Algorithmes en temps polynomial pour les semi-bandits combinatoires : apprentissage par renforcement efficace dans des environnements complexes

Thibaut Cuvelier

université
PARIS-SACLAY

Algorithmes en temps polynomial pour les semi-bandits combinatoires : apprentissage par renforcement efficace dans des environnements complexes

Polynomial-Time Algorithms for Combinatorial Semibandits: Computationally Tractable Reinforcement Learning in Complex Environments

Thèse de doctorat de l'université Paris-Saclay

École doctorale n^o 580, Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : réseaux, information et communications
Unité de recherche : université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190 Gif-sur-Yvette, France
Référent : CentraleSupélec

Thèse présentée et soutenue à Paris-Saclay, le 24 juin 2021, par

Thibaut Cuvelier

Composition du jury

Alexandra Carpentier W2-Professur, Otto-von-Guericke-Universität Magdeburg	Présidente
Odalric-Ambrym Maillard Chargé de recherche, Inria Lille	Rapporteur et examinateur
Vianney Perchet Professeur, ENSAE Paris	Rapporteur et examinateur
Vincent François-Lavet Universitair Docent, Vrije Universiteit Amsterdam	Examinateur
Rémi Munos Directeur de recherche, Inria Lille Directeur de laboratoire, Google DeepMind	Examinateur

Direction de la thèse

Zwi Altman Ingénieur de recherche, Orange Labs	Directeur
Richard Combes Professeur adjoint, CentraleSupélec	Codirecteur
Éric Gourdin Ingénieur de recherche, Orange Labs	Codirecteur

Thèse de doctorat

NNT : 2021UPASG020

Polynomial-Time Algorithms for Combinatorial Semibandits: Computationally Tractable Reinforcement Learning in Complex Environments

Abstract

Sequential decision making is a core component of many real-world applications, from computer-network operations to online ads. The major tool for this use is reinforcement learning: an agent takes a sequence of decisions in order to achieve its goal, with typically noisy measurements of the evolution of the environment. For instance, a self-driving car can be controlled by such an agent; the environment is the city in which the car manœuvers. Bandit problems are a class of reinforcement learning for which very strong theoretical properties can be shown. The focus of bandit algorithms is on the exploration-exploitation dilemma: in order to have good performance, the agent must have a deep knowledge of its environment (exploration); however, it should also play actions that bring it closer to its goal (exploitation).

In this dissertation, we focus on combinatorial bandits, which are bandits whose decisions are highly structured (a 'combinatorial' structure). These include cases where the learning agent determines a path to follow (on a road, in a computer network, etc.) or ads to display on a Website. Such situations share their computational complexity: while it is often easy to determine the optimum decision when the parameters are known (the time to cross a road, the monetary gain of displaying an ad at a given place), the bandit variant (when the parameters must be determined through interactions with the environment) is more complex.

We propose two new algorithms to tackle these problems by mathematical-optimisation techniques. Based on weak hypotheses, they have a polynomial time complexity, and yet perform well compared to state-of-the-art algorithms for the same problems. They also enjoy excellent statistical properties, meaning that they find a balance between exploration and exploitation that is close to the theoretical optimum. Previous work on combinatorial bandits had to make a choice between computational burden and statistical performance; our algorithms show that there is no need for such a quandary.

Résumé

La prise de décision séquentielle est une composante essentielle de nombreuses applications, de la gestion des réseaux informatiques aux annonces en ligne. L'outil principal est l'apprentissage par renforcement : un agent prend une séquence de décisions afin d'atteindre son objectif, avec des mesures typiquement bruitées de son environnement. Par exemple, un agent peut contrôler une voiture autonome ; l'environnement est la ville dans laquelle la voiture se déplace. Les problèmes de bandits forment une classe d'apprentissage de renforcement pour laquelle on peut démontrer de très forts résultats théoriques. Les algorithmes de bandits se concentrent sur le dilemme exploration-exploitation : pour avoir une bonne performance, l'agent doit avoir une connaissance approfondie de son environnement (exploration) ; cependant, il doit aussi jouer des actions qui le rapprochent de son but (exploitation).

Dans cette thèse, nous nous concentrons sur les bandits combinatoires, qui sont des bandits dont les décisions sont très structurées (une structure « combinatoire »). Il s'agit notamment des cas où l'agent détermine un chemin à suivre (sur une route, dans un réseau informatique, etc.) ou des publicités à afficher sur un site Web. De telles situations partagent leur complexité algorithmique : alors qu'il est souvent facile de déterminer la décision optimale lorsque les paramètres sont connus (le temps pour traverser une route, le profit généré par l'affichage d'une publicité à un endroit donné), la variante bandit (lorsque les paramètres doivent être déterminés par des interactions avec l'environnement) est bien plus complexe.

Nous proposons deux nouveaux algorithmes pour aborder ces problèmes par des techniques d'optimisation mathématique. Basés sur des hypothèses faibles, ils présentent une complexité temporelle polynomiale, tout en étant performants par rapport aux algorithmes de pointe pour les mêmes problèmes. Ils présentent également d'excellentes propriétés statistiques, ce qui signifie qu'ils trouvent un équilibre entre exploration et exploitation proche de l'optimum théorique. Les travaux précédents sur les bandits combinatoires ont dû faire un choix entre le temps de calcul et la performance statistique ; nos algorithmes montrent que ce dilemme n'a pas lieu d'être.

Acknowledgements

Being a PhD student is not the easiest part of one's life, but it still leaves memorable moments, especially those shared with coworkers. I was lucky enough to work partly at Orange Labs, an exceptional place to perform applied research. I would like to thank my former colleagues there for the insightful and/or fun discussions we had, be them fellow PhD students — Paul, Yassine, Julien, Jalal, Ahlam, Marie, Raquel, and Wesley — or permanent researchers — Éric, Zwi, Amal, Nancy, Yannick, Adam, Stéphane, to name a few. In particular, I really thank Éric for offering me these research opportunities. I also have to thank Richard, for he was guiding me for most of the work in this manuscript.

I could not have finished this work without the support of my parents, family, and friends, or without Sarah's continuous presence, in moments when everything was fine... or slightly less OK, notwithstanding writing and technical support, even well outside her comfort zone. These years could not have been the same without you!

I am grateful to the people who helped me polish this manuscript: my advisors (Richard, Éric, Zwi) and Sarah. I take full responsibility for the remaining mistakes.

