N\right) = 13.9\%$. Also,

$$\begin{aligned}\boldsymbol{k}^*_{N,C-1} &= \underset{\boldsymbol{k}_{N,C-1}\in\mathcal{W}^{c1}}{\operatorname{argmax}}\, \mu_\Sigma\left(\boldsymbol{k}_{N,C-1}\right) = (\underbrace{2,2,2,2,2}_{\text{cluster 1}}, \underbrace{3,3,3}_{\text{cluster 2}}, \underbrace{1,1}_{\text{cluster 3}})' \\ &= (\mathbf{2}_5, \mathbf{3}_3, \mathbf{1}_2)',\end{aligned}$$

and this leads to $\mu_\Sigma\left(\boldsymbol{k}^*_{N,C-1}\right) = 19\%$. So when it comes to assigning same arms to all neighborhoods, the best we can do is to assign strategy 2, to all neighborhoods, which gives us 13.9% overall reduction in crime. But if we consider cluster-wise uniform assignment, we can do better, by assigning strategy 2 to cluster 1, strategy 3 to cluster 2 and strategy 1 to cluster 3, which gives us 19% overall reduction in crime. So, in this case, we can order the strategies, from most preferred to least preferred as, $\boldsymbol{k}^* \succ \boldsymbol{k}^*_{N,C-1} \succ \boldsymbol{k}^*_N$,

and in terms of the sum of expected means, $\mu^*_\Sigma > \mu_\Sigma\left(\boldsymbol{k}^*_{N,C-1}\right) > \mu_\Sigma\left(\boldsymbol{k}^*_N\right)$.

Note that, for $N = 10$, we have a relatively small network here. So for $K = 3$ arms, when calculating the optimal arm vector, we could loop over all possible combinations or arm vectors in the set $\mathcal{W}^N$, where there are $|K^N| = 3^{10} = 59,049$ number of vectors or combinations. And then we find the optimal arm vector $\boldsymbol{k}^*$. For large network this won't be this easy. In this case the $C-1$ optimal could be useful.

As discussed before, for finding $C-1$ optimal arm vector, we only need to loop over $|\mathcal{W}^{c1}| = K^M = 3^3 = 27$ arm vectors, which is much smaller.

However, our problem is even simpler in this case. Recall that the the clustered interference assumption. This means we can find the optimal uniform arm assignment $\boldsymbol{k}^{(m)*}_{N_m}$ for each cluster separately, and then combine the solution. This will reduce the search space even further. In particular, for three clusters with three arms, we only need to loop over $M \times K = 3 \times 3 = 9$ arm cluster level $\boldsymbol{k}^{(m)}_{N_m}$ arm vectors to find the optimal $C-1$ arm vector. So in cluster 1 we will look for 3 arm vectors $\mathbf{1}^{(1)}_{N_1}$, $\mathbf{2}^{(2)}_{N_2}$, and $\mathbf{3}^{(3)}_{N_3}$ and find $\boldsymbol{k}^{(1)*}_{N_1} = \mathbf{2}^{(1)}_5$. Similarly for cluster 2, we find $\boldsymbol{k}^{(2)*}_{N_2} = \mathbf{3}^{(2)}_3$. Finally for cluster 3, we find the optimal uniform assignment $\boldsymbol{k}^{(3)*}_{N_3} = \mathbf{1}^{(3)}_2$.

Concatenating these three cluster-specific optimal uniform assignments will give us the optimal $C-1$ arm vector. So we write all these notations in detailed as

$$\begin{aligned}
\boldsymbol{k}^*_{N,C-1} &= [\boldsymbol{k}^{(1)*}_{N_1}, \boldsymbol{k}^{(2)*}_{N_2}, \boldsymbol{k}^{(3)*}_{N_3}] \\
&= (\mathbf{2}_5, \mathbf{3}_3, \mathbf{1}_2)' \\
&= (\underbrace{2,2,2,2,2}_{\text{cluster 1}}, \underbrace{3,3,3}_{\text{cluster 2}}, \underbrace{1,1}_{\text{cluster 3}})'
\end{aligned}$$

This property, which we can summarize as - *the optimal $C-1$ arm vector can be constructed by combining the optimal uniform assignments for each cluster separately or playing cluster level optimal $G-1$ assignments, e.g., $\boldsymbol{k}^{(m)}_{N_m}$ separately*. This is an important property for the $C-1$ vectors and later we will see this plays an important role in the algorithm design for $C-M$ learner in Section 2.3.4.

///

2.2.4 K-Arm Linear-In-Means and No Switching Case

Starting from the running example in Section 2.1 we have been using the linear-In-means (LIM) model, proposed by Manski (1993) to illustrate different concepts. In the literature the two-arm setting, or more widely known as the *treatment-control* setup has been widely

used in different areas, e.g., see Chin (2019), Leung (2022) for application related to interference and Bramoullé, Djebbari, and Fortin (2020) for general peer effects in networks. If we have arm vector $\boldsymbol{W}_t \in \{0,1\}^N$ at time t, then following the two-arm LIM model, for each node $i \in \mathcal{I}$ the reward can be written as,

$$Y_{i,t}(\boldsymbol{W}_t) = \alpha_i + \beta \cdot W_{i,t} + \delta \sum_{j \in \mathcal{I}} A_{ij} \cdot W_{j,t} + \epsilon_{i,t}, \tag{2.14}$$

where $W_{i,t} \in \{0,1\}$ indicates whether node i is treated or not at time t, α_i is the baseline mean reward for unit i for no treatment, β is the direct effect parameter for the treatment arm, and δ is the interference effect parameter. The matrix $\mathbf{A}$ is typically the adjacency matrix (could be also normalized adjacency matrix) of the network representing interference structure among the nodes, i.e., the entry A_{ij} indicates the connection between node i and node j, assuming $A_{ii} = 0$, i.e., no self loop. This makes the term $\sum_{j \in \mathcal{I}} A_{ij} \cdot W_{j,t}$ to be the number of treated neighbors of node i at time t (if normalized then fraction of treated neighbors). The error term $\epsilon_{i,t}$ is assumed to be i.i.d. across nodes and time with mean zero and some variance. The model is called *linear-In-means (LIM)* since it is linear in both its own treatment and also in the interference effects coming from the neighbors. Note that the basic version of the model assumes *anonymous interference*, i.e., the interference effect depends only on the number (or fraction) of treated neighbors, and not on the specific identity of the neighbors.

Although this is a very well known model, existing works in the literature focuses only on the two arm case. However, the multi-armed bandit settings typically involve more than two arms, hence, in this work we extend this model to the K arm version which already we used in the motivating example in Section 2.1 (where $K = 3$). This extension is relatively straightforward, but we believe this is a minor yet novel contribution of this chapter as well. In general, for a treatment vector $\boldsymbol{W}_t \in [K]^N$, the K arm LIM model can be written as follows,

$$Y_{i,t}(\boldsymbol{W}_t) = \alpha_i + \sum_{k=1}^{K} \beta_k \cdot \mathbb{1}\{W_{i,t} = k\} + \sum_{k=1}^{K} \delta_k \sum_{j \in \mathcal{I}} A_{ij} \cdot \mathbb{1}\{W_{j,t} = k\} + \epsilon_{i,t}, \tag{2.15}$$

where following the same idea, now for each arm k, we have a direct effect parameter β_k, and an interference effect parameter δ_k. Now in this case term $\sum_{j \in \mathcal{I}} A_{ij} \cdot \mathbb{1}\{W_{j,t} = k\}$ captures the number (or fraction) of neighbors or nodes that received arm k at time t. Now assuming we only have mean level interference (see (i) of Remarks 2.4), then we can further write the model as follows,

$$Y_{i,t}(\boldsymbol{W}_t) = \mu_i(\boldsymbol{W}_t) + \epsilon_{i,t}, \tag{2.16}$$

$$\text{where }, \mu_i(\boldsymbol{W}_t) = \alpha_i + \sum_{k=1}^{K} \beta_k \cdot \mathbb{1}\{W_{i,t} = k\} + \sum_{k=1}^{K} \delta_k \sum_{j \in \mathcal{I}} A_{ij} \cdot \mathbb{1}\{W_{j,t} = k\} \tag{2.17}$$

where the error term $\epsilon_{i,t}$ assumed to be independent across nodes and time. Finally, the vector / matrix version of the model can be written as follows,

$$\boldsymbol{Y}_t(\boldsymbol{W}_t) = \boldsymbol{\alpha} + \mathbf{K}_t\boldsymbol{\beta} + \mathbf{A}\mathbf{K}_t\boldsymbol{\delta} + \boldsymbol{\epsilon}_t \tag{2.18}$$

where $\boldsymbol{Y}_t$ is a vector of rewards at time t when arm vector $\boldsymbol{W}_t$ is pulled, $\mathbf{K}_t$ is $N \times K$ one-hot encoding matrix[||] of the treatment assignment $\boldsymbol{W}_t$. The vectors $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_N)'$, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_K)'$ and $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_K)'$, are $N \times 1$ vector of baseline means, $K \times 1$ vector of direct effect parameters and $K \times 1$ vector of spillover effect parameters, respectively. Finally, $\boldsymbol{\epsilon}_t$ is the error vector at time t.

Since in our example we assumed the adjacency matrix $\mathbf{A}$ is row-normalized, when the learner assigns any $G-1$ arm vector $\boldsymbol{k}_N$ to any node i, the expected reward for i can be easily calculated. In particular, we have,

$$\mu_i(\boldsymbol{k}_N) = \begin{cases} \alpha + \beta_k + \delta_k & \text{if } \sum_{j \in \mathcal{I}} A_{ij} = 1 \\ \alpha + \beta_k & \text{otherwise} \end{cases} \tag{2.19}$$

It's important to note that this is only for $G-1$ arm vectors and not in general.

We now explain a special situation for the optimal arm vectors in the LIM model, which we call *No-Optimal Switching* (No-OS) case[**]. This is important in our study, since in this setting, individual optimal vector for any node is same as the batch optimal vector $\boldsymbol{k}^*$, moreover, this is same as the $G-1$ optimal arm vector $\boldsymbol{k}^*_N$ and also he $C-1$ optimal arm vector $\boldsymbol{k}^*_{N,C-1}$, i.e.,

$$\boldsymbol{k}^* = \underset{\boldsymbol{k} \in \mathcal{W}^N}{\operatorname{argmax}}\, \mu_\Sigma(\boldsymbol{k}) = \underset{\boldsymbol{k} \in \mathcal{W}^N}{\operatorname{argmax}}\, \mu_i(\boldsymbol{k}) \quad \forall i \in \mathcal{I},$$
$$\text{also } \boldsymbol{k}^* = \boldsymbol{k}^*_N = \boldsymbol{k}^*_{N,C-1}$$

We explain this with an example below.

[||] In matrix $\mathbf{K}_t$, column goes from $k = 1, 2, \ldots, K$ and each column contains indicators of whether each node received treatment k at time t, i.e., the entry (i,k) of the matrix is $\mathbb{1}\{W_{i,t} = k\}$. Note that this means each row has only single 1, the rest of the positions are 0.

[**] The complete characterization of the parameters with adjacency matrix requires a detailed analysis, which we don't do here, but keep for future investigation.

Example 2.11 (No-OS Example in LIM Model)**.** Consider the same setup of our running example in Section 2.1, but now suppose we change the strategy set. Assume we have three different strategies, which are installing surveillance cameras with different intensity levels across the neighborhoods, i.e., *strategy* 1*: high intensity surveillance, strategy* 2*: medium intensity surveillance and strategy* 3*: low intensity surveillance*. Table 2.2 has the parameters for this case. Here, the direct effect of surveillance cameras on neighborhood i is - reducing crime in the neighborhood i, but if implemented in i, then the *interference effect on j is also reducing crime in neighborhood j*. So for each strategy k, both the direct effect β_k and the spillover effect δ_k are negative, indicating reduction in crime. And what's important is, the higher the intensity of surveillance, the higher the reduction in crime both directly and through spillover.

Arm / Action k	**Strategy** k	**Direct Effect** (β_k)	**Spillover Effect** (δ_k)
1	High Surveil.	-5.2% | $\downarrow\downarrow\downarrow$	-6.2% | $\downarrow\downarrow\downarrow$
2	Moderate Surveil.	-3.0% | $\downarrow\downarrow$	-2.3% | $\downarrow\downarrow$
3	Low Surveil.	-1.0% | $\downarrow$	0.0

Table 2.2: No optimal switching case for the policing strategies with their direct and spillover effects. In this case, if strategy 1 is implemented in neighborhood i, then neighborhood i experiences strongest decrease in crime (due to direct effect), and at the same time neighboring area $j \neq i$ also experience the strongest decrease in crime (due to spillover effect).

In this case, if we take the same network structure of the neighborhoods as in Figure 2.2, then for neighborhood 2, being an isolated node, the optimal arm is strategy-1, since it has the highest direct effect. If we think in terms of the vector for this isolated node, then the optimal $G-1$ arm vector for this node is simply $\mathbf{1}_N$, i.e., even if everyone gets strategy-1, it doesn't change anything since there are no neighbors.

Now, let's look at any connected node, for example neighborhood 1. It has neighbors 3, 4 and 5. Here we see also the optimal $G-1$ arm vector for this node is $\mathbf{1}_N$. This is because, assigning strategy-1 to itself is optimal for the direct effect. But assigning this same arm to the neighbors also gives the highest positive spillover effect, i.e., reduction in crime from neighbors. So overall, assigning strategy-1 to everyone gives the highest total expected reward for neighborhood 1.

So we are in a case when the optimal $G-1$ arm vector for each node or neighborhood is same, regardless of being isolated or not, and this is $\boldsymbol{k}_N^* = (1,1,1,1,1,1,1,1,1,1)'$. And

this is also the optimal arm vector $\boldsymbol{k}*$ here and the optimal $C-1$ arm vector $\boldsymbol{k}^*_{N,C-1}$ as well. We call this case *No-Optimal Switching* in the LIM model, where the $G-1$ optimal arm for each node **does not switch** depending on whether it has neighbors or not, or what strategies the neighbors get.

///

So the No-OS is simply a special scenario in the LIM model, where the $G-1$ optimal arm vector switches for the isolated and connected nodes. Or in other words, if an arm $k \in \mathcal{W}$ is the optimal arm for an isolated node, then for any connected node in the batch, the arm vector $k \cdot \mathbf{1}_N$ (from the same arm) is also the optimal $G-1$ arm vector. This vector is also then optimal vector $\boldsymbol{k}^*$ and also the $C-1$ optimal arm vector $\boldsymbol{k}^*_{N,C-1}$.

In general, the optimal $G-1$ arm vector **may switch** depending on whether the node is isolated or connected[††]. We call this case *Optimal Switching (OS) case*. We already have the example in the original setup in Table 2.1. There, what's best for the isolated nodes or neighborhoods is different from what's best for the connected nodes. For example, think about neighborhood 2 and 1 again. For neighborhood 2, being an isolated node, $\mathbf{1}_N$ is the optimal arm vector (again what's assigned to the neighbors don't matter for node 2 since it's isolated).

However, for neighborhood 1, since it's connected to neighborhoods 3, 4 and 5, assigning strategy 1 to itself and also to its neighbors cannot be $G-1$ optimal, since this also gives the highest negative spillover effects. In this case the $G-1$ optimal arm vector for this node is assigning strategy-3 to all the nodes, i.e., $\mathbf{3}_N$. This is because by assigning strategy-3 to its neighbors, neighborhood 1 gets zero spillover effect, which is better than negative spillover effects from other strategies.

Since in this case the optimal $G-1$ arm vector for an isolated node is different from the connected nodes, we call this case *Optimal Switching (OS)* case. Note that in this case the global optimal arm vector $\boldsymbol{k}^*$, for neighborhood 1 would be playing strategy-1 for itself and then strategy-3 for its neighbors. So this is also different here.

The reason why No-OS case appears can be understood from looking at the parameters in Table 2.2. Here we see that both direct and spillover effects are aligned in the same direction, i.e., both reduce crime, and the ranking of the arms is the same for both direct and spillover effects. So this means for any node, whether it has neighbors or not, the best arm is always the same arm. However, if the parameters are such that direct and spillover effects are not aligned in the same direction, e.g., one increases reward while the other decreases reward, then we can have switching behavior as we explained before.

†† Although we explained the No-OS case in terms of $G-1$ optimal arm vector switching, another way to look at the case is to see whether the $G-1$ optimal arm vector for any individual node is same as the batch $G-1$ optimal arm vector. In the No-OS case this also holds, but in OS case as we explain it does not.

2.3 Algorithms Under Interference

In this section, we extend the classic Thompson sampling algorithm, that we outlined in Algorithm 1, to different batched setups where interference is possibly present. We propose four alternative algorithms (or we call *learners*) that extend Thompson sampling by simplifying the action space and balancing exploration and exploitation tradeoff in different ways. Depending on the underlying interference structure, these algorithms can help us find the optimal or near-optimal arm vectors efficiently in different scenarios. The first one is the direct batched extension of the classic Thompson sampling that ignores interference, we call this the *identically and independently distributed (IID) learner*. This is not a novel idea per se, but we include this as a baseline algorithm to compare with the other three algorithms that specifically leverage the interference structure. The other three algorithms - *global one arm ($G-1$) learner*, *clustered one arm ($C-1$) learner*, and *cluster multi-arm ($C-M$) learner*. These algorithms leverage the special arm vectors defined in Definition 2.6 and search for optimal arm vectors within the restricted action spaces. The key motivation for us to develop the latter three algorithms is realizing that in many practical scenarios, interference effects are often structured in a way that allows us to simplify the action space without losing much in terms of performance. While the complete action space consists of all possible arm vectors of size K^N, which is computationally infeasible to explore for large N, the $G-1$ and $C-1$ arm vectors provide a way to reduce this complexity significantly and still help us to learn effective policies. Below we explain each of these algorithms in detail with intuitions, outline them with the underlying assumptions and also give concrete examples with Bernoulli and Gaussian rewards.

2.3.1 Identically and Independently Distributed or IID Learner

We first discuss the direct batched version of the no interference Thompson sampling that we call *identically and independently distributed learner or IID learner*. This is a direct extension to a batch version that we outlined in Algorithm 1, which can be thought as the single node / single unit setup where we have K parameters for K arms. If we extend this idea directly to the interference setup where for K arms and N nodes we have K^N arm vectors, then we should have $N \times K^N$ parameters. This is clearly becoming very large even for moderate size of N and K. The idea of the IID learner is to ignore this complexity and interference structure and learn K parameters for K arms just like the no interference case. For IID learner we have the following assumptions,

Assumption 2.12 (ASSUMPTIONS BEHIND *IID* LEARNER)**.**

- ***(A1) Independence**: For any arm vector $\boldsymbol{k} \in \mathcal{W}^N$, the stochastic potential rewards for all node $i \in \mathcal{I}$ are independent, i.e.,*

$$Y_{i,t}(\boldsymbol{k}) \perp Y_{j,t}(\boldsymbol{k}), \quad \forall i, j \in \mathcal{I} \text{ and } i \neq j \tag{2.20}$$

- ***(A2) No Interference and No Heterogeneity**: For any arm vector $\boldsymbol{k} \in \mathcal{W}^N$ and any node $i \in \mathcal{I}$, the stochastic potential reward depends only on the arm assigned to that node i, i.e.,*

$$Y_{i,t}(k_i, \boldsymbol{k}_{-i}) \stackrel{d}{=} Y_{i,t}(k_i, \widetilde{\boldsymbol{k}}_{-i}) \quad \forall \boldsymbol{k}_{-i}, \widetilde{\boldsymbol{k}}_{-i} \in \mathcal{W}^{N-1} \tag{2.21}$$

where k_i is the arm assigned to node i, $\boldsymbol{k}_{-i}$ and $\widetilde{\boldsymbol{k}}_{-i}$ are the arms assigned to all other nodes except node i. Here $\stackrel{d}{=}$ means equality in distribution. Also the potential rewards for all node $i \in \mathcal{I}$ are identically distributed, i.e.,

$$Y_{i,t}(k) \sim \mathrm{F}_k, \quad \forall i \in \mathcal{I} \tag{2.22}$$

where F_k is the potential reward distribution of arm $k \in \mathcal{W}$.

The assumption (A1) states that the stochastic potential reward of node i is independent of the stochastic potential reward of any other node j in the batch for any arm vector $\boldsymbol{k} \in \mathcal{W}^N$. The Assumption (A2) states two points, a) only the arm assigned to node i matters for i, this is in fact no interference assumption, and b) the reward distributions are identical across all nodes for each arm $k \in \mathcal{W}$. So the *IID* learner just ignores the complexity and interference altogether and treats all units or nodes are identical and independent. We give the following concrete example what does these assumptions mean with the running example in Section 2.1.

Example 2.13. We recall that the reward distributions from the crime reduction example in Section 2.1 are assumed to follow the linear-in-means model with interference effects, i.e.,

$$\begin{aligned} Y_{i,t} = \alpha_i + \beta_1 \cdot \mathbb{1}\{W_{i,t} = 1\} + \beta_2 \cdot \mathbb{1}\{W_{i,t} = 2\} + \beta_3 \cdot \mathbb{1}\{W_{i,t} = 3\} + \\ \delta_1 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 1\} + \delta_2 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 2\} + \delta_3 \sum_{j=1}^{10} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 3\} + \epsilon_{i,t} \end{aligned} \tag{2.23}$$

Now, under the assumption (A1), the rewards are independent across the neighborhoods, so $\epsilon_{i,t} \perp \epsilon_{j,t}$ for any $i \neq j$. Under the assumption (A2), the interference effects are nullified, i.e., $\delta_1 = \delta_2 = \delta_3 = 0$ and also all neighborhoods are identical, so we can assume $\alpha = \alpha_i = \alpha_j$ for any $i, j \in \mathcal{I}$ and $\epsilon_{i,t} \sim \mathcal{N}(0, \sigma^2)$ for all $i \in \mathcal{I}$, where σ^2 is some fixed constant. Now, with the Gaussian assumption on the error terms, the reward

Algorithm 6: TS IID Learner (General Case)

Input: $\mathcal{I}$, $\mathcal{W}$, T and $\widetilde{\theta}(k), \forall k \in \mathcal{W}$
Initialize: $\widehat{\theta}_0(k) = \widetilde{\theta}(k), \forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  for $i \in \mathcal{I}$ **do**
    sample $\mu'_t(k) \sim Q^{\widehat{\theta}_{t-1}(k)}, \forall k \in \mathcal{W}$
    select $W_{i,t} = \text{argmax}_{k \in \mathcal{W}} \mu'_t(k)$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  update $\widehat{\theta}_t(k)\ \forall k \in \mathcal{W}$ // for each k using rewards from nodes play arm k
Output: $\boldsymbol{W}_t, \boldsymbol{Y}_t, \forall t \in [T]$

distribution for any neighborhood i at time t reduces to,

$$Y_{i,t}(k) \sim \mathcal{N}(\mu(k), 1) \text{ where } \mu(k) = \alpha + \beta_k \text{ for } k = 1, 2, 3 \tag{2.24}$$

where $\mu(k) = \mathbb{E}[Y_{i,t}(k)] = \alpha + \beta_k$ is the mean reward for any neighborhood i when it is assigned arm k. This is exactly the same as the regular stochastic bandit setup without interference where we have $K = 3$ arms only and a batch of $N = 10$ identical and independent neighborhoods arrive at each time period.

///

Under Assumption 2.12, the general procedure for the IID learner is outlined in Algorithm 6. Compared to the no interference / single unit Thompson sampling (Algorithm 1), it has similar steps that is - *sampling, selection, arm pull and update* - but the difference now is, we have a batched setup. First at time $t = 0$, the learner starts with a time horizon T, an arm set $\mathcal{W} = \{1, 2, \ldots, K\}$, and now additionally a batch $\mathcal{I}$ of N nodes. She sets some fixed values $\widetilde{\theta}(k), \forall k \in \mathcal{W}$ to initialize the parameters. Now, at each time t, for each node i, the learner first samples a random mean $\mu'_t(k)$ for each arm $k \in \mathcal{W}$ from the prior $Q^{\widehat{\theta}_{t-1}(k)}$ that has been updated at end of time $t-1$. Then for each node, the learner selects the arm with the highest sampled mean. After all selections, the learner is able to construct an arm vector $\boldsymbol{W}_t = (W_{1,t}, W_{2,t}, \ldots, W_{N,t})'$. Then the learner pulls this arm vector for all nodes in the batch, and observes the reward from all nodes, i.e., observes $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$. Finally, she updates the shared prior parameter $\widehat{\theta}_t(k)$, (which is now posterior parameter) for all arms based on the observed rewards from all nodes.

The general algorithm for the IID learner in Algorithm 6 doesn't specify how to update each time step, this actually depends on the distributional assumptions. However, what makes it different from the single unit case, and also from other learners is that, here the sampling will be done for all nodes independently but the updates for any arm k is pooled

/ shared from all nodes since they are assumed to be identical. In this way, we have K parameters to update only but simultaneously learning from N nodes. Below we give two concrete examples when the reward distributions are Bernoulli and Gaussian respectively. In both cases, the updates can be expressed in terms of *cumulative rewards* and *cumulative counts*, denoted with $S_t(k)$ and $n_t(k)$ respectively. For each arm k, at the end of time t, the cumulative reward and cumulative count updates are,

$$\begin{aligned} S_t(k) &= \sum_{s=1}^{t}\sum_{i=1}^{N} \mathbb{1}\{W_{i,s} = k\} Y_{i,s} \\ &= S_{t-1}(k) + \sum_{i=1}^{N} \mathbb{1}\{W_{i,t} = k\} Y_{i,t} \end{aligned} \tag{2.25}$$

$$\begin{aligned} n_t(k) &= \sum_{s=1}^{t}\sum_{i=1}^{N} \mathbb{1}\{W_{i,s} = k\} \\ &= n_{t-1}(k) + \sum_{i=1}^{N} \mathbf{1}\{W_{i,t} = k\} \end{aligned} \tag{2.26}$$

So at the end of time t, sum of the rewards and counts are pooled from all the nodes in the batch for all arm $k \in \mathcal{W}$. We now give two examples of prior and posterior updates for the IID learner Bernoulli and Gaussian.

Example 2.14 (TS - IID Learner with Bernoulli Rewards)**.** Like single unit setup, for Bernoulli rewards, the IID learner uses a Beta prior conjugate. The learner starts with prior parameters $\widetilde{\alpha}(k)$ and $\widetilde{\beta}(k)$ for each $k \in \mathcal{W}$, and initializes parameter updates with $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k)$, $\widehat{\beta}_0(k) = \widetilde{\beta}(k)$ and cumulative statistics with $S_0(k) = 0$, and $n_0(k) = 0$. At each round $t = 1, 2, \ldots, T$, for each node $i \in \mathcal{I}$, the learner first *samples* $p'_t(k) \sim \text{Beta}(\widehat{\alpha}_{t-1}(k), \widehat{\beta}_{t-1}(k))$ for all $k \in \mathcal{W}$ independently. Then for each node she *selects* $W_{i,t} = \text{argmax}_{k\in\mathcal{W}}\, p'_t(k)$. She does these two steps independently for all nodes $i \in \mathcal{I}$. After that she *observes* rewards for all nodes, $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$. Then finally she *updates* the cumulative statistics for each arm $k \in \mathcal{W}$ recursively using the formula given in eq. (2.25) and eq. (2.26). With this she also updates the posterior parameters for each arm $k \in \mathcal{W}$ using,

$$\begin{aligned} \widehat{\alpha}_t(k) &= \widetilde{\alpha}(k) + S_t(k), \text{ and} \\ \widehat{\beta}_t(k) &= \widetilde{\beta}(k) + n_t(k) - S_t(k) \end{aligned} \tag{2.27}$$

In practice, we can set $\widetilde{\alpha}(k) = 1$ and $\widetilde{\beta}(k) = 1$ for all $k \in \mathcal{W}$, we have a uniform prior $\text{Beta}(1,1)$ for each arm. In this case, the posterior parameters simplify to $\widehat{\alpha}_t(k) = 1 + S_t(k)$, $\widehat{\beta}_t(k) = 1 + n_t(k) - S_t(k)$.

///

Example 2.15 (TS - *IID* Learner with Gaussian Rewards)**.** For Gaussian rewards, the *IID* learner uses a Gaussian prior conjugate. The learner starts with prior parameters $\widetilde{\mu}(k)$ and $\widetilde{\sigma}^2(k)$ for each arm $k \in \mathcal{W}$, and initializes $\widehat{\mu}_0(k) = \widetilde{\mu}(k)$, $\widehat{\sigma}_0^2(k) = \widetilde{\sigma}^2(k)$, $S_0(k) = 0$, and $n_0(k) = 0$. At each round $t = 1, 2, \ldots, T$, for each node $i \in \mathcal{I}$, the learner first *samples* $\mu'_t(k) \sim \mathcal{N}(\widehat{\mu}_{t-1}(k), \widehat{\sigma}^2_{t-1}(k))$ for each arm $k \in \mathcal{W}$. Then it *selects* $W_{i,t} = \text{argmax}_{k \in \mathcal{W}} \mu'_t(k)$ for each unit i. After *observing* rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, the learner *updates* the cumulative statistics for each arm $k \in \mathcal{W}$ recursively using eq. (2.25) and eq. (2.26), and then updates the posterior parameters for each arm $k \in \mathcal{W}$ using,

$$\begin{aligned} \widehat{\sigma}_t^2(k) &= \left(\frac{n_t(k)}{\sigma^2(k)} + \frac{1}{\widetilde{\sigma}^2(k)} \right)^{-1} \text{ and} \\ \widehat{\mu}_t(k) &= \left(\frac{S_t(k)}{\sigma^2(k)} + \frac{\widetilde{\mu}(k)}{\widetilde{\sigma}^2(k)} \right) \widehat{\sigma}_t^2(k) \end{aligned} \tag{2.28}$$

If we set $\sigma^2(k) = 1$, $\widetilde{\mu}(k) = 0$, and $\widetilde{\sigma}^2(k) = 1$ for all $k \in \mathcal{W}$, then the posterior parameters simplify to $\widehat{\sigma}_t^2(k) = \frac{1}{n_t(k)+1}$ and $\widehat{\mu}_t(k) = \frac{S_t(k)}{n_t(k)+1}$.

///

We now discuss some important aspects of the *IID* learner in the following remark.

Remarks 2.16 (Discussion on TS *IID* Learner)**.**

Sampling, Selection and Update: If we think what happens for each arm $k \in \mathcal{W}$ at each time t, we see that each arm is sampled N times independently, each arm can be selected multiple times (at max N times) for different nodes, and all arms will be updated based on the pooled rewards from all nodes.

Eventually Reaching to a $(G-1)$ Arm Vector: Although initially *IID* learner will play combination of different arms for different nodes, but eventually it will reach to a $G-1$ arm vector by design. This is because of the Thompson sampling procedure, it starts with the same selection probabilities, but the priors are updated based on the observed rewards and like the single node case and eventually one arm will be played more often than others. For a single node case the situation has been illustrated graphically in Chapter 1, Figure 1.4 and Figure 1.5. So if runs long enough, the *IID* learner will eventually play the same arm for all nodes in the batch. However, in the optimal $G-1$ sense this may not be the optimal arm vector due to interference effects as we will discuss next.

Strength in No-Interference and No Switching Setup: The algorithm definitely has the most strength in the no-interference setup. This is expected since the *IID* learner is designed for this setup. In this case the node level best arm is also the global best arm vector, i.e.,

if k is the best arm for any node $i \in \mathcal{I}$, then $\boldsymbol{k}_N$ is the best arm vector for the whole batch $\mathcal{I}$. So IID learner will eventually converge to this unanimous best arm vector. Also, in the no-optimal-switching case, discussed in Example 2.11, even in the presence of interference, the node level optimal arm is same as the global optimal arm vector when all nodes get the same arm. Recall this is the case when optimal arm remains optimal with or without interference (i.e., the interference doesn't change which arm is best for any unit), in this case as well, IID learner will converge to the correct quickly despite the interference effects.

May be Misleading in the Presence of Interference: However, in the presence of interference, the IID learner is not designed to do proper updates for the priors and may converge to a suboptimal $G-1$ arm vector. Let us illustrate this with an example. Assume the learner selects arm 1 for node i and arm 2 for its neighbor j. Now due to interference if node i's reward becomes high because of the positive spillover effect from j receiving arm 2, the learner will attribute the high reward entirely to arm 1, even though arm 1 may not yield high rewards on its own (i.e., without the interference from j). This may lead to wrong conclusions. So in the interference setup, the IID learner is not expected to work properly in all settings.

///

So in summary, the IID Learner is a natural extension of the single unit Thompson sampling algorithm to the batched setup. It works well in the no-interference setup and in some interference settings, however it may not work in all interference settings due to wrong updates. This motivates us to design other algorithms which can work in the presence of interference. We will discuss these algorithms next.

2.3.2 *Global One Arm or $G-1$ Learner*

We already discussed that by construction in the presence of interference, the IID learner will attribute high or low rewards to wrong arms, and this may cause the convergence of the algorithm to a suboptimal $G-1$ arm vector. So, why not devise a policy such that it finds the optimal $G-1$ vector? This motivates our *global one Arm Learner* or in short we say $G-1$ learner.

$G-1$ learner is a simple extension of the classical Thompson sampling algorithm to a batched setup where at each time t the learner assigns only one single arm $k \in \mathcal{W}$ to all the nodes in the batch $\mathcal{I}$, or equivalently the learner pulls a $G-1$ arm vector denoted with $\boldsymbol{k}_N \in \mathcal{W}^{g1}$ for the whole batch, here $\boldsymbol{k}_N = k \cdot \mathbf{1}_N$ for some $k \in \mathcal{W}$. In the presence of interference, this algorithm is designed to directly find the optimal $G-1$ arm vector without any wrong updates from interference effects. First, we introduce some notations. Let $\overline{Y}_t(\boldsymbol{k}_N) := \frac{1}{N}\sum_{i=1}^{N} Y_{i,t}(\boldsymbol{k}_N)$ be the batch average of rewards at time t for arm vector

Algorithm 7: TS $G-1$ Learner (General Case)

Input: $\mathcal{I}$, $\mathcal{W}$, T and $\widetilde{\theta}(k), \forall k \in \mathcal{W}$
Initialize: $\widehat{\theta}_0(k) = \widetilde{\theta}(k), \forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  sample $\overline{\mu}'_t(k) \sim Q^{\widehat{\theta}_{t-1}(k)}, \forall k \in \mathcal{W}$
  select $k' = \text{argmax}_{k \in \mathcal{W}} \overline{\mu}'_t(k)$
  set $\boldsymbol{W}_t = \boldsymbol{k}_N = k' \cdot \mathbf{1}_N$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  update $\widehat{\theta}_t(k')$ // for k' only using rewards from all nodes
Output: $\boldsymbol{W}_t, \boldsymbol{Y}_t, \forall t \in [T]$

$\boldsymbol{k}_N$, with the mean $\overline{\mu}(\boldsymbol{k}_N) := \mathbb{E}\left[\overline{Y}_t(\boldsymbol{k}_N)\right] = \frac{1}{N}\sum_{i=1}^{N} \mu_i(\boldsymbol{k}_N)$. This is the average of the individual means $\mu_i(\cdot)$ in the batch. The $G-1$ learner directly put's prior on this average reward rather than the individual means. In the algorithm this prior / posterior is denoted by $Q^{\widehat{\theta}_t(\boldsymbol{k}_N)}$ where $\widehat{\theta}_t\ (\boldsymbol{k}_N)$ is the parameter of the prior updated at time t. We also use simply k instead of $\boldsymbol{k}_N$ since we only consider $G-1$ arm vectors and both set $\mathcal{W}^{g1}$ and $\mathcal{W}$ have same cardinality. We outline the general procedure for the $G-1$ learner in Algorithm 7.

The $G-1$ learner also has *sampling, selection, arm pull and update* steps like other Thompson sampling algorithms. The learner starts with a time horizon T, an arm set $\mathcal{W} = 1, 2, \ldots, K$, a batch $\mathcal{I}$ of N nodes, and initialization values $\widetilde{\theta}(k)$ for the prior distributions of each arm. At each round t, the learner draws only one sample $\overline{\mu}'_t(k)$ from each arm's prior distribution $Q^{\widehat{\theta}_{t-1}(k)}$ for all nodes. Based on these samples, the learner then selects the arm with the largest sampled mean, denoted with $k' = \text{argmax}_{k \in \mathcal{W}} \overline{\mu}'_t(k)$. This selected arm is then assigned to all nodes simultaneously, yielding a $G-1$ arm vector $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$ where every node pulls the same arm k'. After observing the rewards $Y_{i,t}$ from all nodes in the batch, the learner updates only the prior corresponding to the pulled arm k'. The priors for arms that were not selected remain unchanged. Note that this contrasts with the IID learner where all arm priors may be updated based on selections across nodes.

We can write the cumulative rewards and count updates for the $G-1$ learner as follows. Let $k' \in \mathcal{W}$ be the selected arm after time t, then updates are,

$$\begin{aligned} S_t(k') &= \sum_{s=1}^{t} \mathbb{1}\left\{\boldsymbol{W}_s = k' \cdot \mathbf{1}_N\right\} \sum_{i=1}^{N} Y_{i,s} \\ &= S_{t-1}(k') + \mathbb{1}\{\boldsymbol{W}_t = k' \cdot \mathbf{1}_N\} \sum_{i=1}^{N} Y_{i,t} \end{aligned} \tag{2.29}$$

$$\begin{aligned} n_t(k') &= N \times \sum_{s=1}^{t} \mathbb{1}\left\{\boldsymbol{W}_s = k' \cdot \mathbf{1}_N\right\} \\ &= n_{t-1}(k') + N \cdot \mathbb{1}\{\boldsymbol{W}_t = k' \cdot \mathbf{1}_N\} \end{aligned} \tag{2.30}$$

Again, we give these two updates since both Gaussian and Bernoulli updates can be expressed in terms of these two statistics. Below, we give two concrete examples of prior and posterior updates for the $G-1$ learner with Bernoulli and Gaussian rewards respectively.

Example 2.17 (TS $G-1$ Learner with Bernoulli Rewards)**.** For Bernoulli rewards, the $G-1$ learner models the batch average reward using a Beta prior. Let $\overline{p}(\boldsymbol{k}_N)$ be the mean of the batch average reward when the arm vector $\boldsymbol{k}_N$ is played, i.e., $\overline{p}(\boldsymbol{k}_N) := \mathbb{E}[\overline{Y}_t(\boldsymbol{k}_N)]$. Note that while individual rewards follow $Y_{i,t}(\boldsymbol{k}_N) \sim \text{Bernoulli}(p_i(\boldsymbol{k}_N))$, the batch average $\overline{Y}_t(\boldsymbol{k}_N)$ is not Bernoulli. This is is one of the theoretical issues related to the $G-1$ learner, however, since $\overline{p}(\boldsymbol{k}_N) \in [0,1]$, we can use still Beta prior on the mean of the batch average and assume approximately the underlying priors are close. We write $\overline{p}(k)$ instead of $\overline{p}(\boldsymbol{k}_N)$ for simplicity since we only consider $G-1$ arm vectors. The learner starts with prior parameters $\widetilde{\alpha}(k)$ and $\widetilde{\beta}(k)$ for each arm $k \in \mathcal{W}$, and initializes $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k)$, $\widehat{\beta}_0(k) = \widetilde{\beta}(k)$, $S_0(k) = 0$, and $n_0(k) = 0$. At each round $t = 1, 2, \ldots, T$, the learner first *samples* $\overline{p}'_t(k) \sim \text{Beta}(\widehat{\alpha}_{t-1}(k), \widehat{\beta}_{t-1}(k))$ for each arm $k \in \mathcal{W}$. Then it *selects* $k' = \text{argmax}_{k \in \mathcal{W}} \overline{p}'_t(k)$ and sets the arm vector $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$. After *observing* rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, the learner *updates* the cumulative statistics only for the played arm k' using the cumulative statistics in their recursive forms in eq. (2.29) and eq. (2.30), and then updates the posterior parameters for the played arm k' using,

$$\begin{aligned} \widehat{\alpha}_t(k') &= \widetilde{\alpha}(k') + S_t(k') \text{ and} \\ \widehat{\beta}_t(k') &= \widetilde{\beta}(k') + n_t(k') - S_t(k') \end{aligned} \tag{2.31}$$

Note that, this is same as the updates for the TS *IID* learner in Example 2.14, but now the updates are only for the played arm k'.

///

Example 2.18 (TS $G-1$ Learner with Gaussian Rewards)**.** For Gaussian rewards, the $G-1$ learner models the batch average reward using a Gaussian prior. Let the batch average reward follows $\overline{Y}_t(\boldsymbol{k}_N) \sim \mathcal{N}(\overline{\mu}(\boldsymbol{k}_N), \overline{\sigma}^2(\boldsymbol{k}_N))$, where $\overline{\sigma}^2(\boldsymbol{k}_N)$ is the known variance of the batch average, i.e., $\overline{\mu}(\boldsymbol{k}_N) := \mathbb{E}[\overline{Y}_t(\boldsymbol{k}_N)]$. Note that $\overline{\sigma}^2(\boldsymbol{k}_N)$ incorporates any within-batch correlations arising from interference effects, however in almost all of our examples we will assume interference only affects the mean and not the variance so we can write $\overline{\sigma}^2(k)$. For the mean, although there are interference effects, but we write $\overline{\mu}(k)$, to simplify the notation since we only consider G-1 arm vectors. The learner starts with prior parameters $\widetilde{\mu}(k)$ and $\widetilde{\sigma}^2(k)$ for each arm $k \in \mathcal{W}$, and initializes $\widehat{\mu}_0(k) = \widetilde{\mu}(k)$, $\widehat{\sigma}^2_0(k) = \widetilde{\sigma}^2(k)$, $S_0(k) = 0$, and $n_0(k) = 0$. At each round $t = 1, 2, \ldots, T$, the learner first *samples* $\overline{\mu}'_t(k) \sim \mathcal{N}(\widehat{\mu}_{t-1}(k), \widehat{\sigma}^2_{t-1}(k))$ for each arm $k \in \mathcal{W}$. Then it *selects*

$k' = \text{argmax}_{k \in \mathcal{W}} \overline{\mu}'_t(k)$ and sets the arm vector $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$. After *observing* rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, the learner *updates* the cumulative statistics only for the played arm k' using eq. (2.29) and eq. (2.30), and then updates the posterior parameters for the played arm k' using,

$$\begin{aligned} \widehat{\sigma}_t^2(k') &= \left(\frac{n_t(k')/N}{\overline{\sigma}^2(k')} + \frac{1}{\widetilde{\sigma}^2(k')} \right)^{-1} \text{ and} \\ \widehat{\mu}_t(k') &= \left(\frac{S_t(k')/N}{\overline{\sigma}^2(k')} + \frac{\widetilde{\mu}(k')}{\widetilde{\sigma}^2(k')} \right) \widehat{\sigma}_t^2(k') \end{aligned} \tag{2.32}$$

If we assume interference only affects the mean, e.g., in the linear-in-means model, the error terms are independent across nodes and each node has variance $\sigma^2(k)$ for arm k, then the variance of the batch average is $\overline{\sigma}^2(k) = \sigma^2(k)/N$. In this case, the posterior parameters simplify to

$$\begin{aligned} \widehat{\sigma}_t^2(k') &= \left(\frac{n_t(k')}{\sigma^2(k')} + \frac{1}{\widetilde{\sigma}^2(k')} \right)^{-1} \text{ and} \\ \widehat{\mu}_t(k') &= \left(\frac{S_t(k')}{\sigma^2(k')} + \frac{\widetilde{\mu}(k')}{\widetilde{\sigma}^2(k')} \right) \widehat{\sigma}_t^2(k') \end{aligned} \tag{2.33}$$

Note that this is the same as the TS IID learner's update in Example 2.15, except in this case the updates are only for the played arm k'.

///

Some important aspects of the TS $G-1$ Learner are discussed in the following remarks.

Remarks 2.19 (Discussion on TS $G-1$ Learner)**.**

Sampling, Selection and Update: Perhaps a simple way to understand the $G-1$ learner is, $\forall k \in \mathcal{W}$, at every time period t, the $G-1$ learner samples one random mean following $\overline{\mu}'_t(k) \sim Q^{\widehat{\theta}_{t-1}(k)}$ for all nodes, i.e., each arm is sampled once. Then for all nodes selection is done with $k' = \arg\max_{k \in \mathcal{W}} \overline{\mu}'_t(k)$ and the learner assigns $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$. So for each arm, sampling is done once, and then all nodes gets that arm. Finally, after observing the rewards from all nodes, the learner updates only the played arm k' (since only that arm has been played). This is in sharp contrast to the IID learner.

Eventually Reaching to Optimal $G-1$ Arm Vector: Since only one arm will be played for all nodes at each time t, eventually the priors will be around a single arm which is the best in terms of the batch average $\overline{\mu}(\boldsymbol{k}_N)$. So the algorithm is designed to find the arm vector $\boldsymbol{k}_N$ which has the highest batch average mean $\overline{\mu}(\boldsymbol{k}_N)$. So if we ran the algorithm long enough, it will eventually find the optimal $G-1$ arm vector.