Contents

CHAPTER 1

Industrial Context

Artificial intelligence has become increasingly popular within Internet service providers (ISPs) to manage their networks. Indeed, they have to operate in a highly competitive environment where user churn is a major problem, while the networks themselves have to handle a large number of devices with sometimes strict quality-of-service (low delay, high bandwidth, etc.) and quality-of-experience (good video-call quality) constraints. To this end, ISPs have to lower their operational costs while providing a superior service, but they also have to prepare for the next generations of network technologies (for instance, fibre access networks for end users and 5G networks). In this context, artificial intelligence is used to take better decisions in complex situations than human operators, but also to deal with uncertainty. Two main categories of tools are used: those from operational research (mostly mixed-integer optimisation, but also linear, convex, continuous, and nonsmooth optimisation) and those from machine learning (especially supervised learning, although reinforcement learning is proving popular).

Artificial-intelligence-based systems are used in many network-related contexts:

operational level: the actual operations of the network of a data centre, ideally based on its current usage. This definition encompasses several domains, among which the most important ones include:

- *network-routing optimisation*: this is probably one of the first applications of optimisation in computer networks, where the goal is to find the best route for each packet in the network [**1, 2, 3**]. Handling the uncertainty is a major challenge [**4, 5**]. However, machine-learning approaches have also been explored in the literature, mostly based on reinforcement learning [**6, 7, 8**]. Supervised learning plays a role in estimating the quality of experience of the users [**9**] or in classifying packets [**10**], including detecting attacks [**11**].
- *resource-usage optimisation*: near-optimum deployments are paramount in data-centre operations, especially to reduce their 'total ownership costs' (TOC or TCO, i.e. the sum of the hardware costs, maintenance, and other indirect costs like social or environmental costs). If fewer servers are bought in the first place, and fewer are in operation (which reduces electricity consumption and heat dissipation, for instance), the data-centre owner can reduce their costs, independent of whether optimisation is performed offline [**12, 13**] or online [**14**].
- *cache optimisation*: to improve the quality of experience of users, a very common technique is to hold a copy of the content in a server near the user (i.e. to *cache* it). Building such a distributed system can be very expensive, and making the best possible use of the resources is of the utmost importance: increasing the capacity of the cache is extremely costly [**15, 16**].

real-time decision making: the control of the elements in the network. This level does not allow more than a few milliseconds to take a decision. This time lapse typically precludes the direct use of mathematical-optimisation tools from operational research. Reinforcement-learning methods have been used for congestion control in transport-level protocols like TCP to regulate the bitrate of applications [**17, 18**] and in beam allocation in 5G technologies to maximise throughput [**19**]. The same techniques can be used to optimise ancillary goals that are not visible for the user, like energy consumption when cooling data centres [**20**] or when transmitting video traffic in a core network [**21, 22**]. Mathematical optimisation can still be used for quick decision making, like network routing: this paradigm can be used to compute one or several plans that are applied in real time [**1, 23**] or to decide which combination of previously determined network routings should be used [**24**].

Many of these tasks tend to be deployed at the edge of the network (a paradigm often called *edge computing*), where less computing power is available, with very little electrical power accessible when compared to a traditional data centre [**25**].

Several problems in this short list hide a very complex combinatorial structure, and a good deal of them are even $\mathcal{NP}$-hard (unsplittable network flows [**2**], virtual-machine placement [**12**], cache optimisation [**26**], etc.). Informally, $\mathcal{NP}$-hardness indicates that the computer-science community as a whole thinks it it unlikely that an efficient algorithm ('polynomial-time') exists for this problem. However, $\mathcal{NP}$-hardness does not indicate that it is not possible to solve exactly such problems, even for instance sizes that are relevant to the industrial practice.

1.1. Machine learning and combinatorial optimisation

Machine learning, as a field, is based on many continuous-optimisation tools, and could not have progressed to the point it stands nowadays without these techniques: variations of gradient descent are the most effective algorithm to train neural networks currently, convex duality is crucial for SVM efficiency. However, both domains have been cross-pollinating for a long time, as results from machine learning are also being applied in optimisation, this time mostly combinatorial (i.e. when discrete decisions must be taken: for example, a user is shown a given number of ads, but not halves or thirds of ads).

There are mostly two ways to combine machine learning and combinatorial optimisation: either the machine-learning algorithm is responsible for (parts of) a solution, a scenario which is called 'principal learning', or for guiding other combinatorial-optimisation techniques, 'joint learning' [**27**].

Principal learning: the machine-learning model is responsible for at least a part of the combinatorial solution. The major problem to tackle is that the parameters for the combinatorial problem (the input to the model) have a variable size, and similarly the combinatorial solution might have a nonconstant number of unknowns too. A typical solution is to 'encode' the solution into a representation that is amenable to learning [**28, 27**]. More specific solutions must be used for graph problems, due to their very specific structure [**29, 30, 31, 32**]. Similar techniques can be used to predict some features of the solution, like the value of multipliers [**33**]. In general, the obtained models are very specific to one kind of combinatorial problem, i.e. a user cannot change the underlying structure (even just adding a few constraints) without retraining the models.

Joint learning: in this case, machine learning is only responsible for guiding a traditional optimisation algorithm. It might simply be determining some parameters of the algorithm

to tune it for the instance to solve [**34**]. More evolved schemes use reinforcement learning within the combinatorial algorithm, to evaluate the quality of choices that are made by the combinatorial algorithm and to improve the next ones; this approach has been used with some success in constraint programming [**35**] and mixed-integer programming [**36**]. To foster new developments in this area, ECOLE is an environment to ease building new components for an existing solver [**36**].

This thesis lies at the intersection of machine learning and combinatorial optimisation by providing two new algorithms for reinforcement learning. These are based on advanced mathematical-optimisation tools.

1.2. Contributions of the thesis

The main scientific contributions of this thesis are two new algorithms, efficient both in theory and in practice, to solve the bandit version of several combinatorial problems, with realistic assumptions. We detail these two new methods in Chapters 3 and 4. They lead to two publications to top-tier conferences in artificial intelligence:

- Thibaut Cuvelier, Richard Combes, Éric Gourdin. Statistically efficient, polynomial-time Algorithms for combinatorial semi-bandits. Accepted in *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, March 2021.
- Thibaut Cuvelier, Richard Combes, Éric Gourdin. Asymptotically optimal strategies for combinatorial semi-bandits in polynomial time. Accepted in *Algorithmic Learning Theory*, January 2021. Proceedings of Machine Learning Research.

The tool that we developed to solve both problems is budgeted optimisation, i.e. solving budgeted linear maximization problems over the combinatorial structures of interest. We build a generic methodology to minimise structured convex functions on combinatorial sets (Section 3.1.4), which is central to our bandit algorithms. This approach can be used in more general situations, as it can outperform existing optimisation technology (Section 3.4).

Another contribution of this thesis is open-source software. All developments were done in Julia [**37**], a technical programming language with a very open community. In particular, all numerical experiments have been carried out with CombinatorialBandits.jl, a package that has been developed throughout the research work. Some of its functionality has been split into easily reusable packages: one for combinatorial optimisation, Kombinator.jl, and one for nonsmooth optimisation, NonsmoothOptim.jl.

CHAPTER 2

Introduction to Bandits

This thesis describes two theoretical breakthroughs in combinatorial bandits, a field of the vast domain of machine learning, more specifically reinforcement learning. This chapter introduces the current context of combinatorial bandits, starting with reinforcement learning (Section 2.1), then specialising this paradigm to bandit problems (Section 2.2). Several kinds of bandit environments are considered: k-armed bandits, the most classical version of the problem, are detailed in Section 2.2, while Section 2.3 introduces contextual bandits and Section 2.4 structured bandits. Combinatorial bandits are a part of these structured bandits, but with specific statistical and computational properties that are explained in Section 2.5. Several combinatorial sets on which these combinatorial bandits work are described in Section 2.6.

2.1. Reinforcement learning

Reinforcement learning is the part of machine learning that learns through trial and error. The most common domain of machine learning is supervised learning: the algorithm is given a data base of input-output pairs, and its goal is to provide a model that approximates this input-output relationship. In reinforcement learning, the data is gathered sequentially, and depends on the actions taken by the learner.

The reinforcement-learning algorithm controls an *agent* that interacts with its *environment*. 'Positive' interactions are *reward*ed, so that the agent knows what it should strive for (similarly, 'negative' interactions can be penalised, i.e. given a negative reward).

2.1.1. Formalisation. Usually, in reinforcement learning, time is discretised: the agents do not take actions continuously, but every time step.

At time step n, the agent is asked to take a decision, i.e. pick an action $x(n)$ in the action space $\mathcal{A}$. This space might be either finite in size or infinite.

In order to take a decision, the agent observes the environment: these observations are a part of the *state* of the environment, denoted by $s(n) \in \mathcal{S}$ where $\mathcal{S}$ is the state space. Each action of the agent influences the state of the environment.

The environment evolution is usually formalised as a probability distribution that relates several variables [**38**]:

- the decision of the agent: the action $x(n)$
- the history of the environment: the current state $s(n)$, all the previous states $\{s(n-1), s(n-2) \dots s(1), s(0)\}$, all the previous rewards $\{r(n-1), r(n-2) \dots r(1)\}$
- the future of the environment: the reward $r(n)$ that is obtained when playing $x(n)$, and the next state $s(n+1)$

The probability distribution can therefore be written as:

$$\mathbb{P}\left\{ r(n)\,, s(n+1) \;\middle|\; \begin{array}{c} s(n)\,, s(n-1)\,, s(n-2) \dots s(1)\,, s(0)\,, \\ r(n-1)\,, r(n-2) \dots r(1)\,, \\ x(n) \end{array} \right\}. \tag{2.1.1}$$

An *episode* corresponds to a series of time steps, starting at an environment-defined initial state $s_0 \in \mathcal{S}$ and ending at some point T where the agent can take no further actions. Not all environments have this notion of episode. For instance, the game of go certainly does (s_0 corresponds to a blank board and T to the turn when one player wins or both players come to a draw).

The *policy* π refers to the way a given reinforcement-learning algorithm takes decisions. A policy is a function that maps a state $s(n) \in \mathcal{S}$ to a probability distribution over actions $\mathcal{A}$, usually based on all previous observations.

An agent interacts with the environment to achieve its goal, defined by the experimenter. This goal is formalised as maximising its total reward over time:

$$\max \sum_{t=1}^{T} r(t)\,. \tag{2.1.2}$$

At each time step, the agent takes one action that should, in expectation, bring it closer to that goal. However, not all actions are directed towards maximising the objective: if the agent only explores one strategy, it might not be able to reach a very high total reward in all cases. The agent must also explore other strategies to get a good grasp of sequences of actions that work well or not. This conundrum is known as the 'exploration-exploitation dilemma'.

Initially ($n = 1$), the agent has barely no knowledge about the environment. It only knows the state space $\mathcal{S}$ and the action space $\mathcal{A}$. It can also observe the initial state $s(0)$. However, it has no information about the impact of its actions on the environment or on the rewards. When the agent interacts with the environment, it sequentially gets more information about the probability distribution relating actions, current state, state transitions, and rewards.

In order to provide a better theoretical understanding of the behaviour of the agent, a metric has proved to be very useful: the regret $R(n)$. It is defined as the difference in total reward between the agent and an 'oracle' **[39]**, i.e. an agent that has a perfect prior knowledge of the environment (at each time step, it fully understands the impact of an action on the state and on the reward). At each time step n, the oracle therefore gets the optimum reward $r^\star(s(n))$ for the current state $s(n)$. The regret can then be written as:

$$R(n) = \sum_{t=1}^{n} \left[r^\star(s(n)) - r(n) \right]. \tag{2.1.3}$$

This definition gave many theoretical results for the analysis of reinforcement learning **[40]**, especially for bandit algorithms (Section 2.2).

2.1.2. Markov decision process. The most common hypothesis to simplify the setting is Markov assumption: the environment is entirely described by a probability distribution over the reward $r(n)$ and the next state $s(n+1)$ that only depends on the current state $s(n)$ and the action that the agent takes $x(n)$ **[38]**:

$$\mathbb{P}\{ r(n)\,, s(n+1) \mid s(n)\,, x(n) \}\,. \tag{2.1.4}$$

In particular, this assumption indicates that the probability distribution does not depend on the current time step: the probability of a given transition from $s(n)$ to $s(n+1)$ does not depend on

n. In other words, it must be stationary. This simplifying assumption leads to *Markov decision processes* (MDP).