Strength in Different Interference Structure: The strength of the $G-1$ learner is, in general this can be applied to wide variety of interference setting, since we are looking for the optimal $G-1$ arm vector. So if there is an optimal $G-1$ arm vector, this learner will find it. It does not require any additional assumption or information about how the nodes interfere with each other. Also in the no optimal switching case discussed in Example 2.11, the $G-1$ learner will find the global optimal arm vector. This is because the global optimal arm vector is also the optimal $G-1$ arm vector in this case. However, it will suffer for high initial regret as we will discuss next.

Disadvantage of High Starting Regret: Because at each time t, only one arm is played for all nodes, the $G-1$ learner has low exploration capability per time period but high exploitation. So initially the $G-1$ learner suffers high starting regret. This happens because, if the pulled arm is not the best arm, for the whole batch the regret will be high because we are summing on individual regrets. This is in contrast to the IID learner where different arms can be played for different nodes at each time t, which helps to have low starting regret.

Theoretical Issues Related to Prior-Posterior: There is a theoretical issue regarding the prior-posterior setup of the $G-1$ learner. This came in the discussion of Example 2.17, that is we are placing direct prior assumption on the mean $\overline{\mu}(\boldsymbol{k}_N)$, this is the mean of the batch average reward $\overline{Y}_t(\boldsymbol{k}_N)$. For example, we are assuming whether this mean is Beta distributed or Gaussian distributed depending on the reward distribution. However, in practice this may not happen if we start from node level or individual level, e.g., average of Bernoulli random variables (possibly dependent) is not Bernoulli distributed. So applying Beta-Bernoulli relationship directly on the batch average is not straightforward. However, the algorithm can be seen as an approximation of the unknown theoretical setup that we have underneath.

///

2.3.3 *Clustered One Arm or $C-1$ Learner*

So far we talked about two algorithms, which in terms of sampling and arm selections are at two extremes. At one extreme is the IID learner, where at each time t, for each unit (node) the learner individually samples and selects arms. At the other extreme is the $G-1$ learner, where at each time t, for all N nodes, the learner jointly samples and selects the same arm for all nodes in the batch. We now propose a novel algorithm which is somewhat in-between of these two extremes, which we call the *Clustered One* arm learner or in short the $C-1$ learner.

The $C-1$ learner is based on the assumption that the nodes are partitioned into clusters. We already defined the details in Definition 2.3., however, we repeat the setup here for com-

pleteness. In particular, we assumed that the nodes in $\mathcal{I}$ can be partitioned into M disjoint clusters $\mathscr{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M\}$ such that $\bigcup_{m=1}^{M} \mathcal{C}_m = \mathcal{I}$ and $\mathcal{C}_m \cap \mathcal{C}_{m'} = \emptyset$ for all $m \neq m'$. Also let $N_m = |\mathcal{C}_m|$ be the number of nodes in cluster $\mathcal{C}_m$, then we have $\sum_{m=1}^{M} N_m = N$. Note that, this allows us to define $C-1$ arm vectors (see Definition 2.6), where the same arm is assigned to all nodes within a cluster, but different clusters can receive different arms.

The goal of the $C-1$ learner is same as the $G-1$ learner, that is to find the optimal $G-1$ arm vector. However, the $C-1$ learner leverages the cluster structure to improve exploration and learning. In particular, $C-1$ learner assumes that the clusters are informative about the batch average means. We formalize these assumptions in Assumption 2.20.

Assumption 2.20 (ASSUMPTIONS FOR CLUSTERED LEARNERS)**.**

- ***(C1) Cluster Mean Equality:*** *For any arm vector $\boldsymbol{k}_N \in \mathcal{W}^N$, the average of the node means in the cluster is the same as average of node means for the batch i.e.,*

$$\overline{\mu}_m(\boldsymbol{k}_N) = \overline{\mu}(\boldsymbol{k}_N) \quad \forall m \in [M] \tag{2.34}$$

  where $\overline{\mu}_m(\boldsymbol{k}_N) = \frac{1}{N_m} \sum_{i \in \mathcal{C}_m} \mu_i(\boldsymbol{k}_N)$ and $\overline{\mu}(\boldsymbol{k}_N) = \frac{1}{N} \sum_{i=1}^{N} \mu_i(\boldsymbol{k}_N)$.

- ***(C2) Cluster Mean Sufficiency:*** *For any cluster $\mathcal{C}_m$, the cluster average mean for cluster specific vector $\boldsymbol{k}_{N_m} \in \mathcal{W}^{N_m}$ is sufficient to learn about the cluster average mean for the corresponding $G-1$ arm vector $\boldsymbol{k}_N \in \mathcal{W}^N$, i.e.,*

$$\overline{\mu}_m(\boldsymbol{k}_N) = \overline{\mu}_m(\boldsymbol{k}_{N_m}, \bullet) \tag{2.35}$$

  where $\bullet$ indicates that the arms for nodes outside cluster $\mathcal{C}_m$ can be arbitrary any vector of size $N - N_m$.

The first assumption, cluster mean equality simply says that if we play arm $k \in \mathcal{W}$ for all N units in the batch, i.e., play arm vector $\boldsymbol{k}_N$, then average mean of any cluster is same as the batch average mean. This assumption allows us to use the rewards from any cluster to learn about the batch.

The second assumption, cluster mean sufficiency says that we don't have to play the same arm for all nodes in the batch to calculate $\overline{\mu}_m(\cdot)$, rather for any cluster $\mathcal{C}_m$, we can only focus on the nodes in that cluster and focus on arm vectors like $\boldsymbol{k}_{N_m} = k \cdot \mathbf{1}_{N_m}$ for some $k \in \mathcal{W}$, which means playing the same arm k for all nodes in that cluster. So this also means, the arms played outside that cluster doesn't matter. In other words, under this assumption, the cluster average mean when all N units in the batch play the same arm k, will be same as the cluster average mean when only the nodes in that cluster play arm k, and the nodes outside that cluster can play any arbitrary arms.

Note that, cluster mean sufficiency directly holds in the clustered interference structure in

Algorithm 8: TS $C-1$ Learner (General Case)

Input: $\mathcal{I}$, T, $\mathcal{W}$, $\{\mathcal{C}_m\}_{m\in[M]}$, $\widetilde{\theta}(k)$, $\forall k \in \mathcal{W}$
Initialize: $\widehat{\theta}_0(k) = \widetilde{\theta}(k)$, $\forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  for $m = 1, 2, \ldots, M$ **do**
    sample $\overline{\mu}'_t(k) \sim Q^{\widehat{\theta}_{t-1}(k)}$, $\forall k \in \mathcal{W}$
    select $k^{(m)} = \text{argmax}_{k\in\mathcal{W}} \overline{\mu}'_t(k)$
    set $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$ // here $\boldsymbol{k}_{N_m}^{(m)} = k^{(m)} \cdot \mathbf{1}_{N_m}$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$, $\forall i \in \mathcal{I}$
  update $\widehat{\theta}_t(k)$, $\forall k \in \mathcal{W}$ // for each k using rewards from nodes play arm k
Output: $\boldsymbol{W}_t$, $\boldsymbol{Y}_t$ $\forall t \in [T]$

Definition 2.3. The *clustered interference* means that the interference effects are contained within clusters. In our setting, this means that for each node $i \in \mathcal{C}_m$, the interference exists only from nodes within the same cluster $\mathcal{C}_m$. So if we assume that the interference is present through the mean functions $\mu_i(\cdot)$, then for $i \in \mathcal{C}_m$ clustered interference implies (see Remarks 2.4),

$$\mu_i\left(\boldsymbol{k}_{N_m}, \boldsymbol{k}_{-\mathcal{C}_m}\right) = \mu_i\left(\boldsymbol{k}_{N_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m}\right) \text{ for all } \boldsymbol{k}_{-\mathcal{C}_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m} \in \mathcal{W}^{N-N_m}, \tag{2.36}$$

where $\boldsymbol{k}_{N_m} \in \mathcal{W}^{N_m}$ is the arm vector for nodes in cluster $\mathcal{C}_m$ and $\boldsymbol{k}_{-\mathcal{C}_m} \in \mathcal{W}^{N-N_m}$ and $\widetilde{\boldsymbol{k}}_{-\mathcal{C}_m} \in \mathcal{W}^{N-N_m}$ are arbitrary arm vectors for nodes outside cluster $\mathcal{C}_m$. This means, in the presence of mean level clustered interference, if all nodes in cluster $\mathcal{C}_m$ play the same arm $k \in \mathcal{W}$, i.e., play arm vector $\boldsymbol{k}_{N_m} = k \cdot \mathbf{1}_{N_m}$, then for any $i \in \mathcal{C}_m$, the mean $\mu_i(\cdot)$ does not depend on the arms played by nodes outside $\mathcal{C}_m$. Now, since this holds for any node, this also holds for the average, so for any cluster $\mathcal{C}_m$, we have,

$$\overline{\mu}_m(\boldsymbol{k}_N) = \overline{\mu}_m\left(\boldsymbol{k}_{N_m}, \boldsymbol{k}_{-\mathcal{C}_m}\right) = \overline{\mu}_m\left(\boldsymbol{k}_{N_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m}\right) \text{ for all } \boldsymbol{k}_{-\mathcal{C}_m}, \widetilde{\boldsymbol{k}}_{-\mathcal{C}_m} \in \mathcal{W}^{N-N_m} \tag{2.37}$$

This allows us to have,

$$\overline{\mu}_m(\boldsymbol{k}_N) = \overline{\mu}_m\left(\boldsymbol{k}_{N_m}, \bullet\right) \tag{2.38}$$

We will see that the learner will only play arm vectors $\boldsymbol{k}_{N_m}$ for cluster $\mathcal{C}_m$ in the procedure. We outline the procedure for the $C-1$ learner in Algorithm 8.

The basic steps of the algorithm are mixture of IID and $G-1$ learner. The learner starts with a time horizon T, an arm set $\mathcal{W} = 1, 2, \ldots, K$, a batch $\mathcal{I}$ of N nodes partitioned into M clusters $\{\mathcal{C}_m\}_{m=1}^{M}$, and initialization values $\widetilde{\theta}(k)$ for the prior distributions of

each arm. At each round t, for each cluster $\mathcal{C}_m$, the learner draws a sample $\overline{\mu}'_t(k)$ from each arm's prior distribution $Q^{\widehat{\theta}_{t-1}(k)}$. Based on these samples, again for each cluster $\mathcal{C}_m$ separately, the learner selects the arm with the largest sampled mean, denoted with $k^{(m)} = \text{argmax}_{k\in\mathcal{W}} \overline{\mu}'_t(k)$. If you compare this with the IID learner, here the sampling is done similarly. For the IID learner the learner samples for each node separately, whereas here, it is done for each cluster separately. This gives in total of M samples for each arm. The selection step is also similar, for the IID learner selection has been done node-wise, but here, it is done cluster-wise. So again M times an arm k can be selected.

Next, the learner *assigns* the selected arm $k^{(m)}$ to all nodes in cluster $\mathcal{C}_m$, i.e., $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$.

Finally, the learner can pull the arm $\boldsymbol{W}_t = (\boldsymbol{W}_{\mathcal{C}_1,t}, \boldsymbol{W}_{\mathcal{C}_2,t}, \ldots, \boldsymbol{W}_{\mathcal{C}_M,t})$ and observe the rewards $Y_{i,t}$ from all nodes in the batch. After observing the rewards, the learner updates the prior parameters for all arms based on the observed rewards from all nodes with $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$. The updates are similar to the IID learner. In particular all arms will be updated. But the difference is here the update for any arm k is done by pooling the rewards from all clusters that plat arm k, not all nodes.

For any arm $k \in \mathcal{W}$ the cumulative rewards and count updates for the $C-1$ learner after time t are given as follows.

$$\begin{aligned} S_t(k) &= \sum_{s=1}^{t}\sum_{m=1}^{M} \mathbb{1}\left\{\boldsymbol{W}_{\mathcal{C}_m,s} = k\cdot\mathbf{1}_{N_m}\right\} \sum_{i\in\mathcal{C}_m} Y_{i,s} \\ &= S_{t-1}(k) + \sum_{m=1}^{M} \mathbb{1}\left\{\boldsymbol{W}_{\mathcal{C}_{m,t}} = k\cdot\mathbf{1}_{N_m}\right\} \sum_{i\in\mathcal{C}_m} Y_{i,t} \qquad (2.39) \\ n_t(k) &= \sum_{s=1}^{t}\sum_{m=1}^{M} \mathbb{1}\left\{\boldsymbol{W}_{\mathcal{C}_m,s} = k\cdot\mathbf{1}_{N_m}\right\} \cdot N_m \\ &= n_{t-1}(k) + \sum_{m=1}^{M} \mathbb{1}\left\{\boldsymbol{W}_{\mathcal{C}_{m,t}} = k\cdot\mathbf{1}_{N_m}\right\} \cdot N_m \qquad (2.40) \end{aligned}$$

Below we give two concrete examples of prior and posterior updates for the TS $C-1$ learner with Bernoulli and Gaussian rewards respectively.

Example 2.21 (TS $C-1$ Learner with Bernoulli Rewards)**.** Like the other Bernoulli case, for Bernoulli rewards, the $C-1$ learner uses Beta prior. In terms of the prior assumptions this is similar to the TS $G-1$ learner, but now sampling is done at the cluster level. The learner starts with prior parameters $\widetilde{\alpha}(k)$ and $\widetilde{\beta}(k)$ for all arm $k \in \mathcal{W}$, and initializes $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k)$, $\widehat{\beta}_0(k) = \widetilde{\beta}(k)$, $S_0(k) = 0$, and $n_0(k) = 0$. At each round $t = 1, 2, \ldots, T$, the learner now proceeds cluster by cluster. For each cluster m, it first *samples* $\overline{p}'_t(k) \sim \text{Beta}(\widehat{\alpha}_{t-1}(k), \widehat{\beta}_{t-1}(k))$ for each arm $k \in \mathcal{W}$. Then it *se-*

lects $k^{(m)} = \text{argmax}_{k \in \mathcal{W}} \overline{p}'_t(k)$ for cluster $\mathcal{C}_m$. These two steps are done for all clusters $m = 1, 2, \ldots, M$. After she has selected arms for all clusters, the learner sets the arm vector $\boldsymbol{W}_t$ such that $W_{i,t} = k^{(m)}$ for all units $i \in \mathcal{C}_m$. Then she *observes* rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, and finally *updates* the cumulative statistics for all arms but using the recursive version of the cluster-wise updates in eq. (2.39) and eq. (2.40), and then updates the posterior parameters for all arms $k \in \mathcal{W}$ using,

$$\begin{aligned} \widehat{\alpha}_t(k) &= \widetilde{\alpha}(k) + S_t(k) \text{ and} \\ \widehat{\beta}_t(k) &= \widetilde{\beta}(k) + n_t(k) - S_t(k) \end{aligned} \tag{2.41}$$

///

Example 2.22 (TS $C-1$ Learner with Gaussian Rewards)**.** For Gaussian rewards, the $C-1$ learner uses Gaussian prior. The learner starts with prior parameters $\widetilde{\mu}(k)$ and $\widetilde{\sigma}^2(k)$ for each arm $k \in \mathcal{W}$, and initializes $\widehat{\mu}_0(k) = \widetilde{\mu}(k)$, $\widehat{\sigma}^2_0(k) = \widetilde{\sigma}^2(k)$, $S_0(k) = 0$, and $n_0(k) = 0$. At each round $t = 1, 2, \ldots, T$, the learner now proceeds cluster by cluster. For each cluster m, it first *samples* $\overline{\mu}'_t(k) \sim \mathcal{N}(\widehat{\mu}_{t-1}(k), \widehat{\sigma}^2_{t-1}(k))$ for each arm $k \in \mathcal{W}$. Then it *selects* $k^{(m)} = \text{argmax}_{k \in \mathcal{W}} \overline{\mu}'_t(k)$ for cluster $\mathcal{C}_m$. These two steps are done for all clusters $m = 1, 2, \ldots, M$. After she has selected arms for all clusters, the learner sets the arm vector $\boldsymbol{W}_t$ such that $W_{i,t} = k^{(m)}$ for all units $i \in \mathcal{C}_m$. Then she *observes* rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, and finally *updates* the cumulative statistics for all arms but using cluster-wise updates in eq. (2.39) and eq. (2.40), and then updates the posterior parameters for all arms $k \in \mathcal{W}$ using,

$$\begin{aligned} \widehat{\sigma}^2_t(k) &= \left(\frac{n_t(k)}{\sigma^2(k)} + \frac{1}{\widetilde{\sigma}^2(k)} \right)^{-1} \text{ and} \\ \widehat{\mu}_t(k) &= \left(\frac{S_t(k)}{\sigma^2(k)} + \frac{\widetilde{\mu}(k)}{\widetilde{\sigma}^2(k)} \right) \widehat{\sigma}^2_t(k) \end{aligned} \tag{2.42}$$

///

Note that in both examples, the updates have the same form as the IID (given in eq. (2.27) and eq. (2.28)), and $G-1$ learners (given in eq. (2.31) and eq. (2.33)). However unlike the $G-1$ learners, $C-1$ learners update all arms at every round and unlike the IID learners, here the cumulative statistics are aggregated at the cluster level. We now discuss some important aspects of the TS $C-1$ learner in Remarks 2.23.

Remarks 2.23 (Discussion on TS $C-1$ Learner)**.**

Sampling, Selection and Update: The $C-1$ learner falls in between the IID and $G-1$

learner. In terms of sampling this is similar to the IID learner, difference is here for each arm $k \in \mathcal{W}$, sampling is done M times (once for each cluster) rather than N. Selection is also similar, same arm can be selected for multiple times, maximum M. In terms of setting the arms to nodes, this is similar to $G-1$ learner, difference is here the same arm is set only within a cluster rather than for all nodes. Update is also similar to IID learner, all arms are updated at each time t.

Generalizes both IID and $G-1$ Learners: It generalizes both the IID and $G-1$ learners as special cases. In particular, if we set $M = N$ (i.e., each cluster has one node), then the $C-1$ learner reduces to the IID learner. On the other hand, if we set $M = 1$ (i.e., all nodes belong to a single cluster), then the $C-1$ learner reduces to the $G-1$ learner.

Eventually Reaching to Optimal $G-1$ Arm Vector: This is same as the $G-1$ learner, but initially the exploration is better than the $G-1$ learner since different clusters can play different arms at each time t, and this is why suffers less starting regret than the $G-1$ learner (we explain this more below).

Strength of Interference Structure, Particularly Clustered Interference: The $C-1$ learner is motivated from the idea that *the strength of interference decays or vanishes outside the cluster*. As we explained this known as *clustered interference*. By playing a $C-1$ arm vectors, we can explore multiple $G-1$ arms at the same time in different clusters and under Assumption 2.20, learn the optimal $G-1$ arm vector. So this algorithm is particularly useful when the interference is clustered.

Solves High Starting Regret and Misleading Updates Issues: Since the $C-1$ learner has better exploration than the $G-1$, this gives some advantage that it doesn't suffer from the high starting regret like the $G-1$ learner, since it will explore all arms in different clusters at the beginning. Also, if we compare this to IID learner then, given that clustering can properly capture the interference structure, it won't suffer from misleading updates like IID learner. So it can be good choice when the interference is clustered.

Requires Information about Clustering: One of the limitations of the $C-1$ Learner is it requires the information regarding clusters, in particulars clusters related to the interference structure. In other words, to properly run the $C-1$ learner, we need to know the clusters such that the interference is mostly contained within clusters. In practice, we often don't have enough information about the interference structure in the environment, so this creates a challenge in applying the $C-1$ learner. In this case the algorithm suffers misleading updates from interfering units outside the cluster. In practice, it is possible to use any off the shelf clustering techniques (e.g., K-means, hierarchical clustering etc.) based on some other variables related to the nodes. And perhaps also possible to correct the clusters over time based on the observed rewards. However, we think these are

beyond the scope of this work and we leave these for future research.

///

2.3.4 Clustered Multi Arm or $C-M$ Learner

The $G-1$ and $C-1$ learners both aim to find the optimal $G-1$ arm vector. This suggests when our target is to minimize the $G-1$ regret, these learners or algorithms are reasonable to run. However, when the global optimal arm vector is not a $G-1$ arm vector, we can do better by exploring in a different action space. As we discussed in Section 2.2.3 the best solution is to find $\boldsymbol{k}^* = \text{argmax}_{\boldsymbol{k}\in\mathcal{W}^N} \sum_{i\in\mathcal{I}} \mu_i(\boldsymbol{k})$. However, in many practical scenarios, in general finding this optimal arm vector can be computationally very expensive since the action space $\mathcal{W}^N$ grows exponentially with N. We now propose an algorithm which finds the best $C-1$ arm vector defined in Definition 2.6 assuming clustered interference structure for a given partition of units in $\mathscr{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_M\}$. We call this algorithm the *Clustered Multi Arm Learner or $C-M$ Learner*. The idea is for each cluster the learner will find the best uniform assignment for that cluster, and then combine these cluster-specific uniform assignments to form a $C-1$ arm vector for the whole batch. Recall that the $C-1$ optimal arm vector for a given partition $\mathscr{C}$, is defined as,

$$\boldsymbol{k}^*_{N,C-1} = \underset{\boldsymbol{k}_{N,C-1}\in\mathcal{W}^{c1}}{\text{argmax}} \sum_{i\in\mathcal{I}} \mu_i(\boldsymbol{k}_{N,C-1})$$

where $\mathcal{W}^{c1}$ is the set of all $C-1$ arm vectors. We showed a detailed example of $C-1$ arm vectors and the optimal $C-1$ arm vector in Example 2.7 and Example 2.10 with our motivating crime reduction scenario and in that case for three clusters the optimal $C-1$ arm vector is given by,

$$\boldsymbol{k}^*_{N,C-1} = (\underbrace{2,2,\ldots,2}_{N_1 \text{ times}}, \underbrace{3,3,\ldots,3}_{N_2 \text{ times}}, \underbrace{1,1,\ldots,1}_{N_3 \text{ times}})$$

Now when we have clustered interference structure the optimal $C-1$ arm vector can be written in terms of cluster-specific uniform optimal assignments as follows,

$$\boldsymbol{k}^*_{N,C-1} = (\boldsymbol{k}^{(1)*}_{N_1}, \boldsymbol{k}^{(2)*}_{N_2}, \ldots, \boldsymbol{k}^{(M)*}_{N_M})' \tag{2.43}$$

where for example $\boldsymbol{k}^{(1)*}_{N_1}$ is the optimal uniform arm vector for cluster $\mathcal{C}_1$ of cluster size N_1. Simply put, eq. (2.43) says - for each cluster if we find the best uniform arm assignments of size N_m, then combining these cluster-specific uniform assignments will give us the optimal $C-1$ arm vector for the whole batch. Note that this only happens under the clustered interference structure (see example Example 2.10), otherwise this may not hold.

Algorithm 9: TS $C-M$ Learner (General Case)

Input: $\mathcal{I}$, T, $\mathcal{W}$, $\{\mathcal{C}_m\}_{m\in[M]}$, $\widetilde{\theta}_m(k)$, $\forall m \in [M]$ and $\forall k \in \mathcal{W}$
Initialize: $\widehat{\theta}_{m,0}(k) = \widetilde{\theta}_m(k)$, $\forall m \in [M]$ *and* $\forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  for $m = 1, 2, \ldots, M$ **do**
    sample $\overline{\mu}'_{m,t}(k) \sim Q^{\widehat{\theta}_{m,t-1}(k)}$, $\forall k \in \mathcal{W}$
    select $k^{(m)} = \text{argmax}_{k\in\mathcal{W}}\, \overline{\mu}'_{m,t}(k)$
    set $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$ // here $\boldsymbol{k}^{(m)}_{N_m} = k^{(m)} \cdot \mathbf{1}_{N_m}$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$, $\forall i \in \mathcal{I}$
  for $m = 1, 2, \ldots, M$ **do**
    update $\widehat{\theta}_{m,t}(k^{(m)})$ // only for played arm $k^{(m)}$ in $\mathcal{C}_m$ using rewards from $\mathcal{C}_m$
Output: $\boldsymbol{W}_t$, $\boldsymbol{Y}_t$ $\forall t \in [T]$

The $C-M$ learner will try to exploit this structure to find the optimal $C-1$ arm vector. The only assumption for the $C-M$ learner is that the interference is clustered as defined in Definition 2.3. We outline the procedure for the $C-M$ learner in Algorithm 9.

Clustered interference assumption allows us to focus on cluster specific distributions. In particular, let $\overline{Y}_{m,t}(\boldsymbol{k}_{N_m})$ be the cluster average reward for cluster $\mathcal{C}_m$ at time t when all units in the cluster is assigned to arm k (or we can write $\boldsymbol{k}_{N_m}$). At the distributional level now we have $\overline{Y}_{m,t}(\boldsymbol{k}_{N_m}) \sim \mathrm{F}_{\overline{\mu}_m(\boldsymbol{k}_{N_m})}$, where $\overline{\mu}_m(\boldsymbol{k}_{N_m})$ is the mean of the cluster. The $C-M$ learner places priors on this mean $\overline{\mu}_m(\boldsymbol{k}_{N_m})$ for each cluster $\mathcal{C}_m$, and updates this prior based on the observed rewards from the nodes *only* belong that cluster. We denote the cluster-specific prior for cluster m and arm k with $Q^{\widehat{\theta}_{m,t}(k)}$, where $\widehat{\theta}_{m,t}(k)$ is the parameter updated at time t.

The learner shares the similar steps like the other learners, but differs overall since we have M separate learners for M clusters now. The learner starts with a time horizon T, an arm set $\mathcal{W} = \{1, 2, \ldots, K\}$, a batch $\mathcal{I}$ of N nodes partitioned into M clusters $\{\mathcal{C}_m\}_{m=1}^{M}$, and crucially, *cluster-specific* initialization values $\widetilde{\theta}_m(k)$ for the prior distributions of each arm k in each cluster m. This means the learner maintains $M \times K$ separate prior distributions, one for each cluster-arm pair, rather than K shared priors as the previous learners.

At each round t, for each cluster $\mathcal{C}_m$, the learner draws a cluster specific sample $\overline{\mu}'_{m,t}(k)$ from the cluster's own prior distribution $Q^{\widehat{\theta}_{m,t-1}(k)}$ for each arm $k \in \mathcal{W}$. Based on these cluster-specific samples, the learner selects the arm with the largest sampled mean for that particular cluster, $k^{(m)} = \text{argmax}_{k\in\mathcal{W}}\, \overline{\mu}'_{m,t}(k)$. The selected arm is then assigned to all nodes within cluster $\mathcal{C}_m$. This is the similar to $G-1$ learner but now, for each cluster we have a separate $G-1$ learner.

After one arm gets selected for each cluster, the learner forms $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$, and finally she pulls the arm vector $\boldsymbol{W}_t = (\boldsymbol{W}_{\mathcal{C}_1,t}, \boldsymbol{W}_{\mathcal{C}_2,t}, \ldots, \boldsymbol{W}_{\mathcal{C}_M,t})$, and observe the rewards $Y_{i,t}$ from all nodes in the batch. After observing rewards $Y_{i,t}$ from all nodes in the batch, the learner updates *only the prior corresponding to the played arm for each cluster*. Specifically, for cluster $\mathcal{C}_m$ that played arm $k^{(m)}$, only the parameter $\widehat{\theta}_{m,t}(k^{(m)})$ is updated using the rewards from nodes in $\mathcal{C}_m$. The priors for arms not played in that cluster remain unchanged. This is a key distinction from the $C-1$ learner, where the $C-1$ learner updates all arms (pooled across clusters), and also similar to the $G-1$ learner, where only the played arm is updated.

Since now we have M learners (rather than one learner) for M clusters, the cumulative rewards and cumulative counts should be computed separately for each learner. So after time t, for cluster $\mathcal{C}_m$, the cumulative rewards and counts for played arm $k^{(m)} \in \mathcal{W}$ are given as follows,

$$\begin{aligned} S_{m,t}(k^{(m)}) &= \sum_{s=1}^{t} \mathbb{1}\{\boldsymbol{W}_{\mathcal{C}_m,s} = k^{(m)} \cdot \mathbf{1}_{N_m}\} \sum_{i \in \mathcal{C}_m} Y_{i,s} \\ &= S_{m,t-1}(k^{(m)}) + \sum_{i \in \mathcal{C}_m} Y_{i,t} \end{aligned} \tag{2.44}$$

$$\begin{aligned} n_{m,t}(k^{(m)}) &= \sum_{s=1}^{t} \mathbb{1}\{\boldsymbol{W}_{\mathcal{C}_m,s} = k^{(m)} \cdot \mathbf{1}_{N_m}\} N_m \\ &= n_{m,t-1}(k^{(m)}) + N_m \end{aligned} \tag{2.45}$$

Below we give the details for the Bernoulli and Gaussian rewards case for the $(C-M)$ Learner. The details are mixture of $G-1$ and $C-1$ learners.

Example 2.24 (TS $C-M$ Learner with Bernoulli Rewards)**.** For Bernoulli rewards, the $C-M$ learner models the cluster-level average reward using a Beta prior. In terms of the prior assumptions this is similar to the $G-1$ learner, but now placed for each cluster separately. Let $\overline{p}_m(\boldsymbol{k}_{N_m}) := \mathbb{E}[\overline{Y}_{m,t}(\boldsymbol{k}_{N_m})]$. Similar to $G-1$, assume $\overline{p}_m(\boldsymbol{k}_{N_m}) \in [0,1]$ follows a Beta prior. From here, we again write $\overline{p}_m(k)$ instead of $\overline{p}_m(\boldsymbol{k}_{N_m})$ for simplicity since the cluster uniform assignments $\boldsymbol{k}_{N_m}$ can be indexed by $k \in \mathcal{W}$. The learner starts with prior parameters $\widetilde{\alpha}_m(k)$ and $\widetilde{\beta}_m(k)$, for each cluster $m \in [M]$, for each arm $k \in \mathcal{W}$, and initializes $\widehat{\alpha}_{m,0}(k) = \widetilde{\alpha}_m(k)$, $\widehat{\beta}_{m,0}(k) = \widetilde{\beta}_m(k)$, $S_{m,0}(k) = 0$, and $n_{m,0}(k) = 0$ for all clusters $m \in \{1, \ldots, M\}$. At each round $t = 1, 2, \ldots, T$, the learner proceeds cluster by cluster. For each cluster m, it first *samples* $\overline{p}'_{m,t}(k) \sim \text{Beta}(\widehat{\alpha}_{m,t-1}(k), \widehat{\beta}_{m,t-1}(k))$ for each arm $k \in \mathcal{W}$. Then it *selects* $k^{(m)} = \text{argmax}_{k \in \mathcal{W}} \overline{p}'_{m,t}(k)$ for cluster $\mathcal{C}_m$. After selecting arms for all clusters, the learner sets the arm vector $\boldsymbol{W}_t$ such that $W_{i,t} = k^{(m)}$ for all units $i \in \mathcal{C}_m$. After *observing* rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, the learner *updates* the cumulative statistics for all arms using eq. (2.44) and eq. (2.45), and then updates the posterior parameters only for the chosen arm $k^{(m)}$ in each cluster $m \in \{1, \ldots, M\}$ using,

$$\begin{aligned}\widehat{\alpha}_{m,t}(k^{(m)}) &= \widetilde{\alpha}_m(k^{(m)}) + S_{m,t}(k^{(m)}) \text{ and} \\ \widehat{\beta}_{m,t}(k^{(m)}) &= \widetilde{\beta}_m(k^{(m)}) + n_{m,t}(k^{(m)}) - S_{m,t}(k^{(m)}).\end{aligned} \tag{2.46}$$

Note that similar to the $G-1$ learner, the $C-M$ learner only updates the played arm for each cluster, and definitely this is different from the $C-1$ learner which updates all arms at each round.

///

Example 2.25 (TS $C-M$ Learner with Gaussian Rewards)**.** For Gaussian rewards, the $C-M$ learner models the cluster-level average reward using a Gaussian prior. Let $\overline{\mu}_m(\boldsymbol{k}_{N_m}) := \mathbb{E}[\overline{Y}_{m,t}(\boldsymbol{k}_{N_m})]$. The cluster average reward follows $\overline{Y}_{m,t}(\boldsymbol{k}_{N_m}) \sim \mathcal{N}(\overline{\mu}_m(\boldsymbol{k}_{N_m}), \overline{\sigma}^2_m(\boldsymbol{k}_{N_m}))$, where $\overline{\sigma}^2_m(\boldsymbol{k}_{N_m})$ is the known variance of the cluster average. Similar to the $G-1$ case, we assume interference only affects the mean and not the variance. We write $\overline{\mu}_m(k)$ and $\overline{\sigma}^2_m(k)$ instead of $\overline{\mu}_m(\boldsymbol{k}_{N_m})$ and $\overline{\sigma}^2_m(\boldsymbol{k}_{N_m})$ for simpler notation. The learner starts with prior parameters $\widetilde{\mu}_m(k)$ and $\widetilde{\sigma}^2_m(k)$ for each arm $k \in \mathcal{W}$, and initializes $\widehat{\mu}_{m,0}(k) = \widetilde{\mu}_m(k)$, $\widehat{\sigma}^2_{m,0}(k) = \widetilde{\sigma}^2_m(k)$, $S_{m,0}(k) = 0$, and $n_{m,0}(k) = 0$ for all clusters $m \in \{1, \ldots, M\}$. At each round $t = 1, 2, \ldots, T$, the learner proceeds cluster by cluster. For each cluster m, it first *samples* $\overline{\mu}'_{m,t}(k) \sim \mathcal{N}(\widehat{\mu}_{m,t-1}(k), \widehat{\sigma}^2_{m,t-1}(k))$ for each arm $k \in \mathcal{W}$. Then it *selects* $k^{(m)} = \text{argmax}_{k \in \mathcal{W}} \overline{\mu}'_{m,t}(k)$ for cluster $\mathcal{C}_m$. After selecting arms for all clusters, the learner sets the arm vector $\boldsymbol{W}_t$ such that $W_{i,t} = k^{(m)}$ for all units $i \in \mathcal{C}_m$. After *observing* rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, the learner *updates* the cumulative statistics for arm $k^{(m)}$ that is played for the cluster $\mathcal{C}_m$ at time t using eq. (2.44) and eq. (2.45), and then updates the posterior parameters for arm $k^{(m)} \in \mathcal{W}$ using,

$$\begin{aligned}\widehat{\sigma}^2_{m,t}(k^{(m)}) &= \left(\frac{n_{m,t}(k^{(m)})}{\sigma^2(k^{(m)})} + \frac{1}{\widetilde{\sigma}^2_m(k^{(m)})}\right)^{-1} \text{ and} \\ \widehat{\mu}_{m,t}(k^{(m)}) &= \left(\frac{S_{m,t}(k^{(m)})}{\sigma^2(k^{(m)})} + \frac{\widetilde{\mu}_m(k^{(m)})}{\widetilde{\sigma}^2_m(k^{(m)})}\right) \widehat{\sigma}^2_{m,t}(k^{(m)})\end{aligned} \tag{2.47}$$

///

Below we discuss some important aspects of the TS $C-M$ learner in Remarks 2.26.

Remarks 2.26 (Discussion on TS $C-M$ Learner)**.**

Sampling, Selection and Update: The $C-M$ learner can be seen as running M separate $G-1$ learners, one for each cluster. In terms of sampling, selection and update this exactly like the $G-1$ learner, difference is here everything is done M times (one for each cluster). So number of parameters to be learned is $M \times K$ rather than K in $G-1$

(or $C-1$) learner. This makes this learner a little less efficient than the $G-1$ learner, however we aim to find the optimal $C-1$ arm vector which is a combination arm vector and in general setting can give higher reward for the batch sum than the optimal $G-1$ arm vector (we illustrated this with our motivating example in Example 2.10).

Eventually Reaching to $C-1$ Arm Vector: Since the $C-M$ learner is running M separate $G-1$ learners, for each cluster the algorithm will find the cluster-wise best uniform assignment. So eventually the learner will find a $C-1$ arm vector and if the interference is clustered, this will be the optimal $C-1$ arm vector.

Finds Optimal $C-1$ Arm Vector Under Clustered Interference: Since we have K arms and M clusters, the total number of $C-1$ arm vectors is K^M. So we could loop over all K^M possible $C-1$ arm vectors to find the optimal $C-1$ arm vector, this is how we defined the $C-1$ optimal arm vector in Definition 2.6. However, under clustered interference we could go for each cluster separately and this would reduce the search space from K^M to $M \times K$, since for each cluster we only need to find the best arm from K cluster-wise uniform assignments. The $C-M$ learner exploits this structure to find the optimal $C-1$ arm vector efficiently.

Required Knowledge of Clusters: Like the $C-1$ learner, the key limitation of the $C-M$ learner is it requires the information regarding clusters, in particulars clusters related to the interference structure. In other words, to properly run the $C-M$ learner, we need to know the clusters such that the interference is mostly contained within clusters. However as we discussed in Remarks 2.23, in practice, we often don't have enough information about the clustered interference structure but it is possible to use any off the shelf clustering techniques.

No Advantage for $G-1$ Optimal: If the $G-1$ optimal and $C-1$ optimal vectors are same, then running the $C-M$ learner will not give any advantage over the $G-1$ learner and in fact will be less efficient due to more parameters to learn. This is the case in IID interference setup in Assumption 2.12 or for the no optimal switching case, discussed in example Example 2.11.

///

2.4 Bayesian Bandit with Interference & Bayesian Regret Analysis

In this section we first define the Bayesian bandit model and Bayesian regret under interference, and then do the Bayesian regret analysis for the $G-1$ learner. We recall, from the frequentist point of view, the environment is defined by $\mathscr{F}_{\boldsymbol{\mu}} = \{\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$, where

$\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})}$ is the joint distribution with mean vector $\boldsymbol{\mu}(\boldsymbol{k}) = (\mu_1(\boldsymbol{k}), \mu_2(\boldsymbol{k}), \ldots, \mu_N(\boldsymbol{k}))'$ when the learner pulls arm vector $\boldsymbol{k}$ and observes reward vector $\boldsymbol{Y}_t(\boldsymbol{k})$, i.e., we have $\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})}$ and the set $\mathscr{F}_{\boldsymbol{\mu}}$ contains all such joint distributions for all possible arm vectors $\boldsymbol{k} \in \mathcal{W}^N$. In the Bayesian setup, we just need the additional idea that the frequentist environment $\mathscr{F}_{\boldsymbol{\mu}}$ is an instance that is randomly drawn form a set of possible instances, and there is a prior distribution over all possible instances. First, we define few notations, for a fixed arm vector $\boldsymbol{k}$ the set of all possible mean vectors is denoted with $\boldsymbol{\mathcal{E}}(\boldsymbol{k})$, and the prior distribution over these mean vectors is denoted with $Q_{\boldsymbol{k}}$. Then, we denote the set of all possible mean vector combinations for all arm vectors with $\boldsymbol{\mathcal{E}} = \prod_{\boldsymbol{k} \in \mathcal{W}^N} \boldsymbol{\mathcal{E}}(\boldsymbol{k})$ and then $\boldsymbol{Q} := \prod_{\boldsymbol{k} \in \mathcal{W}^N} Q_{\boldsymbol{k}}$ as the product distribution of all prior distributions. Now, for a fixed instance $\boldsymbol{\nu} \in \boldsymbol{\mathcal{E}}$, the set of all possible joint distributions can be denoted with $\mathscr{F}_{\boldsymbol{\nu}} = \{\mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$, where $\mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})}$ is the joint distribution for arm vector $\boldsymbol{k}$. $\mathscr{F}_{\boldsymbol{\nu}}$ is exactly like the frequentist bandit environment $\mathscr{F}_{\boldsymbol{\mu}}$, but now we define it for a fixed instance $\boldsymbol{\nu}$[‡‡]. Then the set of joint distributions for all possible instances can be denoted with $\mathscr{F}_{\mathcal{B}} = \{\mathscr{F}_{\boldsymbol{\nu}} : \boldsymbol{\nu} \in \boldsymbol{\mathcal{E}}\}$. The Bayesian bandit environment with interference can now be defined as the triplet $(\boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathscr{F}_{\mathcal{B}})$. We give the formal definition in Definition 2.27. In terms of the Bayesian structure, the definition is similar to the setup without interference Definition 1.9, but now we have interference and vector arm assignments. In terms of the interaction between the learner and the environment and the other components, the definition is also similar to the frequentist Bandit model with interference that we have defined in Definition 2.5, except now we have a prior distribution over all possible instances and at time $t = 0$ only one bandit instance is randomly drawn to the learner.

‡‡ In the Bayesian setup we index the distributions with the instance that is drawn, definitely for a fixed instance, we can also index with the mean like in the frequentist setup, but since the mean vectors change depending on the instances, we don't do it here.

Definition 2.27 (BAYESIAN BANDIT MODEL UNDER INTERFERENCE)**.** *A Bayesian bandit environment under interference is defined by a tuple* $(\boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathscr{F}_{\mathcal{B}})$ *where* $\boldsymbol{\mathcal{E}}$ *is the parameter space (set of all possible parameter vectors for the batch),* $\boldsymbol{Q}$ *is the prior distribution over* $\boldsymbol{\mathcal{E}}$ *and* $\mathscr{F}_{\mathcal{B}} := \{\mathscr{F}_{\boldsymbol{\nu}} : \boldsymbol{\nu} \in \boldsymbol{\mathcal{E}}\}$ *is the collection of all set of reward distributions, where for each instance* $\boldsymbol{\nu} \in \boldsymbol{\mathcal{E}}$*, we have* $\mathscr{F}_{\boldsymbol{\nu}} := \{\mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$ *as the set of all reward distributions for all arm vectors, and* $\mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})}$ *is the joint distribution for arm vector* $\boldsymbol{k}$*. A Bayesian bandit model is a repeated interaction between a learner and a Bayesian bandit environment where,*

for $t = 0$ **do**
  a bandit instance $\boldsymbol{\nu} \sim \boldsymbol{Q}$ *is drawn for the learner from the parameter space* $\boldsymbol{\mathcal{E}}$ *according to prior* $\boldsymbol{Q}$*.*
for $t = 1, 2, \ldots, T$ **do**
  the learner selects an arm vector $\boldsymbol{W}_t \in \mathcal{W}^N$ *for the batch*
  the environment generates potential reward vector $\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})}$ *for all* $\boldsymbol{k} \in \mathcal{W}^N$*, where* $\boldsymbol{\nu}$ *is a fixed instance drawn at* $t = 0$
  the learner observes only the realized outcome $\boldsymbol{Y}_t = \boldsymbol{Y}_t(\boldsymbol{W}_t)$ *for instance* $\boldsymbol{\nu}$*, not the counterfactual outcomes* $\boldsymbol{Y}_t(\boldsymbol{k})$ *for* $\boldsymbol{k} \neq \boldsymbol{W}_t$ *for instance* $\boldsymbol{\nu}$ *or any outcomes for any other instance* $\widetilde{\boldsymbol{\nu}} \neq \boldsymbol{\nu}$

Interaction Protocol 4: *Bayesian Bandit With Interference Interaction*

We have the following information structure,

- ***Known or Observable to the Learner:*** *The learner knows the set of arm vectors* $\mathcal{W}^N$*, time horizon* T*, parameter space* $\boldsymbol{\mathcal{E}}$*, prior* $\boldsymbol{Q}$ *and also class of distributions in* $\mathscr{F}_{\mathcal{B}}$*, but does not know which* $\boldsymbol{\nu}$ *appeared at time* $t = 0$*. She observes all actions and outcomes till time* T*, i.e.,* $\{(\boldsymbol{W}_t, \boldsymbol{Y}_t)\}_{t=1}^T$*.*
- ***Known to the Environment:*** *The environment knows the complete set of counterfactual distributions* $\mathscr{F}_{\mathcal{B}} = \{\mathscr{F}_{\boldsymbol{\nu}} : \boldsymbol{\nu} \in \boldsymbol{\mathcal{E}}\}$*, with* $\mathscr{F}_{\boldsymbol{\nu}} := \{\mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$ *for each* $\boldsymbol{\nu} \in \boldsymbol{\mathcal{E}}$ *and observes what the learner observes.*

With the Bayesian bandit environment under interference defined above, we can now define the Bayesian regret for a policy π. We define the $G-1$ Bayesian regret below,

Definition 2.28 (BAYESIAN $G-1$ REGRET UNDER INTERFERENCE)**.** *Consider a Bayesian bandit environment under interference* $(\boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathscr{F}_{\mathcal{B}})$*, for a policy* π *interacting with this environment over time horizon* T*, the Bayesian* $G-1$ *regret after horizon* T *is defined as*

$$\mathcal{BR}_T^{g1}(\pi, \boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathscr{F}_{\mathcal{B}}) = \mathbb{E}_{\boldsymbol{\nu} \sim \boldsymbol{Q}, \pi}\left[\sum_{t=1}^{T} \left(\mu_{\boldsymbol{\nu},\Sigma}\left(\boldsymbol{k}_N^*\right) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{W}_t)\right)\right] \tag{2.48}$$

where $\mu_{\boldsymbol{\nu},\Sigma}\left(\boldsymbol{k}_N^*\right) = \max_{\boldsymbol{k} \in \mathcal{W}^{g1}} \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k})$ *is the maximum batch sum of expected reward over all* $G-1$ *arm vectors for the instance* $\boldsymbol{\nu}$ *drawn from prior* $\boldsymbol{Q}$*, and* $\boldsymbol{W}_t$ *is the arm vector selected by policy* π *at time* t*. The expectation is taken over both the randomness in the bandit instance* $\boldsymbol{\nu}$ *drawn from prior* $\boldsymbol{Q}$ *and the randomness in the policy* π*.*

This regret can be similarly explained as we discussed after Definition 1.11, since this fol-

lows similarly, we skip the details here. With the Bayesian $G-1$ regret defined above, we now do the Bayesian regret analysis for the TS $G-1$ learner defined in Algorithm 7. First, we need the concentration lemma for batch sums when playing $G-1$ arm vectors. Essentially, we need to show that the good event holds with high probability. We state and prove the lemma below. But before that, let's clear with some notations. If we assume for each $G-1$ arm vector $\boldsymbol{k}_N \in \mathcal{W}^{g1}$, the joint reward vector follows Gaussian distribution, i.e., $\boldsymbol{Y}_t(\boldsymbol{k}_N) = (Y_{1,t}(\boldsymbol{k}_N), \ldots, Y_{N,t}(\boldsymbol{k}_N))^\top \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{k}_N), \boldsymbol{\Omega}(\boldsymbol{k}_N))$, then the sum $\boldsymbol{Y}_{\Sigma,t}(\boldsymbol{k}_N)$ also follows Gaussian, i.e.,

$$\boldsymbol{Y}_{\Sigma,t}(\boldsymbol{k}_N) \sim \mathcal{N}\left(\mu_\Sigma(\boldsymbol{k}_N), \sigma^2_\Sigma(\boldsymbol{k}_N)\right),$$

where the batch variance $\sigma^2_\Sigma(\boldsymbol{k}_N) = \mathbf{1}_N^\top \boldsymbol{\Omega}(\boldsymbol{k}_N)\mathbf{1}_N$ expands to:

$$\sigma^2_\Sigma(\boldsymbol{k}_N) = \sum_{i=1}^{N} \sigma_i^2(\boldsymbol{k}_N) + 2\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \sigma_i(\boldsymbol{k}_N)\sigma_j(\boldsymbol{k}_N)\rho_{ij}(\boldsymbol{k}_N).$$

Here, $\sigma_i^2(\boldsymbol{k}_N) = \mathbb{V}(Y_{i,t}(\boldsymbol{k}_N))$ and $\rho_{ij}(\boldsymbol{k}_N) = \text{Corr}\left(Y_{i,t}(\boldsymbol{k}_N), Y_{j,t}(\boldsymbol{k}_N)\right)$

In the following we let $N_{t-1}(\boldsymbol{k}_N)$ be the number of times arm vector $\boldsymbol{k}_N$ has been played up to time $t-1$.

Lemma 3 (Concentration for Batch Sum Rewards For $G-1$ Learner)**.** *Consider a Bayesian bandit environment under interference $(\boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathcal{F}_\mathcal{B})$ for N units, K arms and horizon $T \geq 2$. For a fixed instance $\boldsymbol{\nu} \sim \boldsymbol{Q}$, under Gaussian reward assumption for any $G-1$ arm vector $\boldsymbol{k}_N \in \mathcal{W}^{g1}$, the reward sum $\boldsymbol{Y}_{\Sigma,t}(\boldsymbol{k}_N) \sim \mathcal{N}\left(\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N), \sigma^2_\Sigma(\boldsymbol{k}_N)\right)$. For a fixed $\delta > 0$, assuming $N_{t-1}(\boldsymbol{k}_N) \geq 1$ for all $\boldsymbol{k}_N \in \mathcal{W}^{g1}$, define the clean event for the batch sum as,*

$$\mathbf{E}_\Sigma(\delta) = \left\{\forall t \in [T], \forall \boldsymbol{k}_N \in \mathcal{W}^{g1} : \left|\overline{\boldsymbol{Y}}_{\Sigma,t-1}(\boldsymbol{k}_N) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N)\right| < \sigma_\Sigma(\boldsymbol{k}_N)\sqrt{\frac{2\log(1/\delta)}{N_{t-1}(\boldsymbol{k}_N)}}\right\} \tag{2.49}$$

where $\overline{\boldsymbol{Y}}_{\Sigma,t-1}(\boldsymbol{k}_N)$ is empirical mean of the batch sum for arm vector $\boldsymbol{k}_N$ played up to time $t-1$, $\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N)$ and $\sigma^2_\Sigma(\boldsymbol{k}_N)$ are the true mean and variance of the batch sum for playing arm vector $\boldsymbol{k}_N$ under fixed instance $\boldsymbol{\nu}$. Then conditional on $\boldsymbol{\nu}$ we can show that

$$\mathbb{P}\left(\mathbf{E}_\Sigma(\delta)^c\right) \leq 2TK\delta \tag{2.50}$$

Proof. The proof follows similarly to Lemma 2 in Chapter 1. However, here we need to take care of the fact that we are only playing $G-1$ arm vectors. For a fixed $G-1$ vector $\boldsymbol{k}_N$, the clean event can be written as,

$$\mathbf{E}_{\Sigma}(\delta, \boldsymbol{k}_N) = \left\{ \forall t \in [T] : \left| \overline{\boldsymbol{Y}}_{\Sigma,t-1}(\boldsymbol{k}_N) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N) \right| < \sigma_{\Sigma}(\boldsymbol{k}_N) \sqrt{\frac{2\log(1/\delta)}{N_{t-1}(\boldsymbol{k}_N)}} \right\} \tag{2.51}$$

$$= \bigcap_{t=1}^{T} \left\{ \left| \overline{\boldsymbol{Y}}_{\Sigma,t-1}(\boldsymbol{k}_N) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N) \right| < \sigma_{\Sigma}(\boldsymbol{k}_N) \sqrt{\frac{2\log(1/\delta)}{N_{t-1}(\boldsymbol{k}_N)}} \right\} \tag{2.52}$$

Note there is a variance term $\sigma^2_{\Sigma}(\boldsymbol{k}_N)$ in the concentration bound, this is the variance of the batch sum of the rewards here. Now we look at the complement event $\mathbf{E}(\delta, \boldsymbol{k}_N)^c$.

$$\mathbf{E}_{\Sigma}(\delta, \boldsymbol{k}_N)^c = \left\{ \exists t \in [T] : \left| \overline{\boldsymbol{Y}}_{\Sigma,t-1}(\boldsymbol{k}_N) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N) \right| \geq \sigma_{\Sigma}(\boldsymbol{k}_N) \sqrt{\frac{2\log(1/\delta)}{N_{t-1}(\boldsymbol{k}_N)}} \right\} \tag{2.53}$$

$$= \bigcup_{t=1}^{T} \left\{ \left| \overline{\boldsymbol{Y}}_{\Sigma,t-1}(\boldsymbol{k}_N) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N) \right| \geq \sigma_{\Sigma}(\boldsymbol{k}_N) \sqrt{\frac{2\log(1/\delta)}{N_{t-1}(\boldsymbol{k}_N)}} \right\} \tag{2.54}$$

Here we face the same issue like Lemma 2, when we try to apply the concentration bound for the batch sum of Gaussian random variables[§§], that is $N_{t-1}(\boldsymbol{k}_N)$ is a random variable and selected by the $G-1$ learner. So we can not directly apply the concentration bound. Again we apply here the same pre-generated rewards technique and then taking the union bound over all $t \in [T]$ we get desired result,

$$\mathbb{P}\left(\mathbf{E}_{\Sigma}(\delta, \boldsymbol{k}_N)^c\right) \leq 2T\delta \tag{2.55}$$

Finally taking union bound over all $G-1$ arm vector $\boldsymbol{k}_N \in \mathcal{W}^{g1}$, we get the final result as,

$$\mathbb{P}\left(\mathbf{E}_{\Sigma}(\delta)^c\right) = \mathbb{P}\left(\bigcup_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \mathbf{E}_{\Sigma}(\delta, \boldsymbol{k}_N)^c \right) \leq \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \mathbb{P}\left(\mathbf{E}_{\Sigma}(\delta, \boldsymbol{k}_N)^c\right) \leq \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} 2T\delta = 2TK\delta \tag{2.56}$$

where we used that $|\mathcal{W}^{g1}| = K$ since there are only K possible $G-1$ arm vectors.

■

Note if we had a general policy $\boldsymbol{k} \in \mathcal{W}^N$, we would have K^N arms, then the probability of bad event would have a higher bound as $\mathbb{P}\left(\mathbf{E}_{\Sigma}(\delta)^c\right) \leq 2TK^N\delta$, this is going to be quite large. However, in the $G-1$ learner setup we only have K arm vectors which makes the analysis feasible. We now state and prove the Bayesian regret bound for the $G-1$ learner.

§§ The details for the Gaussian tail bound for the batch sum is provided in Section A2.3

Theorem 2.29 (G-1 THOMPSON SAMPLING REGRET BOUND). *Consider a Bayesian bandit environment under interference $(\boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathscr{F}_{\mathcal{B}})$ for N units, K arms and horizon $T \geq 2$. Under Gaussian reward assumption, for an instance $\boldsymbol{\nu} \sim \boldsymbol{Q}$, for any $\boldsymbol{k}_N \in \mathcal{W}^{g1}$, we have $\boldsymbol{Y}_{\Sigma,t}(\boldsymbol{k}_N) \sim \mathcal{N}\left(\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N), \sigma^2_{\Sigma}(\boldsymbol{k}_N)\right)$ with $\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}_N) \in [\widetilde{B}_{\Sigma}, \widetilde{B}_{\Sigma}]$. Assuming $N_{t-1}(\boldsymbol{k}_N) \geq 1$ for all $\boldsymbol{k}_N \in \mathcal{W}^{g1}$, the Bayesian $G-1$ regret of the $G-1$ learner outlined in Algorithm 7 satisfies,*

$$B\mathcal{R}^{g1}_T = \mathbb{E}_{\pi,\boldsymbol{\nu}\sim\boldsymbol{Q}}\left[\sum_{t=1}^{T}\left(\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}^*) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{W}_t)\right)\right] \leq 8\sigma_{\Sigma,max}\sqrt{KT\log(T)} + 8K\widetilde{B}_{\Sigma} \tag{2.57}$$

where $\sigma^2_{\Sigma,max} := \max_{\boldsymbol{k}_N\in\mathcal{W}^{g1}} \sigma^2_{\Sigma}(\boldsymbol{k}_N)$.

Proof. We followed the similar style of the proof given in Theorem 1.12, start by decomposing the Bayesian $G-1$ regret:

$$\mathcal{BR}^{g1}_T = \sum_{t=1}^{T}\mathbb{E}\left[\mathbb{E}\left[\mu_{\boldsymbol{\nu},\Sigma}\left(\boldsymbol{k}^*_N\right) - \mu_{\boldsymbol{\nu},\Sigma}\left(\boldsymbol{W}_t\right) \mid \mathcal{F}_{t-1}\right]\right] \tag{2.58}$$

Now define the upper confidence bound for the batch sum rewards for arm vector $\boldsymbol{k}_N$ at time t as,

$$U^{\Sigma}_t(\boldsymbol{k}) = \text{clip}_{[-\widetilde{B}_{\Sigma},\widetilde{B}_{\Sigma}]}\left(\overline{\boldsymbol{Y}}_{\Sigma,t-1}(\boldsymbol{k}) + \sigma_{\Sigma}(\boldsymbol{k}_N)\sqrt{\frac{2\log(1/\delta)}{N_{t-1}(\boldsymbol{k})}}\right) \tag{2.59}$$

where $\sigma^2_{\Sigma}(\boldsymbol{k}_N)$ is the variance of the batch sum rewards for arm vector $\boldsymbol{k}_N$, i.e., depends on the specific arm vector played. The clipping here ensures the upper confidence bound of the batch sum lies within the range where the batch sum lies. Now, together with the probability matching property of the $G-1$ learner, we use this bound to decompose the inner expectation term in eq. (2.58) as follows,

$$\begin{aligned}\mathbb{E}[\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}^*_N) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{W}_t) \mid \mathcal{F}_{t-1}] &= \mathbb{E}[\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}^*_N) - U^{\Sigma}_t(\boldsymbol{W}_t) + U^{\Sigma}_t(\boldsymbol{W}_t) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{W}_t) \mid \mathcal{F}_{t-1}] \\ &= \mathbb{E}[\mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{k}^*_N) - U^{\Sigma}_t(\boldsymbol{k}^*_N) \mid \mathcal{F}_{t-1}] + \mathbb{E}[U^{\Sigma}_t(\boldsymbol{W}_t) - \mu_{\boldsymbol{\nu},\Sigma}(\boldsymbol{W}_t) \mid \mathcal{F}_{t-1}]\end{aligned} \tag{2.60}$$

In this case the probability matching condition becomes,

$$\mathbb{P}\left(\boldsymbol{W}_t = \boldsymbol{k}_N \mid \mathcal{F}_{t-1}\right) = \mathbb{P}\left(\boldsymbol{k}^*_N = \boldsymbol{k}_N \mid \mathcal{F}_{t-1}\right) \quad \forall \boldsymbol{k}_N \in \mathcal{W}^{g1} \tag{2.61}$$

Now from here defining two sets A and B and then analyzing the good event $\mathbf{E}_{\Sigma}(\delta)$, the proof goes exactly same as Theorem 1.12, but we outline the details for completeness. First define,

$$A = \sum_{t=1}^{T} (\mu_{\nu,\Sigma}(\boldsymbol{k}_N^*) - U_t^{\Sigma}(\boldsymbol{k}_N^*)), \quad B = \sum_{t=1}^{T} (U_t^{\Sigma}(\boldsymbol{W}_t) - \mu_{\nu,\Sigma}(\boldsymbol{W}_t)), \tag{2.62}$$

because of the linearity of expectation and probability matching property in eq. (2.61) we have,

$$\mathcal{BR}_T^{g1} = \mathbb{E}_{\pi,\nu\sim Q}[A + B] = \mathbb{E}_{\pi,\nu\sim Q}[A] + \mathbb{E}_{\pi,\nu\sim Q}[B] \tag{2.63}$$

Next we bound A and B separately by considering two cases a) when the clean event $\mathbf{E}_\Sigma(\delta)$ holds and b) when its complement $\mathbf{E}_\Sigma(\delta)^c$ holds.

Case a) On Event $\mathbf{E}_\Sigma(\delta)$:

On the event $\mathbf{E}_\Sigma(\delta)$, the concentration holds for all t and $\boldsymbol{k}_N$, so A is already bounded by 0 since

$$\mu_{\nu,\Sigma}\left(\boldsymbol{k}_N^*\right) - U_t^{\Sigma}\left(\boldsymbol{k}_N^*\right) \leq 0$$

We only need to bound term B, we get

$$U_t^{\Sigma}(\boldsymbol{k}_N) - \mu_{\nu,\Sigma}(\boldsymbol{k}_N) \leq \sigma_\Sigma^2(\boldsymbol{k}_N)\sqrt{\frac{8\log(1/\delta)}{N_{t-1}(\boldsymbol{k}_N)}}$$

$$\begin{aligned}
B &= \sum_{t=1}^{T} \left(U_t^{\Sigma}(\boldsymbol{W}_t) - \mu_{\nu,\Sigma}(\boldsymbol{W}_t)\right) \\
&= \sum_{t=1}^{T} \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \mathbb{1}\{\boldsymbol{W}_t = \boldsymbol{k}_N\} \left(U_t^{\Sigma}(\boldsymbol{k}_N) - \mu_{\nu,\Sigma}(\boldsymbol{k}_N)\right) \\
&\leq \sum_{t=1}^{T} \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \mathbb{1}\{\boldsymbol{W}_t = \boldsymbol{k}_N\} \sigma_\Sigma(\boldsymbol{k}_N)\sqrt{\frac{8\log(1/\delta)}{N_{t-1}(\boldsymbol{k}_N)}} \\
&= \sqrt{8\log(1/\delta)} \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma_\Sigma(\boldsymbol{k}_N) \sum_{t=1}^{T} \mathbb{1}\{\boldsymbol{W}_t = \boldsymbol{k}_N\} \frac{1}{\sqrt{N_{t-1}(\boldsymbol{k}_N)}} \\
&\leq \sqrt{8\log(1/\delta)} \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma_\Sigma(\boldsymbol{k}_N) \int_0^{N_T(\boldsymbol{k}_N)} \frac{1}{\sqrt{s}}\, ds \\
&= \sqrt{8\log(1/\delta)} \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma_\Sigma(\boldsymbol{k}_N) \left[2\sqrt{s}\right]_0^{N_T(\boldsymbol{k}_N)} \\
&= \sqrt{32\log(1/\delta)} \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma_\Sigma(\boldsymbol{k}_N)\sqrt{N_T(\boldsymbol{k}_N)} \\
&\leq \sqrt{32\log(1/\delta)} \sqrt{\left(\sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma_\Sigma^2(\boldsymbol{k}_N)\right)\left(\sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} N_T(\boldsymbol{k}_N)\right)}
\end{aligned}$$

$$
\begin{aligned}
&= \sqrt{32\log(1/\delta) \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma^2_\Sigma(\boldsymbol{k}_N) \cdot T} \\
&= \sqrt{32T\log(1/\delta) \sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma^2_\Sigma(\boldsymbol{k}_N) \cdot} \\
&\leq \sigma_{\Sigma,max} \sqrt{32KT\log(1/\delta)}
\end{aligned}
$$

where similarly to the proof of Theorem 1.12, we used Cauchy-Schwarz and the last line comes from $\sum_{\boldsymbol{k}_N \in \mathcal{W}^{g1}} \sigma^2_\Sigma(\boldsymbol{k}_N) \leq K \cdot \sigma^2_{\Sigma,max}$. This means on the good event $\mathbf{E}_\Sigma(\delta)$ we have,

$$
A + B \leq \sigma_{\Sigma,max} \sqrt{32KT\log(1/\delta)}
$$

Case b) On Event $\mathbf{E}_\Sigma(\delta)^c$:

On $\mathbf{E}_\Sigma(\delta)^c$, the proof goes exactly same as Theorem 1.12. We use a worst-case bound. Since $\mu_{\nu,\Sigma}(\boldsymbol{k}_N) \in [-\widetilde{B}_\Sigma, \widetilde{B}_\Sigma]$ and $U^\Sigma_t(\boldsymbol{k}_N) \in [-\widetilde{B}_\Sigma, \widetilde{B}_\Sigma]$ (due to clipping), for each t:

$$
\begin{aligned}
\mu_{\nu,\Sigma}(\boldsymbol{k}^*_N) - U^\Sigma_t(\boldsymbol{k}^*_N) &\in [-2\widetilde{B}_\Sigma, 2\widetilde{B}_\Sigma], \\
U^\Sigma_t(\boldsymbol{W}_t) - \mu_{\nu,\Sigma}(\boldsymbol{W}_t) &\in [-2\widetilde{B}_\Sigma, 2\widetilde{B}_\Sigma]
\end{aligned}
$$

So for each t,

$$
\mu_{\nu,\Sigma}(\boldsymbol{k}^*_N) - U^\Sigma_t(\boldsymbol{k}^*_N) + U^\Sigma_t(\boldsymbol{W}_t) - \mu_{\nu,\Sigma}(\boldsymbol{W}_t) \in [-4\widetilde{B}_\Sigma, 4\widetilde{B}_\Sigma] \tag{2.64}
$$

Thus, on event $\mathbf{E}_\Sigma(\delta)^c$:

$$
A + B = \sum_{t=1}^{T} (\mu_{\nu,\Sigma}(\boldsymbol{k}^*_N) - U^\Sigma_t(\boldsymbol{k}^*_N) + U^\Sigma_t(\boldsymbol{W}_t) - \mu_{\nu,\Sigma}(\boldsymbol{W}_t)) \leq 4T\widetilde{B}_\Sigma \tag{2.65}
$$

Since $\mathbb{P}(\mathbf{E}_\Sigma(\delta)^c) \leq 2TK\delta$, the contribution from $\mathbf{E}_\Sigma(\delta)^c$ is;

$$
\begin{aligned}
\mathbb{E}\left[(A+B) \cdot \mathbb{1}_{\mathbf{E}_\Sigma(\delta)^c}\right] &\leq 4T\widetilde{B}_\Sigma \cdot \mathbb{P}\left(\mathbf{E}_\Sigma(\delta)^c\right) \\
&\leq 4T\widetilde{B}_\Sigma \cdot 2TK\delta \\
&= 8T^2K\widetilde{B}_\Sigma\delta
\end{aligned}
$$

Combining both cases, we have;

$$
\begin{aligned}
\mathcal{BR}^{g1}_T &= \mathbb{E}[A+B] \\
&= \mathbb{E}[(A+B) \mid \mathbf{E}_\Sigma(\delta)]\mathbb{P}(\mathbf{E}_\Sigma(\delta)) + \mathbb{E}[(A+B) \mid \mathbf{E}_\Sigma(\delta)^c]\mathbb{P}(\mathbf{E}_\Sigma(\delta)^c)
\end{aligned}
$$

$$\leq \sigma_{\Sigma,max}\sqrt{32KT\log(1/\delta)} + 8T^2K\widetilde{B}_\Sigma\delta$$
$$= \sigma_{\Sigma,max}\sqrt{32KT\log(1/\delta)} + 8T^2K\widetilde{B}_\Sigma\delta$$

If we plug $\delta = \frac{1}{T^2}$, then

$$\begin{aligned}\mathcal{BR}_T^{g1} &\leq \sigma_{\Sigma,max}\sqrt{64KT\log(T)} + 8K\widetilde{B}_\Sigma \\ &= 8\sigma_{\Sigma,max}\sqrt{KT\log(T)} + 8K\widetilde{B}_\Sigma\end{aligned} \tag{2.66}$$

■

2.5 Numerical Experiments

In this section, with some numerical experiments, we evaluate the performance of the four learners with different interference scenarios. The reward model is the linear-in-means model with three arms as we used in our motivating example in Section 2.1 and the parameter setups are taken from Table 2.1 and Table 2.2 which are essentially an optimal switching (OS) and no optimal switching (No-OS) case respectively. Also, we consider the zero-interference case as well. In the following first, we describe the setup of the numerical experiment, this includes the details of the adjacency matrix and parameter configuration of the three interference models of varying interference effects. After that we compare the performance of the proposed algorithms under these different interference setups.

2.5.1 Network Structure and Clustering

For this experiment, we consider a random geometric graph (RGG), which gives us a *spatial interference* structure as proposed in Leung (2022). First, $N = 250$ spatial positions (or nodes) are randomly generated from uniform distribution $\mathcal{U}\left[-\sqrt{N},\sqrt{N}\right]^2 \subset \mathbb{R}^2$, and then the edges are formed when the distance between node i and node j, is less than or equal to a certain threshold works as an *interference radius*, denoted with $\bar{\rho}$. After that, we normalize the adjacency matrix such that each row sums to one. Then, the quantity $\sum_{j\in\mathcal{I}} A_{ij}\mathbb{1}\{W_{i,t} = k\}$ represents fraction of neighbors within $\bar{\rho}$ who are assigned to arm k at time t for unit i. As we increase the interference radius $\bar{\rho}$, the number of edges increase and the network becomes more and more connected and the interference effect spreads. In plot Figure 2.3 we plot the networks for $\bar{\rho} \in \{1, 1.5, 2.0, 2.5\}$.

Following Leung (2022), we use spectral clustering technique to generate clusters[¶¶]. This partitioned the set of 250 units in $M = N^{2/3} \approx 40$ clusters. Since the network is a spatial network, given these positions, the spectral clustering can capture the spatial proximity and create clusters accordingly. In Figure 2.4 we show the plot of the network for $\bar{\rho} =$

¶¶ An introductory summary of spectral clustering technique can be found in Von Luxburg (2007)

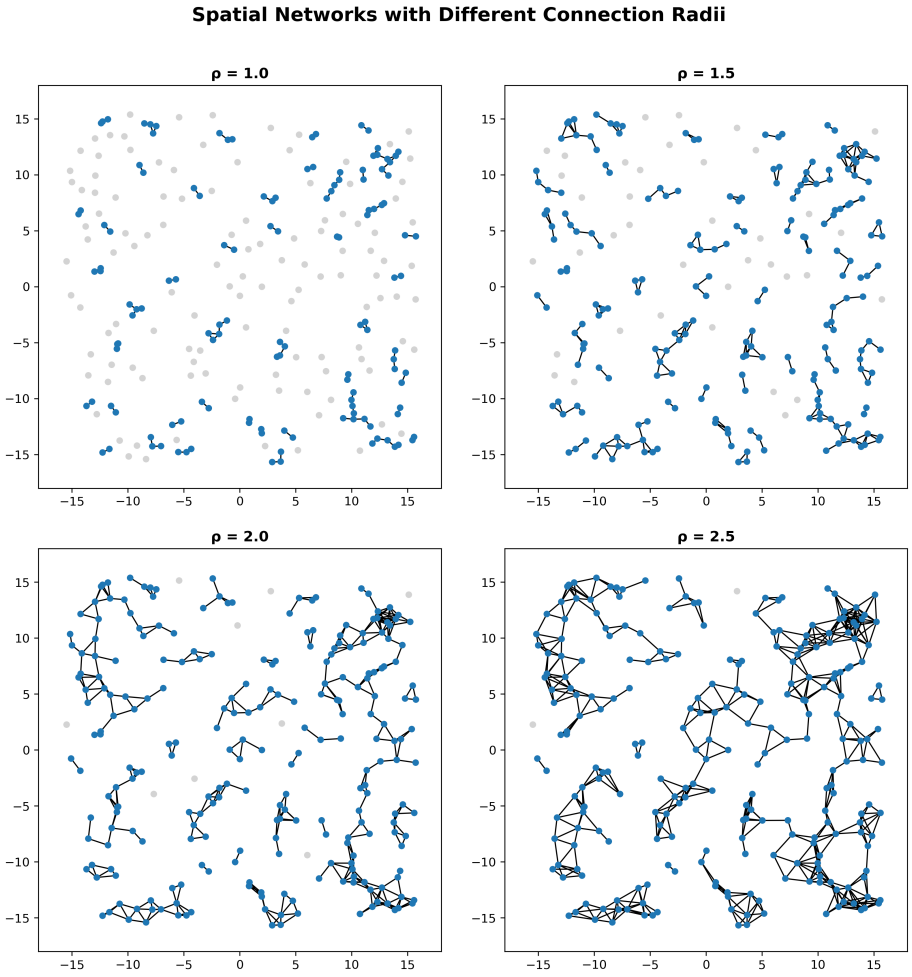


Figure 2.3: Network structure for different interference radius, in particular for $\bar{\rho} \in \{1, 1.5, 2.0, 2.5\}$ from left to right.

1.3 with clusters obtained from spectral clustering method shown in different colors. In the simulation in the next section, we consider the adjacency matrix generated from this network.

2.5.2 *Reward Generation and Interference Parameters*

For the reward generation, we consider the linear-in-means model with three arms $\mathcal{W} = \{1, 2, 3\}$. This is essentially same model as we used in our motivating example in Section 2.1. So if the learner played $\boldsymbol{W}_t \in \mathcal{W}^{250}$ at time t, then the reward for unit i at time t is generated with,

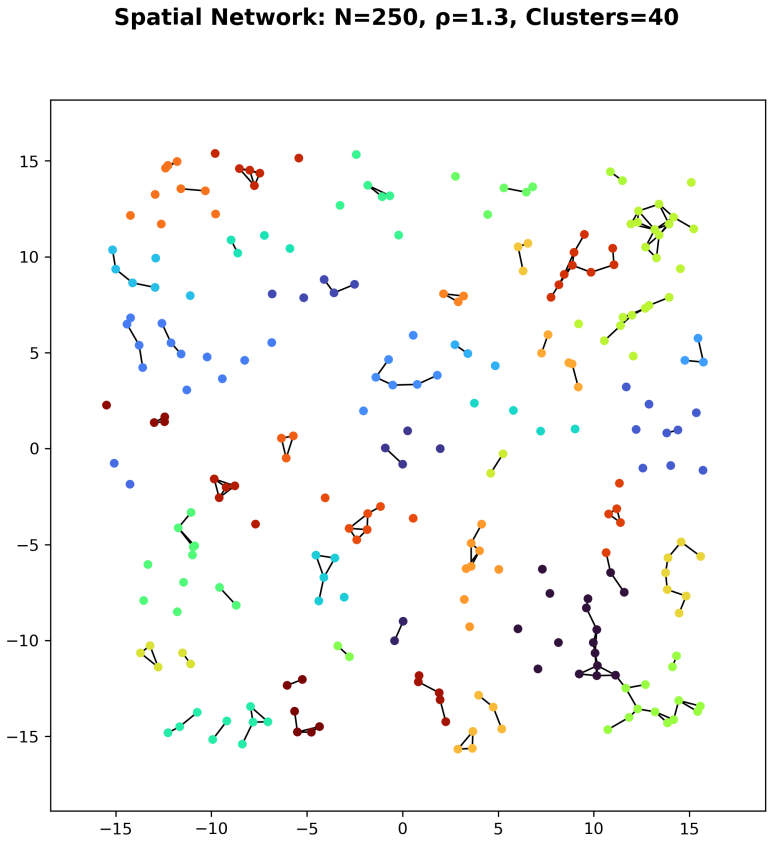


Figure 2.4: 40 clusters obtained using spectral clustering for $\overline{\rho} = 1.3$. Different colors represent different clusters.

$$Y_{i,t}(\boldsymbol{W}_t) = \alpha_i + \beta_1 \cdot \mathbb{1}\{W_{i,t} = 1\} + \beta_2 \cdot \mathbb{1}\{W_{i,t} = 2\} + \beta_3 \cdot \mathbb{1}\{W_{i,t} = 3\} + \delta_1 \sum_{j=1}^{250} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 1\} + \delta_2 \sum_{j=1}^{250} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 2\} + \delta_3 \sum_{j=1}^{250} A_{ij} \cdot \mathbb{1}\{W_{j,t} = 3\} + \epsilon_{i,t} \tag{2.67}$$

where we assume $\epsilon_{i,t} \overset{i.i.d.}{\sim} \mathcal{N}(0, 9)$ across all i and t, for the OS case, and $\epsilon_{i,t} \overset{i.i.d.}{\sim} \mathcal{N}(0, 64)$, for zero spillover and No-OS case. The No-OS case is relatively easier to find, so we set higher variance to make the learning slightly more challenging. We consider following three configurations for the direct and interference effect parameters to represent different interference models. These are essentially the same configurations we used in Table 2.1 and Table 2.2 along with a zero spillover case.

Arm	Baseline α	Direct Effect β_k	(1) Zero		(2) No-OS		(3) OS	
			δ_k	$\bar{\mu}(\boldsymbol{k}_N)$	δ_k	$\bar{\mu}(\boldsymbol{k}_N)$	δ_k	$\bar{\mu}(\boldsymbol{k}_N)$
1	0.0	−5.2%	0.0	5.20	−6.2%	9.86	6.2%	0.54
2		−3.0%	0.0	3.00	−2.3%	4.73	2.3%	1.27
3		−1.0%	0.0	1.00	0.0	1.00	0.0	1.00

Table 2.3: Parameter configurations for the three interference scenarios. The direct effect in absolute value, increases as ($|\beta_1| > |\beta_2| > |\beta_3|$). The spillover effect δ_k determines the interference type: **(1) zero spillover** ($\delta_k = 0$), **(2) no optimal switching** ($\delta_k < 0$), and **(3) optimal switching** ($\delta_k > 0$). The column $\bar{\mu}(\boldsymbol{k}_N)$ shows the global average reward when all N units play arm k.

In principle, the baseline parameter α_i could be different for different units in the batch, but we set it to zero to focus on the direct and spillover effects. The direct effect parameters are set such that Arm 1 is the best arm, followed by Arm 2 and Arm 3, in all three configurations. We follow the same more negative is better convention as same as the motivating example in Section 2.1. The three configurations differ in the spillover effect parameters δ_k. Configuration (1) is the zero spillover effect case, i.e., no interference across all arms. Configuration (2) is no optimal switching (**No-OS**) case is such that the spillover effects have the same ranking as the direct effects for the three arms[***]. Configuration (3) is the optimal switching (**OS**) case, such the spillover effects are positive and working in somewhat opposite direction.

In the No-OS case, for this parameter configuration, we have $\boldsymbol{k}_N^* = \boldsymbol{k}_{N,C-1}^* = \mathbf{1}_N$. So the $G-1$ optimal arm vector, the global optimal arm vector and the $C-1$ optimal arm vector all coincide to $\mathbf{1}_N$, and this is also the unit level optimal arm vector for any unit, regardless of being isolated or connected. However, in the OS case, the $G-1$ optimal arm vector is $\mathbf{2}_N$. The $C-1$ optimal arm vector is a combination vector with 7 arm 1, 16 arm 2 and 17 arm 3. The mean for the $C-1$ optimal arm vector is 1.48, which is higher than $G-1$ optimal which is 1.27.

2.5.3 Comparison of All Learners

For each learner we ran Monte Carlo simulations with 1000 runs, with horizon $T = 500$, in the three interference scenarios. The individual reward variance for the OS case is $\sigma^2(k) =$

*** In example Example 2.11 we already discussed the No-OS case, essentially this is the case where the optimal $G-1$ arm is same for an isolated node and a connected node, and this is also the global optimal vector.

9 for all $k \in \{1, 2, 3\}$. And, for the No-OS and zero spillover case is $\sigma^2(k) = 64$. The No-OS case is relatively easier to find, so we set higher variance to make the learning slightly more challenging. For the Thompson sampling learners, we set the prior means to 0 and prior variance to 100 for all arms. After running the simulations, we computed the empirical average and standard deviations of cumulative $G-1$ and $C-1$ regret for all the learners. The values at the final time point $T = 500$ are gathered in Table 2.4.

Algorithm	(1) **Zero** $G-1$ / $C-1$ Regret	(2) **No-OS** $G-1$ / $C-1$ Regret	(3) **OS** $G-1$ Regret	(3) **OS** $C-1$ Regret
IID Learner	772.34 (184.33)	1331.58 (272.77)	91390.40 (126.85)	118640.40 (126.85)
$G-1$ Learner	2245.95 (997.59)	5049.59 (2715.74)	589.39 (269.03)	27839.39 (269.03)
$C-1$ Learner	793.89 (200.08)	1192.92 (166.46)	501.51 (419.32)	27751.51 (419.32)
$C-M$ Learner	14164.66 (2035.19)	10209.97 (1141.92)	−19575.78 (1063.37)	7674.22 (1063.37)

Table 2.4: Mean and Standard Deviations (in parentheses) of cumulative regret at $T = 500$ across 1000 Monte Carlo replications. For Zero (1) and No-OS (2) scenarios, $G-1$ and $C-1$ regret coincide since $\boldsymbol{k}^*_N = \boldsymbol{k}^*_{N,C-1}$. For OS (3), they differ since these two optimal arm vectors are different.

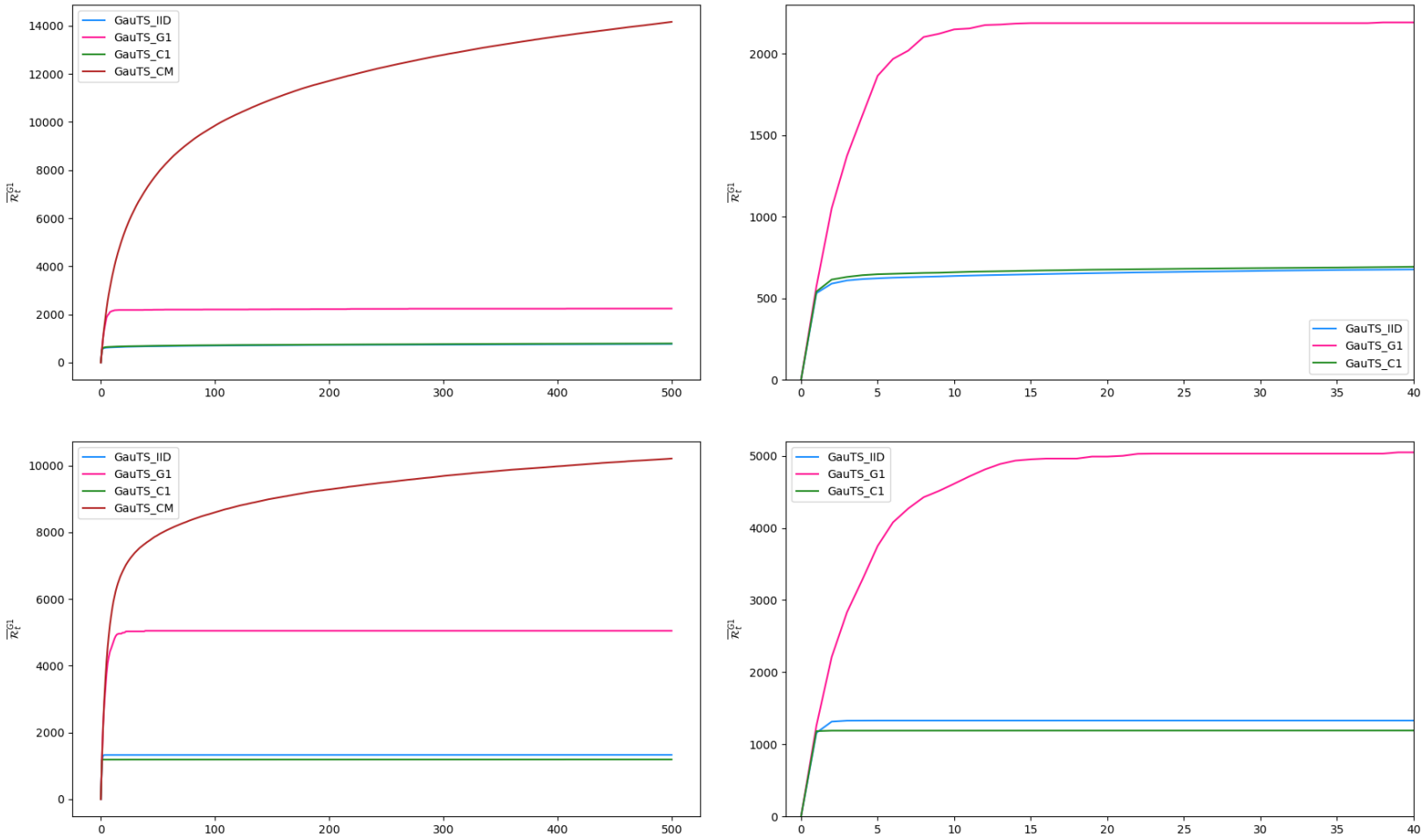


Figure 2.5: Comparison of $G-1$ regret for four learners across zero and No-OS interference scenarios. **Top left:** $G-1$ Regret for zero spillover (1) all learners. **Top right:** $G-1$ regret for zero spillover (1) for the IID, $G-1$ and $C-1$ learners. **Bottom left:** $G-1$ regret for No-OS case (2) all learners. **Bottom right:** $G-1$ regret for No-OS case (2) for the IID, $G-1$ and $C-1$ learners.

From the table, we see for the zero interference case, the IID learner performs best as expected, both mean and standard deviations of the regret is lowest among all. The $C-1$ learner is also close and $C-M$ is the worst. The $G-1$ learner also suffers for high regret.

The reason $C-1$ learner performs good is, by construction it finds the optimal $G-1$ arm vector. In particular, for any arm $k \in \mathcal{W}$, if $k \cdot \mathbf{1}_N$ is the optimal $G-1$ arm vector, then in the no interference case, the best arm is the same $k \in \mathcal{W}$. So the $C-1$ learner is also able to learn this best arm. The $G-1$ learner also has the same target, but suffers because of the high exploitation and the lack of exploration at the initial stage. The $C-M$ learner is designed to find a combination arm as opposed to one best arm, so it has really high regret in this case. The regret plot shown in Figure 2.5 (top row) also shows similar patterns. On the top left, we have the regret plot for all four learners for the zero interference case. However, since the $C-M$ learner has really high regret, it makes the other plots hard to see in a same scale. So we also have a separate plot for the $G-1$, $C-1$ and IID learners in the top right of Figure 2.5. The mean regret plots also confirm, the IID learner performs the best, followed by the $C-1$ learner and then the $G-1$ and $C-M$ learners. The $G-1$ and $C-M$ learners, although have high regret, but seem to be sublinear as time progresses.

The second column of table Table 2.4 and the bottom row of Figure 2.5 show the results for the no optimal switching case. The results are similar, however here the $C-1$ learner performs the best, followed by the IID learner. The $G-1$ learner again has high regret for the same reason and, the $C-M$ learner is the worst. The regret plots in the bottom row of Figure 2.5 also confirm these observations.

Finally, we discuss the results for the optimal switching case, shown in the last two columns of Table 2.4 and the regret plots are in Figure 2.6. For the $G-1$ regret, note that technically the $C-M$ learner achieves the lowest regret since this is negative. However, this negative regret comes because the $C-M$ learner looks for the $C-1$ optimal arm, which is in general close to the global optimal vector and gives lower regret than $G-1$ optimal. However, for this column our target is finding the $G-1$ optimal, and also $G-1$ regret is defined based on this[†††]. So we can ignore the result for the $C-M$ learner, and bring the discussion again when we discuss the $C-1$ regret. We give the brown color to highlight that we should not really compare this with other learners, since it is designed to find better than the defined target.

[†††] One might ask, why we define $G-1$ regret based on $G-1$ optimal and compare algorithms based on this rather than standard regret? Our argument is, playing to minimize the standard regret defined for the global best arm vector is hard. In fact finding this vector itself is hard, since we have to look for all K^N arm vectors. The $G-1$ optimal arm vector, although suboptimal in global sense, but gives us a simple solution and the $G-1$ regret is defined based on this. Then the $C-1$ optimal arm vector and the regret that we defined, is also feasible to calculate in many settings. These two optimal vectors are somewhat compromise solutions between the global optimal and simple solutions, and although not global optimal but can still give us high expected reward in practice.

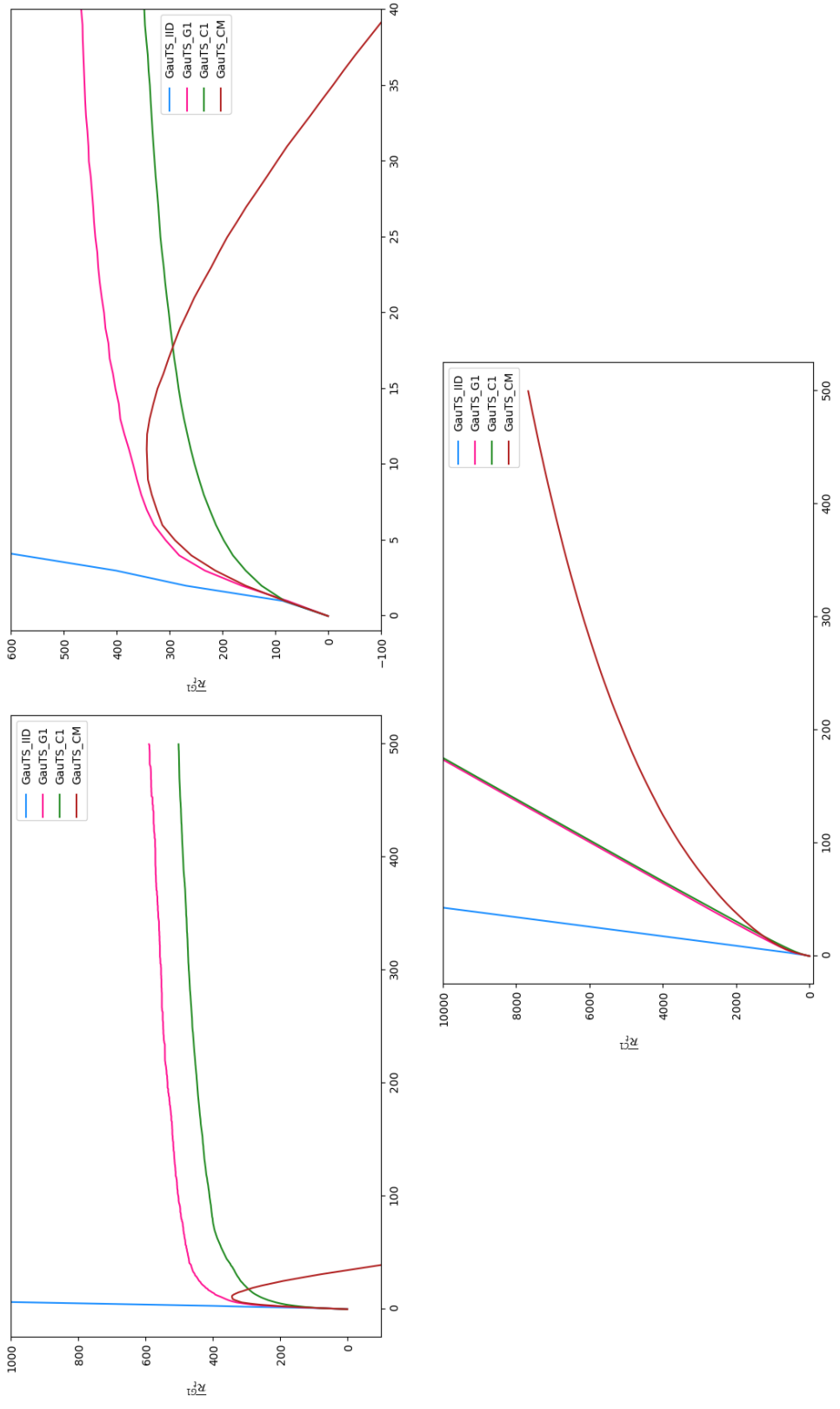


Figure 2.6: Comparison of $G-1$ and $C-1$ regret of four learners for the OS scenario (1). **Top left:** $G-1$ regret for the OS, till $T = 500$. **Top right:** $G-1$ regret for the OS, till $T = 40$, $G-1$ and $C-1$ learners. **Bottom:** $C-1$ regret for the OS, till $T = 500$.