2.1.3. Examples. Reinforcement learning can be applied to car driving. However, to fit the paradigm, an artificial discretisation must be applied: for instance, the agent might be asked to take a decision every 50 ms. The state $s(n)$ is composed of the observations of the car, i.e. what is in front of it (a building, a pedestrian, another car, etc.).

$$\mathcal{S} = \{\text{pedestrian 2 m ahead}, \text{car 50 cm ahead}, \text{clear road} \dots\}. \tag{2.1.5}$$

The action $x(n)$ can be 'turn right', 'turn left', or 'continue straight ahead', among others.

$$\mathcal{A} = \{\text{turn right}, \text{turn left}, \text{continue straight ahead} \dots\}. \tag{2.1.6}$$

When a car driver steers, their vehicle moves in the environment (its state evolves), and pedestrians might stop to avoid accidents.

Another example can be fighting in video games, to propose fierce opponents without the bias usually induced by human-crafted artificial-intelligence scripts: a learning agent might think of uncommon strategies for a human, e.g. novel ways of playing the game. In this case, the agent is a fighter, and the environment contains the opponent (and it might also be controlled by reinforcement learning!), while the state corresponds to the position of the opponent and the design of the battlefield. Rewards are given if the agent wins the match **[41]**.[1]

The best-known example, however, is probably the game of go, where a reinforcement-learning-lead algorithm won against the best human players in 2016. The agent is a player, the environment the *goban* board and the other player. The agent must decide, at each round, where to position its next stone **[42]**.

2.1.4. Problem structure. In general, there is very little structure to reinforcement learning. The only hypothesis is Markov assumption, thanks to which only the state for the current time step n is needed to take decisions and evaluate how the environment behaves; it also implies stationarity.

However, this assumption implies no restriction on the action: the agent might take a single discrete action at a time (e.g., in go) or several continuous actions (like car driving). Not all algorithms may handle all these situations, though.

The reward function is supposed to follow some stationary probability distribution and to be only influenced by the current state $s(n)$ and the new action $x(n)$. In particular, this forbids the situation where an adversary chooses the rewards (a scenario studied in Section 2.2.2). Some researchers try to generalise current algorithms to nonstationary environments**[43]** (see also Section 2.2.2).

This thesis studies a very specific case of reinforcement learning, bandit algorithms, that impose a very strong structure on the problem at hand. The major difference is that bandit algorithms consider that there is only one possible state. Many classes of bandit algorithms exist: for instance, k-armed bandits consider that only discrete actions can be taken (Section 2.2); linear bandits suggest that the reward $r(n)$ is a linear function of the action $x(n)$ (Section 2.4); contextual bandits extend the paradigm by adding some state, but it is not influenced by the actions taken by the agent (Section 2.3). Apart from k-armed bandits, all bandit environments can be extended to the case of continuous actions **[44, 45]**, but we will not consider them further.

[1] Ubisoft tried to include reinforcement-learning-based car drivers in Watch Dogs 2 and fighters in For Honor, but the technological advancement when these games were released (2016-2017) was not sufficient. Instead, these agents were used against script-controlled agents and to tune them manually **[41]**.

2.1.5. On-policy and off-policy learning. Certain reinforcement-learning algorithms learn continuously: each action they take and each state transition they see have an impact on the way they will behave for the next time step. The policy of these algorithms therefore changes at each time step; these algorithms include Q-learning and TD-learning, for instance [**38**]. Other algorithms must gather data from one episode at a time to improve. For these algorithms (mostly, the Monte-Carlo family [**38**]), the policy remains constant throughout an episode.

A distinction often made on reinforcement-learning algorithms is that of on-policy and off-policy learning [**38**]. On-policy learning corresponds to situations where the new data points to train the reinforcement-learning system were decided by the current policy: for continuously learning algorithms, the policy decides to take an action, and is then updated from the received reward and the new state; no other data is used to update the policy. To the contrary, off-policy learning corresponds to situations where the reinforcement-learning algorithm is fed with data that does not come from the policy that is being updated: this data may come from previous episodes or interactions with the environment, or from a completely unrelated policy.

Both kinds of learning have very different convergence properties, and off-policy learning tends to show slower convergence in general: an off-policy algorithm may need more interactions with the environment than an on-policy one [**38**]. In the specific case of bandit algorithms, many techniques can perform off-policy learning, with little harm to asymptotic convergence [**46**]: many bandit algorithms can learn from any sample taken from the environment (from any action that is performed on the environment), they do not need to decide which action to perform; the only requirement is to perform sufficient exploration.

2.2. k-armed bandits

The k-armed-bandit problem is a large simplification of reinforcement learning. There is still an agent that takes actions on an environment to gather rewards, but the state disappears [**47**]. With a well-defined mathematical structure, bandit problems are easier to study than the complete reinforcement-learning paradigm.

This simplification allows to focus on one specific issue: the exploration-exploitation dilemma. Should the agent play the best action found so far ('exploit') or rather experiment with other actions in case it missed the true optimum action ('explore')?

Bandit problems became a research topic in 1933 with William Thompson [**48**], who compared the effectiveness of two 'treatments': based on the current data, in order to maximise the number of people that survive, what treatment should the doctor give? The major assumption is that the individuals being treated are valuable: it is not advisable to sacrifice patients by using an inferior treatment while a better one is known.

This setting has become a standard use of bandits. Considering that observing exactly the 'state' of a patient is very difficult, Thompson considered that all that matters is the effectiveness of each cure. This bandit has two arms: the first one corresponds to the first treatment, and the second one to the second medicine. In order to take the best decisions in the long term, the doctor may, at some point, use a medication that seems inferior, but that has not been used as often as the other; this situation corresponds to 'exploration'.

The name 'bandit' comes from fruit/slot machines that are commonly found in casinos (they are given many names depending on the country). These machines typically have one arm, with some probability of reward. The rational player should seek the machine that gives the best reward out of the k available machines. One explanation for the name 'bandit' is given by Lai and Robbins [**49**]:

> Ordinary slot machines with one arm are one-armed bandits, since in the long run they are as effective as human bandits in separating the victim from his money.

2.2.1. Formalisation of the bandit setting. The bandit setting is a large simplification of the reinforcement-learning one (Section 2.1.1): there is only one state, and the number of actions is typically finite.