For the other three learners, again $C-1$ learner performs the best and also the $G-1$ learner, although initially struggles because initial over exploitation, but eventually learns and achieves sublinear regret. The IID learner performs the worst here, since it is not designed to handle interference, the regret becomes linear. This is one of the major take-aways from this experiment, that in presence of interference, the standard IID learner can perform really bad, and the algorithms like $G-1$ and $C-1$ learners that are really easy to implement, can perform better than IID learner, even if they are not designed to find the global optimal arm vector.

The $C-1$ regret column and the plot in the bottom of Figure 2.6 is easy to interpret. Here as expected, only the $C-M$ learner performs the best, since it is designed to find the $C-1$ optimal arm vector. The other three learners have really high regret, and don't seem to become sublinear.

2.6 Conclusion

In this chapter, we considered stochastic multi-armed bandit problem under interference, where a learner sequentially assigns treatment vectors to a batch of units with possible interference among them. We believe we have made some foundational contribution to this new area of research, which just started to gain attention in very recent years. We define the stochastic potential rewards and stochastic bandit model under interference, different notions of optimal arm vectors and regrets, connect the setups and definitions with a motivating real life example, proposed four different algorithms that one can apply in this setting to learn different optimal vectors, analyzed the Bayesian regret for one of the proposed algorithms, and finally performed numerical experiments to compare the performance of all the proposed algorithms under different interference scenarios. We believe this opens new research directions and many interesting questions both for us and for the research community. Our future work includes analyzing the Bayesian regret for other proposed algorithms both in Bayesian and frequentist settings. One can also try to incorporate the idea that whether the clusters can be learned adaptively over time rather than being fixed a priori. Another interesting direction is to consider more complex interference structures, for example dynamic interference structure that can change over time.

Chapter 3.

Nonparametric Learners for Stochastic Bandits Under Interference

(Unpublished)

The Thompson sampling algorithm that we discussed in Chapter 1 and Chapter 2 critically depends on the parametric assumptions on the reward distributions. In particular, we assumed that the rewards follow some parametric class of distributions, e.g., Bernoulli or Gaussian, and then the updates are derived using Bayesian prior-posterior relationship. This is restrictive in a sense that, in many practical situations the true underlying reward distributions may not follow these parametric assumptions, or we have enough information to specify a prior distribution. In such cases, it is desirable to have more flexible approaches that can adapt to different reward distributions without relying on strong parametric assumptions. One such approach involves resampling techniques like *bootstrapping or subsampling* (Efron, 1979; Efron & Tibshirani, 1994; Politis, Romano, & Wolf, 1999). These are flexible methods, completely data dependent and do not require any parametric assumptions on the reward distributions. In the stochastic bandit literature, already there exist some works using resampling techniques and some notable works include Kveton, Szepesvari, Vaswani, et al. (2019); Riou and Honda (2020), Chan (2020) and Baudry et al. (2020). In this chapter we extend our contributions in the last chapter and incorporate some resampling techniques that have been proposed in the bandit without interference into our batched bandit setup with interference.

As we discussed in Chapter 2, including our contributions, there already have been some works on stochastic bandit algorithms with interference, particularly the works of Agarwal et al. (2024) and Gleich et al. (2025) are similar to our contributions. However, so far none of these works consider nonparametric or resampling techniques with interference. We fill this gap by extending two notable resampling bandit algorithms to the batched setting with interference. In particular, we extend nonparametric Thompson sampling algorithm (NPTS) proposed by Riou and Honda (2020) and subsampling dueling algorithm (SDA) proposed by Baudry et al. (2020) to the batched setting with interference. The general idea of these extensions are very similar to Section 2.3, but they differ quite a lot in terms of the detail. Thanks to the parametric extensions of Chapter 2, we now have a good guidance on how to proceed with extensions when it comes to interference using resampling techniques.

The novel contribution of this chapter is, extending the four learners that we proposed in Chapter 2, namely, the IID, $G-1$, $C-1$, and $C-M$ learners, to the nonparametric setting using NPTS and SDA algorithms. This gives us eight new algorithms, i.e., NPTS and SDA variants of each learner. Together, these provide a wide range of nonparametric bandit

algorithms capable of handling interference under varying structural assumptions.

The motivation and general principles underlying these extensions mirror the algorithms we proposed in Section 2.3, i.e., we exploit interference by learning optimal treatment assignment vectors tailored to different interference structures. However, the technical details differ substantially from Chapter 2, as the nonparametric framework doesn't depend on the Bayesian updates and for each learner, we had to carefully think about how to keep the original spirit that motivated the design. We present detailed algorithms and explanations for each proposed extension. Finally, we conduct Monte Carlo simulation studies under various interference structures to illustrate the strengths and limitations of the proposed algorithms.

The rest of the chapter is organized as follows. In Section 3.1, we start with the review of two algorithms that are based on resampling techniques for stochastic bandits without interference. In particular, the nonparametric Thompson sampling (NPTS), proposed in Riou and Honda (2020) and the subsampling dueling algorithm (SDA), proposed in Baudry et al. (2020). Both algorithms are quite different in their design principles, but have the common goal, that is, relax the parametric distributional assumption and use resampling methods with good exploration-exploitation trade-off. We don't have a separate section for the problem setup in this chapter, since they follow the exact same stochastic bandit setup in Chapter 1 for the algorithms without interference, and Chapter 2 for the algorithms for stochastic bandits under interference.

The next four sections are essentially extensions of these two algorithms in the batched setting where interference is possibly present. In Section 3.2, we give the proposed extensions to the batched setting, first assuming no interference, we call them *nonparametric IID learners*. NPTS is a direct-batched extension, whereas for SDA we propose some modifications, that is efficient in terms of implementation and help us to adjust the round based structure to the batched setting. These are essentially benchmark algorithms and we evaluate other algorithms against these IID benchmarks. But still these are novel contributions of our work. In Section 3.3, we give the proposed extensions for the global uniform best arm learner, or we call $G-1$ arm learner. We already discussed the parametric version of this in the last chapter in Section 2.3.2, here we propose the *nonparametric global one arm or* $G-1$ *learners*. The key motivation is to search for one best uniform arm vector that maximizes the overall expected reward and performs better than the IID learners in the setup under interference. Next, in Section 3.4 we propose the *nonparametric clustered one arm or* $C-1$ *learners*, where the goal is same as the $G-1$ learner, but the learner has additional cluster information of the batch here and leverages the cluster interference structure to learn the optimal $G-1$ arm vector. These algorithms turn out to be very efficient for optimal $G-1$ learning and interestingly this also performs good in the no optimal switching scenario, which is close to the no interference setting. Finally, we also propose *nonpara-*

metric clustered multi arm or C − M learners in Section 3.5, which learns a combination arm vector, essentially the $C-1$ optimal arm vector for the batch that generally gives higher expected reward since here we are learning the best uniform arm vector per cluster under the clustered interference structure. In section Section 3.6 we give the comparison tables for all the eight algorithms that we proposed in this chapter. In Section 3.7 we evaluate the performance of these algorithms under various interference structures using Monte Carlo simulations. We conclude the chapter in Section 3.8, where we summarize our contributions and discuss some future directions.

3.1 Nonparametric Algorithms Without Interference

When it comes to applying resampling techniques in a bandit learning, the first thing that comes to mind is, applying Efron's bootstrap for each arm from the observed rewards and calculating some bootstrap statistics. In particular, at any time t, for any arm $k \in \mathcal{W}$, we can generate bootstrap samples of equal size as the history, compute some bootstrap statistics, e.g., bootstrap mean, variance or quantiles, and then select the arm that is best* based on these bootstrap statistics. As simple and intuitive as the idea sounds, unlike the statistical inference setup, this approach may fail in case of bandit learning. We explain this below using a conceptual example of a two armed bandit,

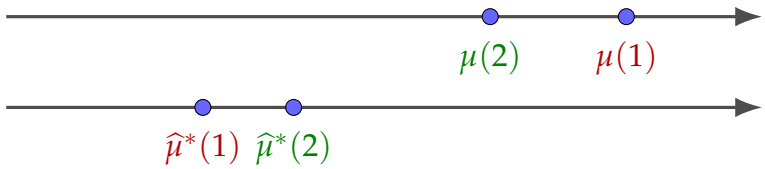


Figure 3.1: Two armed bandit case with empirical mean, arm 1 is optimal but empirical mean of arm 2 is higher than arm 1. Because of less exploration, arm 1 may get stuck indefinitely as a suboptimal arm.

Example 3.1 (Conceptual Example: Two Armed Bandit with Bootstrap Mean)**.** Consider a two-armed bandit model, where arm 1 is optimal, i.e., $\mu(1) > \mu(2)$. Let $\widehat{\mu}^*(1)$ and $\widehat{\mu}^*(2)$ be the bootstrap mean reward for arm 1 and 2 respectively after time t. Suppose that, by mere chance, the initial rewards for arm 2 are unusually high, so when learner computer she finds $\widehat{\mu}^*(2) > \widehat{\mu}^*(1)$. Now being optimistic, she will exploit this situation, and selects arm 2 since it has highest bootstrap mean. Now, if she plays arm 2, she will see more rewards from arm 2 and from here on wards, this creates a vicious cycle: arm 2 accumulates more samples and this makes its estimate more precise and because bootstrap estimates are good estimates, this will converge toward $\mu(2)$. Whereas for

* Here best could be mean different things based on the context, but in general in the bandit the best arm is the arm which has the highest expected reward.

arm 1, if the learner pulls it less frequently, then the bootstrap mean of this arm remains more or less at the same place and might underestimate the true mean $\mu(1)$. So without sufficient playing or exploration of arm 1, possibly the learner will never discover that arm 1 is actually optimal, and it may be the case she may never finds it and gets stuck in arm 2 forever.

///

The Example 3.1 illustrates that the learning in a bandit setting not strictly a statistical learning or inference problem, that typically we face in Statistics. In bandit learning, there is an inherent unfairness among the history of different arms and this may come by construction from different procedures. On one hand, we want to detect the reward maximizing arm as quickly as possible, on the other hand, we also need to explore all arms sufficiently to avoid getting stuck in suboptimal arms. Yes! the learner does need a good estimator, such that her estimates converge to the true mean, but at the same time she also needs to cleverly balance exploration and exploitation trade-off. Efron's bootstrap perhaps will give us a good estimator, which solves the inference problem, but it may not be sufficient to solve the exploration-exploitation trade-off issues, unless we incorporate some additional mechanisms to encourage more exploration or penalize excessive exploitation. This issue has been discussed in Osband and Van Roy (2015), Kveton, Szepesvari, Ghavamzadeh, and Boutilier (2019) and Kveton, Szepesvari, Vaswani, et al. (2019).

Of course in the literature, the scholars have already proposed many solutions for this particular problem. Broadly, we can classify them in three categories, **1** : algorithms based on additional or synthetic data, or sometimes also called *fake rewards*, this will encourage more exploration for all arms. Then **2** : algorithms based on fair comparison primarily by subsampling, and **3** : algorithms based on upper confidence bound (UCB) with resampling. This is modifying the UCB algorithms using resampling techniques. In this section we review nonparametric Thompson sampling (NPTS) and subsampling duelling (SDA) algorithms which fall under the first two categories respectively. We don't discuss the third category here, simply because the extensions that we propose later for interference are based on the first two categories. But we refer the reader to the work of Hao, Wen, Abbasi-Yadkori, and Cheng (2019) for more details on UCB with resampling.

3.1.1 *Fake Rewards with NPTS*

We start with the first category, which is adding fake rewards, and explain the nonparametric Thompson sampling (NPTS) algorithm in detail. The key idea of "*fake rewards*" is to add some additional data to solve under-exploration problem that we would otherwise face. Essentially these additional data encourage more exploration at the beginning of the learning and eventually helps the algorithm to escape from suboptimal arms. Other algorithms in this approach include - Bootstrap Thompson Sampling (BTS) (Osband &

Algorithm 10: NPTS (Single Unit)

Input: $T, \mathcal{W}, y^b(k), \forall k \in \mathcal{W}$
Initialize: $n_0(k) = 0, \boldsymbol{y}_0^h(k) = y^b(k), \forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  sample $\boldsymbol{\omega}_t(k) \sim \text{Dirichlet}(\mathbf{1}_{n_{t-1}(k)+1}), \forall k \in \mathcal{W}$ // probability sampling from simplex
  compute $\hat{\mu}^*(k) = \boldsymbol{\omega}_t(k)' \cdot \boldsymbol{y}_{t-1}^h(k), \forall k \in \mathcal{W}$ // weighted bootstrap mean
  select $W_t = \text{argmax}_{k \in \mathcal{W}} \hat{\mu}^*(k)$
  observe $Y_t = Y_t(W_t)$
  update
    $\boldsymbol{y}_t^h(W_t) = (\boldsymbol{y}_{t-1}^h(W_t); Y_t)$ // concat only for chosen arm
    $n_t(W_t) = n_{t-1}(W_t) + 1$
Output: $W_t, Y_t, \forall t \in [T]$

Van Roy, 2015), Perturbed History Exploration (PHE) and Garbage in, Reward Out (GIRO) (Kveton, Szepesvari, Ghavamzadeh, & Boutilier, 2019; Kveton, Szepesvari, Vaswani, et al., 2019) and Residual Bootstrap (Wang, Yu, Hao, & Cheng, 2020). The complete NPTS algorithm is summarized in Algorithm 10 and we explain the details below. For the other algorithms we refer the readers to the original works.

Here is how NPTS works, at $t = 0$, for each arm $k \in \mathcal{W}$, the learner or algorithm initializes each arm's history with a single fake reward $y^b(k)$. This is actually the upper bound on the support of the rewards; and this is particularly the element of the algorithm which encourages more exploration. Then at each time $t = 1, 2, \ldots, T$, for each arm $k \in \mathcal{W}$, the algorithm generates a weight vector from a Dirichlet distribution with parameters $\mathbf{1}_{n_{t-1}(k)+1} = (1, 1, \ldots, 1)$, where $n_{t-1}(k)$ is the number of times arm k has been played up to time $t-1$. Note that the additional 1 in the parameter vector is because of the fake reward we added at $t = 0$. Then using this weight vector and the history of rewards for arm k up to time $t-1$, the algorithm computes a weighted mean or bootstrap mean for arm k. We denote bootstrap mean with $\hat{\mu}^*(k) = \boldsymbol{\omega}_t(k)' \cdot \boldsymbol{y}_{t-1}^h(k)$, where $\boldsymbol{y}_{t-1}^h(k)$ is the vector of rewards (i.e., the history) for arm k after time $t-1$.

After computing the weighted means for all arms, the algorithm selects the arm with the highest weighted mean and pulls that arm to observe the reward. Finally, the history of only the selected arm is updated with the newly observed reward and the number of times that arm has been played is incremented by one. We give some additional remarks on the NPTS algorithm below.

Remarks 3.2 (Discussion on NPTS)**.**

Weighted Bootstrap Mean: The bootstrap mean is written as a weighted mean $\widehat{\mu}^*(k) :=$

$\boldsymbol{\omega}_t(k)' \cdot \boldsymbol{y}^h_{t-1}(k)$, where $\boldsymbol{\omega}_t(k)$ is the weight vector of length $n_{t-1}(k)+1$ and $\boldsymbol{y}^h_{t-1}(k)$ is the history vector for arm k up to time $t-1$ plus the upper bound. The weight is generated from Dirichlet($\mathbf{1}_{n_{t-1}(k)+1}$), which means the weights are uniformly distributed over the simplex. This is different from the traditional Efron's bootstrap where the weights are generated from a multinomial distribution. The history vector $\boldsymbol{y}^h_{t-1}(k)$ contains the fake reward at index 0 and then the observed rewards for arm k up to time $t-1$. So the length of this vector is also $n_{t-1}(k)+1$. The bootstrap mean is simply the weighted average of these rewards using the generated weights.

From Dirichlet-Multinomial to Dirichlet-Continuous: The idea of the Dirichlet weights actually comes from the Dirichlet-Multinomial relationship. This is very similar to the Beta-Bernoulli relationship that we discussed in Chapter 1 for Thompson sampling. In particular, Riou and Honda (2020) first proposed a Multinomial Thompson sampling (MTS) algorithm for discrete rewards using Dirichlet-Multinomial relationship. Then, the authors extended this to continuous rewards using Dirichlet weights. We briefly summarize this relationship here. Fix any arm k, let the support of the reward $Y_t(k)$ be $\{1,2,3,\ldots,M\}$ and $Y_t(k) \sim$ Multinomial$(1, \boldsymbol{p}(k))$, where $\boldsymbol{p}(k) = (p_1(k), p_2(k), \ldots, p_M(k))'$ is the Multinomial probability vector for the reward distribution of arm k. This means $Y_t(k)$ is a discrete random variable and at each time t and one of $m = 1,2,\ldots,M$ categories is observed with probability $p_m(k)$. Now if the prior is $\boldsymbol{p}(k) \sim$ Dirichlet$(\boldsymbol{\alpha}(k))$, where $\boldsymbol{\alpha}(k) = (\alpha_1(k), \alpha_2(k), \ldots, \alpha_M(k))'$, then using the Bayesian updates, the posterior will be Dirichlet$(\alpha_1(k) + n^1_{t-1}(k), \alpha_2(k) + n^2_{t-1}(k), \ldots, \alpha_M(k) + n^M_{t-1}(k))$, where $n^m_{t-1}(k)$ is the number of times category m has been observed in these rewards. So these updates can be used to design a Thompson sampling algorithm for Multinomial rewards. This is still parametric and Bayesian. The authors then departed from this in two ways, first, they assumed the rewards are continuous and bounded in $[0,1]$, but kept Dirichlet weights for the observed rewards as we explained. This can be viewed as situation, where the observed rewards adaptively becomes the support of the Multinomial. Second, they added the fake reward at the beginning, such that it always remains in the history and encourages exploration. In this way, the algorithm becomes nonparametric and there is actually no Bayesian updates involved.

Choice of Fake Reward: The choice of fake reward $y^b(k)$ is important in the NPTS algorithm. Riou and Honda (2020) suggest using the upper bound of the reward support for this purpose. The key result of the paper assumed the rewards are bounded in $[0,1]$, so they used $y^b(k) = 1$ for all arms.

Weight Decreases With More Data: Note that, although If arm k is pulled more often, then the length of $\boldsymbol{y}^h_{t-1}(k)$ increases, this means the weight vector $\boldsymbol{\omega}_t(k)$ also increases in length and each individual weight decreases in magnitude, so we will still take a

weighted average but over more data points. This also has influence for the weight of the fake reward in the bootstrap mean. Over time, as more data is collected for arm k, the influence of the fake reward on the bootstrap mean also decreases.

Theoretical Guarantees: NPTS algorithm proposed in Riou and Honda (2020) has good theoretical properties. In particular, the authors showed that NPTS achieves asymptotic optimality in terms of achieving the Lai-Robbins lower bound (see eq. (1.6)) for K arms bandit, but an important limitation is this holds only for bounded rewards in $[0, 1]$. The algorithms we discuss next also has similar theoretical guarantees but it relaxes this boundedness assumption.

///

3.1.2 Fair Comparison with SDA

We now discuss the second approach, that is, comparing arms *"fairly"*. The algorithm we discuss in this approach is known as *subsampling dueling algorithm* or SDA in short, proposed by Baudry et al. (2020), also see the manuscript Baudry (2022) for more details. Why subsampling? The short answer is - it's another way to balance exploration and exploitation, or in particular exploitation. Recall, the first approach or NPTS in particular, encourages more exploration by adding fake rewards and then reduces exploitation by adjusting the Dirichlet weights. This approach, or SDA in particular, controls exploitation of the large history arms by taking subsamples, and then brings more exploration by the round based structure. As we briefly alluded at the beginning of this section, *"fairness"* here means whether the history or data of arms we compare are of similar size, all the algorithms in this category ensures that this holds in general. Other than SDA, there are also Best Empirical Sampled Average (BESA) proposed by Baransi, Maillard, and Mannor (2014) and Subsample Mean Comparison (SSMC) proposed by Chan (2020) falling in this category. However, since SDA can be seen as a combination of BESA and SSMC algorithms, we start with SDA, which is also our key focus, and then briefly discuss BESA and SSMC in Remarks 3.3

Algorithm 11 implements the core SDA procedure. One key exception is, since it's a round based approach, so there is a separate round counter and time counter, denoted with r and t respectively. The definition of the round is, it's a state when the learner forms a set called the *winner set* of the arms. This is denoted with $\mathcal{A}_r$. At the end of each round all the arms in the winner set are pulled. The time t is conceptually same object as before, so whenever the learner pulls an arm, the time t increases by one. So if we set horizon T, then in total there will be T pulls, but the number of rounds could be less than T.

Let's explain the procedure, the learner starts with a time horizon T, an arm set $\mathcal{W} = \{1, 2, \ldots, K\}$ and a subsampling strategy $\mathcal{S}$ (there are different ways to subsample, we give

Algorithm 11: SDA (Single Unit Case)

Input: T, $\mathcal{W} = [K]$, $\mathcal{S}$
Initialize:
  $\mathcal{D}_1(k) = \{Y_k\}, n_1(k) = 1, \forall k \in \mathcal{W}$
  $r = 2, t = K + 1$
while $t \leq T$ **do**
  select leader: $\ell_r = \text{argmax}_{k \in \mathcal{W}} n_{r-1}(k)$ // break ties with max. $\widehat{\mu}_{r-1}(k)$, else keep ℓ_{r-1}
  draw leader subsamples: $\mathcal{D}^*_{r,k} \sim \mathcal{S}(\mathcal{D}_{r-1}(\ell_r), n_{r-1}(k)), \forall k \in \mathcal{W} \backslash \{\ell_r\}$
  compute winner set: $\mathcal{A}_r = \{k \in \mathcal{W} \backslash \{\ell_r\} : \widehat{\mu}_{r-1}(k) \geq \widehat{\mu}^*_{r-1,k}\}$, or $\mathcal{A}_r = \{\ell_r\}$ if $\mathcal{A}_r = \emptyset$
  foreach $k \in \mathcal{A}_r$ **do**
    pull arm $W_t = k$
    observe reward $Y_t = Y_t(W_t)$
    update
      $\mathcal{D}_r(k) = \mathcal{D}_{r-1}(k) \cup \{Y_t\}$ // update history only for arms in the winnerset
      $n_r(k) = n_{r-1}(k) + 1$ // update count only for arms in the winnerset
      increment $t = t + 1$ // time will increase when an arm is pulled
      if $t > T$ **then**
        break // exit if pulls exceeds horizon
  increment $r = r + 1$
Output: $W_t, Y_t, \forall t \in [T]$

some details in Remarks 3.3). At the beginning, when round $r = 1$, the learner plays all arms once, so this is simply the initialization step. Here the histories for each arm will be created with a random draw from arm k, i.e., $\mathcal{D}_1(k) = \{Y_k\}$, where we denote the history of arm k at round r with $\mathcal{D}_r(k)$ and Y_k is a randomly generated reward from arm k. Since we have k arms, the learner pulls K times, one for each arm.

After this, the count for each arm increases to 1, i.e., $n_1(k) = 1$. However, we have already passed K times, so now the time counter is set to $t = K + 1$. The round counter in this case $r = 2$. This is written in line $2 - 4$ of Algorithm 11, essentially these lines are after the initialization step of SDA.

Now, from $t = K + 1$, at each round r, the main loop starts. First task is to form a winner set. There are essentially three key steps to form the winner set $\mathcal{A}_r$ at round r, a) *leader selection:* the learner selects a leader arm based on the longest history, b) *subsampling:* draw a subsample of the leader's history equal to the size of **every** challengers' history, and then c) *duelling:* play a duel between the leader and every challenger by comparing their empirical means and subsampled mean, i.e., compare $\widehat{\mu}_{r-1}(k)$ and $\widehat{\mu}^*_{r-1,k}$ where the first one is the empirical mean of challenger arm k and second one is the subsampled mean of the

leader arm against challenger arm k. So for every challenger arm, learner will subsample the leader's history and then organize a duel. In total this will give $K-1$ duels. And whenever the challenger's empirical mean is higher than the subsampled leader's empirical mean, the challenger arm qualifies for the winner set. If no arm qualifies for the winner set, then the leader arm is added to the winner set by default. If in the leader selection there are ties, then ties are broken by choosing the arm with the highest empirical mean, if there is no arm in the winner set, this means, even if we are subsampling, the leader's subsampled mean is higher than all challenger arms, then previous leader is kept in the winner set. These three steps are written in line $6-8$ of Algorithm 11.

Now after the learner has the winner set $\mathcal{A}_r$ at round r, it's time for the leaner to pull all the arms one by one. After each pull, the history and count of that arm is updated and also the time counter t is incremented by one. This is written in line $10-15$ of Algorithm 11. Total increment of time t will be $|\mathcal{A}_r|$ and after pulling all arms in the winner set, the round counter r is incremented by one and the main loop continues until time horizon T is reached. But if there is a round where after pulling some arms in the winner set $\mathcal{A}_r$, the time counter t exceeds the time horizon T, then the algorithm breaks out of the inner loop and also the main loop terminates since we have already reached the time horizon T. We provide some additional remarks on SDA below.

Remarks 3.3 (Discussion on SDA)**.**

SDA is an Evolution from BESA and SSMC: SDA can be seen as a combination of BESA and SSMC algorithms. BESA, proposed in Baransi et al. (2014), the authors first propose an algorithm for a two arm bandit and the initial idea of *subsampling* has been proposed to ensure the fair comparison takes place between two arms. At each time t, the algorithm organizes a match between two arms by comparing the subsamples from the longer-history arm with full history size of the shorter one. The authors then proposed an idea of conducting a *tournament* for more than two arms setting. In particular, they proposed to organize matches between randomly paired arms, the winners of each pair move and plays matches to the next rounds until one arm remains, and then the winning arm is pulled. This is a useful extension but as Baudry et al. (2020) argued when there are many arms there are some issues with the tournament idea. In particular with an example they showed it gets difficult to select the best arm since there is a possibility that it will have to pass through multiple duels against other arms to reach the final round. In SSMC, proposed by Chan (2020), the author brings this fairness idea, but proposed a more scalable algorithm. The novel idea is to have a *leader arm* and many possible *challenger arms*. At every round, the arm with the longest history of play will be leader and the other arms are challenger, the leader will play duels with every challenger which is again a comparison of means of equal sized histories, leader subsampled history Vs. challenger's full

history. However, the difference is leader's the worst history sequence is considered, so there is no randomness in the subsampling. The authors provide strong results for arms coming from single-parameter exponential family of distributions. Note that, SDA combines the ideas from BESA (in particular the idea of subsampling) and SSMC (in particular the idea of a leader Vs. challenger) to propose a more general algorithm.

Subsampling Strategies: The subsampling strategy $\mathcal{S}$ is an important part of the SDA algorithm. A subsampling strategy is essentially a function that takes two inputs, i) the history of the leader arm $\mathcal{D}_{r-1}(\ell_r)$ and ii) the size of the challenger's history $n_{r-1}(k)$, and then outputs a subsample $\mathcal{D}^*_{r,k}$ of size $n_{r-1}(k)$ from the leader's history. There can be different ways to subsample, Baudry (2022) discussed some strategies, e.g., *sampling without replacement:* where the $n_{r-1}(k)$ samples are taken uniformly from the leader's history without replacement, *random block sampling:* randomly a starting point is selected first and then $n_{r-1}(k)$ consecutive elements are taken from the history of the leader, and *deterministic sampling:* e.g., last block sampling, where the last $n_{r-1}(k)$ elements from the leader's history are taken.

Details on Duelling: We give some details on duelling. At round r, for each challenger arm $k \in \mathcal{W}\backslash\{\ell_r\}$, a *duel* occurs between the leader arm ℓ_r and the challenger. In this duel, the leader's subsampled mean reward is compared against the challenger's full mean reward. Fix a challenger arm k, we let $\widehat{\mu}^*_{r-1,k}$ be the subsampled mean reward of the leader arm at round r for this challenger arm,i.e., $\widehat{\mu}^*_{r-1,k} := \frac{1}{n_{r-1}(k)} \sum_{y \in \mathcal{D}^*_{r,k}} y$. Note that this is based on the subsample $\mathcal{D}^*_{r,k}$, where $\mathcal{D}^*_{r,k} \sim \mathcal{S}(\mathcal{D}_{r-1}(\ell_r), n_{r-1}(k))$. This means we always have $n_{r-1}(k)$ samples in this subsample. Also since subsampling will always be for the leader's history we simply write $\widehat{\mu}^*_{r-1,k}$ without adding the leader index and k refers to the challenger arm, for which the subsample is drawn. Now, for the k, let $\widehat{\mu}_r(k)$ be the empirical mean reward after round r with full history, i.e., $\widehat{\mu}_r(k) = \frac{1}{n_r(k)} \sum_{s=1}^{r} \mathbb{1}\{W_s = k\} Y_s = \frac{1}{n_r(k)} \sum_{y \in \mathcal{D}_r(k)} y$. A challenger arm k wins the duel and gets a place in the winner set if at round r, $\widehat{\mu}_{r-1}(k) \geq \widehat{\mu}^*_{r-1,k}$. And all the arms that win their respective duels are added to the winner set $\mathcal{A}_r$. If no arm wins a duel, then the leader arm is added to the winner set by default.

Theoretical Guarantees of SDA: According to Baudry et al. (2020), SDA has a strong theoretical guarantee under two conditions, *a) diversity of the subsamples:* this means the subsampling strategy should be such that it generates diverse subsamples that do good exploration of the leader's history. The last block and random block sampling strategy satisfies this condition, however, the sampling without replacement may or may not. The second condition is, *b) balanced distribution*: here the authors defined a property called balancing property that essentially says an optimal arm cannot lose duels against the leader for a long time. They show that if a sampler satisfies the diversity property,

then for any balanced distribution of rewards, SDA achieves the asymptotic optimality in terms of achieving the Lai-Robbins lower bound for K arms bandit. Compared to NPTS, this is more flexible result, since it doesn't require bounded rewards like $[0, 1]$.

///

In the next four sections we extend the NPTS and SDA algorithms to the batched setting where there is possible treatment interference among the units, i.e., treatment or arm selection of one unit may affect the reward of another unit. In Chapter 2 we extended the classic Thompson sampling algorithm to the interference setup, and then proposed three algorithms that can efficiently learn different optimal assignments under different interference structures along with the IID learner that works best for the batch under no interference. Our motivation is same here, we also follow a similar approach. However now we leave the parametric setting and propose resampling based or nonparametric versions instead. This gives us nonparametric versions of IID, $C-1$, $G-1$, and $C-M$ learners.

The bandit setup under interference and the regret definitions are same as Section 2.2, so we will avoid here for repetition. For a context, we still can think about the motivating example given in Section 2.1. This gives us an idea about where to apply these algorithms in practice. The learner's goal is to maximize cumulative batch sum of the reward over the horizon T, or equivalently minimize the cumulative regret, where different notions of regret are given in Section 2.2.

3.2 Nonparametric IID Learners

We start with the nonparametric versions of the IID learners. These learners are designed to work best under no interference setting. First extension is from the basic NPTS algorithm outlined in Algorithm 10. We call this nonparametric Thompson sampling or NPTS IID learner and the steps are outlined in Algorithm 12. To explain, at round t, the IID learner maintains a shared history $\boldsymbol{y}^h_{t-1}(k)$ and a shared count number $n_{t-1}(k)$, for each arm $k \in \mathcal{W}$. The shared history vector essentially contains the rewards observed from all the units up to round $t-1$ for any arm k, and the same goes for the shared count. At $t=0$, the history for each arm is initialized with a fake reward $y^b(k)$, so the initial history and count are $\boldsymbol{y}^h_0(k) = (y^b(k))$ and $n_0(k) = 0$ for all arms $k \in \mathcal{W}$.

The algorithm proceeds as follows. At each t, for each unit $i \in \mathcal{W}$, the learner samples weights $\boldsymbol{\omega}_t(k) \sim \text{Dirichlet}\left(\mathbf{1}_{n_{t-1}(k)+1}\right)$ for all arms $k \in \mathcal{W}$ independently, computes the bootstrap mean $\hat{\mu}^*_t(k) = \boldsymbol{\omega}_t(k)' \cdot \boldsymbol{y}^h_{t-1}(k)$ using unit-specific weights but shared history, and selects the arm for the unit with $W_{i,t} = \text{argmax}_{k \in \mathcal{W}} \hat{\mu}^*_t(k)$, i.e., the arm that gives the highest bootstrap mean for the particular unit. Here what's important is, for each unit, the weight vector is different, but they have the shared history, and because of the different weights the bootstrap means will be different for different units. So it's likely

Algorithm 12: NPTS *IID* Learner

Input: $\mathcal{I}$, T, $\mathcal{W}$, $y^b(k), \forall k \in \mathcal{W}$
Initialize: : $n_0(k) = 0, \boldsymbol{y}_0^h(k) = (y^b(k)), \forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  for $i \in \mathcal{I}$ **do**
    sample $\boldsymbol{\omega}_t(k) \sim \text{Dirichlet}(\mathbf{1}_{n_{t-1}(k)+1}), \forall k \in \mathcal{W}$ // probability sampling from simplex
    compute $\hat{\mu}_t^*(k) = \boldsymbol{\omega}_t(k)' \cdot \boldsymbol{y}_{t-1}^h(k), \forall k \in \mathcal{W}$ // weighted bootstrap mean
    select $W_{i,t} = \text{argmax}_{k \in \mathcal{W}} \hat{\mu}_t^*(k)$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  update
    $\boldsymbol{y}_t^h(k) = (\boldsymbol{y}_{t-1}^h(k); (Y_{i,t})_{i:W_{i,t}=k}), \forall k \in \mathcal{W}$ // concat from all units to reward vector
    $n_t(k) = n_{t-1}(k) + \sum_{i=1}^N \mathbb{1}\{W_{i,t} = k\}, \forall k \in \mathcal{W}$
Output: $\boldsymbol{W}_t, \boldsymbol{Y}_t, \forall t \leq [T]$

Algorithm 13: SD *IID* Learner

Input: $\mathcal{I}$, T, $\mathcal{W} = [K]$, $\mathcal{S}$
Initialize:
  $\mathcal{D}_1(k) = \{Y_{i,1} : i \in \mathcal{I}, W_{i,1} = k\}$ with $W_{i,1} \sim \mathcal{U}(\mathcal{W}), \forall k \in \mathcal{W}$
  $n_1(k) = \sum_{i=1}^N \mathbb{1}\{W_{i,1} = k\}, \forall k \in \mathcal{W}$
for $t \in \{2, \ldots, T\}$ **do**
  select leader: $\ell_t = \text{argmax}_{k \in \mathcal{W}} n_{t-1}(k)$ // break ties with max. $\hat{\mu}_{t-1}(k)$, else keep ℓ_{t-1}
  draw leader subsamples: $\mathcal{D}_{t,k}^* \sim \mathcal{S}(\mathcal{D}_{t-1}(\ell_t), n_{t-1}(k)), \forall k \in \mathcal{W} \backslash \{\ell_t\}$
  compute winner set: $\mathcal{A}_t = \{k \in \mathcal{W} \backslash \{\ell_t\} : \hat{\mu}_{t-1}(k) \geq \hat{\mu}_{t-1,k}^*\}$, or $\mathcal{A}_t = \{\ell_t\}$ if $\mathcal{A}_t = \emptyset$
  for $i \in \mathcal{I}$ **do**
    pull / select arm: $W_{i,t} \sim \mathcal{U}(\mathcal{A}_t)$ // uniform sampling from winner set
  observe rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  update
    $\mathcal{D}_t(k) = \mathcal{D}_{t-1}(k) \cup \{Y_{i,t} : i \in \mathcal{I}, W_{i,t} = k\}, \forall k \in \mathcal{W}$ // pulling across units
    $n_t(k) = n_{t-1}(k) + \sum_{i=1}^N \mathbb{1}\{W_{i,t} = k\}, \forall k \in \mathcal{W}$
Output: $\boldsymbol{W}_t, \boldsymbol{Y}_t, \forall t \leq [T]$

that the learner will select different arms for different units at time t. After selecting all arms, the learner observes the rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, and then updates the shared history vector and count for all arm $k \in \mathcal{W}$. The history vector is being updated by appending or concatenating rewards from all units for whom the learner selects arm k. The count is incremented by the number of units for whom the learner pulled arm k. The update equations are as follows:

$$\left.\begin{aligned} \boldsymbol{y}_t^h(k) &= \left(\boldsymbol{y}_{t-1}^h(k); (Y_{i,t})_{i:W_{i,t}=k}\right) \\ n_t(k) &= n_{t-1}(k) + \textstyle\sum_{i=1}^N \mathbb{1}\{W_{i,t} = k\} \end{aligned}\right\} \quad \forall k \in \mathcal{W}$$

where $\left(\boldsymbol{y}_{t-1}^h(k); (Y_{i,t})_{i:W_{i,t}=k}\right)$ means concatenation of the previous history vector with the new rewards from all units that pulled arm k at time t. The count $n_t(k)$ is updated by adding the number of units that pulled arm k at time t. The algorithm continues till time horizon T is reached.

Next, we give the details for the subsampling duelling or SD *IID* learner. However, unlike NPTS *IID* extension, this is not as straightforward as one might think. Recall the SDA in Algorithm 11 has a round based structure and at each round the single unit learner selects many arms so question how to do this for each node? We come back to this question later in Remarks 3.4. It turns out this becomes very expensive procedure with no significant benefit in performance.

The algorithm that we propose has the same single winner set approach, which we call *common winner set* for all units at each time step and then for each unit the learner selects an arm uniformly at random from the common winner set. The procedure for the SD *IID* learner is outlined in Algorithm 13. The algorithm initializes by taking a uniform random sample of arms for all units at time $t = 1$, i.e., $W_{i,1} \sim \mathcal{U}(\mathcal{W})$ for all $i \in \mathcal{I}$. Then the history and count for each arm k are formed based on which units selected arm k at time $t = 1$. So the initial history and count for all arms $k \in \mathcal{W}$ are $\mathcal{D}_1(k) = \{Y_{i,1} : i \in \mathcal{I}, W_{i,1} = k\}$, which means the set of rewards that picked arm k and $n_1(k) = \sum_{i=1}^N \mathbb{1}\{W_{i,1} = k\}$, which means the number of units that picked arm k at time $t = 1$. Then, the main loop starts.

In Algorithm 11, or for the single unit case at every round r, the winner set $\mathcal{A}_r$ is formed using three steps: *a) leader selection b) subsampling and c) dueling*. In our proposed SD *IID* learner, at each time step t, the learner forms one *common winner set* $\mathcal{A}_t$ using the same three steps. Then once the learner has the winner set, for each unit $i \in \mathcal{I}$, she selects an arm with $W_{i,t} \sim \mathcal{U}(\mathcal{A}_t)$. After selecting arms for all units, the learner observes rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, and finally updates the shared history and counts for each arm k based on for which units she selected arm k at time t. The update equations are as follows:

$$\left.\begin{aligned} \mathcal{D}_t(k) &= \mathcal{D}_{t-1}(k) \cup \{Y_{i,t} : i \in \mathcal{I}, W_{i,t} = k\} \\ n_t(k) &= n_{t-1}(k) + \textstyle\sum_{i=1}^N \mathbb{1}\{W_{i,t} = k\}, \end{aligned}\right\} \quad \forall k \in \mathcal{W}$$

We give some important remarks on the NPTS and SD *IID* learners below.

Remarks 3.4 (Discussion on Nonparametric *IID* Learners)**.**

IID Structure of NPTS and SDA: Both algorithms have the independent and identical structure, comparable to Remarks 2.16. In particular, for both algorithms, since the histories are shared across all units, this can be seen as identical structures in terms of the *empirical distribution*. For NPTS, the weights are generated for each unit separately, this is the *independent structure*. Now, when it comes to SDA, the complete independent structure would require us to form separate winner set for each unit at each round and then also have a round for each unit. However, what we find is, in terms of computation this is quite expensive to implement and also don't give significant improvement compared to the common winner set approach we took here. So for SDA we don't the have complete independent structure for all units since arms will be selected from a common winner set at each time step. But, since we do this uniformly at random, this brings some uniform variability in the arm selection for the units.

Has Strength and Weaknesses as IID learner: The NPTS *IID* learner has the same strengths and weaknesses of parametric *IID* learner discussed in Remarks 2.16. In particular, it works best when there is no interference or negligible interference among the units. Also in the setup with interference it assumes all units are independent and identical. So this brings the same mis-attribution issue as TS *IID* learner when interference is present. However, the SD *IID* learner, has some blessings in disguise. Since this is not fully independent across all units, it actually shows better performance compared to NPTS *IID* learner in scenarios when interference is present. We see this in the Section 3.7. The NPTS *IID* learner explores similar to the like TS *IID* learner, whereas the SD *IID* learner has a little less exploration due to common winner set across all units.

No Rounds for SDA IID Learner: As we have seen in the SD *IID* learner's algorithm, we do not need round counters anymore. This is because we compute the winner set at every time step and multiple arms can be played in parallel across units. An important point for the SD *IID* learner is, it doesn't guarantee that the learner pulls all arms in $\mathcal{A}_t$ at time t, since each unit independently samples from $\mathcal{A}_t$ so some arms may remain unselected. However, since usually we have $|\mathcal{A}_t| \ll N$, i.e., the size of the winner set is typically much smaller than N. And also for each unit we have $W_{i,t} \sim \mathcal{U}(\mathcal{A}_t)$, the probability is very high that all arms will be selected. Formally, $\mathbb{P}\left(\bigcap_{i=1}^{N} \{W_{i,t} \neq k\}\right) = (1 - 1/|\mathcal{A}_t|)^N$, which decays exponentially in N.

Direct but Asynchronous Extension of SDA: Now we get to the question, why not proposing SD *IID* as a direct extension of SDA. As we mentioned, the key reason is, this doesn't give significant improvement in performance compared to the common winner set approach we took here and brings more computational burden when implementing. This is what we find in numerical experiments. But we explain here how this direct exten-

sion could be implemented. The key difference would be, instead of forming a common winner set at each time, we keep the round based structure of SDA and keep a separate round counter for each unit. Then for each unit, for its respective round, we need to perform the three steps: *a) leader selection b) subsampling and v) dueling* separately to form a unit specific winner set. When it comes to pulling the arms, for each unit the learner pulls arms one by one from its own winner set sequentially over multiple time steps until the winner set for each unit is exhausted, then the learner forms new winner sets for each unit again. This creates an asynchronous version of SDA with independent round counters and winner sets. As we mentioned, the common winner set approach is simpler to implement and has similar performance, so we stick to that version here.

///

3.3 Nonparametric $G-1$ Learners

We now propose two nonparametric $G-1$ learners, NPTS $G-1$ and SD $G-1$ learners. The key principle of the $G-1$ learners is, the learners here assign the same arm to all the units at each time and the goal is to learn the optimal $G-1$ arm vector that minimizes the $G-1$ regret†. These algorithms are simplest to implement, essentially we proceed like the single unit cases and then assign $G-1$ arm vector to everyone, finally the update is performed only for the selected arm.

We start with the NPTS $G-1$ learner first. The core procedure is shown in Algorithm 14. The algorithm proceeds exactly like the single unit NPTS outlined in Algorithm 10. The only difference is, at line no. 6 when assigning arms, the learner assigns the same arm to all units in the batch, i.e., $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$ where $k' = \text{argmax}_{k \in \mathcal{W}} \hat{\mu}_t^*(k)$. Then after observing rewards from all units, the learner updates the shared history and count **only** for the selected arm k', and not for other arms. The update equations are as follows:

$$\left.\begin{aligned} \boldsymbol{y}_t^h(k') &= (\boldsymbol{y}_{t-1}^h(k'); \boldsymbol{Y}_t) \\ n_t(k') &= n_{t-1}(k') + N \end{aligned}\right\} \text{ only for selected } k' \in \mathcal{W}$$

where $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{N,t})$ is the reward vector observed at time t.

The procedure for the SD $G-1$ learner is outlined in Algorithm 15. It follows the same round based procedure as the single unit SDA, but now at every round, all the units will play the same arm from the winner set sequentially. At the initialization step, the learner initializes the history and count for each arm k by pulling each arm once across all units, i.e., $\mathcal{D}_1(k) = \{Y_{1,k}, \ldots, Y_{N,k}\}$, where $Y_{i,k}$ is the reward observed from unit i when arm k is assigned to all units, and $n_1(k) = N$ for all $k \in \mathcal{W}$. Then the main loop starts.

† The $G-1$ optimal arm vector and the $G-1$ regret is defined in Chapter 2, in particular in section Section 2.2.

Algorithm 14: NPTS $G-1$ Learner

Input: $\mathcal{I}$, T, $\mathcal{W}$, $y^b(k), \forall k \in \mathcal{W}$
Initialize: $n_0(k) = 0, \boldsymbol{y}_0^h(k) = (y^b(k)), \forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  sample $\boldsymbol{\omega}_t(k) \sim \text{Dirichlet}(\mathbf{1}_{n_{t-1}(k)+1}), \forall k \in \mathcal{W}$
  compute $\hat{\mu}_t^*(k) = \boldsymbol{\omega}_t(k)' \cdot \boldsymbol{y}_{t-1}^h(k), \forall k \in \mathcal{W}$
  select $k' = \text{argmax}_{k \in \mathcal{W}} \hat{\mu}_t^*(k)$
  set $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$ // assign same arm k' to all units
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  update
    $\boldsymbol{y}_t^h(k') = (\boldsymbol{y}_{t-1}^h(k'); \boldsymbol{Y}_t)$
    $n_t(k') = n_{t-1}(k') + N$

Algorithm 15: SD $G-1$ Learner

Input: $\mathcal{I}$, T, $\mathcal{W} = [K]$, $\mathcal{S}$
Initialize:
  $\mathcal{D}_1(k) = \{Y_{1,k}, \ldots, Y_{N,k}\}, n_1(k) = N, \forall k \in \mathcal{W}$
  $r = 2, t = K + 1$
while $t \leq T$ **do**
  select leader: $\ell_r = \text{argmax}_{k \in \mathcal{W}} n_{r-1}(k)$ // break ties with max. $\widehat{\mu}_{r-1}(k)$, else keep ℓ_{r-1}
  draw leader subsamples: $\mathcal{D}_{r,k}^* \sim \mathcal{S}(\mathcal{D}_{r-1}(\ell_r), n_{r-1}(k)), \forall k \in \mathcal{W} \backslash \{\ell_r\}$
  compute winner set: $\mathcal{A}_r = \{k \in \mathcal{W} \backslash \{\ell_r\} : \widehat{\mu}_{r-1}(k) \geq \widehat{\mu}_{r-1,k}^*\}$, or $\mathcal{A}_r = \{\ell_r\}$ if $\mathcal{A}_r = \emptyset$
  foreach $k' \in \mathcal{A}_r$ **do**
    pull arm vector $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$
    observe reward $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
    update
      $\mathcal{D}_r(k') = \mathcal{D}_{r-1}(k') \cup \{Y_{i,t} : \forall i \in \mathcal{I}\}$ // at time t all units play arm k'
      $n_r(k') = n_{r-1}(k') + N$ // increment count by number of units
      increment $t = t + 1$
      if $t > T$ **then**
        break // exit if t exceeds horizon
  increment $r = r + 1$ // move to next round
Output: $W_t, Y_t, \forall t \in [T]$

In the main loop, the procedure is same as the single unit SDA Algorithm 11 until line no. 9, where after the learner has formed the winner set $\mathcal{A}_r$, she selects an arm $k' \in \mathcal{A}_r$, and then unanimously assigns this same arm to all the units in the batch, i.e., sets $\boldsymbol{W}_t = k' \cdot \mathbf{1}_N$. The learner then observes rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$ and finally, she updates the shared history and count only for the selected arm k' and not for others. Then, she moves

to another arm in the winner set until all arms in the winner set are played. The update equations are as follows:

$$\left.\begin{aligned} \mathcal{D}_r(k') &= \mathcal{D}_{r-1}(k') \cup \{Y_{i,t} : \forall i \in \mathcal{I}\} \\ n_r(k') &= n_{r-1}(k') + N, \end{aligned}\right\} \quad \text{one } k' \in \mathcal{A}_r \text{ at time } t$$

Since both the $C-1$ learner and the $G-1$ learner have same targets, we give some important remarks on the $G-1$ learners after the discussion on nonparametric $C-1$ learners in Remarks 3.5.

3.4 Nonparametric $C-1$ Learners

Like the $G-1$ learners, the $C-1$ learners also target the optimal $G-1$ arm vector. However, in this setting the learner leverages the cluster information of the batch and clustered interference structure among the units to learn efficiently. Recall, we already defined the clustered interference in the bandit setup in Definition 2.3. In particular let $\mathscr{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_M\}$ be a clustering of the batch $\mathcal{I}$ into M clusters. Then, for this learner to work we need two assumptions stated in Assumption 2.20. The first one says, the interference exists only within clusters, this helps the learner to focus on learning from the cluster. The second assumption says, the cluster mean for any arm $k \in \mathcal{W}$, i.e., assigning arm k to all units in a cluster and then calculating the cluster average, is same as the global mean for arm k, i.e., assigning arm k to all units in the batch and then calculating the overall $G-1$ average. Under these two assumptions, we now propose the two nonparametric $C-1$ learners. First, let's discuss the NPTS $C-1$ learner, the procedure is outlined in Algorithm 16.

We start with the NPTS $C-1$ learner. The core procedure is shown in Algorithm 16. First couple of steps are exactly same as the IID learner. The only difference is, now the Dirichlet sampling and bootstrap mean calculations are done at the cluster level. So for each cluster $\mathcal{C}_m \in \mathscr{C}$ (rather than for each unit), for each arm $k \in \mathcal{W}$, the learner first generates the bootstrap weight vector $\boldsymbol{\omega}_t(k)$ from Dirichlet distribution. Then, she computes the weighted bootstrap mean $\hat{\mu}_t^*(k)$. After that she selects the arm that gives the highest bootstrap mean for the cluster and assigns this arm uniformly for all the units in the cluster i.e., she sets $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$, where $\boldsymbol{W}_{\mathcal{C}_m,t}$ is the treatment vector for all units in cluster $\mathcal{C}_m$ at time t. Note that these steps are combination of IID learner and $G-1$ learner, like IID learner we generate cluster specific weights and means, but like $G-1$ learner we assign the same arm to all units in a cluster. This resembles similar idea we had for the Thompson sampling $C-1$ learner in Section 2.3.3

After assigning arms to all clusters, the learner observes the rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ for all $i \in \mathcal{I}$, where $\boldsymbol{W}_t = (\boldsymbol{W}_{\mathcal{C}_1,t}; \ldots; \boldsymbol{W}_{\mathcal{C}_M,t})$, is the treatment vector for all units at time t. Finally,

Algorithm 16: NPTS $C-1$ Learner

Input: $\mathcal{I}$, T, $\mathcal{W}$, $\mathscr{C} = \{\mathcal{C}_m\}_{m\in[M]}$, $y^b(k), \forall k \in \mathcal{W}$
Initialize: : $n_0(k) = 0, \boldsymbol{y}_0^h(k) = (y^b(k)), \forall k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  for $m \in M$ **do**
    sample $\boldsymbol{\omega}_t(k) \sim \text{Dirichlet}(\mathbf{1}_{n_{t-1}(k)+1}), \forall k \in \mathcal{W}$ // probability sampling from simplex
    compute $\hat{\mu}_t^*(k) = \boldsymbol{\omega}_t(k)' \cdot \boldsymbol{y}_{t-1}^h(k), \forall k \in \mathcal{W}$ // weighted bootstrap mean
    select $k^{(m)} = \text{argmax}_{k\in\mathcal{W}}\, \hat{\mu}_t^*(k)$
    set $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  update
    $\boldsymbol{y}_t^h(k) = \left(\boldsymbol{y}_{t-1}^h(k);\, (\boldsymbol{Y}_{\mathcal{C}_m,t})_{m:\,k^{(m)}=k}\right), \forall k \in \mathcal{W}$ // where $\boldsymbol{Y}_{\mathcal{C}_m,t} = (Y_{i,t})_{i\in\mathcal{C}_m}$
    $n_t(k) = n_{t-1}(k) + \sum_{m\in[M]} N_m \cdot \mathbb{1}\{k^{(m)} = k\}, \forall k \in \mathcal{W}$
Output: $\boldsymbol{W}_t, \boldsymbol{Y}_t, \forall t \leq [T]$

Algorithm 17: SD $C-1$ Learner

Input: $\mathcal{I}$, T, $\mathcal{W} = [K]$, $\mathscr{C} = \{\mathcal{C}_m\}_{m\in[M]}$, $\mathcal{S}$
Initialize:
  $\mathcal{D}_1(k) = \left\{Y_{i,t} : i \in \mathcal{C}_m, k^{(m)} = k\right\}$, with $k^m \sim \mathcal{U}(\mathcal{W})$ and $\boldsymbol{W}_{\mathcal{C}_m,1} = k^{(m)} \cdot \mathbf{1}_{N_m}, \forall k \in \mathcal{W}$
  $n_1(k) = \sum_{m\in[M]} N_m \cdot \mathbb{1}\{k^{(m)} = k\}, \forall k \in \mathcal{W}$
for $t \in \{2, \ldots, T\}$ **do**
  select leader: $\ell_t = \text{argmax}_{k\in\mathcal{W}}\, n_{t-1}(k)$ // break ties with max. $\hat{\mu}_{t-1}(k)$, else keep ℓ_{t-1}
  draw leader subsamples: $\mathcal{D}_{t,k}^* \sim \mathcal{S}(\mathcal{D}_{t-1}(\ell_t), n_{t-1}(k)), \forall k \in \mathcal{W}\backslash\{\ell_t\}$
  compute winner set: $\mathcal{A}_t = \{k \in \mathcal{W}\backslash\{\ell_t\} : \hat{\mu}_{t-1}(k) \geq \hat{\mu}_{t-1,k}^*\}$, or $\mathcal{A}_t = \{\ell_t\}$ if $\mathcal{A}_t = \emptyset$
  for $m = 1, 2, \ldots M$ **do**
    select $k^{(m)} \sim \mathcal{U}(\mathcal{A}_t)$ // uniform sampling from winner set
    set $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$
  observe rewards $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  update
    $\mathcal{D}_t(k) = \mathcal{D}_{t-1}(k) \cup \left\{Y_{i,t} : i \in \mathcal{C}_m, k^{(m)} = k\right\}, \forall k \in \mathcal{W}$ // pulling across units
    $n_t(k) = n_{t-1}(k) + \sum_{m\in[M]} N_m \cdot \mathbb{1}\{k^{(m)} = k\}, \forall k \in \mathcal{W}$
Output: $\boldsymbol{W}_t, \boldsymbol{Y}_t, \forall t \leq [T]$

the learner updates the shared history and counts for all arms. The updates are similar to the IID learner but now the rewards and counts are gathered from the clusters completely, rather than for each unit separately. In particular, following are the updates,

$$\left.\begin{aligned} \boldsymbol{y}_t^h(k) &= \left(\boldsymbol{y}_{t-1}^h(k);\, (\boldsymbol{Y}_{\mathcal{C}_m,t})_{m:\, k^{(m)}=k}\right), \\ n_t(k) &= n_{t-1}(k) + \textstyle\sum_{m\in[M]} N_m \cdot \mathbb{1}\{k^{(m)} = k\}, \end{aligned}\right\} \quad \forall k \in \mathcal{W}$$

where for each arm $k \in \mathcal{W}$, the learner updates the shared history vector $\boldsymbol{y}_t^h(k)$ by concatenating the cluster reward vector whenever arm k is selected for cluster $\mathcal{C}_m$. This is what we wrote as $(\boldsymbol{Y}_{\mathcal{C}_m,t})_{m:\, k^{(m)}=k}$ where $\boldsymbol{Y}_{\mathcal{C}_m,t}$ is the reward vector for cluster $\mathcal{C}_m$ at time t, i.e., $\boldsymbol{Y}_{\mathcal{C}_m,t} = (Y_{i,t})_{i\in\mathcal{C}_m}$. And for the count update, the learner increments the count for arm k by the total number of units in all clusters where arm k was selected. This is written as $\sum_{m\in[M]} N_m \cdot \mathbb{1}\{k^{(m)} = k\}$.

Next, we explain the SD $C-1$ learner outlined in Algorithm 17. This is similar to the SD IID learner discussed in Algorithm 13 and follows the same *common winner set approach*. However rather than sampling uniformly for each unit separately from the winner set, the learner samples for each cluster now and then assigns the same arm uniformly for all the units in the cluster.

At the initialization step, the learner randomly samples an arm $k^{(m)}$ uniformly for each cluster $\mathcal{C}_m \in \mathscr{C}$, and then assigns this arm to all units in the cluster, i.e., sets $\boldsymbol{W}_{\mathcal{C}_m,1} = k^{(m)} \cdot \mathbf{1}_{N_m}$. Then she observes rewards from all units and initializes the history and counts for each arm k based on the cluster wise assignments. The history for each arm k is initialized as $\mathcal{D}_1(k) = \left\{Y_{i,t} : i \in \mathcal{C}_m, k^{(m)} = k\right\}$, i.e., collecting rewards from all clusters where arm k was assigned. The count for each arm k is initialized as $n_1(k) = \sum_{m\in[M]} N_m \cdot \mathbb{1}\{k^{(m)} = k\}$, i.e., summing up the number of units in all clusters where arm k was assigned.

Then the main loop starts. Here also, like NPTS IID, the algorithm closely follows the SD IID learner till the formation of the *common winner set* in line no. 8. After that, rather than sampling for each unit separately, the learner samples for each cluster uniformly from the winner set, i.e., for each cluster $\mathcal{C}_m$, she selects $k^{(m)} \sim \mathcal{U}(\mathcal{A}_t)$, where $\mathcal{A}_t$ is the common winner set formed at time t. Then she assigns this arm to all units in the cluster, i.e., sets $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$. After assigning arms to all clusters, the learner observes rewards from all units in the batch. So the key difference compared to the IID is, the arm selections are done uniformly from the winner set at the cluster level. Finally, the updated are done for all arm, but now the rewards and counts are gathered from the clusters. The update equations are as follows:

$$\left.\begin{aligned} \mathcal{D}_t(k) &= \mathcal{D}_{t-1}(k) \cup \left\{Y_{i,t} : i \in \mathcal{C}_m, k^{(m)} = k\right\} \\ n_t(k) &= n_{t-1}(k) + \textstyle\sum_{m\in[M]} N_m \cdot \mathbb{1}\{k^{(m)} = k\} \end{aligned}\right\} \quad \forall k \in \mathcal{W}$$

We now discuss some important remarks on the nonparametric $G-1$ and $C-1$ learners below.

Remarks 3.5 (Discussion on Nonparametric $C-1$ Learners)**.**

Strength and Weaknesses as $G-1$ and $C-1$ Learners: In Chapter 2: Both class of algorithms, the $G-1$ class: NPTS and SDA $G-1$ and the $C-1$ class: NPTS and SDA $C-1$, follow the same strengths as the respective Thompson sampling $G-1$ and $C-1$ learners (see Remarks 2.19 and see Remarks 2.23). That is the $G-1$ learners find the optimal $G-1$ arm vector and in the interference setting they don't suffer from the miss-attribution issue like the IID learners. However, as the parametric case, the $G-1$ learners suffer from high initial regret because of the over exploitation at the initial phase. They assign the whole $G-1$ vector to all units at each time, so initially for any particular arm they don't have much exploration. This creates high regret at the initial phase. Here the $C-1$ learners although having the same target as the $G-1$, take advantage of the cluster structure to explore more efficiently. So generally, they perform better and have less initial regret. However, the $C-1$ learners depend heavily on the cluster assumptions, if these assumptions are violated then their performance affects negatively. SD $C-1$ is not the cluster-wise independent extension, and like the SD IID, has the same common winner set approach. This implementation is easier, however depends on uniform sampling from the winner set, so there are some dependence. We discuss the details regarding this in a separate remark below.

Advantages over the Parametric $G-1$ or $C-1$ learners: The nonparametric $G-1$ learners have a particular advantage over the parametric Thompson sampling $G-1$ learner we proposed in Algorithm 7. That is, we don't need to worry about whether the mean of the rewards follow the same distribution. In fact the learners here are completely distribution free and work for any reward distribution as long as we can sample from the history. Recall, the Thompson sampling $G-1$ learner directly puts prior assumption on the means and then updates the prior sequentially. However, the issue is, the mean or sum may not follow the same class of distribution as the unit wise individual rewards. We gave a concrete example for the Bernoulli distribution in Example 2.17. Recall, we assume Beta prior on the mean of possibly dependent Bernoulli rewards. This problem is solved in this setting, since can avoid the distributional assumptions.

Asynchronous Extension of SD $C-1$ Learner: Like the SD IID version (see Remarks 3.4), it's also possible to propose an asynchronous extension of the SD $C-1$ learner. The idea is similar, we need to keep a separate round counter and form winner sets per cluster, and then progress asynchronously with their rounds. We implemented this initially, and find this don't bring significant improvement in numerical performance compared to the common winner set approach we took here and rather brings more computational burden because of separate round counters and winner sets. So although the common winner set approach has some dependence across clusters because of uniform sampling,

but we stick to this common winner set approach

///

3.5 Nonparametric $C-M$ Learners

Finally, we propose the nonparametric versions of the clustered multi learners or $C-M$ learners. Recall, we discussed the Thompson sampling $C-M$ learner in Chapter 2, Section 2.3.4. Here, like other classes, we propose two nonparametric versions, NPTS $C-M$ and SD $C-M$ learners. The key idea and the underlying assumptions are same, that is, given some cluster information these learners look for the $C-1$ optimal arm vector, but now the learners are resampling based.

The $C-1$ optimal arm vectors are defined based on some clustering. Let $\mathscr{C}=\{\mathcal{C}_1,\ldots,\mathcal{C}_M\}$ be a clustering of $\mathcal{I}$. Then, the $C-1$ optimal arm vector is defined as the arm vector that assigns the optimal uniform arm for each cluster separately[‡]. Note that, it's not unique for the batch, and depends on the clustering. But if the clusters can sufficiently capture the interference structure among the units, then using this clustering, the $C-M$ learner finds a combination arm vector that usually gives higher reward compared to the $G-1$ arm vector since it will find cluster level optimal $G-1$[§]. This is the key advantage of the $C-M$ learners over other learners. Like the parametric case, the nonparametric $C-M$ learners also depend on the clustered interference assumption stated in Assumption 2.20.

We start with the NPTS $C-M$ learner, the procedure is outlined in Algorithm 18. Essentially, the NPTS $C-M$ learner is M separate $G-1$ learners for M clusters. The learner here separately records the shared history and counts for each cluster, for all arms. In particular, at time t, for cluster $\mathcal{C}_m$, where $m\in[M]$, the learner has its own independent history $\boldsymbol{y}^h_{m,t-1}(k)$ and count $n_{m,t-1}(k)$ for each arm $k\in\mathcal{W}$. Then for each cluster, for each arm $k\in\mathcal{W}$, the learner samples weight vector $\boldsymbol{\omega}_{m,t}(k)\sim\text{Dirichlet}\left(\mathbf{1}_{n_{m,t-1}(k)+1}\right)$, computes the bootstrap mean $\hat{\mu}^*_{m,t}(k)=\boldsymbol{\omega}_{m,t}(k)'\cdot\boldsymbol{y}^h_{m,t-1}(k)$ using cluster specific weights and history, selects clusterwise arm $k^{(m)}=\text{argmax}_{k\in\mathcal{W}}\,\hat{\mu}^*_{m,t}(k)$ and assigns this arm to all the units in the cluster, i.e., $\boldsymbol{W}_{\mathcal{C}_m,t}=k^{(m)}\cdot\mathbf{1}_{N_m}$. The key difference from the $C-1$ learner is, here the history are different, whereas in $C-1$ learner the history is shared across clusters.

When the learner pulls arm for all the clusters, the full assignment vector is formed $\boldsymbol{W}_t=(\boldsymbol{W}_{\mathcal{C}_1,t},\ldots,\boldsymbol{W}_{\mathcal{C}_M,t})$ and then the learner can observe rewards $Y_{i,t}=Y_{i,t}(\boldsymbol{W}_t)$ for all $i\in\mathcal{I}$, which might have possible interference, but assumed to be present only within clusters. Finally, the learner updates the history and counts for each cluster separately. The updates are in this case cluster specific, so for a cluster $\mathcal{C}_m$, if the learner selected arm $k^{(m)}$ at time t, then the history and count are updated as follows:

‡ For the detailed definition we refer the author to the discussion in Chapter 2, in particular Definition 2.6.

§ We give a motivating example in Chapter 2, in particular in Example 2.7

$$\left.\begin{aligned} \boldsymbol{y}^{h}_{m,t}\left(k^{(m)}\right) &= \left(\boldsymbol{y}^{h}_{m,t-1}\left(k^{(m)}\right); \boldsymbol{Y}_{\mathcal{C}_m,t}\right), \\ n_{m,t}\left(k^{(m)}\right) &= n_{m,t-1}\left(k^{(m)}\right) + N_m \end{aligned}\right\} \quad \text{for arm } k^{(m)} \in \mathcal{W}, \text{ for cluster } \mathcal{C}_m$$

where $\boldsymbol{Y}_{\mathcal{C}_m,t} = (Y_{i,t})_{i\in\mathcal{C}_m}$ denotes the reward vector for cluster m.

Next, we discuss the SD $C-M$ learner outlined in Algorithm 19. Since this algorithm is somewhat different from other SD learners, we provide a detailed explanation of the procedure here. Following the general idea of the $C-M$ learners, the SD $C-M$ learner has M different SD $G-1$ learners, one for each cluster. For each cluster $\mathcal{C}_m$ and for each arm $k \in \mathcal{W}$, it has a separate history set $\mathcal{D}_{m,t}(k)$ and count $n_{m,t}(k)$. However, now, it has a separate round counter r_m for each cluster, and each cluster calculates its own winner set $\mathcal{A}_{m,r_m}$ at its local round r_m. This is completely different from all other SD learners we discussed so far, in all cases there was one winner set, but here we have M different winner sets, one for each cluster.

Now, at any time t, the winner set sizes for any cluster $\mathcal{C}_m$ can be different for different clusters. So here the learner additionally needs a *winner set arm counter* to track when all arms of any cluster level winner set have been pulled. This is denoted with $\texttt{counter}_m$. Essentially, whenever the learner pulls an arm for cluster m and assigns it to all units in that cluster, the count increments by 1 until it reaches to the size of the winner set $|\mathcal{A}_{m,r_m}|$. Then, it resets the counter to zero. Here the round is complete, so round counter r_m now increases by 1 and here the learner needs to do the winner set calculation for this cluster round again. This suggests, this is in fact the asynchronous version of the SDA. Asynchronous in a sense that we have different round counters for different clusters and the winner set calculation happens at different states in the algorithm.

We explain some steps. For each cluster $\mathcal{C}_m$, the learner pulls all arms once for all the units in the batch. This is essentially the initialization phase for each cluster. This is what we have as $\mathcal{D}_{m,1}(k) = \{Y_{1,k}, \ldots, Y_{N_m,k}\}$, and also the counts are initialized as $n_{m,1}(k) = N_m$, for all $m \in [M]$ and for all $k \in \mathcal{W}$. Now the round counter for each cluster $\mathcal{C}_m$, is $r_m = 1$ and the winner set arm counter for this cluster is $\texttt{counter}_m = 0$. Then from $t = 2, \ldots, T$, at each time step t, for each cluster $\mathcal{C}_m$, the learner first checks whether the winner set arm counter, which is $\texttt{counter}_m = 0$. If yes, then the learner calculates the winner set $\mathcal{A}_{m,r_m}$ for cluster $\mathcal{C}_m$. This calculation is same as the other SD learners, and involves the same three steps (lines 7-11). At line 11, the learner randomly orders the winner set $\mathcal{A}_{m,r_m}$ just to bring some randomness when pulling the arms from the winner set. The $\texttt{counter}_m$ is then incremented by one (line 12), and the learner selects the arm at the position $\texttt{counter}_m$ in the ordered winner set $\mathcal{A}_{m,r_m}$ (line 13). After that the learner assigns this arm to all units in cluster $\mathcal{C}_m$.

Once all units in cluster $\mathcal{C}_m$ have received the cluster level arm. The learner checks whether

Algorithm 18: NPTS $C-M$ Learner

Input: $\mathcal{I}$, T, $\mathcal{W}$, $\mathscr{C} = \{\mathcal{C}_m\}_{m\in[M]}$, $y^b_m(k) \forall m \in [M], k \in \mathcal{W}$
Initialize: $n_{m,0}(k) = 0, \boldsymbol{y}^h_{m,0}(k) = y^b_m(k), \forall m \in [M], k \in \mathcal{W}$
for $t = 1, 2, \ldots, T$ **do**
  for $m = 1, 2, \ldots, M$ **do**
    sample $\boldsymbol{\omega}_{m,t}(k) \sim \text{Dirichlet}(\mathbf{1}_{n_{m,t-1}(k)+1}), \forall k \in \mathcal{W}$
    compute $\hat{\mu}^*_{m,t}(k) = \boldsymbol{\omega}_{m,t}(k)' \cdot \boldsymbol{y}^h_{m,t-1}(k), \forall k \in \mathcal{W}$
    select $k^{(m)} = \text{argmax}_{k\in\mathcal{W}} \hat{\mu}^*_{m,t}(k)$
    set $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  for $m = 1, 2, \ldots, M$ **do**
    update
      $\boldsymbol{y}^h_{m,t}(k^{(m)}) = \left(\boldsymbol{y}^h_{m,t-1}(k^{(m)}); \boldsymbol{Y}_{\mathcal{C}_m,t}\right)$ // where $\boldsymbol{Y}_{\mathcal{C}_m,t} = (Y_{i,t})_{i\in\mathcal{C}_m}$
      $n_{m,t}(k^{(m)}) = n_{m,t-1}(k^{(m)}) + N_m$

Algorithm 19: SD $C-M$ Learner

Input: T, N, $\mathcal{W} = [K]$, $\mathcal{I}$, $\mathscr{C} = \{\mathcal{C}_m\}_{m\in[M]}$, $\mathcal{S}$
Initialize:
  $\mathcal{D}_{m,1}(k) = \{Y_{1,k}, \ldots, Y_{N_m,k}\}, n_{m,1}(k) = N_m, \forall m \in [M], \forall k \in \mathcal{W}$ // randomly order $\mathcal{W}$
  set $r_m = 1$ and $\texttt{counter}_m = 0, \forall m \in [M]$ // $\texttt{counter}_m$ tracks arms for cluster m
for $t = 2, \ldots, T$ **do**
  for $m = 1, 2, \ldots, M$ **do**
    if $\texttt{counter}_m = 0$ **then**
      select leader: $\ell_{m,r_m} = \text{argmax}_{k\in\mathcal{W}} n_{m,t-1}(k)$
      draw subsamples: $\mathcal{D}^*_{m,r_m,k} \sim \mathcal{S}(\mathcal{D}_{m,t-1}(\ell_{m,r_m}), n_{m,t-1}(k)), \forall k \in \mathcal{W}\backslash\{\ell_{m,r_m}\}$
      compute the set of winners: $\mathcal{A}_{m,r_m} = \{k \in \mathcal{W}\backslash\{\ell_{m,r_m}\} : \widehat{\mu}_{m,t-1}(k) \geq \widehat{\mu}^*_{m,r_m-1,k}\}$
       or $\mathcal{A}_{m,r_m} = \{\ell_{m,r_m}\}$if $|\mathcal{A}_{m,r_m}| = 0$
      randomly order $\mathcal{A}_{m,r_m}$
    increment index: $\texttt{counter}_m = \texttt{counter}_m + 1$
    select $k^{(m)} = \mathcal{A}_{m,r_m}[\texttt{counter}_m]$
    set $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$
    if $\texttt{counter}_m = |\mathcal{A}_{m,r_m}|$ **then**
      reset $\texttt{counter}_m = 0$
      increment $r_m = r_m + 1$
  observe $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t), \forall i \in \mathcal{I}$
  for $m = 1, 2, \ldots, M$ **do**
    update
      $\mathcal{D}_{m,t}(k^{(m)}) = \mathcal{D}_{m,t-1}(k^{(m)}) \cup \boldsymbol{Y}_{\mathcal{C}_m,t}$
      $n_{m,t}(k^{(m)}) = n_{m,t-1}(k^{(m)}) + N_m$

the size of $\texttt{counter}_m$ is same as the size of the winner set $|\mathcal{A}_{m,r_m}|$, if yes, for this cluster the winner set is exhausted, so the learner resets the counter to zero and increments the round counter r_m by one (lines 15-17). After that, the learner moves to the next cluster and repeats the same procedure. Once all clusters have selected their arms and assigned to their units, the learner can observe the rewards for all units in the batch with possible interference, this is in line 18. Finally, the learner updates the history and counts for each cluster separately but only for the arms that were selected in each cluster. So the updates equations are

$$\left.\begin{aligned} \mathcal{D}_{m,t}(k^{(m)}) &= \mathcal{D}_{m,t-1}(k^{(m)}) \cup \mathbf{Y}_{\mathcal{C}_m,t} \\ n_{m,t}(k^{(m)}) &= n_{m,t-1}(k^{(m)}) + N_m \end{aligned}\right\} \quad \text{for arm } k^{(m)} \in \mathcal{A}_{m,r_m} \text{ for cluster } \mathcal{C}_m$$

We discuss some important remarks on the nonparametric $C-M$ learners below.

Remarks 3.6 (Discussion on Nonparametric $C-M$ Learners)**.**

Strengths and Weaknesses as $C-M$ Learners: The nonparametric $C-M$ learners have the same strengths and weaknesses as the $C-M$ learner in the parametric setting we discussed in Remarks 2.26. In particular, under clustered interference both nonparametric $C-M$ learners find the optimal $C-1$ arm vector efficiently. However, if our target is to learn the global optimal $G-1$ arm vector, then these learners may not perform well, simply because it's learning cluster level optimal arms independently.

SD $C-M$ Learner is an Asynchronous Extension: The SD $C-M$ learner is a bit involved and not a straightforward extension compared to other SD learners. One could think of a common winner set approach for all clusters similar to SD $C-1$ learner, but that would not be aligned with the $C-M$ learner idea that is to learn cluster specific optimal arms independently, i.e., for each arm $k \in \mathcal{W}$, it will be learning same parameter for all clusters, whereas we need to learn cluster specific parameters here. The SD $C-M$ learner overcomes this issue by having separate round counters and winner set arm counters for each cluster. So here as if we are learning M separate SD $G-1$ learners simultaneously. But the problem is, at any time step t, different clusters can be at different rounds, and their winner sets can be of different sizes. So we need to track when all arms in a cluster level winner set have been pulled completely, and only then we can do the winner set calculation for that cluster again. This idea makes the SD $C-M$ learner somewhat different and asynchronous in a sense that different clusters can be at different rounds at any time step t.

///

3.6 Comparison Tables

Finally we give a comparison table of all the NPTS algorithms discussed so far in Table 3.1.

	Nonparametric Thompson sampling learners			
Feature	IID	$G-1$	$C-1$	$C-M$
History per arm k	Shared: $\boldsymbol{y}_t^h(k)$	Shared: $\boldsymbol{y}_t^h(k)$	Shared: $\boldsymbol{y}_t^h(k)$	Cluster-specific: $\boldsymbol{y}_{m,t}^h(k)$
Weight generation	per unit	per batch	per cluster	per cluster
Arm assignment	Unit-wise: $W_{i,t}$ may vary	Global: $\boldsymbol{W}_t = k \cdot \mathbf{1}_N$	Clustered: $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$	Clustered: $\boldsymbol{W}_{\mathcal{C}_m,t} = k^{(m)} \cdot \mathbf{1}_{N_m}$
History update	For each arm played by different units	Only for the selected arm from all units	For each arm played by different clusters	For each cluster for the selected arm
History growth per arm k	Sum of units that selected arm k at t	Exactly N per t if k is selected for the batch	Sum of cluster sizes for clusters that selected arm k at t	For cluster m, sum of cluster size if k selected for that cluster at t
Exploration diversity	High	Low	Medium	Medium
Best use case	Best single arm under no interference	Uniform optimal arm under interference	Uniform optimal arm under interference (cluster $\approx$ batch means)	Cluster-specific optimal (different from uniform optimal), under clustered interference

Table 3.1: Comparison of NPTS Learners for Batched Settings

	Subsampling Dueling Learners			
Feature	IID	$G-1$	$C-1$	$C-M$
History per arm k	Shared	Shared	Shared	Cluster-specific
Round-based	No	Yes (single round)	No	Cluster-wise
Winner set formation	Per time	Per round	Per time	Per cluster per round
Arm assignment	Uniformly random per unit from common winner set	Sequentially assign from single winner set	Uniformly random per cluster from common winner set	For each cluster, sequentially assign from cluster-specific winner set
History update	For each arm from all units uniformly from same winner set	One by one for all arms in the winner set	For each arm from each cluster uniformly from same winner set	For each cluster for the selected arm in the winner set
History growth per arm k	Sum of units that selected arm k from winner set uniform sampling at t	Exactly N per round if k is selected for the batch in the winner set	Sum of cluster sizes for clusters that selected arm k from winner set uniform sampling at t	For each cluster, sum of cluster size if k selected for that cluster at t in the winner set
Exploration diversity	Medium	Low	Medium	Medium
Best use case	Best single arm under no interference	Uniform optimal arm under interference	Uniform optimal arm under interference (cluster $\approx$ batch means)	Cluster-specific optimal (different from uniform optimal), under clustered interference

Table 3.2: Comparison of SD Learners for Batched Settings

3.7 Numerical Experiments

In this section we evaluate the performance of the nonparametric learners that we have proposed in this chapter through numerical experiments. As benchmark algorithms, we consider the Thompson Sampling (TS) learners discussed in Chapter 2. The direct effect and spillover effect parameters are set similarly for two interference scenarios as in Section 2.5, i.e., No Optimal Switching (No-OS) and Optimal Switching (OS). However, we consider the 5-arm bandit setting.

3.7.1 Network Structure and Clustering

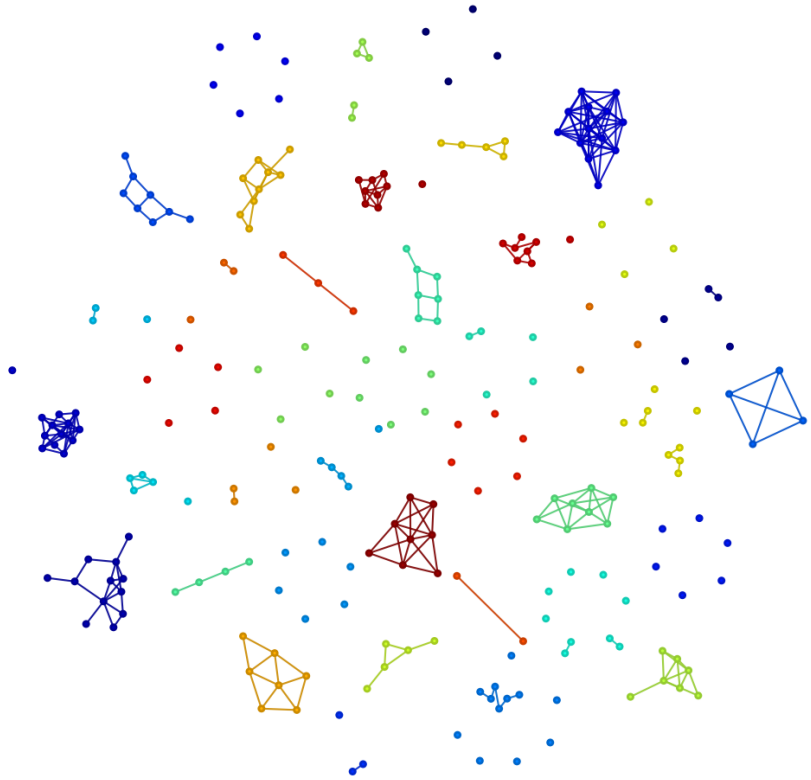


Figure 3.2: Network structure for the 5-arm bandit simulation with $N = 250$ units divided into $M = 40$ clusters.

We set the batch size $N = 250$ and total cluster number $M = 40$. In this chapter. In this chapter, unlike the random geometric graph (RGG) structure used in Chapter 2, we use stochastic block model (SBM) to generate a block-structured network. The intra-cluster connection probabilities are varying in $\{0.05, 0.1, 0.3, 0.5, 0.7\}$, while the inter-cluster probabilities are set to 0, so that we have strict clustered interference structure. The plot is shown in Figure 3.2. The node locations in the network don't have any spatial meaning. The plot has been generated using the `networkx` library in Python with a layout such that

nodes within the same cluster are grouped together and the clusters are visually separated. In total there are 170 nodes which are connected and 80 isolated nodes.

3.7.2 Reward Generation and Interference Parameters

The rewards are generated using a 5 arms linear-in-means model ¶. According the model, if the learner pulls the arm vector $\boldsymbol{W}_t$, then the reward for unit i at time t is given by:

$$Y_{i,t}(\boldsymbol{W}_t) = \alpha + \sum_{k=1}^{5} \beta_k \cdot \mathbb{1}\{W_{i,t} = k\} + \sum_{k=1}^{5} \delta_k \sum_{j \in \mathcal{I}} A_{ij} \cdot \mathbb{1}\{W_{j,t} = k\} + \epsilon_{i,t} \quad \epsilon_{i,t} \sim \mathcal{N}(0,1), \tag{3.1}$$

where the adjacency matrix $\mathbf{A}$ is a row normalized matrix with $A_{ii} = 0$, and entries $A_{ij} \cdot \mathbb{1}\{W_{j,t} = k\}$ represent the fraction of neighbors of unit i that received arm k at time t. The experimental scenarios are similar to Section 2.5, that we have two interference scenarios: No Optimal Switching (No-OS) and Optimal Switching (OS). The parameter configurations along with the batch average of $G-1$ arm vectors, are given in Table 3.3.

Arm k	Baseline	No-OS (1)			OS (2)		
		Direct	Spillover	Global Avg	Direct	Spillover	Global Avg
	α	β_k	δ_k	$\overline{\mu}(\boldsymbol{k}_{250})$	β_k	δ_k	$\overline{\mu}(\boldsymbol{k}_{250})$
1	0.5	0.4	0.0	0.9000	0.9	0.0	1.4000
2		0.5	0.1	1.0680	0.8	0.21	1.4428
3		0.6	0.22	1.2496	0.7	0.42	1.4856
4		0.7	0.33	1.4244	0.6	0.64	1.5352
5		0.8	0.44	1.5992	0.5	0.85	1.5780

Table 3.3: Parameter configurations for the five-arm bandit with two interference scenarios. The baseline effect is $\alpha = 0.5$ for both cases. In the No Optimal Switching case, both direct and spillover effects increase together. In the Optimal Switching case, direct effects decrease while spillover effects increase. The global average $\overline{\mu}(\boldsymbol{k}_{250})$ shows the batch-level mean potential outcome for each arm. In both configurations, arm 5 is the $G-1$ optimal arm, highlighted in green.

For the No-OS (1) scenario, the $G-1$ optimal vector is same as the $C-1$ optimal vector, and also this best arm for each cluster. However for the OS scenario, the $G-1$ optimal arm is arm 5, but the cluster optimal arms are both arm 1 (for total 13 clusters) and arm 5 (for total 27 clusters), making the $C-1$ optimal arm vector an arm combination of these

¶ We discussed some details on the linear-in-means model in Section 2.2.4. This is a well known model in the economics and social sciences literature to capture peer effects and interference, initially proposed by Manski (1993). We also proposed K arm version of this model.

Cluster Index m	Count N_m	# Edges $\lvert E_m \rvert$	Optimal Arm $\boldsymbol{k}_{N_m}^{(m)*}$	Cluster Avg Arm 1 $\overline{\mu}_m(\mathbf{1}_{N_m})$	Cluster Avg Arm 5 $\overline{\mu}_m(\mathbf{5}_{N_m})$
1	4	0	1	1.4000	1.0000
2	5	1	1	1.4000	1.3400
3	12	18	5	1.4000	1.8500
4	13	33	5	1.4000	1.7846
5	13	54	5	1.4000	1.8500
6	6	0	1	1.4000	1.0000
7	6	0	1	1.4000	1.0000
8	3	1	5	1.4000	1.5667
9	8	9	5	1.4000	1.8500
10	4	6	5	1.4000	1.8500
11	12	5	5	1.4000	1.4250
12	6	0	1	1.4000	1.0000
13	5	3	5	1.4000	1.6800
14	3	1	5	1.4000	1.5667
15	5	5	5	1.4000	1.6800
16	9	2	1	1.4000	1.3778
17	5	1	1	1.4000	1.3400
18	7	8	5	1.4000	1.8500
19	4	3	5	1.4000	1.8500
20	8	18	5	1.4000	1.8500
21	6	0	1	1.4000	1.0000
22	4	0	1	1.4000	1.0000
23	5	4	5	1.4000	1.8500
24	7	12	5	1.4000	1.8500
25	5	5	5	1.4000	1.8500
26	4	0	1	1.4000	1.0000
27	9	4	5	1.4000	1.5667
28	5	5	5	1.4000	1.8500
29	9	14	5	1.4000	1.8500
30	7	12	5	1.4000	1.8500
31	4	1	5	1.4000	1.4250
32	3	0	1	1.4000	1.0000
33	3	1	5	1.4000	1.5667
34	2	1	5	1.4000	1.8500
35	3	2	5	1.4000	1.8500
36	6	0	1	1.4000	1.0000
37	5	0	1	1.4000	1.0000
38	8	8	5	1.4000	1.7438
39	9	15	5	1.4000	1.7556
40	8	19	5	1.4000	1.8500

Table 3.4: Cluster-level summary for OS (2) case, showing, cluster index m, unit counts in N_m, edge counts $\lvert E_m \rvert$, optimal arm assignments $\boldsymbol{k}_{N_m}^{(m)*}$, and average rewards when the whole cluster is assigned to arm 1 and arm 5, $\overline{\mu}_m(\mathbf{1}_{N_m})$ and $\overline{\mu}_m(\mathbf{5}_{N_m})$ respectively. Note that cluster averages are close to the average for $G-1$ arm vectors, which are 1.4 for arm 1 and 1.578 for arm 5.

two arms. In Table 3.4, we provide a cluster-level summary for the OS scenario, showing the optimal arm for each cluster along with the average reward for arm 1 and 5. We can see that for some clusters, the optimal arms are arm 1 and some are 5 and also the cluster means for these two arms are close to the batch means for these arms, which are 1.4 and

1.578 respectively. This makes the OS scenario a favorable case for the $C-1$ learners, since by learning the cluster optimal arms, they can also learn a good approximation of the $G-1$ optimal arm.

3.7.3 Comparison of All Learners

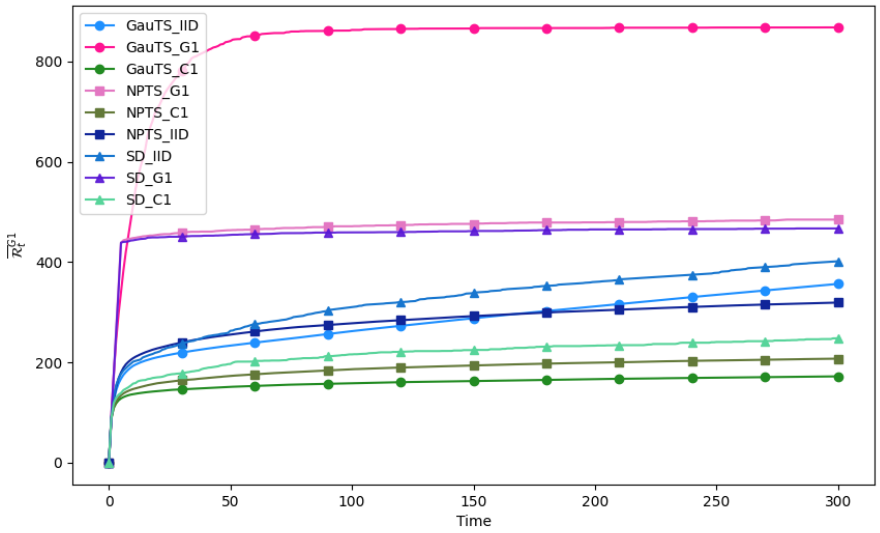


Figure 3.3: Comparison of $G-1$ regrets with $T=300$, over 1000 Monte Carlo simulations for the No-OS (1) case.