Mathematically, for each patient n (i.e. for each *round*), the doctor takes a decision $x(n) \in \mathcal{A}$, where $\mathcal{A}$ is the set of actions (named *arms* for bandits). In this particular case, the reward the doctor gets, denoted as $r(n)$, depends on whether the patient survived or not. The outcome is stochastic: treating two patients with the same medicine does not always guarantee two identical results; therefore, $r(n)$ is a random variable. Arbitrarily, the reward might be defined as 1 for a survivor and 0 otherwise. Each medicine $a_i \in \mathcal{A}$ has its own probability distribution of rewards: in this case, it is a Bernoulli distribution, with a probability θ_i of survival. Therefore, the reward can be written as:

$$r(n) \sim \mathcal{B}(\theta_i) \quad \text{if} \quad x(n) = a_i. \tag{2.2.1}$$

The goal of the doctor (the agent) is to find the best medicine (the best arm to play), i.e. the one with the highest probability of survival (the highest average reward), denoted by $r^\star$. Their reasoning is formalised by the notion of *regret*. This metric compares the (total) reward the agent did actually get (i.e. how many patients survived, $\sum_{t=1}^{n} r(t)$ at round n) and the outcome had the doctor known *in advance* the best medicine, i.e. as if the doctor was an oracle with perfect knowledge about the environment. The regret after n rounds is formally defined as:

$$R(n) = \underbrace{n\,r^\star}_{\substack{\text{the optimum reward,}\\ n \text{ rounds}}} - \underbrace{\mathbb{E}\left\{\sum_{t=1}^{n} r(t)\right\}}_{\substack{\text{the reward the bandit}\\ \text{received}}} . \tag{2.2.2}$$

To get the lowest possible regret, the doctor (our agent) cannot continue using the same medicine (i.e. play the same arm) over and over again, unless they have proven that the other ones are not as effective.

The *expected* discrepancy between the optimum policy and the bandit is called the *gap*, for each round:

$$\Delta(n) = \mathbb{E}\{r^\star - r(n)\}\,. \tag{2.2.3}$$

The regret can be written as the sum of the gaps:

$$R(n) = \sum_{t=1}^{n} \Delta(t)\,. \tag{2.2.4}$$

The gap can also be defined for the ith arm, and corresponds to the average reward that is lost when playing a_i (whose average reward is r_i) instead of the optimum arm (whose average reward is $r^\star$), in expectation. This value is thus zero for the optimum arms and greater than zero for the others.

$$\Delta_i = \mathbb{E}\{r^\star - r_i\}\,. \tag{2.2.5}$$

FIGURE 2.2.1. A typical fruit machine. (*1899 "Liberty Bell" machine, manufactured by Charles Fey*, photo by Nazox, distributed under a Creative Commons BY-SA 3.0 license. Available online.)

A classic decomposition of the regret is based on the gap for each arm and the number of times it has been played up to round n, which is denoted by $T_i(n)$[2]:

$$R(n) = \sum_{i=1}^{k} \Delta_i \, \mathbb{E}\{T_i(n)\} \, . \tag{2.2.6}$$

A direct intuition of this decomposition is that arms with a low gap should be played more often than those with higher gaps (i.e. worse arms).

Typically, a harder bandit problem has a lower *minimum gap*, which is defined as:

$$\Delta_{\min} = \min_{\substack{i \in \{1,2 \dots k\}: \\ \Delta_i > 0}} \Delta_i. \tag{2.2.7}$$

If the minimum gap is very small, the bandit has to sample very often the optimum arm and the one corresponding to $\Delta_{\min}$ in order to be able to distinguish them. This intuition is formalised as the Lai-Robbins bound (Section 2.2.4). Similarly, the *maximum gap* is defined as:

$$\Delta_{\max} = \max_{i \in \{1,2 \dots k\}} \Delta_i. \tag{2.2.8}$$

2.2.2. Stochastic and adversarial settings. There are two major settings for k-armed bandits, depending on how rewards are chosen: stochastic and adversarial. The introductory example happened in the stochastic setting: before the bandit algorithm, the environment is set with probability distributions for each arm; the rewards the bandit gets are drawn from these distributions. In particular, they are not allowed to change over the course of the bandit execution.

On the other hand, in an adversarial setting, the bandit is playing against an opponent (hence the name). This scenario has been less studied in the literature, and it is only recently that researchers investigated the setting, starting in 1995 with [**50**] (while the field of stochastic bandits started in the 1930s [**48**]). Before the agent starts interacting with the environment, the antagonist chooses the reward vectors for the T rounds to come, with one component per arm that can be played. This *oblivious* competitor thus cannot adapt to the behaviour of the bandit. At each round, the bandit agent takes its decision, based on the previous rewards it was able to get; the reward is taken from the reward vector of the round. This setting is significantly harder than the stochastic case: for instance, a deterministic policy is guaranteed to suffer from a linear regret. Algorithms exploit different principles than in the stochastic case: there should no more be optimism in the face of uncertainty.

2.2.3. Applications. Apart from the historical and very classical application of drug testing, k-armed bandits are very useful in many cases, and not only in theory. For instance, these algorithms are used to replace A/B testing of websites, mobile applications, and many other computer applications [**51**].

In small-cell cellular networks like some deployments of 4G or 5G, the base station emits data on the beam level, i.e. for a small angle and a specific distance with respect to the antenna (whereas a cell corresponds to a large angle and a sizeable distance). Beams may interfere with those of other antennae: in this case, the transmission has a poor quality, and the packets must be sent again [**19**]. The retransmission may happen on the same beam or not: if both antennae stay on the same beam, they will interfere with each other again; moreover, if both switch beams at the same time, some

[2]The origins of this decomposition are not exactly clear [**47**]. One of its earliest use is in [**49**].

spectrum will be lost, while these antennae may interfere with others. This decision might be taken by a bandit.

2.2.4. Lai-Robbins bound. A fundamental result of k-armed-bandit theory is the *Lai-Robbins bound* [**49**]. It gives a lower bound on the regret of the best implementable bandit algorithm, a 'consistently good policy'. Without prior information on the bandits, such a policy cannot have a zero regret, or even a regret that is constant with the number of rounds.

Formally, a *consistently good policy* [**49**] is a bandit algorithm whose asymptotic regret $R(n)$ when $n \to +\infty$ is such that, for all reward distributions θ in the compact set Θ,

$$\mathbb{E}\{R(n)\} \in o(n^{\alpha}), \qquad \forall \alpha > 0. \tag{2.2.9}$$

Even though the regret highly depends on the choice of reward distribution θ, a consistently-good policy must respect this condition for all distributions in Θ.

Stating the Lai-Robbins bound requires the definition of the *Kullback-Leibler divergence* $\mathrm{kl}(p, q)$ between two probability distributions p and q [**52**]. $\mathrm{kl}(p, q)$ will be close to zero if p and q are highly similar probability distributions. This divergence is defined as follows, for two continuous probability distributions:

$$\mathrm{kl}(p, q) = \int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x. \tag{2.2.10}$$

For two discrete distributions, the integral is replaced by a sum:

$$\mathrm{kl}(p, q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}. \tag{2.2.11}$$

The particular case of Bernoulli distributions is very useful for bandit problems: in this case, arms only have two possible values for the reward, either 0 or 1. For Bernoulli distributions, denoting by p and q the win probability for each distribution and abusing these symbols to refer to the probability distributions, the Kullback-Leibler divergence is:

$$\mathrm{kl}(p, q) = p \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q}. \tag{2.2.12}$$

The Lai-Robbins bound is the following [**49**, Theorem 1]: for all distributions $\theta \in \Theta$ such that there is at least one arm i with $\Delta_i > 0$ (i.e. one suboptimum arm), a uniformly-good policy must have the following lower bound on the number of times each arm $j \in \{1, 2 \dots k\}$ is played:

$$\liminf_{n \to +\infty} \mathbb{E}\left\{ \frac{T_j(n)}{\log n} \right\} \geq \frac{1}{\mathrm{kl}(r_j, r^{\star})}. \tag{2.2.13}$$

In particular, if j is the optimum arm, the limit value is infinite. Therefore, in order to be optimum, a bandit algorithm must explore.

2.2.5. Efficient algorithms for stochastic bandits. The literature provides many algorithms for stochastic bandits, many of them having similar regret bounds (within a constant factor with respect to each other). Many algorithms are based on the notion of 'optimism in face of uncertainty' introduced in [**49**], like UCB-1 [**53**]. In practice, this methodology implies that the algorithm uses an *index* to choose the arm to play. This index is a function of the arm and has two components: the average reward for this arm $\hat{\theta}_i(n)$, and a 'confidence bonus' (or radius) $\hat{\rho}_i$. The algorithm is designed so that the true reward for the arm is contained with high probability in the interval $\left[\hat{\theta}_i + \hat{\rho}_i, \hat{\theta}_i - \hat{\rho}_i\right]$. Usually, the confidence bonus is a function of the *inverse* of the number of times

the arm has been played, so that an arm that has not been played often (i.e. its reward estimation is probably poor) has a higher confidence bonus. All the ingenuity in designing such bandit algorithms is in finding the right expression for the bonus so that exploration is balanced with respect to exploitation.

UCB-1 [**54**] has a confidence bonus tuned to minimise the regret while still having a very simple expression:

$$x(n) \in \operatorname*{arg\,max}_{i\in\{1,2\dots k\}} \overbrace{\underbrace{\hat{\theta}_i}_{\substack{\text{average}\\ \text{reward}\\ \text{of arm } i}} + \underbrace{\sqrt{\frac{2\,\log n}{T_i(n)}}}_{\text{confidence bonus}}}^{\text{index of arm } i} . \tag{2.2.14}$$

The algorithm is formally stated in Algorithm 1. This algorithm can be proved to have the following regret bound [**54**, Theorem 1]:

$$\mathbb{E}\{R(n)\} \leq 8 \sum_{i=1}^{k} \mathbb{1}_{\Delta_i\neq 0} \frac{\log n}{\Delta_i} + \left(1+\frac{\pi^2}{3}\right) \sum_{i=1}^{k} \Delta_i. \tag{2.2.15}$$

A simpler form is $R(n) \in \mathcal{O}(\log n)$.

KL-UCB [**55, 56, 57**] still belongs to the same family of algorithms, but has deeper roots in probability. It is based on the Kullback-Leibler divergence, a measure of distance between two probability distributions (already introduced in Section 2.2.4). The confidence interval for each reward is written in terms of the Kullback-Leibler divergence, with the same effect as in UCB-1: if the arm has been played often, the interval should be small. The arm is chosen as:

$$x(n) \in \operatorname*{arg\,max}_{i\in\{1,2\dots k\}} \left\{ q_i \,\middle|\, q_i \in \Theta \quad \text{and} \quad T_i(n)\ \mathrm{kl}\Big[\hat{\theta}_i(n)\,, q_i\Big] \leq \log n \right\}. \tag{2.2.16}$$

The algorithm is formally stated in Algorithm 2. KL-UCB can be proved to have the following regret bound [**55**]:

$$\limsup_{n\to+\infty} \frac{\mathbb{E}\{R(n)\}}{\log n} \leq \sum_{i=1}^{k} \mathbb{1}_{\Delta_i\neq 0} \frac{\Delta_i}{\mathrm{kl}(\theta_i, \theta^\star)}. \tag{2.2.17}$$

As UCB-1, it scales as $R(n) \in \mathcal{O}(\log n)$. The difference is in the constant factor between $\log n$ and the regret the algorithm exhibits: in some cases like Bernoulli rewards, this algorithm is optimum and reaches the Lai-Robbins lower bound, i.e. it is asymptotically optimum for these problems.