For the numerical experiments, we ran Monte Carlo simulations with 1000 runs for each learner in two interference scenarios, with the time horizon set to $T=300$. The mean and standard deviation of the final $G-1$ and $C-1$ regrets at time $T=300$ for all learners are given in Table 3.5. First, we explain the no optimal switching or **No-OS (1)** case. Recall, this is the situation when the optimal $G-1$ arm remains same with or without interference. Also, both $G-1$ and $C-1$ optimal arm vectors are same, and this is the global optimal for the entire batch and also unit-wise optimal. We didn't consider the $C-M$ learners here, since they target $C-1$ optimal arms by construction, and will incur higher regret. Among the other three types, IID, $G-1$ and $C-1$ learners, from the table we see that overall the $C-1$ learners perform the best, followed by the IID learners and then the $G-1$ learners. This is similar to the pattern what we have found in the parametric case in Section 2.5. Among the three $C-1$ learners, TS, NPTS and SD, the TS or Thompson sampling $C-1$ learner is the best, but NPTS is close, and then we have the SD $C-1$ learner.

Scenario	*IID*			$G-1$			$C-1$			$C-M$		
	TS	NPTS	SD	TS	NPTS	SD	TS	NPTS	SD	TS	NPTS	SD
(1) No Switching ($G-1$ / $C-1$ regret coincide)												
	356.6	**319.4**	401.7	867.8	484.8	**467.1**	**172.3**	207.6	248.2	—	—	—
	(1160.83)	(74.73)	(1149.92)	(466.06)	(41.73)	(35.34)	(47.00)	(35.65)	(196.40)			
(2) Optimal Switching — $G-1$ regret												
	13170.3	13123.1	**4064.4**	534.7	471.5	**416.7**	487.4	**435.2**	522.0	−2969.1	−2279.5	−2988.2
	(57.85)	(68.70)	(2527.20)	(243.50)	(140.08)	(173.13)	(535.28)	(228.45)	(571.61)	(299.72)	(218.71)	(275.22)
(2) Optimal Switching — $C-1$ regret												
	19410.3	19363.1	10304.4	6774.7	6711.5	6656.7	6727.4	6675.2	6762.0	3270.9	3960.5	**3251.8**
	(57.85)	(68.70)	(2527.20)	(243.50)	(140.08)	(173.13)	(535.28)	(228.45)	(571.61)	(299.72)	(218.71)	(275.22)

Table 3.5: Mean and standard deviations (in parentheses) of cumulative regret at $T = 300$ across 1000 Monte Carlo replications. Columns are grouped by learner type (IID, $G-1$, $C-1$, $C-M$) with subcolumns for each algorithm (TS, NPTS, SD). For No Switching (1), $G-1$ and $C-1$ regret coincide. For OS (2), they differ since cluster optimal arm vectors differ. The $C-M$ learners are only applicable in the OS (2) scenario.

For the IID learners, TS and NPTS learners are better than SD IID, and for the $G-1$ learners, SD $G-1$ performs best. We show the $G-1$ regret plots in Figure 3.3. The regret plots show similar conclusion. That is overall the $C-1$ learners (shown in green) outperform both $G-1$ and IID learners in this case. One reason why this happens is, by construction when learning the $G-1$ optimal arm, the $C-1$ learners explore in small batches so have less initial regrets while still being able to learn the $G-1$ optimal arm with some exploration.

When it comes to **Optimal Switching** scenario, which is the case when the optimal $G-1$ arm vector changes due to interference, we need to separately discuss the results for the $G-1$ and $C-1$ regret. First, for the $G-1$ regret, from the Table 3.5 and, first we can discard the IID learners since they perform poorly here. Also, the regret plot Figure 3.4 (top left) and (top right) shows the TS IID and NPTS IID learners have linear regret. Recall the reason for this is, we are now in the optimal switching case where the optimal arm is different from the no interference case, so the IID learners suffer from the wrong updates. The SD IID also has relatively high regret, but shows a sublinear pattern. We think this is because the SD IID does not have the complete independent structure like other two IID learners, since it has a common winner set approach. Overall both the $G-1$ and $C-1$ learners perform well here. From Table 3.5, the SD $G-1$ learner performs best, and then we have the NPTS $C-1$ and $G-1$ learners. The TS $C-1$ learner also performs well. We have a separate plot in Figure 3.4 (bottom left), which compares the $G-1$ and $C-1$ learners only. We also don't discuss the result for the $C-M$ learners here since by construction they target the $C-1$ optimal arm vector. However, the learner's goal in this case is to find the $G-1$ optimal or minimize $G-1$ regret and $C-M$ learners find a different target, which maybe better, but not intended. The proper regret to look at for $C-M$ learners in this case is the $C-1$ regret, which we discuss next.

Finally, for the $C-1$ regret in the OS scenario, from Table 3.5, we see that the SD $C-M$ learner performs the best among all learners. Recall, in this case, we are looking for a combination arm vector, which is a harder problem, so in general the regrets are higher than the $G-1$ regrets. The SD $C-M$ learner which is an asynchronous round-based learner, performs better than the Thompson sampling based $C-M$ learner. We think this is a remarkable result, since the SD $C-M$ learner is completely nonparametric.

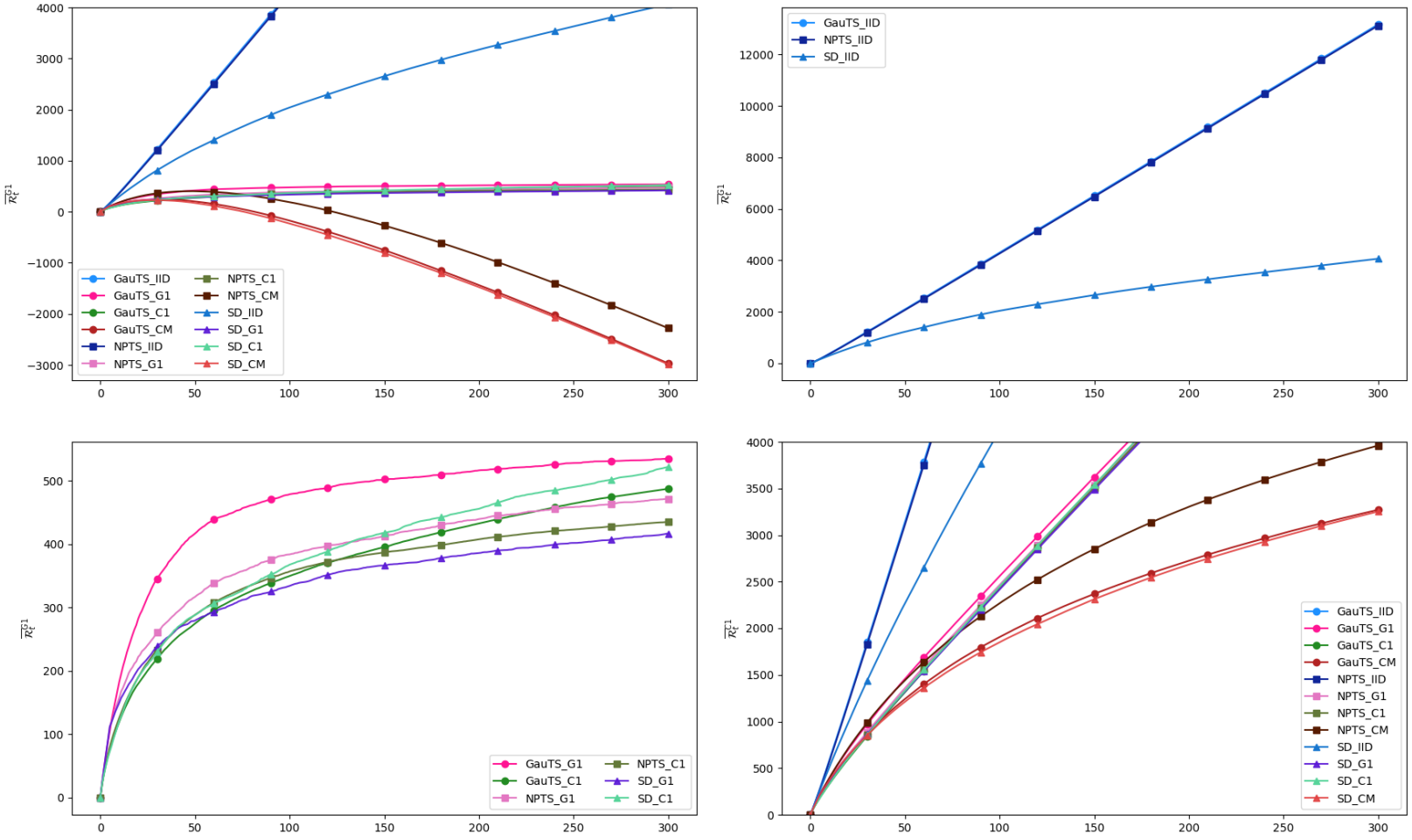


Figure 3.4: Comparison of $G-1$ and $C-1$ regrets computed with 1000 Monte Carlo simulations for $T = 300$ across learners in the Optimal Switching case. Top Left : All learners together for $G-1$ regret. Top Right: IID learners for $G-1$ regret. Bottom Left: $G-1$ and $C-1$ learners for $G-1$ regret. Bottom Right: All learners together for $C-1$ regret.

3.8 Conclusion

In this chapter, we proposed different nonparametric learners for batched interference setup that we have proposed in Chapter 2. The algorithms are primarily based on two resampling based bandit algorithms, nonparametric Thompson sampling (NPTS) and subsampling dueling algorithm (SDA). Both have strong theoretical guarantees and very different in design. We have adapted these algorithms to our batched interference setup and proposed four types of learners in each case, targeting different optimal arm vectors, i.e., IID, $G-1$, $C-1$ and $C-M$, following the similar principle as in Chapter 2. We outlined all the algorithms using the same potential reward framework, and provided some detailed description of the key features. Finally, we evaluated the performance of these learners through numerical experiments and compared them with the Thompson sampling based learners from Chapter 2. The results show that overall the nonparametric learners perform competitively with the Thompson sampling learners, and in some cases even better, especially the subsampling dueling learners. We believe, these algorithms can be applied to many real worlds batched bandit problems under interference, where the parametric assumptions may not hold. Further work includes theoretical analysis of these learners in our batched interference setup.

Chapter 4.

Learning Optimal Price Under Interference

(Unpublished)

In this chapter, we return to Example 1.1, the hawker example from Chapter 1. The example has been primarily motivated from Kleinberg and Leighton (2003) in the context of dynamic pricing. The story goes like this: a seller has an unlimited supply of some good, interacts with T buyers sequentially over $t = 1, \ldots, T$ periods, offers a price between 0 and 1, the buyer t buys or leaves, if buys the seller earns revenue at t, otherwise not. The goal of the seller is: *maximizing cumulative revenue from sales over T periods*, hence the seller needs to find optimal price that can be charged. Kleinberg and Leighton (2003) was interested to investigate: *what is the value of knowing the demand curve in this setting?* In particular, the question they investigated is related to - whether there is a difference for the seller in expected revenues, when the seller is informed, i.e., knows the demand curve Vs. the seller is uninformed and sets the prices according to some online strategy like multi armed bandit*. The answers they found is - it depends particularly on the buyers' valuation, types, i.e., whether, identical, random or adversarial and there will be always some cost for this discretization of the prices.

In this chapter, we take this problem, and question whether interference has any role in buyer's purchasing decision. In particular, we ask: what if the buyers are connected in a network and the purchase decision of a buyer depends not only on the price she has been offered, but also on the prices offered to her neighbors or someone connected to her in the network. Then how does the seller solve his problem. This is in fact, *the connection of dynamic pricing with interference*. We try to formalize the dynamic pricing problem raised by Kleinberg and Leighton (2003) under interference with the framework that we developed in Section 2.2, and also evaluate the algorithms in solving the seller's maximization problem.

This chapter can be seen as an illustration of our proposed framework in Chapter 2 and algorithms that we have developed and discussed in Chapter 2 and Chapter 3. In the dynamic pricing setting, there are several questions one can ask coming from an economic theory perspective, in particular, how this is connected to the mechanism design, auction theory or industrial organization (IO) literature. In the selling process often there could be bargaining, and how does this come into play. We don't attempt to answer all these ques-

* From the classical Microeconomic theory perspective, if the seller knows the demand curve, then it's possible to equate marginal revenue and marginal cost to get the profit maximizing quantity and also profit maximizing price! This is a very classical result, however the issue is, in practice we often don't know the market demand curve. One solution is *estimating market demand curve* from data and this falls under the literature of industrial organization (IO). However, in this work we don't go in that direction of demand curve estimation, and request the reader to look at Aguirregabiria (2021) for a recent review if interested. Our approach is more towards online learning with sequential experimentation.

tions here neither we try to convince the reader that our approach is the only approach for this modeling. But, we think there exists a gap here when it comes to interference in purchasing decision and pricing, and in this chapter we open some interesting directions for future research, both for us and others, and we would like to investigate these questions in our future work.

There are two novel aspects of this chapter, compared to the previous chapters. First, we formulate the dynamic pricing bandit problem with interference among buyers, where, a buyer's purchase decision depends not only on the price offered to her, but also on the prices offered to her neighbors. Second, we bring the random network in the setting. Until now, in Chapter 2 and Chapter 3, we have assumed a fixed set of nodes or units arriving at each time period. But this setup is inappropriate in the dynamic pricing case. Since in practice, a seller may have a large population of potential buyers connected through a social network, and at each time period only a random sample of buyers arrives to him. So we take this random sampling in our consideration in this chapter and see how the proposed learners can be applied in this setting. The recent work proposed by Gleich et al. (2025) also considers random sampling of nodes[†], but their approach is quite different from ours in several ways. First, they specify a probabilistic model of the network and generate samples independently over time, whereas we consider a fixed large spatial population network and then buyer meets a random sample from here at each time. This is motivated from the practical consideration related to the hawker example, where hawker has a target population in mind, and meets a subset of random buyers at each day. Second, they consider linear model underneath where the network is known and then learn the parameters using Thompson sampling based approach, where our algorithms are model agnostic, we don't assume the learner knows the adjacency matrix and use the clustered interference structure where the learner only knows the cluster information. Also, we apply the non-parametric learners that we have developed in Chapter 3, which makes the learners more flexible in terms of distributional assumptions.

This chapter is organized as follows. In Section 4.1 we outline the interaction protocol and model purchase decision under interference. In Section 4.2 we discuss how the proposed learners from Chapter 2 and Chapter 3 can be applied in this setting. In Section 4.3, we present simulation results to evaluate the performance of the learners. Finally, we conclude in Section 4.4 with a discussion on future research directions.

4.1 Protocol and Modeling Purchase Decisions Under Interference

Since this is an online learning problem. We need to clearly outline the interaction protocol and the seller's goal in this setting. In the following we first explain the case without

[†] To the best of our knowledge, except our attempt, this is the only paper which considers random network in the stochastic bandit setting.

interference, then we extend the model to include interference.

4.1.1 Modeling Purchasing Without Interference

Seller's Goal: Maximize Cumulative Revenue $\sum_{t=1}^{T} Y_t$

for $t = 1, 2, \ldots, T$ **do**
- a buyer arrives with a purchase probability function: $q_t(\cdot)$
- seller offers a price: $W_t \in \mathcal{W}$
- buyer buys or not: $X_t \in \{0, 1\}$
- the seller earns revenue: $Y_t = W_t X_t$

Interaction Protocol 5: Dynamic Pricing (Without Interference)

If we don't consider interference, the problem is simple and similar to the setting given in Kleinberg and Leighton (2003), except the author modeled with continuum of prices over $[0, 1]$ and a random buyer comes from a distribution of valuations. Whereas, we keep the buyer as a random entity, but apply for a discrete price set and model the purchase decision probabilistically with a Bernoulli coin flip. In particular, we model the buyer's purchase decision through potential outcome framework, where the probability of purchase for a buyer depends on the price offered to the buyer. We outline how the interaction takes place in Line 24.

The seller starts with K prices with a set $\mathcal{W} = \{p_1, \ldots, p_K\}$. At each time, a single buyer arrives, seller offers a price $W_t \in \mathcal{W}$, buyer decides whether to buy or not, this is denoted with a binary variable $X_t \in \{0, 1\}$. The seller earns revenue $Y_t = W_t \cdot X_t$, i.e., revenue is exactly equal to the price offered if the buyer purchases the good, otherwise zero. Now, since $X_t \in \{0, 1\}$, we propose to model the purchase decision with a Bernoulli coin flip, i.e.,

$$\begin{aligned} X_t &\sim \text{Bernoulli}\left(q\left(W_t\right)\right) \\ q\left(W_t\right) &= \sigma\left(\eta\left(W_t\right)\right) \\ \eta\left(W_t\right) &= \beta_0 - \beta_1 \log W_t \end{aligned} \tag{4.1}$$

where $q\left(W_t\right)$ is the purchase probability at price W_t. The sigmoid function $\sigma(\cdot)$ ensures the probability lies between 0 and 1. The term $\eta\left(W_t\right)$ is a semi-log function of price W_t, where increase in price leads to decrease in $\eta\left(W_t\right)$ and hence decrease in purchase probability[‡]. If we invoke the potential reward framework in this setup, then for k prices we have k potential purchase decisions for a buyer. So for each price $p_k \in \mathcal{W}$, we have a potential purchase decision variable $X_t(p_k) \sim \text{Bernoulli}\left(q(p_k)\right)$. Here $q(p_k)$ is the purchase probability at price p_k. So for K prices we have K potential purchase distributions. Since $X_t(p_k) \in \{0, 1\}$, the mean of this distribution simply tells what is the purchase probability at price p_k for a

‡ The interpretation here is the classic semi-log interpretation, so if price offered for unit i goes up by 1% then $\eta\left(W_t\right)$ goes down by approximately $\beta_1/100$.

buyer coming randomly from a population. Note that, here we have $\mathbb{E}\left[X_t(p_k)\right] = q(p_k)$. Now let $Y_t\left(p_k\right) = p_k \cdot X_t(p_k)$ be the corresponding potential revenue or potential reward for the seller, then the expected potential revenue is $\mu(p_k) = \mathbb{E}\left(Y_t\left(p_k\right)\right) = p_k \cdot q(p_k)$. When the seller offers price W_t, he observes the revenue $Y_t = Y_t\left(W_t\right)$, while the counterfactual revenues $\left\{Y_t\left(p_k\right) : p_k \neq W_t\right\}$ remain unobserved. The seller's goal is to maximize cumulative revenue over T days, i.e., maximize $\sum_{t=1}^{T} Y_t$, or in expectation

$$\text{maximize} \quad \sum_{t=1}^{T} \mathbb{E}\left[Y_t(W_t)\right] = \sum_{t=1}^{T} \mu(W_t) = \sum_{t=1}^{T} W_t \cdot q(W_t)$$

Or, equivalently, if we have an optimal price, i.e., then

$$\text{minimize regret, where pseudo regret is} \quad \mathcal{R}_T = \sum_{t=1}^{T} \left(\mu(p^*) - \mu(W_t)\right).$$

where $p^* = \text{argmax}_{p_k \in \mathcal{W}} \mathbb{E}\left[Y_t\left(p_k\right)\right] = \text{argmax}_{p_k \in \mathcal{W}} \mu\left(p_k\right) = \text{argmax}_{p_k \in \mathcal{W}} p_k \cdot q(p_k)$, is the best price or optimal price that the seller would like to learn over the selling horizon of T days and if this is offered the seller maximizes his expected revenue.

Remarks 4.1 (Connecting to Valuation Framework of Kleinberg and Leighton (2003))**.** There is a close connection with the potential outcome framework that we use to motivate the problem, and the valuation setup discussed in Kleinberg and Leighton (2003). In particular, let F_v be the population distribution of valuations for all buyers over $[0, 1]$. Under this setup a random buyer comes with valuation $v_t \sim \mathrm{F}_v$ and then the seller offers the price p_k. If the $v_t \geq p_k$, i.e., buyer's valuation is greater than or equal to the price offered, then the buyer purchases the good if. The proportion of buyers who purchase at price p_k is given by

$$q^D(p_k) = \mathbb{P}(v_t \geq p_k) = 1 - \mathrm{F}_v(p_k).$$

which is conventionally called the demand function at price p_k, since it shows what fraction of buyers would like to purchase at price p_k and if price increases, the fraction (or demand) decreases.

In our potential outcome framework, this is exactly the purchase probability at price p_k, i.e., $q(p_k) = q^D(p_k) = 1 - \mathrm{F}_v(p_k)$. So in this sense, our potential outcome framework models what is the probability that the random buyer would like to purchase at price p_k, and in the valuation framework of Kleinberg and Leighton (2003), this is same as the percentage of buyers whose valuation is greater than or equal to the price offered. Hence, our modeling of the purchase probability can be seen as an alternative way of

modeling the purchase decision. This would be useful when we extend the model to include interference.

///

4.1.2 Modeling Purchasing Under Interference

Now, we explain the problem under interference. Suppose, the seller with a possible price set $\mathcal{W} = \{p_1, p_2, \ldots, p_K\}$ now goes to sell his products in a market consisting of N potential buyers denoted with a set $\mathcal{I}$. This is the population set of buyers. At each day t, the seller meets a random sample of $n < N$ buyers in a set $\mathcal{I}_t \subset \mathcal{I}$ i.e., $|\mathcal{I}_t| = n$. The goal is same, that is: maximizing cumulative revenue over a selling horizon of T days. Here is how the interaction takes place,

Seller's Goal: Maximize Cumulative Revenue $\sum_{t=1}^{T} \sum_{i \in \mathcal{I}_t} Y_{i,t}$

for $t = 1, 2, \ldots, T$ **do**
  a random sample of buyers $\mathcal{I}_t \subset \mathcal{I}$ arrives with purchase probability function $\{q_{i,t}(\cdot)\}_{i \in \mathcal{I}_t}$
  foreach *buyer* $i \in \mathcal{I}_t$ **do**
    seller offers price $W_{i,t} \in \mathcal{W}$
    buyer buys or not $X_{i,t} \in \{0, 1\}$
    the seller earns revenue $Y_{i,t} = W_{i,t} X_{i,t}$

Interaction Protocol 6: Dynamic Pricing with Interference

At each time t, a random sample of buyers $\mathcal{I}_t$ arrives with a set of purchase probability functions $\{q_{i,t}(\cdot)\}_{i \in \mathcal{I}_t}$. For each buyer $i \in \mathcal{I}_t$, the seller offers a price $W_{i,t} \in \mathcal{W}$, buyer i decides whether to purchase or leave. The decision variable is denoted with $X_{i,t} \in \{0, 1\}$, where $X_{i,t} = 1$ mean the buyer i purchases the product, and $X_{i,t} = 0$ means otherwise. However, now the purchase decision of buyer i depends not only on the price $W_{i,t}$ offered to her, but also on the prices offered to other buyers in the sample $\mathcal{I}_t$. The revenue that the seller receives from a particular buyer i is $Y_{i,t} = W_{i,t} X_{i,t}$. The goal of the seller is to maximize cumulative revenue over the selling horizon of T days, i.e., $\sum_{t=1}^{T} \sum_{i \in \mathcal{I}_t} Y_{i,t}$.

As mentioned the interference arises because the purchase decisions of buyers are influenced by the prices offered to their neighbors or other buyers in the social network. For example, it could be the case that if a buyer finds out that her neighbor has been offered a lower price, she may perceive the pricing is unfair and decides not to buy at all. This is just one example of different interference scenarios that may arise. Like the setup without interference, we model the purchase variable $X_{i,t}$ through a Bernoulli coin flip, however now because of interference, the purchase probability of buyer i, denoted with $q_{i,t}(\cdot)$ depends, not only on the price offered to $i \in \mathcal{I}_t$, but also on the other units in $\mathcal{I}_t$, i.e., on the whole price vector $\boldsymbol{W}_t = (W_{1,t}, W_{2,t}, \ldots, W_{n,t})'$ at time t. The complete model can be written as follows:

$$
\begin{aligned}
X_{i,t} &\sim \text{Bernoulli}\left(q_{i,t}\left(\boldsymbol{W}_t\right)\right) \\
q_{i,t}\left(\boldsymbol{W}_t\right) &= \sigma\left(\eta_{i,t}\left(\boldsymbol{W}_t\right)\right), \\
\eta_{i,t}\left(\boldsymbol{W}_t\right) &= \alpha_i + \mu_{m(i)}(W_{i,t}) - \gamma V_{i,t}\left(\boldsymbol{W}_t\right).
\end{aligned} \tag{4.2}
$$

Now for interference effect, the term $\eta_{i,t}\left(\boldsymbol{W}_t\right)$ that goes inside this sigmoid function, can be modeled with a similar idea as the linear-in-means by Manski (1993), also we discussed in Section 2.2.4. We model this as a linear function of three components. First, α_i, this captures buyer i's unique preference over the product. This is something the buyer specific. Second, $\mu_{m(i)}(W_{i,t})$, is a function that models the preference of buyer i based on some groups or clusters that she belongs to. Third the term, $\gamma V_{i,t}\left(\boldsymbol{W}_t\right)$ has the interference parameter and the interference function $V_{i,t}\left(\boldsymbol{W}_t\right)$, where $V_{i,t}\left(\cdot\right)$ depends on the whole price vector $\boldsymbol{W}_t = (W_{1,t}, W_{2,t}, \ldots, W_{n,t})'$.

Now, we give more details for the two functions $\mu_{m(i)}(W_{i,t})$ and $V_{i,t}\left(\boldsymbol{W}_t\right)$. First, assume that the n buyers coming randomly, may belong to M different clusters or groups based on their characteristics or preferences towards the product. Let $m(i) \in \{1, 2, \ldots, M\}$ be the cluster that buyer i belongs to. The term $\mu_{m(i)}(W_{i,t})$ captures the willingness to purchase the product of buyer i, when offered price $W_{i,t}$, given that the buyer belongs to some group or cluster $m(i)$. This is actually same for all buyers in the same cluster. We call this the cluster specific willingness to purchase function, and as we explained, roughly it captures how buyer i's purchase decision is influenced by the price offered to her, given her cluster membership. This function can take different forms, including some non-linear forms. In our experiments we model this with a simple log-linear function, i.e., $\mu_{m(i)}(W_{i,t}) = \beta_{0,m(i)} - \beta_{1,m(i)} \log W_{i,t}$, this is the same semi-log form we used in eq. (4.1).

Similarly, the interference function $V_{i,t}\left(\boldsymbol{W}_t\right)$ can also take different forms, based on how we model interference. In our experiments we set the interference function as representation of mistrust or perceived unfairness due to price differences. In particular, we set, $V_{i,t}\left(\boldsymbol{W}_t\right) = \frac{1}{d_{i,t}} \sum_{j \in \mathcal{I}_t : A_{ij,t}=1} \left(W_{j,t} - W_{i,t}\right)^2$. Where, $A_{ij,t} = 1$ if buyers i and j are connected in the batch $\mathcal{I}_t$. The entries $A_{ij,t}$ are the elements of the adjacency matrix $\mathbf{A}_t$ for the sampled network at t. The term $d_{i,t} = \sum_j A_{ij,t}$ is the degree of buyer i for batch $\mathcal{I}_t$. So the interference term captures the average of the squared price difference in buyer i's local neighborhood at day t. If neighbors are offered prices very different from $W_{i,t}$, then $V_{i,t}\left(\boldsymbol{W}_t\right)$ penalizes the purchase probability for buyer i due to perceived unfairness or distrust. If all the neighbors are offered the same price as $W_{i,t}$, then there is no effect from the neighbors. So put it simply different prices offered to neighbors at time t influences buyer i's purchase probability negatively.

The idea of the potential purchase decision and potential reward is similar to the setup without interference. However, now because of interference, the potential purchase decision for buyer i at time t depends on the whole price vector $\boldsymbol{W}_t$. In general, we denote a

fixed price vector of length n with $\boldsymbol{p} \in \mathcal{W}^n$, where $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)'$ and $p_i \in \mathcal{W}$ for each $i \in [n]$. Then for a fixed price vector $\boldsymbol{p}$, we can denote potential purchase decision for buyer i at time t with $X_{i,t}(\boldsymbol{p}) \in \{0, 1\}$. The corresponding potential reward or potential revenue for the seller from buyer i at time t is then $Y_{i,t}(\boldsymbol{p}) = p_i \cdot X_{i,t}(\boldsymbol{p})$. The expected potential revenue is then $\mu_{i,t}(\boldsymbol{p}) = \mathbb{E}\left[Y_{i,t}(\boldsymbol{p})\right] = p_i \cdot q_{i,t}(\boldsymbol{p})$. When the seller offers price vector $\boldsymbol{W}_t$ at time t, he observes the revenue $Y_{i,t} = Y_{i,t}(\boldsymbol{W}_t)$ from each buyer $i \in \mathcal{I}_t$, while the counterfactual revenues $\{Y_{i,t}(\boldsymbol{p}) : \boldsymbol{p} \neq \boldsymbol{W}_t\}$ remain unobserved.

Now, to define the seller's goal or optimal price vector in this setting, we think about two possible scenarios, both are motivated from practical considerations and consistent with our optimal arm vectors that we defined in Section 2.2. In particular, consider our hawker example, at any day or time t, the hawker may sell in one location or in different locations in a city. If he goes to different locations, then he can offer different prices. This is the classic *differential pricing* or *price discrimination* in economics. This would help the seller to extract more revenue based on varying willingness to pay of buyers in different locations. However, if he stays in one location, it is unlikely that he can offer different prices to different buyers in the same location per day. Although initially he may start experimenting, but eventually he has to settle with one price for all buyers in the same location at a time period. This motivated us to consider the $G-1$ optimal arm vector and $C-1$ optimal arm vector as targets in this setting. In particular if the buyer offers one price to all buyers in a same day, regardless whether they belong to different locations, then this is similar to the $G-1$ arm vector. And, if he offers different prices to different locations per day, then this is similar to the $C-1$ arm vector, where the locations are the clusters in our setup. Let's denote the $G-1$ price vector be $\boldsymbol{p}_n = p \cdot \mathbf{1}_n$, for some $p \in \mathcal{W}$, i.e., price p is offered to all buyers in the sample $\mathcal{I}_t$ at time t. Now, the $G-1$ optimal price vector, denoted with $\boldsymbol{p}_N^* = p^* \cdot \mathbf{1}_N$ for some $p^* \in \mathcal{W}$, is the uniform price vector that maximizes expected revenue when offered to all buyers in the population, i.e.,

$$p^* = \underset{p_k \in \mathcal{W}}{\operatorname{argmax}} \sum_{i \in \mathcal{I}} \mu_i(p_k \cdot \mathbf{1}_N) = \underset{p_k \in \mathcal{W}}{\operatorname{argmax}} \sum_{i \in \mathcal{I}} p_k \cdot q_i(p_k \cdot \mathbf{1}_N),$$

With this p^*, the seller's goal is to maximize cumulative revenue over T days by learning this optimal uniform price p^*. Or equivalently, minimize $G-1$ regret, where pseudo regret is defined as

$$\begin{aligned} \mathcal{R}_T^{g1} &= \sum_{t=1}^{T} \sum_{i \in \mathcal{I}_t} \left(\mu_{i,t}(p^* \cdot \mathbf{1}_n) - \mu_{i,t}(\boldsymbol{W}_t)\right) \\ &= \sum_{t=1}^{T} \sum_{i \in \mathcal{I}_t} \left(p^* \cdot q_{i,t}(p^* \cdot \mathbf{1}_n) - W_{i,t} \cdot q_{i,t}(\boldsymbol{W}_t)\right). \end{aligned} \tag{4.3}$$

The other options is the seller want to offer location specific prices to different locations per

day. In our setup, this is actually the $C-1$ arm vectors, or in this context, the price vectors. We will denote the $C-1$ price vector as $\boldsymbol{p}_{C-1,N} = (p_1 \cdot \mathbf{1}_{N_1}, p_2 \cdot \mathbf{1}_{N_2}, \ldots, p_M \cdot \mathbf{1}_{N_M})'$, where N_m is the number of buyers in cluster $\mathcal{C}_m$ and $p_m \in \mathcal{W}$ is the price offered to all buyers in cluster $\mathcal{C}_m$. Recall, we call this the clustered one or $C-1$ price vector, since there is one price per cluster. With this we can also define the $C-1$ optimal price vector, denoted with $\boldsymbol{p}^*_{C-1,N}$, which is essentially the combination price vector from K^M price vectors that maximizes expected revenue when offered to all buyers in the population, i.e.,

$$\boldsymbol{p}^*_{C-1,N} = \underset{\boldsymbol{p}_{C-1,N} \in \mathcal{W}^{c1}}{\operatorname{argmax}} \sum_{i \in \mathcal{I}} \mu_i(\boldsymbol{p}_{C-1,N}) = \underset{\boldsymbol{p}_{C-1,N} \in \mathcal{W}^{c1}}{\operatorname{argmax}} \sum_{i \in \mathcal{I}} W_i \cdot q_i(\boldsymbol{p}_{C-1,N})$$

Now, we can also define the $C-1$ pseudo regret similarly as $G-1$ pseudo regret.

$$\mathcal{R}^{c1}_T = \sum_{t=1}^{T} \sum_{i \in \mathcal{I}_t} \left(\mu_{i,t}(\boldsymbol{p}^*_{C-1,n}) - \mu_{i,t}(\boldsymbol{W}_t) \right) \tag{4.4}$$

So when defining the seller's goal we consider maximizing cumulative revenue through learning either the $G-1$ optimal price vector or the $C-1$ optimal price vector given that buyers belong some clusters.

Remarks 4.2 (On the Interference in Dynamic Pricing Setting)**.**

Optimal Prices are Defined for the Population: Note that, the $G-1$ optimal price $\boldsymbol{p}^*_N$ and the $C-1$ optimal price vector $\boldsymbol{p}^*_{C-1,N}$ are defined for the whole population of buyers $\mathcal{I}$, not just for the sampled buyers $\mathcal{I}_t$ at time t. We defined the optimal prices this way because for the global optimal price, this is a fixed notion, i.e., there is one optimal price for the whole population, and also there is one optimal price per cluster for the whole population. Depending on the random sample, the price that would maximize revenue for the sampled buyers may vary. But the seller's goal is to learn the optimal prices for the whole population.

Modeling Interference in Pricing: We proposed a model of distrust, where if a buyer offers different price compared to her neighbor, then this influences the purchase probability negatively. So any kind of price discrimination or differential pricing would lead to some negative spillover effects. However, the nature of spillover effects depends on the product, market and different social scenarios, it could be that if the neighbor is offered a higher price and purchased the product, then this may increase the willingness to purchase for the buyer.

///

Algorithm 20: TS Price Learner (Without Interference)

1 **Input:** $\mathcal{W} = \{p_k, \ldots, p_K\}$, T and $\widetilde{\alpha}(k), \widetilde{\beta}(k)\ \forall k \in [K]$
2 **Initialize:** $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k), \widehat{\beta}_0(k) = \widetilde{\beta}(k), S_0(k) = 0, n_0(k) = 0, \forall k \in [K]$
3 **for** $t = 1, 2, \ldots, T$ **do**
4 sample $q'_t(k) \sim \text{Beta}(\widehat{\alpha}_{t-1}(k), \widehat{\beta}_{t-1}(k)), \forall k \in [K]$ // after the buyer arrives
5 select $k' = \text{argmax}_{k \in [K]}\ p_k \cdot q'_t(p_k)$
6 offers price $W_t = p_{k'} \in \mathcal{W}$,
7 observe purchase decision X_t
8 calculate revenue $Y_t = W_t \cdot X_t$
9 **update**
10 $S_t(k') = S_{t-1}(k') + X_t$ & $n_t(k') = n_{t-1}(k') + 1$
11 $\widehat{\alpha}_t(k') = \widetilde{\alpha}(k') + S_t(k')$ & $\widehat{\beta}_t(k') = \widetilde{\beta}(k') + n_t(k') - S_t(k')$
12 **Output:** $W_t, Y_t, \forall t \in [T]$

4.2 Price Learners Under Interference

We now discuss how we can apply different bandit algorithms in this dynamic pricing setting. First, let's ignore the interference altogether and apply the standard Thompson sampling algorithm for revenue maximization in dynamic pricing problem without interference, i.e., for Line 24. We outline the learner's procedure in Algorithm 20.

So the seller or the hawker starts with a set of prices $\mathcal{W} = \{p_1, \ldots, p_K\}$ of size K. The price set is actually our treatment set. In the parametric setup, the learner assumes the potential decision variable $X_t(p_k)$ that follows Bernoulli distribution i.e., $X_t(p_k) \sim \text{Bernoulli}(q(k))$, where $q(k)$ is the purchase probability at price p_k and then assumes a prior on the purchase probability $q(k)$ for each $k \in [K]$. The algorithm then follows very closely to the standard Thompson sampling with Bernoulli rewards that we have outlined in Algorithm 2, except there is a small difference in the arm selection. In the usual Bernoulli Thompson sampling, arm is selected based on the sampled mean reward from the posterior distribution. Here, the arm is selected based on maximizing the expected revenue. Here expected revenue for price p_k is $\mathbb{E}(Y_t(k)) = p_k \cdot q(k)$. This is in line no. 5 in the algorithm. The rest of the steps and then the updates are similar to the standard Bernoulli Thompson sampling.

Now let's apply the algorithms under interference that we have developed in Chapter 2 and Chapter 3. It is possible to apply all four type of learners, i.e., IID, $G-1$, $C-1$ and $C-M$ learners both in parametric, i.e., Thompson sampling and non-parametric algorithms, NPTS and SDA in this case. Since this is just some minor adjustments of the basic algorithms we have already outlined, we will only outline the parametric versions of the TS $G-1$ and TS $C-M$ learners here and give the TS IID and TS $C-1$ learners in Appendix A3 and also mention how one can apply the non-parametric learners in Chapter 3.

Algorithm 21: TS $G-1$ Price Learner

Input: $\{\mathcal{I}_t\}_{t=1}^{T}$, $\mathcal{W}=\{p_1,\ldots,p_K\}$, T and $\widetilde{\alpha}(k)$, $\widetilde{\beta}(k)$ $\forall k\in[K]$
Initialize: $\widehat{\alpha}_0(k)=\widetilde{\alpha}(k)$, $\widehat{\beta}_0(k)=\widetilde{\beta}(k)$, $S_0(k)=0$, $n_0(k)=0$, $\forall k\in[K]$
for $t=1,2,\ldots,T$ **do**
  a random sample of buyers $\mathcal{I}_t\subset\mathcal{I}$ arrives
  sample $q'_t(k)\sim \text{Beta}(\widehat{\alpha}_{t-1}(k),\widehat{\beta}_{t-1}(k))$, $\forall k\in[K]$
  select $k'=\text{argmax}_{k\in[K]}\, p_k\cdot q'_t(k)$
  offer price $W_{i,t}=p_{k'}\in\mathcal{W}$, $\forall i\in\mathcal{I}_t$
  observe purchase decision $X_{i,t}$, $\forall i\in\mathcal{I}_t$ // $X_{i,t}$ is a function of $\boldsymbol{W}_t$ due to interference
  calculate revenue $Y_{i,t}=W_{i,t}\cdot X_{i,t}$, $\forall i\in\mathcal{I}_t$
  update
    $S_t(k')=S_{t-1}(k')+X_t$ & $n_t(k')=n_{t-1}(k')+n$
    $\widehat{\alpha}_t(k')=\widetilde{\alpha}(k')+S_t(k')$ & $\widehat{\beta}_t(k')=\widetilde{\beta}(k')+n_t(k')-S_t(k')$
Output: $\boldsymbol{W}_t$, $\boldsymbol{Y}_t$, $\forall t\in[T]$

Algorithm 22: TS $C-M$ Price Learner

Input: $\mathcal{I}$, $\mathcal{W}=\{p_1,\ldots,p_K\}$, T, $\{\mathcal{C}_m\}_{m\in[M]}$, $\widetilde{\alpha}_m(k)$, $\widetilde{\beta}_m(k)$, $\forall m\in[M]$ and $\forall k\in[K]$
Initialize: $\widehat{\alpha}_{m,0}(k)=\widetilde{\alpha}_m(k)$, $\widehat{\beta}_{m,0}(k)=\widetilde{\beta}_m(k)$, $S_{m,0}(k)=0$, *and* $n_{m,0}(k)=0$, $\forall m\in[M]$ *and* $\forall k\in[K]$
for $t=1,2,\ldots,T$ **do**
  a random sample of buyers $\mathcal{I}_t\subset\mathcal{I}$ arrives
  for $m=1,2,\ldots,M$ **do**
    sample $q'_{m,t}(k)\sim \text{Beta}(\widehat{\alpha}_{m,t-1}(k),\widehat{\beta}_{m,t-1}(k))$, $\forall k\in[K]$
    select $k^{(m)}=\text{argmax}_{k\in[K]}\, p_k\cdot q'_{m,t}(k)$
    offer price $W_{i,t}=p_{k^{(m)}}$, $\forall i\in\mathcal{C}_m$
  observe purchase decision $X_{i,t}$, $\forall i\in\mathcal{I}_t$
  calculate revenue $Y_{i,t}=W_{i,t}\cdot X_{i,t}$, $\forall i\in\mathcal{I}_t$
  for $m=1,2,\ldots,M$ **do**
    update
      $S_{m,t}(k^{(m)})=S_{m,t-1}(k^{(m)})+\sum_{i\in\mathcal{C}_m}Y_{i,t}$ & $n_{m,t}(k^{(m)})=n_{m,t-1}(k^{(m)})+N_m$
      $\widehat{\alpha}_{m,t}(k^{(m)})=\widetilde{\alpha}_m(k^{(m)})+S_{m,t}(k^{(m)})$
      $\widehat{\beta}_{m,t}(k^{(m)})=\widetilde{\beta}_m(k^{(m)})+n_{m,t}(k^{(m)})-S_{m,t}(k^{(m)})$
Output: $\boldsymbol{W}_t$, $\boldsymbol{Y}_t$, $\forall t\in[T]$

Recall the idea of the $G-1$ learner, here the seller is looking for the best $G-1$ price vector, or simply, what is the optimal price that he can offer to all buyers such that $G-1$ regret we defined eq. (4.3) is minimized. The basic procedure is similar to the Bernoulli example of the $G-1$ learner in Example 2.17, however like Algorithm 20 for the arm selection step learner needs to select the price based on maximizing expected revenue, and then

following the $G-1$ style it assigns the same arm to all buyers in the sample at time t. We outline the TS $G-1$ Price learner in Algorithm 21.

Similarly, we can also modify the TS $C-M$ learner for this dynamic pricing setting with interference. Recall the idea of the $C-M$ learner, here the seller is looking for the optimal $C-1$ price vector, or simply, what is the location-wise best price that he can offer to buyers in different locations such that the $C-1$ regret we defined in eq. (4.4) is minimized. Again the basic procedure is similar to the Bernoulli example of the $C-M$ learner in Example 2.24 with some modifications for the arm selection step and the updates.

The nonparametric learners in Chapter 3 can also be similarly applied. In fact the modifications are easier, since the learner needs to sample the history of the rewards, which in this is the history of the revenues, and the selects are the arm based on relevant steps for SDA and NPTS modifications.

4.3 Numerical Experiments

In this section, we evaluate different parametric and nonparametric price learners through some numerical experiments. For the parametric cases, we run the Thompson sampling (TS) price learners that we have discussed in Section 4.2 and the for the non-parametric case, the prices learners are extensions of the subsampling duelling (SD) algorithms proposed in Chapter 3. In the following, we first explain the experimental setup, in particular how we constructed the population network of buyers and how sampling is done at each time t. Then, we compare different price learners based on their cumulative regrets over T days and discuss the results.

4.3.1 Population Network Construction and Sampling

For this experiment, we generated the population network of buyers distributed across a large spatial area, with small clusters inside. As we discussed before, this is motivated from the hawker example, where the hawker targets this population of buyers in a city and he goes to different locations, or in this case clusters, to sell his products. We keep the generated the clusters sufficiently separated with denser connections within but sparse connections in between. Then at each time t, a random sample of buyers are drawn from this population and the hawker or seller offers his price to learn the optimal price over a horizon of T days. Here are the details of the general construction procedure for the population spatial network. We assume, given that L is the side-length of a single cluster, the separating roads have a uniform width of $rf \times L$, where $rf \in (0,1)$ stands for road fraction (i.e., the fraction of the cluster length used for a road). Along the outer boundary, we also keep additional width of $\frac{rf}{2} \times L$ and the full area of the square is $[0,3L] \times [0,3L]$.

Now, for this experiment we set $L = 1$ and $rf = 0.3$, then the population area is $[0,3] \times$

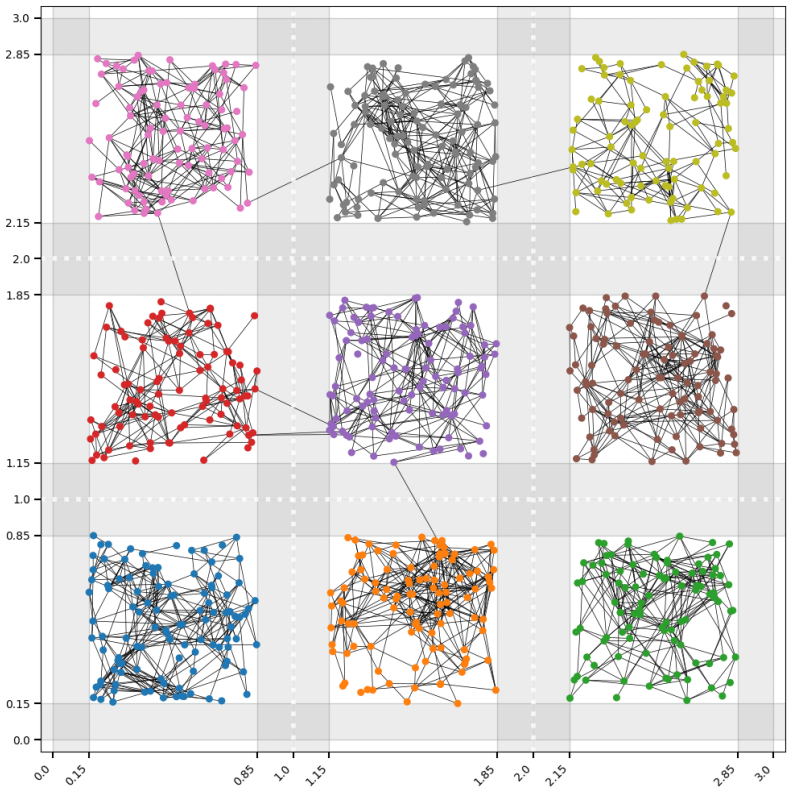


Figure 4.1: An example of a clustered network with $N = 900$ nodes and $M = 9$ clusters, generated using the specified road geometry and connectivity model.

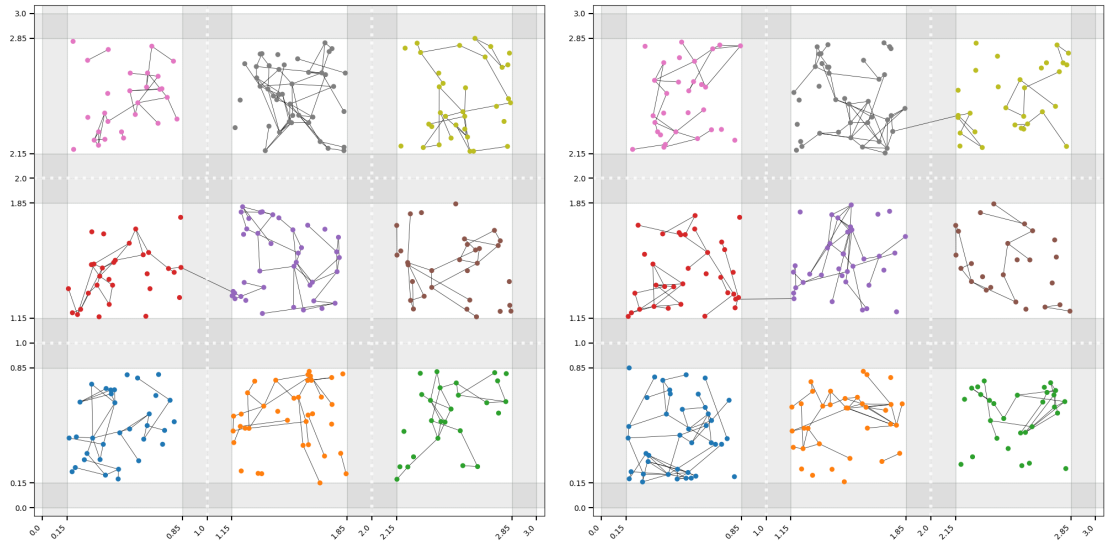
$[0, 3]$, with $3 \times 3 = 9$ clusters separated by roads of width 0.15 between clusters and also along the outer boundary. The full population area with clusters are shown in Figure 4.1.

Now, we generate a network on this space. To generate nodes, we randomly generated $N = 900$ points uniformly within this population region. Initially they are scattered across all clusters along with the roads. Then with some scaling we shifted the points inside the clusters, such that no points are located within the road regions. This separates the clusters sufficiently, and helps us to get the structure of connections that we want. Any two nodes, or points, i and j are then randomly connected with a probability p_{ij},

$$\mathrm{p}_{ij} = \mathrm{p}_{\min} + (\mathrm{p}_{\max} - \mathrm{p}_{\min}) \times e^{-\left(\rho_{ij}/\bar{\rho}\right)^2}, \tag{4.5}$$

where p_{ij} denotes the probability of an edge between node i and j (for $i \neq j$), $\mathrm{p}_{\min}$ and $\mathrm{p}_{\max}$ are the minimum and maximum probabilities for connection respectively, ρ_{ij} is the Euclidean distance between them and $\bar{\rho}$ is the scale parameter (or also called decay parameter) controls how fast the connectivity decays with distance. So, if $\rho_{ij} \to 0$, then $\mathrm{p}_{ij} \to \mathrm{p}_{\max}$, and if $\rho_{ij} \to \infty$, then $\mathrm{p}_{ij} \to \mathrm{p}_{\min}$, i.e., i.e., lower distance lead to higher probability and vice versa. The $\bar{\rho}$ controls the strength of this decay, smaller values means faster

decay hence we will get a sparser graph, while larger values means slower decay hence denser graph. In our experiments, we set $\bar{\rho} = 0.2$, $p_{min} = 0.0$, and $p_{max} = 0.3$. This gives the connections shown in Figure 4.1. The graphs generated in this manner are known as *soft random geometric graphs (soft RGGs)* in the literature (Penrose, 2016; Duchemin & de Castro, 2022) as opposed to the classic random geometric graphs (RGGs) where connections are deterministic and all nodes within a certain threshold or distance are connected and beyond they are not. We used the RGG construction in Chapter 2, following Leung (2023), here we used the soft RGG construction to have more flexibility in controlling the density of connections.



m	1	2	3	4	5	6	7	8	9
N_m	109	105	94	93	107	100	95	110	87
n_m	36	35	31	31	36	33	32	37	29

Figure 4.2: Two examples of sampled networks at two different time t. Population and Sample cluster sizes are N_m and n_m are given in the table.

Now, at each day or time t, the hawker meets a random sample of buyers $\mathcal{I}_t$ or size $n = 300$ from this population. Network sampling is a difficult problem, and one can do sampling in different ways with different objectives in mind[§]. Since our goal is to see how the price learners perform, we avoid the complex sampling issues, and do a simple stratified sampling here. In particular, we fixed the number of units in each cluster first, and then randomly sample from each cluster the specified number of buyers.Let N_m be the number of buyers in population cluster $\mathcal{C}_m$, for $m = 1, 2, \ldots, 9$, such that $\sum_{m=1}^{9} N_m = N$ and n_m be the number of buyers sampled from cluster $\mathcal{C}_m$ at each time t, such that $\sum_{m=1}^{9} n_m = n$.

§ For example, we could keep the same degree distribution as population, or clustering coefficient, or average path length, etc. We found an introductory overview, covering different aspects of network sampling in Izenman (2023, Chapter 3). Interested readers are referred to this chapter for more details.

Then we have the N_m and n_m values as shown in Figure 4.2. Also, the figure shows two sampled networks at two different time points. Note that, since the sampling is random, the sampled networks are different at different time t.

4.3.2 Reward Generation and Parameter Setup

The rewards or the revenues are generated based on the model we discussed in Section 4.1.2. Here we summarize the key elements again for completeness. The treatment set or price set in this case is $\mathcal{W} = \{10, 12, 15\}$[¶]. The revenue from a buyer $i \in \mathcal{I}_t$ at time t, is denoted with $Y_{i,t}$, this depends on the price offered to her, denoted with $W_{i,t}$, times the decision variable, $X_{i,t} \in \{0, 1\}$. The decision variable models whether the buyer purchases or not and this is actually the key element of interference that we discussed before. In particular, the reward generation model is as follows:

$$\begin{aligned} Y_{i,t} &= W_{i,t} \cdot X_{i,t}, \\ X_{i,t} &\sim \text{Bernoulli}\left(q_{i,t}\left(\boldsymbol{W}_t\right)\right) \\ q_{i,t}\left(\boldsymbol{W}_t\right) &= \sigma\left(\eta_{i,t}\left(\boldsymbol{W}_t\right)\right), \\ \eta_{i,t}\left(\boldsymbol{W}_t\right) &= \alpha_i + \mu_{m(i)}(W_{i,t}) - \gamma V_{i,t}\left(\boldsymbol{W}_t\right). \\ \mu_{m(i)}(W_{i,t}) &= \beta_{0,m(i)} - \beta_{1,m(i)} \log W_{i,t} \\ V_{i,t}\left(\boldsymbol{W}_t\right) &= \frac{1}{d_{i,t}} \sum_{j \in \mathcal{I}_t : A_{ij,t}=1} \left(W_{j,t} - W_{i,t}\right)^2 \end{aligned} \tag{4.6}$$

where the purchase probability $q_{i,t}(\cdot)$ depends on individual-specific preference α_i, cluster-specific function $\mu_{m(i)}(\cdot)$ with parameters $\beta_{0,m(i)}$ and $\beta_{1,m(i)}$, and the interference function $V_{i,t}(\cdot)$ with parameter γ. Recall, the function $V_{i,t}(\cdot)$ depends on the entire treatment vector at time t, $\boldsymbol{W}_t = (W_{1,t}, \ldots, W_{n,t})' \in \mathcal{W}^n$ and captures the negative spillover effect from neighbors for differential pricing. The entries $A_{ij,t}$ are the elements of the adjacency matrix $\mathbf{A}_t$ for the sampled network at t, where $A_{ij,t} = 1$ if i and j are connected in the batch $\mathcal{I}_t$, and 0 otherwise and $d_{i,t} = \sum_j A_{ij,t}$ is the degree of buyer i for batch $\mathcal{I}_t$. This means the reward or revenue from buyer i at time t depends on the price offered to batch $\mathcal{I}_t$, and not on the entire population $\mathcal{I}$[||].

The parameter configurations for the experiments are summarized in Table 4.1. We have three sets of parameters with three clusters each. For each cluster set, the individual willingness to purchase (WP), denoted with α_i are generated from Gaussian distributions with different variances. The baseline willingness to purchase, denoted with $\beta_{0,m}$ and price

¶ One could also do a discretization like Kleinberg and Leighton (2003), where the prices are discrete values between 0 and 1. In this setting this is just a generalization to a continuous price set, which essentially means more arms here. Since we don't analysis based different demand curves we keep the setting simple.

|| In the random network case, this assumption makes several things tractable, otherwise the interference term would depend on the entire population network, which would be difficult to handle.

sensitivity $\beta_{1,m}$ are also different for different clusters, but we set the interference effect parameter $\gamma = 0.1$ same for all three sets, or for all clusters.

Cluster set m	**Individual WP** $\alpha_i \sim \mathcal{N}(0, \sigma_m^2)$ σ_m^2	**Baseline WP** $\beta_{0,m}$	**Price Sensitivity** $\beta_{1,m}$	**Interference Effect** γ
$\{1,2,3\}$	4	25	8	0.1
$\{4,5,6\}$	0.25	5	2	
$\{7,8,9\}$	0.01	12	5	

Table 4.1: Parameter values for the dynamic pricing bandit with network interference experiments.

For the first set, clusters $1, 2, 3$ have high baseline WP and also high price sensitivity. In the context of our hawker example, these are essentially some busy commercial locations in the city. So people coming to these locations and have high willingness to purchase in general, but they are also very price sensitive, i.e., if price is suddenly high, they just leave, possibly since they don't have time for a second thought. These areas also have high individual variance, i.e., high variance for α_i. The idea is, since these are some busy locations, so different types of people coming with different individual willingness to purchase. The second group, clusters $4, 5, 6$ are the opposite of the first group in nature. These essentially represent quiet residential zones, so when hawker goes there, people have low baseline WP, but they are also not very price sensitive. So when the hawker charges a bit higher price, maybe people would still buy some goods but maybe not many people would buy. These areas have moderate individual variance for α_i. The last group, clusters $7, 8, 9$ are in between of last two groups. These could be some moderate busy locations like schools or offices.

For the three price arms, i.e., $p_1 = 10$, $p_2 = 12$, and $p_3 = 15$, we can compute the global mean revenues under $G - 1$ arm vector assignments, also we can compute the cluster-wise mean revenues under the $G - 1$ assignments as well. Since there are edges here outside the clusters, we computed the cluster-wise mean revenues for the complete $G - 1$ vectors since in this setting in general these values won't be same as the mean revenues under the cluster-wise constant price vectors. The cluster-wise mean revenues under $G - 1$ arm vector assignments are denoted with $\overline{\mu}_m(p_k \cdot \mathbf{1}_N)$ for each cluster m and price arm p_k. The values are given in Table 4.2. We can see that the cluster revenue values vary significantly compared to the $G - 1$ revenue across clusters for different prices.

Cluster	$\overline{\mu}_m(10 \cdot \mathbf{1}_N)$	$\overline{\mu}_m(12 \cdot \mathbf{1}_N)$	$\overline{\mu}_m(15 \cdot \mathbf{1}_N)$
1	9.95	11.78	13.65
2	9.91	11.62	13.30
3	9.92	11.64	13.31
4	6.16	6.38	6.41
5	5.73	5.83	5.72
6	6.03	6.22	6.21
7	6.17	4.73	2.64
8	6.18	4.74	2.65
9	6.21	4.77	2.67

	$\overline{\mu}(10 \cdot \mathbf{1}_N)$	$\overline{\mu}(12 \cdot \mathbf{1}_N)$	$\overline{\mu}(15 \cdot \mathbf{1}_N)$	**Optimal** $G-1$ price
$\boldsymbol{G-1}$	7.39	7.58	7.49	12

Table 4.2: Global and Cluster-wise mean revenues for $G-1$ arm vector.

According to the mean revenues, the optimal $G-1$ price vector is $\mathbf{12}_N$ since it has the highest batch mean revenue of 7.58. The optimal $C-1$ price vector consists of 15 for all units in the clusters $1, 2, 3$, then 12 for all units in the clusters $4, 5, 6$, and 10 for all units in the clusters $7, 8, 9$, achieving a batch mean revenue of 8.65, which is higher than the optimal $G-1$ revenue of 7.58. Note that, for clusters 4, looking at the $G-1$ vector, it shows price 15 is optimal which gives revenue of 6.41, however since in this case cluster 4 has edges outside the cluster, the optimal $C-1$ arm vector will be different from looking at $G-1$ arm vector for the cluster.

4.3.3 Comparison of Price Learners

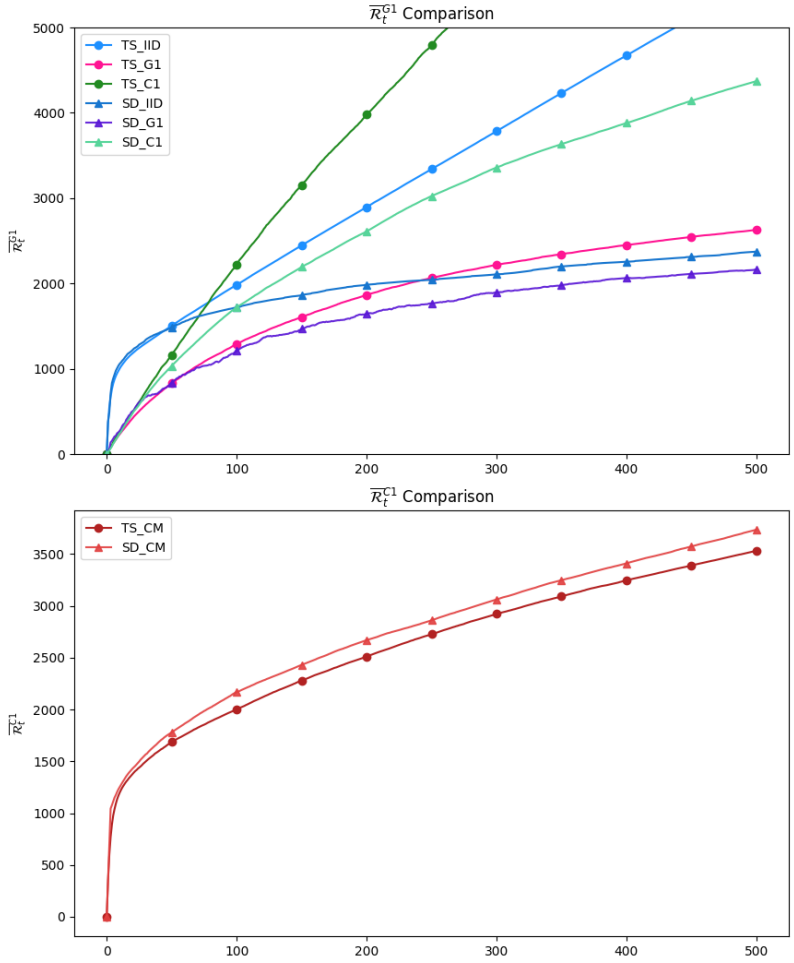


Figure 4.3: Regret comparisons in the dynamic pricing with interference setting. **Top:** $G-1$ regret comparison of different TS and SD (IID, $G-1$, $C-1$) learners. **Bottom:** $C-1$ regret comparison of different TS $C-M$ and SD $C-M$ learners.

In this section, we compare the performance of different price learners through their cumulative regrets over a horizon of $T = 500$ days, with 1000 Monte Carlo simulations. We consider all TS price learners and all SD price learners. The cumulative regret plots for the $G-1$ and $C-1$ regret are shown in Figure 4.3. The worst performance are from TS $C-1$, then TS IID and SD $C1$. Unlike the simulations from Chapter 2 and Chapter 3, here we see that the $C-1$ learners perform poorly. There are actually good reasons for this, and that is in this case, cluster means vary significantly from the $G-1$ means, which breaks the first assumption in Assumption 2.20. Although not many, but we have some edges outside the clusters, which also creates learning more challenging for the $C-1$ learner. The best is SD $G-1$ learner, then SD IID and TS $G-1$. The $G-1$ learners are performing good, and this

is expected but the SD IID learner also performs good. Although this is a bit strange, but as we explained in Remarks 3.4, our SD IID learner is not fully independent because of the common winner set approach. So we think this is the reason why this IID learner is also performing relatively good here. This needs more investigation with difficult dependence structure and more arms.

4.4 Conclusion

In this chapter, we have discussed the dynamic pricing problem under network interference. The problem is motivated from a real life example of a hawker selling his products in a city, where buyers' purchase decisions are influenced not only by the price offered to them but also by the prices offered to their neighbors in the network. We extended the multi-armed bandit setup in the dynamic pricing setting following the work of Kleinberg and Leighton (2003). We first specified how interference appears in the purchase decision. The modeling is motivated from the linear-in-means model, but actually a nonlinear version adapted in the dynamic pricing setting. We then outlined how Thompson sampling price learners with and without interference can be applied in this setting. Finally, we evaluated the performance of different price learners through numerical experiments. This chapter opens up several interesting future research directions including theoretical analysis of the proposed learners, exploring more complex interference structures, and considering more realistic demand models.

Appendix 1.

SOME CONJUGATE PRIOR RESULTS

Here we give some details about Bayesian updates in Thompson Sampling algorithms. Essentially the idea is to use the conjugate prior results, which roughly means, the prior and posterior distributions are from the same family. We will consider two cases, Bernoulli distributed rewards with Beta prior, and Gaussian distributed rewards with Gaussian prior.

A1.1 Bernoulli Rewards with Beta Prior

Suppose we have n plays of an arbitrary arm $k \in \mathcal{W}$, with $Y_t(k) \sim \text{Bernoulli}\,(p(k))$ i.e., the reward of arm k is Bernoulli distributed with parameter $p(k)$. Since the derivation is same for all arm $k \in \mathcal{W}$, for this section, we will drop the arm index k and simply write $Y_t \sim \text{Bernoulli}(p)$. Assume we have n independent plays of the arm and we have observed the rewards $Y_1, Y_2, \ldots, Y_n$, so all of these random variables are now identically and independently distributed (i.i.d) with Bernoulli(p). Next we assume a Beta prior on p, i.e.,

$$
\begin{aligned}
p &\sim \text{Beta}(\alpha, \beta), \\
\text{with density} \quad f(p) &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}, \text{ where } \quad \Gamma(n) = (n-1)!
\end{aligned}
$$

This prior distribution can be quite different depending on different values on α and β, which are the parameters of the prior, often called *hyper-parameters*. For example, we may have high density in some parts of $[0, 1]$, or maybe we may have uniform prior across $[0, 1]$. This depends on our initial belief about the parameter p. The uniform case is often called *uninformative prior*. For the hyper-parameters, if $\alpha >> \beta$ (i.e., α is very large compared to β), then the density is concentrated near 1, which essentially means we believe the success probability is high, on the other hand if $\beta >> \alpha$, then the density is concentrated near 0.

Let $S_n = Y_1 + Y_2 + \ldots, Y_n$, be the cumulative reward after n plays, or the number of success after n trials. Then we know, $S_n \sim \text{Binomial}(n, p)$, and we can write the density of S_n given p as $f(S_n = s \mid p) = \binom{n}{s} p^s (1-p)^{n-s}$. The idea of the Bayesian learning is we will swap this conditional density with the conditional density of p given the data observed after n plays, i.e., $f(p \mid S_n = s)$, that is the distribution of p given the cumulative rewards after n plays. We can use the Bayes' rule to get this,

$$
\begin{aligned}
f(p \mid S_n = s) &= \frac{f(S_n = s \mid p) \times f(p)}{f(S_n = s)} \\
&\propto f(s \mid p) \times f(p)
\end{aligned}
$$

Now we plug $f(p)$ assuming $p \sim \text{Beta}(\alpha_0, \beta_0)$ and start simplifying the right hand side,

$$\begin{aligned} f(s \mid p) \times f(p) &= \binom{n}{s} p^s (1-p)^{n-s} \times \frac{\Gamma(\alpha_0 + \beta_0)}{\Gamma(\alpha_0)\Gamma(\beta_0)} p^{\alpha_0 - 1} (1-p)^{\beta_0 - 1} \\ &\propto p^s (1-p)^{n-s} \times p^{\alpha_0 - 1} (1-p)^{\beta_0 - 1} \\ &= p^{\alpha_0 + s - 1} (1-p)^{\beta_0 + n - s - 1} \end{aligned}$$

where we assumed we started from the prior parameter α_0 and β_0. Now this means the updated density of p, or the posterior, after observing the s success after n trials is given by,

$$f(p \mid S_n = s) \propto p^{\alpha_0 + s - 1} (1-p)^{\beta_0 + n - s - 1}$$

Notice, except the constant, this resembles the Beta distribution with parameter $\alpha_0 + s$ and $\beta_0 + n - s$, i.e., the posterior distribution is also a Beta distribution, in particular $\text{Beta}(\alpha_0 + s, \beta_0 + n - s)$.

A1.2 Gaussian Rewards with Gaussian Priors

Now suppose the rewards follow Gaussian distributions with unknown mean but known variance, again we will skip arm index k, so assume $Y_t \sim \mathcal{N}(\mu, \sigma^2)$. For an i.i.d. random sample $Y_1, Y_2, \ldots, Y_n$ from n independent plays, we can write the joint density as:

$$\begin{aligned} f(y_1, \ldots, y_n \mid \mu, \sigma^2) &= \prod_{t=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{1}{2} \left(\frac{y_t - \mu}{\sigma} \right)^2 \right) \\ &= \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \cdot \exp\left(-\frac{1}{2\sigma^2} \sum_{t=1}^{n} (y_t - \mu)^2 \right) \\ &= \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \cdot \exp\left(-\frac{1}{2\sigma^2} \left(n s^2 + n(\bar{y} - \mu)^2 \right) \right) \quad \text{[using the decomposition below]} \\ &= \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \cdot \exp\left(-\frac{1}{2\sigma^2} n s^2 \right) \cdot \exp\left(-\frac{1}{2\sigma^2} n (\bar{y} - \mu)^2 \right) \\ &\propto \exp\left(-\frac{n}{2\sigma^2} (\bar{y} - \mu)^2 \right) \quad \text{[dropping constants not involving } \mu\text{]} \\ &\propto f(\bar{y} \mid \mu, \sigma^2 / n) \end{aligned}$$

where it shows that the likelihood of the data is proportional to the distribution of the sample mean $\bar{y}$. Also the decomposition that we used above is similar to Bias-Variance decomposition, i.e., MSE $=$ Bias^2 $+$ Variance. We get,

$$\begin{aligned}\sum_{t=1}^{n}(y_t-\mu)^2 &= \sum_{t=1}^{n}\left[(y_t-\bar{y})-(\mu-\bar{y})\right]^2 \\ &= \sum_{t=1}^{n}(y_t-\bar{y})^2+\sum_{t=1}^{n}(\bar{y}-\mu)^2-2\sum_{t=1}^{n}(y_t-\bar{y})(\mu-\bar{y}) \quad \text{[expanding the square]} \\ &= ns^2+n(\bar{y}-\mu)^2 \quad \text{[since } \sum_{t=1}^{n}(y_t-\bar{y})=0\text{]}\end{aligned}$$

where,

$$s^2=\frac{1}{n}\sum_{t=1}^{n}(y_t-\bar{y})^2 \quad \text{and} \quad \bar{y}=\frac{1}{n}\sum_{t=1}^{n}y_t$$

We assume a Gaussian conjugate prior on μ, meaning

$$\mu \sim \mathcal{N}(\mu_0,\sigma_0^2)$$

This gives the density of μ,

$$f(\mu)\propto \exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right)$$

where μ_0 and σ_0^2 are the prior mean and variance. Now we apply Bayes' rule to obtain the posterior distribution:

$$\begin{aligned}f(\mu\mid y_1,\ldots,y_n) &\propto f(y_1,\ldots,y_n\mid\mu)\times f(\mu) \\ &= \exp\left(-\frac{n}{2\sigma^2}(\bar{y}-\mu)^2\right)\times\exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right) \\ &= \exp\left(-\frac{n}{2\sigma^2}(\bar{y}^2-2\bar{y}\mu+\mu^2)\right)\times\exp\left(-\frac{1}{2\sigma_0^2}(\mu^2-2\mu\mu_0+\mu_0^2)\right) \\ &= \exp\left(-\frac{n}{2\sigma^2}\mu^2+\frac{n\bar{y}\mu}{\sigma^2}-\frac{n}{2\sigma^2}\bar{y}^2\right)\times\exp\left(-\frac{1}{2\sigma_0^2}\mu^2+\frac{\mu\mu_0}{\sigma_0^2}-\frac{1}{2\sigma_0^2}\mu_0^2\right) \\ &= \exp\left(-\frac{\mu^2}{2}\left(\frac{n}{\sigma^2}+\frac{1}{\sigma_0^2}\right)+\mu\left(\frac{n\bar{y}}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right)-\left(\frac{n\bar{y}^2}{2\sigma^2}+\frac{\mu_0^2}{2\sigma_0^2}\right)\right)\end{aligned}$$

Now we need to match this to the form of a Gaussian. First we write the posterior density as

$$f(\mu\mid y_1,\ldots,y_n)\propto\exp\left(-\frac{1}{2\sigma_n^2}(\mu-\mu_n)^2\right)=\exp\left(-\frac{1}{2\sigma_n^2}(\mu^2-2\mu\mu_n+\mu_n^2)\right)$$

where μ_n and σ_n^2 are the posterior mean and variance that we need to find. To do this, we will match the coefficients of μ^2 and μ from the two expressions for the posterior density. First, we match the coefficients of μ^2. This will be useful to find σ_n^2:

$$\begin{aligned}
\frac{-\mu^2}{2\sigma_n^2} &= \frac{-\mu^2}{2}\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}\right) \\
\implies \frac{1}{\sigma_n^2} &= \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \\
\implies \sigma_n^2 &= \frac{1}{\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}} = \frac{\sigma^2\sigma_0^2}{n\sigma_0^2 + \sigma^2}
\end{aligned}$$

Next, we match the coefficients of μ to find μ_n:

$$\begin{aligned}
\frac{2\mu\mu_n}{2\sigma_n^2} &= \mu\left(\frac{n\bar{y}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}\right) \\
\implies \frac{\mu_n}{\sigma_n^2} &= \frac{n\bar{y}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2} \\
\implies \mu_n &= \left(\frac{n\bar{y}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}\right)\sigma_n^2
\end{aligned}$$

Therefore, the posterior distribution is:

$$\mu \mid y_1, \ldots, y_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$$

where

$$\begin{aligned}
\mu_n &= \left(\frac{n\bar{y}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}\right)\sigma_n^2 \\
\sigma_n^2 &= \left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}\right)^{-1}
\end{aligned}$$

These are essentially the parameter updates we use in the Thompson sampling algorithms with Gaussian rewards and Gaussian priors.

Appendix 2.

GAUSSIAN CONCENTRATION WITH CHERNOFF TRICK

In this part, we give some details of the concentration bounds / inequalities that we used in Chapter 1. The key idea is to use what's known as *Chernoff's trick*, essentially the idea is - *exponentiating a random variable with a parameter $\lambda > 0$, applying Markov inequality and finally optimizing over λ*. This idea is very powerful and gives exponential bounds for different distributions including Gaussian, Bernoulli, Poisson and more general classes like sub-Gaussian, sub-Exponential random variables.

A2.1 For Single Gaussian Random Variables

Let Y be a mean zero random variable (any distribution). For any $\epsilon > 0$ and any $\lambda > 0$, applying *Markov inequality* we get

$$\mathbb{P}(Y \geq \epsilon) = \mathbb{P}(e^{\lambda Y} \geq e^{\lambda \epsilon}) \leq \frac{\mathbb{E}[e^{\lambda Y}]}{e^{\lambda \epsilon}}$$

Interestingly, this holds for any $\lambda > 0$, so we now optimize λ to get the tightest bound possible. This means we can write:

$$\mathbb{P}(Y \geq \epsilon) \leq \inf_{\lambda > 0} \frac{\mathbb{E}[e^{\lambda Y}]}{e^{\lambda \epsilon}}$$

Now, specific example can be given for different distributions. But before this note that the term $\mathbb{E}[e^{\lambda Y}]$ is known as the moment-generating function (MGF) of Y evaluated at λ. We write

$$M_Y(\lambda) = \mathbb{E}[\exp(\lambda Y)]$$

Now we will give the details for the *mean zero Gaussian random variable* with variance σ^2. In this case, the MGF is given by:

$$M_Y(\lambda) = \exp\left(\frac{\lambda^2 \sigma^2}{2}\right)$$

Now with the Gaussian MGF, we can write the Chernoff bound as:

$$\mathbb{P}(Y \geq \epsilon) \leq \inf_{\lambda > 0} \frac{\exp\left(\frac{\lambda^2 \sigma^2}{2}\right)}{\exp(\lambda \epsilon)} = \inf_{\lambda > 0} \exp\left(\frac{\lambda^2 \sigma^2}{2} - \lambda \epsilon\right)$$

Now we have a concrete expression and we can minimize the right hand side with respect to λ, this gives,

$$\lambda\sigma^2 - \epsilon = 0 \implies \lambda = \frac{\epsilon}{\sigma^2}$$

Now substituting this value of λ back into the bound, we get:

$$\mathbb{P}(Y \geq \epsilon) \leq \exp\left(\frac{\epsilon^2}{2\sigma^2} - \frac{\epsilon^2}{\sigma^2}\right) = \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right)$$

So finally for mean zero Gaussian random variable with variance σ^2, we have the Chernoff bound as:

$$\mathbb{P}(Y \geq \epsilon) \leq \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right) \tag{A2.1}$$

If the variable is not mean zero, i.e., $Y \sim \mathcal{N}(\mu, \sigma^2)$, then we can simply consider $Y - \mu$, and apply the same bound. Similarly, we can get the bound for the left tail as well and this gives the following two-sided bound,

$$\mathbb{P}(|Y - \mu| \geq \epsilon) \leq 2\exp\left(-\frac{\epsilon^2}{2\sigma^2}\right) \tag{A2.2}$$

These bounds are often also used for sub-Gaussian variables. If we assume Y is sub-Gaussian with parameter σ^2, then by definition, for all $\lambda \in \mathbb{R}$, the moment generating function $M_X(\lambda) \leq \exp\left(\frac{\lambda^2\sigma^2}{2}\right)$. Then the same bound holds. Note that the sub-Gaussian variables assume th...

A2.2 For a Sequence of Independent Gaussian Random Variables

If we have a collection of independent Gaussian random variables $Y_1, Y_2, \ldots, Y_n$ with mean and variance $(\mu_1, \sigma_1^2), (\mu_2, \sigma_2^2), \ldots, (\mu_n, \sigma_n^2)$ respectively, then the average $\overline{Y}_n = \frac{1}{n}\sum_{i=1}^n Y_i$ is also a Gaussian random variable. In particular,

$$\overline{Y}_n \sim \mathcal{N}\left(\overline{\mu}, \sigma_{\overline{Y}}^2\right) \tag{A2.3}$$

where $\overline{\mu} = \frac{1}{n}\sum_{i=1}^n \mu_i$ and $\sigma_{\overline{Y}}^2 := \sum_{i=1}^n \sigma_i^2/n^2$. Now applying the bound that we derived above for Gaussian random variables, we have the following bound for the average:

$$\mathbb{P}\left(|\overline{Y}_n - \overline{\mu}| \geq \epsilon\right) \leq 2\exp\left(-\frac{\epsilon^2}{2\sigma_{\overline{Y}}^2}\right) \tag{A2.4}$$

Because average of Gaussian is a Gaussian, we can directly get this bound from the Gaus-

sian concentration that we had in eq. (A2.2)*. In the special case where we have identical distribution assumption as well, i.e., for all i, we have $\mu_i = \mu$ and $\sigma_i^2 = \sigma^2$, then $\sum_{i=1}^n \sigma_i^2 = n\sigma^2$, and $\sigma_{\overline{Y}}^2 = \sigma^2/n$. The tail bound becomes:

$$\mathbb{P}\left(|\overline{Y}_n - \mu| \geq \epsilon\right) \leq 2\exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right) \tag{A2.5}$$

Now this bound tells, given ϵ what is the probability that the average deviates from the mean by at least ϵ. We can ask the *reverse question*, that is, for a given probability level δ, what is the value of ϵ such that the bound holds. This is a simple algebra, we set the right-hand side to δ and solve for ϵ:

$$\begin{aligned}
&\exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right) = \delta \\
&\Rightarrow \exp\left(\frac{n\epsilon^2}{2\sigma^2}\right) = \frac{1}{\delta} \\
&\Rightarrow \frac{n\epsilon^2}{2\sigma^2} = \log\left(\frac{1}{\delta}\right) \\
&\Rightarrow \epsilon^2 = \frac{2\sigma^2\log(1/\delta)}{n} \\
&\Rightarrow \epsilon = \sigma\sqrt{\frac{2\log(1/\delta)}{n}}
\end{aligned}$$

Plugging this value of ϵ back into the tail bound, we get the final result,

$$\mathbb{P}\left(|\overline{Y}_n - \mu| \geq \sigma\sqrt{\frac{2\log(1/\delta)}{n}}\right) \leq 2\delta \quad \text{and} \quad \mathbb{P}\left(|\overline{Y}_n - \mu| \leq \sigma\sqrt{\frac{2\log(1/\delta)}{n}}\right) \geq 1 - 2\delta \tag{A2.6}$$

A2.3 Batched Setup for Gaussian Rewards

Here we give the details for the batched concentration result which we used in Lemma 3. Suppose we have a fixed batch of size N. We observe a sequence of batches, and we are interested in the concentration of the average of the batch sums over time t. Let $\boldsymbol{Y}_t = (Y_{1,t}, Y_{2,t}, \ldots, Y_{N,t})'$ be the vector of rewards of the units in this batch at time t. In the potential reward framework, for a fixed arm vector $\boldsymbol{k} \in \mathcal{W}^N$, we write the reward vector as $\boldsymbol{Y}_t(\boldsymbol{k}) = (Y_{1,t}(\boldsymbol{k}), Y_{2,t}(\boldsymbol{k}), \ldots, Y_{N,t}(\boldsymbol{k}))'$. Here $Y_{i,t}(\boldsymbol{k})$ is the reward of unit i in the batch at time t when arm vector $\boldsymbol{k}$ is pulled by the learner (see Section 2.2 for more details). For the Gaussian case, for any $\boldsymbol{k} \in \mathcal{W}^N$, in principle we have

* For the sub-Gaussian case, we need to do some work to show that the sum and average are also sub-Gaussian random variables, and then we can apply the same bound.

$$\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{k}), \boldsymbol{\Omega}(\boldsymbol{k}))$$

where $\boldsymbol{\mu}(\boldsymbol{k}) = (\mu_1(\boldsymbol{k}), \mu_2(\boldsymbol{k}), \ldots, \mu_N(\boldsymbol{k}))'$ is the mean vector and $\boldsymbol{\Omega}(\boldsymbol{k})$ is the variance-covariance matrix of the batch rewards. Let the batch sum at time t, for pulling arm vector $\boldsymbol{k}$ is defined as,

$$\boldsymbol{Y}_{\Sigma,t}(\boldsymbol{k}) := \sum_{i=1}^{N} Y_{i,t}(\boldsymbol{k})$$

This batch sum is also a Gaussian random variable (from simple Gaussian properties), in particular we have:

$$\boldsymbol{Y}_{\Sigma,t}(\boldsymbol{k}) \sim \mathcal{N}\left(\mu_\Sigma(\boldsymbol{k}), \sigma_\Sigma^2(\boldsymbol{k})\right)$$

where

$$\begin{aligned}
\mu_\Sigma(\boldsymbol{k}) &= \sum_{i=1}^{N} \mu_i(\boldsymbol{k}) \\
\sigma_\Sigma^2(\boldsymbol{k}) &= \mathbb{V}(\boldsymbol{Y}_\Sigma(\boldsymbol{k})) \\
&= \sum_{i=1}^{N} \sigma_i^2(\boldsymbol{k}) + 2 \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \sigma_i(\boldsymbol{k}) \sigma_j(\boldsymbol{k}) \rho_{ij}(\boldsymbol{k})
\end{aligned}$$

where $\sigma_i^2(\boldsymbol{k})$ is the variance of reward of unit i and $\rho_{ij}(\boldsymbol{k})$ is the correlation between rewards of unit i and unit j in the batch when arm vector $\boldsymbol{k}$ is pulled. We apply the concentration for a centered Gaussian random variable as derived above. First we need the MGF of the centered sum $\boldsymbol{Y}_{\Sigma,t}(\boldsymbol{k}) - \mu_\Sigma(\boldsymbol{k})$. In this we can write this following sum of Gaussian is also Gaussian, and the MGF is: $\exp\left(\frac{\lambda^2 \sigma_\Sigma^2}{2}\right)$.

Now suppose we have observed this batch of units over different times, and we observed in total $N_t(\boldsymbol{k})$ batches when the learner pulled the vector $\boldsymbol{k}$. This means we have observed $\boldsymbol{Y}_{\Sigma,1}(\boldsymbol{k}), \boldsymbol{Y}_{\Sigma,2}(\boldsymbol{k}), \ldots, \boldsymbol{Y}_{\Sigma,N_t(\boldsymbol{k})}(\boldsymbol{k})$ batch sums. Then the empirical average of these batch sums is given by:

$$\overline{\boldsymbol{Y}}_{\Sigma,t}(\boldsymbol{k}) = \frac{1}{N_t(\boldsymbol{k})} \sum_{s=1}^{N_t(\boldsymbol{k})} \boldsymbol{Y}_{\Sigma,s}(\boldsymbol{k})$$

Now we apply the inequality eq. (A2.5), we have the following concentration bound for the empirical average of the batch sums:

$$\mathbb{P}\left(|\overline{\boldsymbol{Y}}_{\Sigma,t}(\boldsymbol{k}) - \mu_{\Sigma,t}(\boldsymbol{k})| \geq \epsilon\right) \leq 2\exp\left(-\frac{N_t(\boldsymbol{k})\epsilon^2}{2\sigma_\Sigma^2(\boldsymbol{k})}\right)$$

Following eq. (A2.6), we can also write the bound as

$$\mathbb{P}\left(|\overline{\boldsymbol{Y}}_{\Sigma,t}(\boldsymbol{k}) - \mu_{\Sigma,t}(\boldsymbol{k})| > \sqrt{\frac{2\sigma_\Sigma^2(\boldsymbol{k})\log(1/\delta)}{N_t(\boldsymbol{k})}}\right) \leq 2\delta \tag{A2.7}$$

or,

$$\mathbb{P}\left(|\overline{\boldsymbol{Y}}_{\Sigma,t}(\boldsymbol{k}) - \mu_{\Sigma,t}(\boldsymbol{k})| > \cdot\sigma_\Sigma(\boldsymbol{k}) \cdot \sqrt{\frac{2\log(1/\delta)}{N_t(\boldsymbol{k})}}\right) \leq 2\delta \tag{A2.8}$$

Finally, if we assume only mean level interference exists (see (i) of Remarks 2.4), i.e., there is no variance when the arm vector $\boldsymbol{k}$ is pulled at different times, then the variance of the sum does not depend on $\boldsymbol{k}$, it will be a fixed number which we can write with σ_Σ^2.

Appendix 3.

THOMPSON SAMPLING PRICE LEARNERS

Algorithm 23: TS *IID* Price Learner

Input: $\mathcal{I}$, $\mathcal{W} = \{p_1, \ldots, p_K\}$, T and $\widetilde{\alpha}(k)$, $\widetilde{\beta}(k)$, $\forall k \in [K]$
Initialize: $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k)$, $\widehat{\beta}_0(k) = \widetilde{\beta}(k)$, $S_0(k) = 0$, $n_0(k) = 0$, $\forall k \in [K]$
for $t = 1, 2, \ldots, T$ **do**
  a random sample of buyers $\mathcal{I}_t \subset \mathcal{I}$ arrives
  for $i \in \mathcal{I}_t$ **do**
    sample $q'_t(k) \sim \text{Beta}(\widehat{\alpha}_{t-1}(k), \widehat{\beta}_{t-1}(k))$, $\forall k \in [K]$
    select price index $k' = \text{argmax}_{k \in [K]}\, p_k \cdot q'_t(k)$
    offer price $W_{i,t} = p_{k'} \in \mathcal{W}$
  observe purchase decision $X_{i,t}$, $\forall i \in \mathcal{I}_t$ // $X_{i,t}$ is a function of $\boldsymbol{W}_t$ due to interference
  calculate revenue $Y_{i,t} = W_{i,t} \cdot X_{i,t}$, $\forall i \in \mathcal{I}_t$
  update
    $S_t(k) = S_{t-1}(k) + X_t$ and $n_t(k) = n_{t-1}(k) + 1$, $\forall k \in [K]$
    $\widehat{\alpha}_t(k) = \widetilde{\alpha}(k) + S_t(k)$ & $\widehat{\beta}_t(k) = \widetilde{\beta}(k) + n_t(k) - S_t(k)$, $\forall k \in [K]$
Output: $\boldsymbol{W}_t$, $\boldsymbol{Y}_t$, $\forall t \in [T]$

Algorithm 24: TS $C-1$ Price Learner

Input: $\mathcal{I}$, $\mathcal{W} = \{p_1, \ldots, p_K\}$, T, $\{\mathcal{C}_m\}_{m \in [M]}$ and $\widetilde{\alpha}(k)$, $\widetilde{\beta}(k)$ $\forall k \in [K]$
Initialize: $\widehat{\alpha}_0(k) = \widetilde{\alpha}(k)$, $\widehat{\beta}_0(k) = \widetilde{\beta}(k)$, $S_0(k) = 0$, $n_0(k) = 0$, $\forall k \in [K]$
for $t = 1, 2, \ldots, T$ **do**
  a random sample of buyers $\mathcal{I}_t \subset \mathcal{I}$ arrives
  for $m = 1, 2, \ldots, M$ **do**
    sample $q'_t(k) \sim \text{Beta}(\widehat{\alpha}_{t-1}(k), \widehat{\beta}_{t-1}(k))$, $\forall k \in [K]$
    select $k^{(m)} = \text{argmax}_{k \in [K]}\, p_k \cdot q'_t(k)$
    offer price $W_{i,t} = p_{k^{(m)}} \in \mathcal{W}$, $\forall i \in \mathcal{C}_m$
  observe purchase decision $X_{i,t}$, $\forall i \in \mathcal{I}_t$ // $X_{i,t}$ is a function of $\boldsymbol{W}_t$ due to interference
  calculate revenue $Y_{i,t} = W_{i,t} \cdot X_{i,t}$, $\forall i \in \mathcal{I}_t$
  update
    $S_t(k) = S_{t-1}(k) + \sum_{m=1}^{M} \mathbb{1}\left\{\boldsymbol{W}_{\mathcal{C}_{m,t}} = p_k \cdot \mathbf{1}_{N_m}\right\} \sum_{i \in \mathcal{C}_m} Y_{i,t}$, $\forall k \in [K]$
    $n_t(k) = n_{t-1}(k) + \sum_{m=1}^{M} \mathbb{1}\left\{\boldsymbol{W}_{\mathcal{C}_{m,t}} = p_k \cdot \mathbf{1}_{N_m}\right\} \cdot N_m$, $\forall k \in [K]$
    $\widehat{\alpha}_t(k) = \widetilde{\alpha}(k) + S_t(k)$ & $\widehat{\beta}_t(k) = \widetilde{\beta}(k) + n_t(k) - S_t(k)$, $\forall k \in [K]$
Output: $\boldsymbol{W}_t$, $\boldsymbol{Y}_t$, $\forall t \in [T]$

Appendix 4.