Thompson sampling is an older algorithm based on very different principles [**48**]. It uses Bayesian statistics to estimate the arm rewards [**58**]: each arm is assigned to a probability distribution to encode the 'degree of belief' about the value of the average reward. The *prior distribution* is the one the bandit starts with: after each round, the bandit gathers information about the rewards, the updated distribution is called the *posterior distribution*. For Bernoulli rewards, the beta distribution is used: both distributions are *conjugate*, meaning that updating the beta-posterior parameters based on new information (coming from a Bernoulli distribution) is easy. The beta distribution has two parameters: the number of successes $\hat{\theta}_i^+(n)$ and the number of failures $\hat{\theta}_i^-(n)$. For prediction, an estimated reward is drawn from the posterior distribution and the arm with the

best reward is chosen:

$$x(n) \in \operatorname*{arg\,max}_{i\in\{1,2\ldots k\}} t_i(n)\,, \qquad \text{where } t_i(n) \sim \beta\left[\hat{\theta}_i^+(n)+1, \hat{\theta}_i^-(n)+1\right]. \tag{2.2.18}$$

The algorithm is formally stated in Algorithm 3. It can be proved to have to following regret bound [**59**, Theorem 1]:

$$R(n) \leq (1+\varepsilon)\sum_{i=1}^{k} \mathbb{1}_{\Delta_i\neq 0}\Delta_i \frac{\log n + \log\log n}{\mathrm{kl}(\theta_i,\theta^\star)} + C(\varepsilon,\theta_1,\theta_2\ldots\theta_k) \tag{2.2.19}$$

for all values of $\varepsilon > 0$, where $C(\varepsilon,\theta_1,\theta_2\ldots\theta_k)$ is a constant that only depends on the chosen ε and on the average rewards of each arm. What is uncommon with Thompson sampling is that the analysis came after the algorithm: while the algorithm was developed in the 1930s [**48**], its analysis had to wait until the 2010s [**60, 61, 62, 59**].

UCB-1 and Thompson sampling are very easy to implement and are lightweight to execute. For UCB-1, an implementation only requires to store, for each arm, a counter of times this arm has been played and an average reward (for Bernoulli rewards, it suffices to count the successes): per round and per arm, it only requires a few floating-point operations to evaluate the index (an inverse, a logarithm, a product, and a square root). These operations can be performed in a few milliseconds on very cheap hardware like microcontrollers. Thompson sampling, as long as a conjugate pair of distributions is used, can be implemented very efficiently, but sampling a vector of estimated rewards might be cumbersome to implement. KL-UCB, however, requires a more complex optimisation algorithm to compute the index of each arm, such as a binary search, and therefore requires more computational power. Nevertheless, all these three algorithms can be implemented in polynomial time (the major variable being k, the number of arms).

2.2.6. Efficient algorithms for adversarial bandits. The principle of optimism in face of uncertainty is no more valid for designing adversarial-bandit algorithms: the reward of the arms no longer fluctuate around an unknown average, an empirical average over the previous rewards for a given arm has no reason to be close to the average reward of that arm with high probability. Instead, the algorithm must estimate the reward of the arms it did not decide to play. It might have a mechanism to directly perform this estimation (like EXP3) or delegate this task to 'experts' that decide the best solution to play in their opinion (EXP4).

EXP3 (***exp****onential-weight algorithm for* ***exp****loration and* ***exp****loitation*) [**63, 50**] keeps a weight w_i for each arm $a_i \in \mathcal{A}$. The arm to play is sampled using these weights. The algorithm uses a learning rate $\eta > 0$ to guide the update of the arms and to ensure that all arms have a sufficiently high probability of being played. This principle provides ample exploration, there is no need for mechanisms like ε-greedy or confidence bonus [**64**]. More precisely, the algorithm starts with all weights equal to 1: $w_i(0) = 1$ for each arm $a_i \in \mathcal{A}$. The probability distribution p over the arms is computed as:

$$p_i(n) = (1-\eta)\,\frac{w_i(t)}{\sum_{a_j\in\mathcal{A}} w_j(n)} + \frac{\eta}{k}. \tag{2.2.20}$$

Once the arm $x(n)$ is played, the bandit receives a reward $r(n)$. It only updates the weight of this arm (the others are left untouched):

$$w_{x(n)}(n+1) = w_{x(n)}(n) \times \exp\left(\eta\,\frac{r(n)}{k\,p_i(n)}\right). \tag{2.2.21}$$

Using this algorithm, the regret has the following bound with an appropriate choice of η (whose value depends on the number of rounds the bandit will be used, T) [**50**, Theorem 4.1], [**47**, Theorem 11.1]:

$$R(T) \leq 2\sqrt{T\,k\,\log k} \qquad \text{if} \qquad \eta = \sqrt{\frac{\log k}{T\,k}}. \tag{2.2.22}$$

The algorithm is formally stated in Algorithm 4.

EXP4 (***exp****onential-weight algorithm for* ***exp****loration and* ***exp****loitation with* ***exp****erts*) is a different beast [**53**]. Whereas EXP3 relies on the obtained rewards to take its next decision, EXP4 delegates this task to a set of experts. The experts are fed with the previous rewards, and generate an advice vector indicating which arm is supposed to be optimum for the next round. Then, instead of maintaining a probability distribution over the arms (like EXP3), EXP4 has a probability distribution p over the experts $e \in E$, to determine whether the experts are usually right about the best decision to take or not. EXP4 also uses a learning rate $\eta > 0$. These experts may give constant advice for an arm to play or do complex learning on previous rewards for each arm before advising anything. This probability distribution p modulates the decisions from each expert, each of them giving a probability distribution over the arms $\mathbf{E}_e(n)$. The probability over the arms is therefore:

$$q_i(n) = \sum_{e \in E} p_e(n)\; E_{e,i}(n)\,. \tag{2.2.23}$$

Initially, each expert is given the same probability: $p_e = |E|^{-1}$ for each expert $e \in E$. Once the reward $r(n)$ is received, the rewards for the other actions are estimated, for instance using:

$$\hat{r}_i(n) = \begin{cases} \frac{r(n)}{q_i(n)} & \text{if } x(n) = a_i \\ 0 & \text{otherwise} \end{cases} \qquad \text{so that} \qquad r(n) = \mathbb{E}\{\hat{r}(n)\}\,. \tag{2.2.24}$$

Then, the expected loss for each arm $a_i \in \mathcal{A}$ is:

$$g_i(n) = \sum_{a_i \in \mathcal{A}} e_i(n)\; \hat{r}_i(n)\,. \tag{2.2.25}$$

Finally, the weights are updated as:

$$p_e(n+1) = p_e(n) \times \exp\left(-\eta \sum_{t=1}^{n} g_i(t)\right). \tag{2.2.26}$$

With an appropriate choice of η that depends on the total number of rounds where the bandit will be used (T), this algorithm can be proved to have the following regret:

$$R(T) \leq \mathbb{E}\Big\{\sqrt{2\,T\,k\,\log|E^\star(T)|}\Big\} \qquad \text{if} \qquad \eta = \sqrt{\frac{2\,\log|E^\star(T)|}{T\,k}} \tag{2.2.27}$$

where $E^\star(T) = \sum_{n=1}^{T} \sum_{a_i \in \mathcal{A}} \max_{e \in E} E_{e,i}(n)$ is sum of the maximum expert probability for each bandit round. The algorithm is formally stated in Algorithm 5.