BOX PLOTS FOR PARAMETRIC AND NONPARAMETRIC LEARNERS

In this Appendix, we present additional box plots for the parametric and nonparametric learners.

A4.1 Box plots for TS Learners

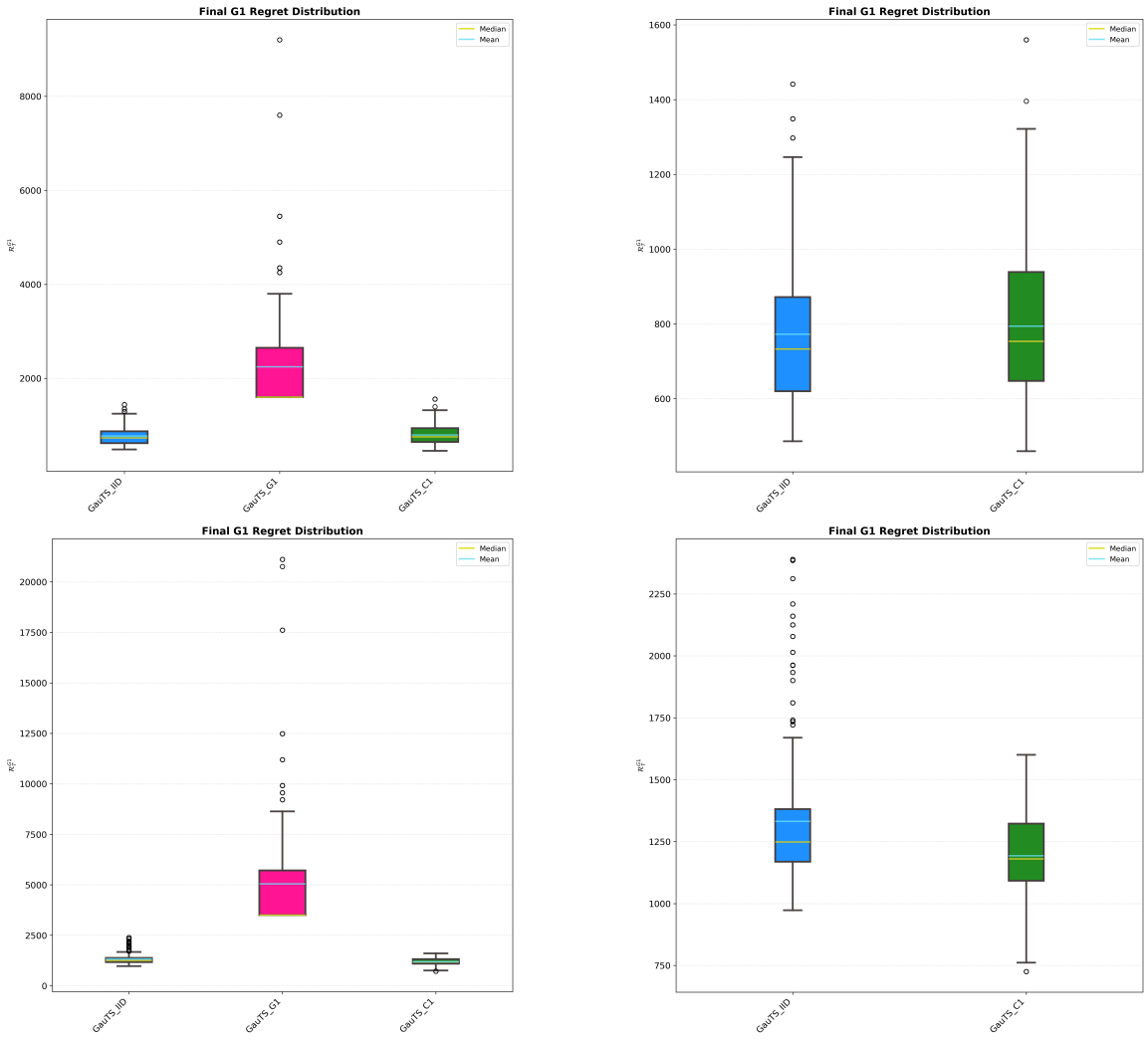


Figure 4.1: Box plots of empirical pseudo $G-1$ regrets at $T=500$. Top Left: Zero Interference: IID, $G-1$, and $C-1$ learners. Top Right: Zero Interference IID and $C-1$ learners only. Bottom Left: No-OS: IID, $G-1$, and $C-1$ learners. Bottom Right: No-OS: IID and $C-1$ learners only.

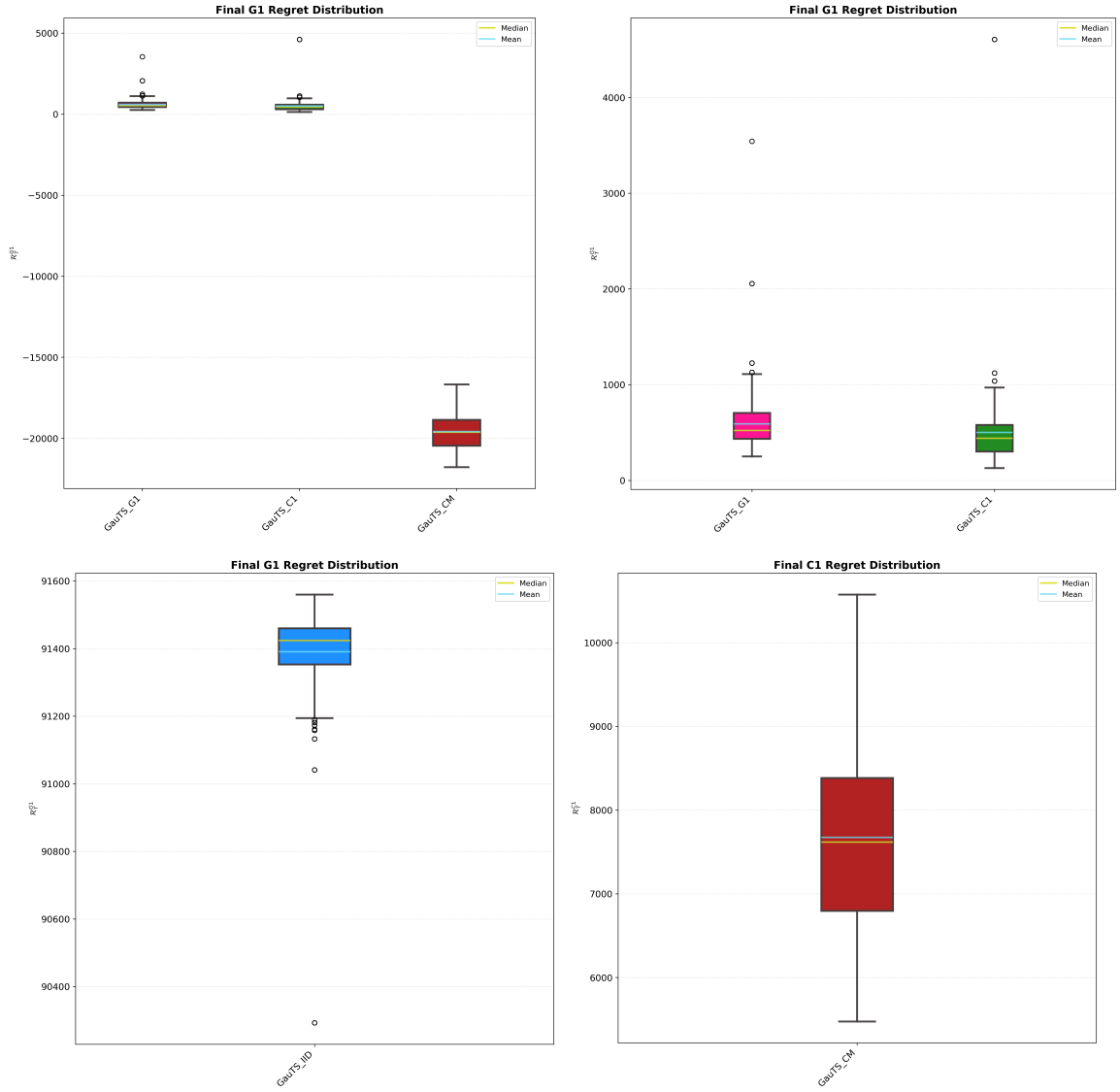


Figure 4.2: Boxplots of empirical pseudo $G-1$ regrets at $T = 500$ for the **OS** scenario. Top left: $G-1$, $C-1$, and $C-M$ learners. Top right: $G-1$ and $C-1$ learners. Bottom left: $C-M$ learner only. Bottom right: IID learner only.

A4.2 Boxplots for NP Learners

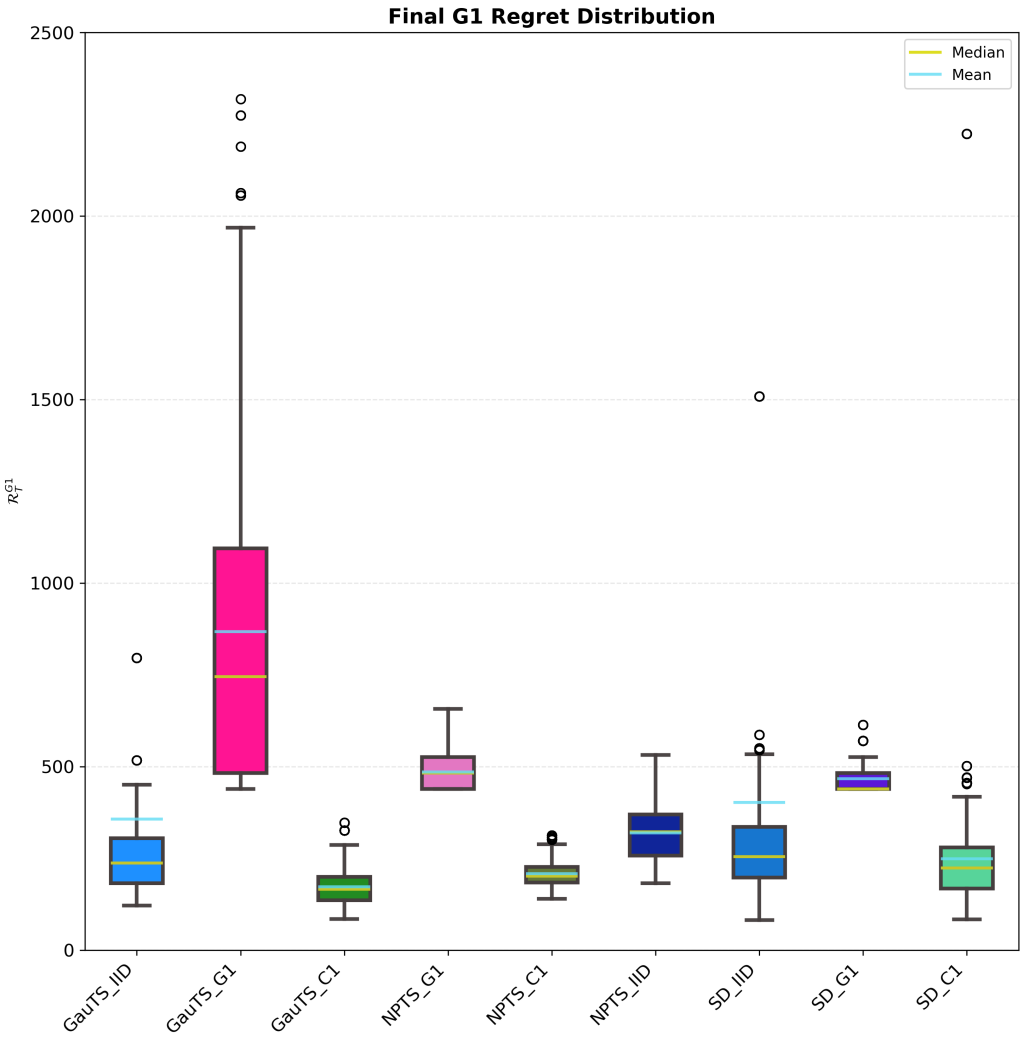


Figure 4.3: Boxplots of $G-1$ regret at time $T=300$ computed with 1000 Monte Carlo simulations for different learners in the No Optimal Switching case.

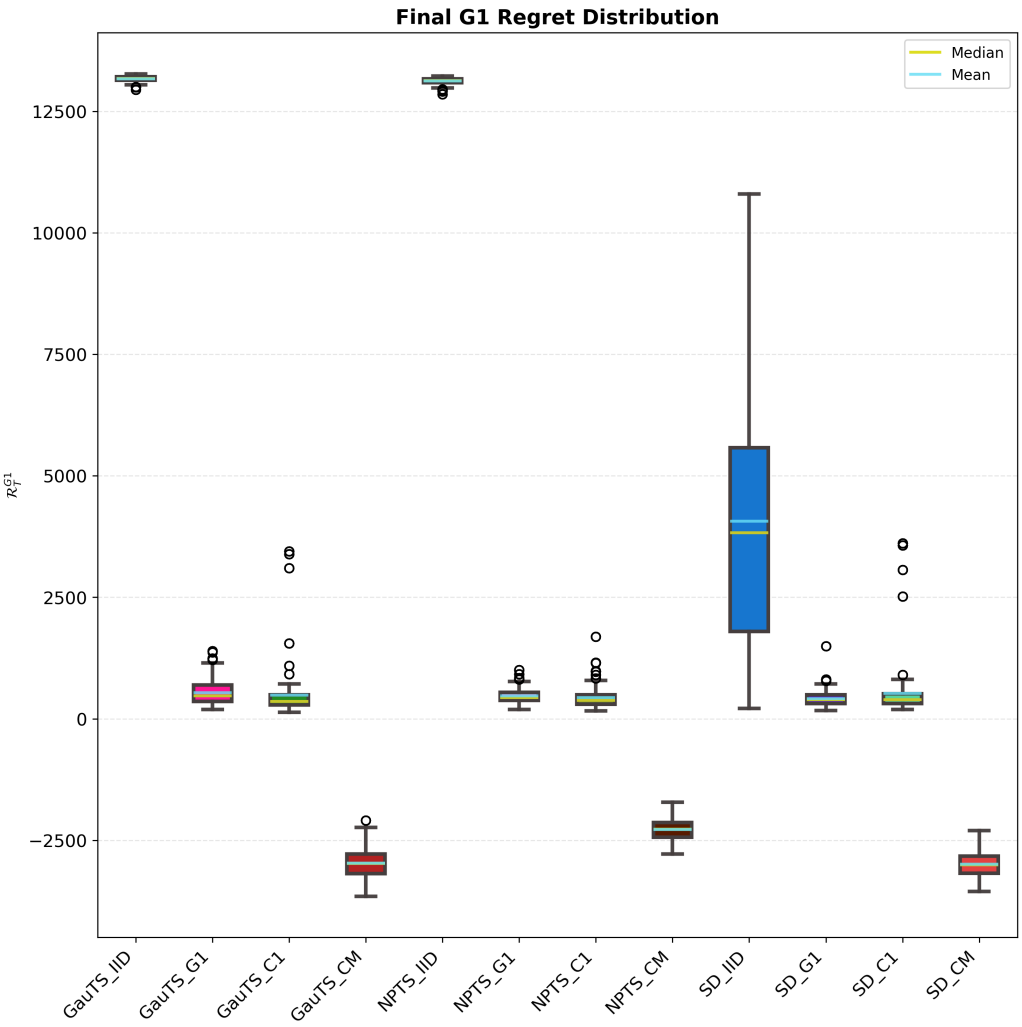


Figure 4.4: Boxplots of $G-1$ regret at time $T=300$ computed with 1000 Monte Carlo simulations for different learners in the Optimal Switching case.

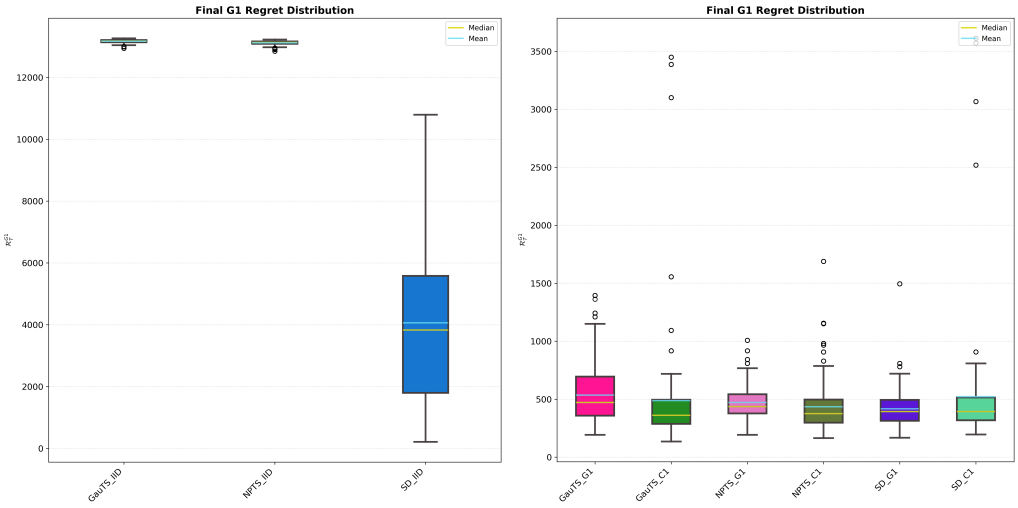


Figure 4.5: We separately show the boxplot of $G-1$ regret at time $T=300$ for (Left) IID learners and (Right) $G-1$ and $C-1$ learners.

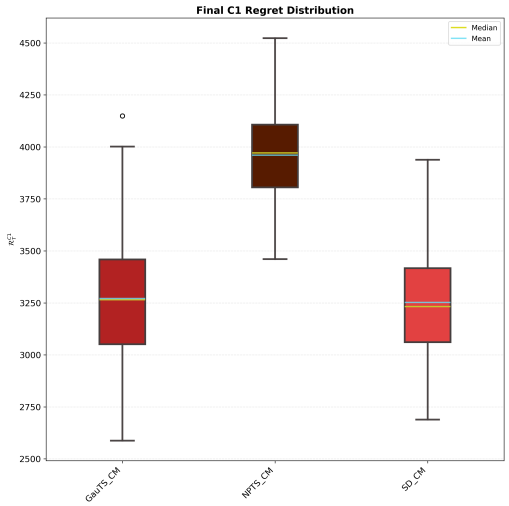


Figure 4.6: Boxplots of $C-1$ regret at time $T=300$ computed with 1000 Monte Carlo simulations for different $C-M$ learners in the Optimal Switching case.

List of Tables and Figures

Tables

Figures

List of Algorithms

List of Symbols and Notation

We gather here some common symbols and notation used throughout this dissertation for easy reference. This list is not exhaustive and mostly focused on stochastic bandit setup.

Common Symbols and Operators

Symbol	**Description**
$[a]$	set of integers from 1 to s, $\{1, 2, \ldots, a-1, a\}$
$a \vee b$	maximum of a and b
$a \wedge b$	minimum of a and b
$\mathrm{clip}_{[a,b]}(x)$	clip function that restricts x to $[a, b]$, $\mathrm{clip}_{[a,b]}(x) := a \vee (b \wedge x)$
$o(\cdot)$	small o notation
$O(\cdot)$	big O notation
$\mathbf{A}$	bold uppercase letters denote matrices
$\mathbb{E}(\cdot)$	expectation operator
$\mathbb{V}(\cdot)$	variance operator
$\mathbb{P}(\cdot)$	probability measure
$D_{\mathrm{kl}}(F_1 \Vert F_2)$	Kullback-Leibler (KL) divergence between distributions F_1 and F_2
$\mathbb{1}\{x\}$	indicator function, equals 1 if x is true, 0 otherwise
$\mathbf{1}_n$	n-dimensional vector of all ones
$\lvert A \rvert$	cardinality of set A
$\mathrm{Bernoulli}(p)$	Bernoulli distribution with parameter p
$\mathrm{Beta}(\alpha, \beta)$	Beta distribution with parameters α and β
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$\mathcal{U}(a, b)$	Uniform distribution on the interval $[a, b]$

Stochastic Bandit Model

Symbol	**Description**
K	number of arms/treatments/actions
T	time horizon or number of time steps
$\mathcal{W}$	set of arms
W_t	arm chosen at time t
$Y_t(k)$	potential reward for arm k at time t
Y_t	observed reward at time t, $Y_t := Y_t(W_t)$
$\mathcal{Y}$	reward space

Stochastic Bandit Model (continued)

Symbol	Description
$\mu(k)$	mean reward of arm k
k^*	optimal arm, $k^* := \arg\max_{k \in \mathcal{W}} \mu(k)$
μ^*	optimal mean reward, $\mu^* := \max_{k \in \mathcal{W}} \mu(k) = \mu(k^*)$
$\Delta(k)$	suboptimality gap of arm k, $\Delta(k) := \mu^* - \mu(k)$
μ	set of mean rewards of all arms, $\mu := \{\mu(k) : k \in \mathcal{W}\}$
$\mathrm{F}_{\mu(k)}$	reward distribution of arm k parameterized $\mu(k)$, i.e., $Y_t(k) \sim \mathrm{F}_{\mu(k)}$
$\mathscr{F}_\mu$	environment parameterized by mean rewards, $\mathscr{F}_\mu := \{F_{\mu(k)} : k \in \mathcal{W}\}$
π_t	policy function of learner at time t
π	policy of learner, $\pi := \{\pi_t\}_{t=1}^T$
$n_t(k)$	number of pulls of arm k up to time t, $n_t(k) := \sum_{s=1}^t \mathbf{1}_{\{W_s=k\}}$
$S_t(k)$	cumulative reward of arm k up to time t, $S_t(k) := \sum_{s=1}^t Y_s(k)\mathbf{1}_{\{W_s=k\}}$
r_t	instantaneous regret at time t, $r_t = \mu^* - \mu(W_t)$
$\mathcal{R}_t$	pseudo or random regret, $\mathcal{R}_T := \sum_{t=1}^T r_t$
$\mathbb{E}(\mathcal{R}_T)$	regret

Bayesian Bandit Model

Symbol	Description
$\mathcal{E}$	parameter space
Q	prior distribution over parameter space $\mathcal{E}$
ν	a bandit/environment instance or parameter vector drawn from prior Q
$\nu(k)$	parameter vector of arm k in instance ν
$\mathrm{F}_{\nu(k)}$	marginal reward distribution of arm k parameterized by $\nu(k)$
F_ν	environment instance parameterized by ν, $\mathrm{F}_\nu := (\mathrm{F}_{\nu(k)})_{k \in \mathcal{W}}$
F	collection of all environment instances, $\mathrm{F} := \{\mathrm{F}_{\nu(k)} : \nu \in \mathcal{E}, k \in \mathcal{W}\}$
$\mathcal{BR}_T$	Bayesian regret over time horizon T, $\mathcal{BR}_T := \mathbb{E}_{\nu \sim Q}[\mathbb{E}(\mathcal{R}_T)]$

Stochastic Bandit Model with Interference

Symbol	Description
N	number of units in batch
$\mathcal{I}$	set of units in batch, $\mathcal{I} := [N]$
i	unit index, $i \in \mathcal{I}$
M	number of clusters
m	cluster index, $m \in [M]$
$\mathcal{C}_m$	set of units in cluster m or m-th cluster, $\mathcal{C}_m \subseteq \mathcal{I}$

Stochastic Bandit Model with Interference (continued)

Symbol	Description
N_m	number of units in cluster m, $N_m := \lvert\mathcal{C}_m\rvert$
$\mathscr{C}$	partition of $\mathcal{I}$ into M clusters, $\mathscr{C} := \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M\}$, $\bigcup_{m=1}^{M} \mathcal{C}_m = \mathcal{I}$, $\mathcal{C}_m \cap \mathcal{C}_{m'} = \emptyset$ for $m \neq m'$
$\mathcal{W}^N$	set of arm vectors of length N with each arm in $\mathcal{W}$
$\boldsymbol{k}$	arm vector generally denoted in boldface, e.g., $\boldsymbol{k} = (k_1, k_2, \ldots, k_N)'$
$W_{i,t}$	arm assigned to unit i at t, $W_{i,t} \in \mathcal{W}$
$\boldsymbol{W}_t$	arm vector assigned to batch at t, $\boldsymbol{W}_t := (W_{1,t}, W_{2,t}, \ldots, W_{N,t})' \in \mathcal{W}^N$
$Y_{i,t}(\boldsymbol{k})$	potential reward of unit i at t for $\boldsymbol{k}$
$\boldsymbol{Y}_t(\boldsymbol{k})$	potential reward vector for batch at t for $\boldsymbol{k}$, $\boldsymbol{Y}_t(\boldsymbol{k}) := (Y_{1,t}(\boldsymbol{k}), Y_{2,t}(\boldsymbol{k}), \ldots, Y_{N,t}(\boldsymbol{k}))'$
$Y_{i,t}$	observed reward of unit i at t, $Y_{i,t} := Y_{i,t}(\boldsymbol{W}_t)$
$\boldsymbol{Y}_t$	observed reward vector at t, $\boldsymbol{Y}_t := \boldsymbol{Y}_t(\boldsymbol{W}_t)$
$\mu_i(\boldsymbol{k})$	mean reward of unit i for arm vector $\boldsymbol{k}$
$\boldsymbol{\mu}(\boldsymbol{k})$	mean reward vector for batch for arm vector $\boldsymbol{k}$, $\boldsymbol{\mu}(\boldsymbol{k}) := (\mu_1(\boldsymbol{k}), \mu_2(\boldsymbol{k}), \ldots, \mu_N(\boldsymbol{k}))'$
$\boldsymbol{\mu}$	set of mean reward vectors, $\boldsymbol{\mu} := \{\boldsymbol{\mu}(\boldsymbol{k}) : \boldsymbol{k} \in \mathcal{W}^N\}$
$\mu_\Sigma(\boldsymbol{k})$	batch sum of mean rewards for arm vector $\boldsymbol{k}$, $\mu_\Sigma(\boldsymbol{k}) := \sum_{i=1}^{N} \mu_i(\boldsymbol{k})$
$\mathrm{F}_{\boldsymbol{k}}$	joint potential reward distribution for arm vector $\boldsymbol{k}$, $\boldsymbol{Y}_t(\boldsymbol{k}) \sim \mathrm{F}_{\boldsymbol{k}}$
$\mathscr{F}$	environment, set of joint potential reward distributions, $\mathscr{F} := \{\mathrm{F}_{\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{W}^N\}$
$\mathrm{F}_{i,\boldsymbol{k}}$	marginal potential reward distribution for unit i and arm vector $\boldsymbol{k}$
$\mathscr{F}_i$	set of marginal potential reward distributions for unit i, $\mathscr{F}_i := \{\mathrm{F}_{i,\boldsymbol{k}} : \boldsymbol{k} \in \mathcal{W}^N\}$
$\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})}$	joint potential reward distribution for arm vector $\boldsymbol{k}$ parameterized by mean reward vector $\boldsymbol{\mu}$
$\mathrm{F}_{\mu_i(\boldsymbol{k})}$	marginal potential reward distribution for unit i parameterized by mean reward $\mu_i(\boldsymbol{k})$
$\mathscr{F}_{\boldsymbol{\mu}}$	environment parameterized by mean reward vector $\boldsymbol{\mu}$, $\mathscr{F}_{\boldsymbol{\mu}} := \{\mathrm{F}_{\boldsymbol{\mu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$
$\boldsymbol{k}_N$	$G-1$ vector with all entries k of length N, e.g., $\mathbf{1}_3 = (1,1,1)'$
$\boldsymbol{k}_{N,C-1}$	general $C-1$ vector under partition $\mathscr{C}$, can be specified as $\boldsymbol{k}_{N,C-1} = (\boldsymbol{k}_{N_1}^{(1)}, \boldsymbol{k}_{N_2}^{(2)}, \ldots, \boldsymbol{k}_{N_M}^{(M)})$ where each $i \in \mathcal{C}_m$ plays arm $k^{(m)}$, for each $m \in [M]$
π	learner policy (possibly randomized), selecting $\boldsymbol{W}_t$ based on past history
$\mathcal{W}^{g1}$	set of $G-1$ vectors of length N
$\mathcal{W}^{c1}$	set of $C-1$ vectors under partition $\mathscr{C}$
$\boldsymbol{k}^*$	optimal arm vector, $\boldsymbol{k}^* := \arg\max_{\boldsymbol{k} \in \mathcal{W}^N} \mu_\Sigma(\boldsymbol{k})$
$\boldsymbol{k}_N^*$	optimal $G-1$ arm vector, $\boldsymbol{k}_N^* := \arg\max_{\boldsymbol{k} \in \mathcal{W}^{g1}} \mu_\Sigma(\boldsymbol{k})$
$\boldsymbol{k}_{N,C-1}^*$	optimal $C-1$ arm vector, $\boldsymbol{k}_{N,C-1}^* := \arg\max_{\boldsymbol{k} \in \mathcal{W}^{c1}} \mu_\Sigma(\boldsymbol{k})$
$\mathcal{R}_T^{g1}(\mathscr{F}_{\boldsymbol{\mu}}, \pi)$	$G-1$ regret over horizon T with respect to arm vector $\boldsymbol{k}_N^*$,
$\mathcal{R}_T^{c1}(\mathscr{F}_{\boldsymbol{\mu}}, \pi)$	$C-1$ regret over horizon T with respect to arm vector $\boldsymbol{k}_{N,C-1}^*$,

Bayesian Bandit With Interference

Symbol	Description
$\boldsymbol{\mathcal{E}}(\boldsymbol{k})$	parameter space for a fixed arm vector $\boldsymbol{k}$ (e.g., set of all possible mean vectors $\boldsymbol{\mu}(\boldsymbol{k})$)
$Q_{\boldsymbol{k}}$	prior distribution over $\boldsymbol{\mathcal{E}}(\boldsymbol{k})$
$\boldsymbol{\mathcal{E}}$	global parameter space across all arm vectors, $\boldsymbol{\mathcal{E}} := \prod_{\boldsymbol{k}\in\mathcal{W}^N} \boldsymbol{\mathcal{E}}(\boldsymbol{k})$
$\boldsymbol{Q}$	global prior over $\boldsymbol{\mathcal{E}}$ (product prior), $\boldsymbol{Q} := \prod_{\boldsymbol{k}\in\mathcal{W}^N} Q_{\boldsymbol{k}}$
$\boldsymbol{\nu}$	a bandit instance/parameter object drawn once at $t = 0$, $\boldsymbol{\nu} \sim \boldsymbol{Q}$
$\boldsymbol{\nu}(\boldsymbol{k})$	the component of instance $\boldsymbol{\nu}$ corresponding to arm vector $\boldsymbol{k}$ (e.g., the mean vector / parameter governing rewards under $\boldsymbol{k}$)
$\mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})}$	joint reward distribution (Bayesian indexing) for arm vector $\boldsymbol{k}$ under instance $\boldsymbol{\nu}$
$\mathscr{F}_{\boldsymbol{\nu}}$	environment for a fixed instance $\boldsymbol{\nu}$, $\mathscr{F}_{\boldsymbol{\nu}} := \{\mathrm{F}_{\boldsymbol{\nu}(\boldsymbol{k})} : \boldsymbol{k} \in \mathcal{W}^N\}$
$\mathscr{F}_{\mathcal{B}}$	collection of all instance-environments, $\mathscr{F}_{\mathcal{B}} := \{\mathscr{F}_{\boldsymbol{\nu}} : \boldsymbol{\nu} \in \boldsymbol{\mathcal{E}}\}$
$(\boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathscr{F}_{\mathcal{B}})$	Bayesian bandit environment under interference
$\boldsymbol{k}_N^*$	optimal $G-1$ arm vector for the realized instance $\boldsymbol{\nu}$, $\boldsymbol{k}_N^* := \arg\max_{\boldsymbol{k}\in\mathcal{W}^{g1}} \mu_\Sigma(\boldsymbol{k})$
$\mathcal{BR}_T^{g1}$	Bayesian $G-1$ regret over horizon T under interference, $\mathcal{BR}_T^{g1}(\pi, \boldsymbol{\mathcal{E}}, \boldsymbol{Q}, \mathscr{F}_{\mathcal{B}}) := \mathbb{E}_{\boldsymbol{\nu}\sim Q,\, \pi}\left[\sum_{t=1}^{T} (\mu_\Sigma(\boldsymbol{k}_N^*) - \mu_\Sigma(\boldsymbol{W}_t))\right]$

REFERENCES

Agarwal, A., Agarwal, A., Masoero, L., & Whitehouse, J. (2024). Multi-Armed Bandits with Network Interference. In *Advances in neural information processing systems* (Vol. 37, pp. 36414–36437).

Agrawal, S., & Goyal, N. (2012). Analysis of Thompson Sampling for the Multi-armed Bandit Problem. In *Proceedings of the 25th Annual Conference on Learning Theory* (pp. 39.1–39.26). JMLR Workshop and Conference Proceedings.

Agrawal, S., & Goyal, N. (2013). Further Optimal Regret Bounds for Thompson Sampling. In *Proceedings of the 16th conference on artificial intelligence and statistics* (pp. 99–107).

Agrawal, S., & Goyal, N. (2017). Near-Optimal Regret Bounds for Thompson Sampling. *Journal of the ACM*, *64*, 1–24.

Aguirregabiria, V. (2021). *Empirical industrial organization: Models, methods, and applications.* (BOOK PROJECT: Unpublished)

Aronow, P. M., & Samii, C. (2017). Estimating Average Causal Effects under General Interference, with Application to a Social Network Experiment. *The Annals of Applied Statistics*, *11*, 1912–1947.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*, 235–256.

Baransi, A., Maillard, O.-A., & Mannor, S. (2014). Sub-Sampling for Multi-Armed Bandits. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 115–131). Springer.

Baudry, D. (2022). *Non-parametric algorithms for multi-armed bandits* (Unpublished doctoral dissertation). Université de Lille.

Baudry, D., Kaufmann, E., & Maillard, O.-A. (2020). Sub-Sampling for Efficient Non-Parametric Bandit Exploration. *Advances in Neural Information Processing Systems*, *33*, 1433–1444.

Besbes, O., & Zeevi, A. (2009). Dynamic Pricing Without Knowing the Demand Function: Risk Bounds and Near-Optimal Algorithms. *Operations Research*, *57*, 1407–1420.

Blattman, C., Green, D. P., Ortega, D., & Tobón, S. (2021). Place-Based Interventions at Scale: The Direct and Spillover Effects of Policing and City Services on Crime. *Journal of the European Economic Association*, *19*, 2022–2051.

Bramoullé, Y., Djebbari, H., & Fortin, B. (2020). Peer Effects in Networks: A Survey. *Annual Review of Economics*, *12*, 603–629.

Bubeck, S., Cesa-Bianchi, N., & Lugosi, G. (2013). Bandits with Heavy Tail. *IEEE Transactions on Information Theory*, *59*, 7711–7717.

Cai, J., Janvry, A. D., & Sadoulet, E. (2015). Social Networks and the Decision to Insure. *American Economic Journal: Applied Economics*, *7*, 81–108.

Chan, H. P. (2020). The Multi-Armed Bandit Problem: An Efficient Nonparametric Solution. *The Annals of Statistics*, *48*, 346–373.

Chapelle, O., & Li, L. (2011). An Empirical Evaluation of Thompson Sampling. In *Advances in Neural Information Processing Systems* (Vol. 24, pp. 2249–2257).

Chin, A. (2019). Regression Adjustments for Estimating the Global Treatment Effect in Experiments with Interference. *Journal of Causal Inference*, *7*, 20180026.

Cox, D. R. (1958). *Planning of experiments*. Oxford, England: Wiley.

Den Boer, A. V. (2015). Dynamic Pricing and Learning: Historical Origins, Current Research, and New Directions. *Surveys in Operations Research and Management Science*, *20*, 1–18.

Duchemin, Q., & de Castro, Y. (2022). *Random geometric graph: Some recent developments and perspectives.*

Eckles, D., Karrer, B., & Ugander, J. (2017). Design and Analysis of Experiments in Networks: Reducing Bias from Interference. *Journal of Causal Inference*, *5*, 20150021.

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, *7*, 1 – 26.

Efron, B., & Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Faruk, O., & Zheleva, E. (2023). *Leveraging Heterogeneous Spillover in Maximizing Contextual Bandit Rewards in Social Networks.* arXiv preprint arXiv:2310.10259. (Under review for TheWebConf 2025.)

Forastiere, L., Airoldi, E. M., & Mealli, F. (2021). Identification and Estimation of Treatment and Interference Effects in Observational Studies on Networks. *Journal of the American Statistical Association*, *116*, 901–918.

Gleich, D. F., Laber, E. B., & Volfovsky, A. (2025). *Scalable Thompson Sampling for Network Interference.* arXiv preprint arXiv:2505.18118. (This is a speculative entry for a future paper.)

Hao, B., Wen, Z., Abbasi-Yadkori, Y., & Cheng, G. (2019). Bootstrapping Upper Confidence Bound. In *Advances in neural information processing systems* (Vol. 32, pp. 12123–12133).

Hudgens, M. G., & Halloran, M. E. (2008). Toward Causal Inference With Interference. *Journal of the American Statistical Association*, *103*, 832–842.

Imbens, G. W., & Rubin, D. B. (2015). *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction* (1st edition ed.). Cambridge New York, NY Port Melbourne New Delhi Singapore: Cambridge University Press.

Izenman, A. J. (2023). *Network models for data science*. Cambridge University Press.

Jagadeesan, R., Pillai, N., & Volfovsky, A. (2017). *Designs for estimating the treatment effect in networks with interference* (No. arXiv:1705.08524). arXiv.

Jia, S., Frazier, P. I., & Kallus, N. (2025). Multi-Armed Bandits with Interference: Bridging Causal Inference and Adversarial Bandits. In *Proceedings of the 42nd international conference on machine learning*. PMLR. (Also available as arXiv:2402.01845. Publication details are speculative.)

Kasy, M., & Sautmann, A. (2021). Adaptive Treatment Assignment in Experiments for Policy Choice. *Econometrica*, *89*, 113–132.

Kaufmann, E., Korda, N., & Munos, R. (2012). Thompson Sampling: An Asymptotically Optimal Finite-Time Analysis. In N. H. Bshouty, G. Stoltz, N. Vayatis, & T. Zeugmann (Eds.), *Algorithmic Learning Theory* (pp. 199–213). Berlin, Heidelberg: Springer.

Kleinberg, R., & Leighton, T. (2003). The Value of Knowing a Demand Curve: Bounds on Regret for Online Posted-Price Auctions. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* (pp. 594–605). Cambridge, MA, USA: IEEE Computer. Soc.

Kock, A. B., Preinerstorfer, D., & Veliyev, B. (2023). Treatment Recommendation with Distributional Targets. *Journal of Econometrics*, *234*, 624–646.

Kveton, B., Szepesvari, C., Ghavamzadeh, M., & Boutilier, C. (2019). Perturbed-History Exploration in Stochastic Multi-Armed Bandits. In *Proceedings of the 35th uncertainty in artificial intelligence conference* (pp. 1058–1068).

Kveton, B., Szepesvari, C., Vaswani, S., Wen, Z., Lattimore, T., & Ghavamzadeh, M. (2019). Garbage in, Reward out: Bootstrapping Exploration in Multi-Armed Bandits. In *International conference on machine learning* (pp. 3601–3610). PMLR.

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

Lattimore, T., & Szepesvári, C. (2020). *Bandit Algorithms* (1st ed.). Cambridge University Press.

Lee, C., Zeng, D., & Hudgens, M. G. (2024). Efficient Nonparametric Estimation of Stochastic Policy Effects with Clustered Interference. *Journal of the American Statistical Association*, , 1–13.

Leung, M. P. (2022). Rate-Optimal Cluster-Randomized Designs for Spatial Interference. *The Annals of Statistics*, *50*, 3064–3087.

Leung, M. P. (2023). Network Cluster-Robust Inference. *Econometrica*, *91*, 641–667.

Manski, C. F. (1993). Identification of Endogenous Social Effects: The Reflection Problem. *The Review of Economic Studies*, *60*, 531.

Osband, I., & Van Roy, B. (2015). *Bootstrapped thompson sampling and deep exploration.* arXiv preprint arXiv:1507.00300.

Papadogeorgou, G., Mealli, F., & Zigler, C. M. (2018). *Causal inference for interfering units with cluster and population level treatment allocation programs* (No. arXiv:1711.01280). arXiv.

Pearl, J. (2009). *Causality: Models, reasoning, and inference* (2nd ed.). Cambridge: Cambridge University Press. (See Ch. 7, Sec. 7.2.2; discussion around p. 220 ("intrinsic nondeterminism").)

Penrose, M. D. (2016). Connectivity of Soft Random Geometric Graphs. *The Annals of Applied Probability*, *26*, .

Politis, D. N., Romano, J. P., & Wolf, M. (1999). *Subsampling*. New York: Springer.

Riou, C., & Honda, J. (2020). Bandit Algorithms Based on Thompson Sampling for Bounded Reward Distributions. In *Algorithmic learning theory* (pp. 777–826). PMLR.

Robbins, H. (1952). Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, *58*, 527–535.

Robins, J. M., & Greenland, S. (1989). The Probability of Causation under a Stochastic Model for Individual Risk. *Biometrics*, *45*, 1125–1138.

Robins, J. M., & Greenland, S. (2000). Causal Inference Without Counterfactuals: Comment. *Journal of the American Statistical Association*, *95*, 431–435.

Rubin, D. B. (1974). Estimating Causal Effects of Treatments in Randomized and Nonrandomized Studies. *Journal of Educational Psychology*, *66*, 688–701.

Rubin, D. B. (1976). Inference and Missing Data. *Biometrika*, *63*, 581–592. (With discussion)

Rubin, D. B. (1980). Randomization Analysis of Experimental Data: The Fisher Randomization Test Comment. *Journal of the American Statistical Association*, *75*, 591–593.

Russo, D., & Roy, B. V. (2016). An Information-Theoretic Analysis of Thompson Sampling. *Journal of Machine Learning Research*, *17*, 1–30.

Russo, D., Roy, B. V., Kazerouni, A., Osband, I., & Wen, Z. (2020). *A Tutorial on Thompson Sampling* (No. arXiv:1707.02038). arXiv.

Russo, D., & Van Roy, B. (2014). Learning to Optimize via Posterior Sampling. *Math. Oper. Res.*, *39*, 1221–1243.

Slivkins, A. (2022). *Introduction to Multi-Armed Bandits.* arXiv. (arXiv:1904.07272 [cs, stat])

Sobel, M. E. (2006). What Do Randomized Studies of Housing Mobility Demonstrate?: Causal Inference in the Face of Interference. *Journal of the American Statistical Association*, *101*, 1398–1407.

Su, L., Lu, W., & Song, R. (2019). Modelling and estimation for optimal treatment decision with interference. *Stat*, *8*, e219.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. The MIT Press.

Tchetgen, E. J. T., & VanderWeele, T. J. (2012). On Causal Inference in the Presence of Interference. *Statistical Methods in Medical Research*, *21*, 55–75.

Thompson, W. R. (1933). On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, *25*, 285–294.

Ugander, J., Karrer, B., Backstrom, L., & Kleinberg, J. (2013). *Graph cluster randomization: Network exposure to multiple universes* (No. arXiv:1305.6979). arXiv.

VanderWeele, T. J., & Robins, J. M. (2012). Stochastic Counterfactuals and Stochastic Sufficient Causes. *Statistica Sinica*, *22*, 379–392.

Verbitsky-Savitz, N., & Raudenbush, S. W. (2012). Causal Inference Under Interference in Spatial Settings: A Case Study Evaluating Community Policing Program in Chicago. *Epidemiologic Methods*, *1*, 1–23.

Vermorel, J., & Mohri, M. (2005). Multi-Armed Bandit Algorithms and Empirical Evaluation. In J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, & L. Torgo (Eds.), *Machine Learning: ECML 2005* (pp. 437–448). Berlin, Heidelberg: Springer.

Von Luxburg, U. (2007). A Tutorial on Spectral Clustering. *Statistics and Computing*, *17*, 395–416.

Wang, C.-H., Yu, Y., Hao, B., & Cheng, G. (2020). Residual Bootstrap Exploration for Bandit Algorithms. In *Proceedings of the 36th conference on uncertainty in artificial intelligence (uai)* (pp. 310–319).

Xu, Y., Lu, W., & Song, R. (2024). *Linear Contextual Bandits with Interference.* arXiv preprint arXiv:2409.15682.

Zhang, Z., & Wang, Z. (2024). *Online Experimental Design With Estimation-Regret Trade-off Under Network Interference.* arXiv preprint arXiv:2412.03727.