2.3. Contextual bandits

The basic k-armed bandit is only designed to work in stationary environments without side information. Nevertheless, not all situations fit this description. For instance, recommender systems provide users with new content they might appreciate. An online streaming platform has access to the content the user has already watched ('side information'). Algorithms for the k-armed-bandit

problem can only recommend new content to all users indistinctively, as the policy to select a new arm does not take the user into account. However, recommending a horror film to people that only watched romance pictures is probably a poor choice: the user will probably not watch this content, which yields a very low reward for the platform owner. As all users have very different tastes regarding recordings, the reward follows a highly nonstationary distribution.

Contextual bandits make use of this supplementary side information, in the hope of bringing some stationarity: if users are clustered (e.g., horror films, romance films), there is, intuitively, a higher chance of having a stationary reward distribution. For each group, a standard k-armed bandit can be used, using the stationarity hypothesis [**65**].

2.3.1. Applications. Recommender systems are a major application of contextual bandits: for personal news feeds [**66**], for music streams [**67**], for ad-format selection on webpages [**68**], personalised learning [**69**], etc. A similar application is active learning [**70**].

2.3.2. Definition of the setting. More formally, a contextual bandit operates as follows. Let $\mathcal{C}$ denote the set of contexts: it may be either finite (e.g., the context is the cluster to which the current user is assigned) or infinite (an element in a vector space). First, the environment chooses a context $c(n) \in \mathcal{C}$; this choice can be performed either stochastically or adversarially. Then, the bandit algorithm chooses the arm to play $x_{c(n)}(n)$. Finally, the bandit receives its reward $r(n)$, drawn from a probability distribution only parametrised by the current context $c(n)$ and the played arm $x(n)$.

The gap $\Delta(n)$ of a played arm $x(n)$ in the context $c(n) \in \mathcal{C}$ is defined as the difference in reward between what the bandit received and what the best context-dependent policy $r^\star_{c(n)}$ can achieve (i.e. the reward the best policy for context $c(n)$ would get):

$$\Delta(n) = r^\star_{c(n)} - r(n)\,. \tag{2.3.1}$$

The total expected regret allows a decomposition according to the states, supposing that there is a finite number of contexts:

$$\begin{aligned} R(n) &= \mathbb{E}\left\{ \sum_{c\in\mathcal{C}} \mathbb{1}_{c(n)=c}\, \Delta(n) \right\} \\ &= \sum_{c\in\mathcal{C}} \sum_{i=1}^{k} \Delta_{i,c}\, \mathbb{E}\big\{ \mathbb{1}_{c(n)=c}\, T_{i,c}(n) \big\}\,. \end{aligned}$$

This decomposition highlights the dependence of the total regret on the frequency of each context.

2.3.3. Relation with adversarial bandits. As the context can be chosen by an opponent, contextual bandits are clearly related to adversarial bandits.

Using one instance of EXP3 per context, the regret can be proved to be [**47**]:

$$R(T) \leq 2\,\sqrt{T\,k\;|\mathcal{C}|\;\log k} \qquad \text{if} \qquad \eta = \sqrt{\frac{\log k}{T\,k}}. \tag{2.3.2}$$

However, the bound for EXP4 (where each expert now takes as input the context) is still [**53**]:

$$R(T) \leq \mathbb{E}\Big\{ \sqrt{2\,T\,k\,\log|E^\star(T)|} \Big\} \qquad \text{if} \qquad \eta = \sqrt{\frac{2\,\log|E^\star(T)|}{T\,k}}. \tag{2.3.3}$$

where $E^\star(T) = \sum_{n=1}^{T} \sum_{a_i \in \mathcal{A}} \max_{e \in E} E_{e,i}(n)$. One can prove that $E^\star(T) \leq T \min\{k, |E|\}$ [**47**]. This bound therefore mostly depends on the agreement between experts: if all experts always agree, $E^\star(T) = T$, which is the lowest possible value [**47**]. Therefore, more experts can be added without having a detrimental effect on the regret.

The bound for EXP3 increases faster than that of EXP4 when k increases: when the number of possible decisions (these can be items to recommend) is large, EXP4 has a significant advantage over EXP3 if the experts tend to express similar opinions. In the worst case, even if experts always disagree, EXP4 has a performance slightly better than that of EXP3, but only within a constant factor.

REMARK 1. $E^\star(T)$ is close to impossible to exactly determine beforehand. A typical solution is to have a learning rate η_n that depends on the round and uses the currently estimated $E^\star(n)$, as this can be computed easily for previous rounds. In this case, though, the learning rate should be chosen as:

$$\eta_n = \sqrt{\frac{2 \log |E^\star(n-1)|}{T\, k}}. \tag{2.3.4}$$

The regret bound is slightly worse:

$$R(T) \leq \mathbb{E}\left\{2\sqrt{T\, k\, \log |E^\star(T)|}\right\}. \tag{2.3.5}$$

2.3.4. Linear contextual bandits. The simple methodology of having one bandit per context is limited in its applicability: this approach only works when the number of contexts is finite. Moreover, if the number of contexts is too large, learning might be slow: information from one context is never shared with other contexts. This methodology cannot be improved if all contexts are very different from one another. Outside this case, the rewards in different contexts are somewhat linked. This hypothesis makes sense in practice: to recommend films, a very good action film might appeal to both horror- and romance-film enthusiasts. A similar reasoning also applies to online ads [**71**].

A common assumption is that of *linearity* when the context is a vector space (each context $\mathbf{c} \in \mathcal{C}$ is a vector of $\mathbb{R}^d$, for instance) [**72**]. Mathematically, this hypothesis implies that the expected reward of playing an arm $a_i \in \mathcal{A}$ in a context $\mathbf{c} \in \mathcal{C}$, which is denoted by $r_{i,\mathbf{c}}$, is a linear function of this context $\mathbf{c}$:

$$r_{i,\mathbf{c}} = \mathbf{c}^T\, \boldsymbol{\theta}_i, \tag{2.3.6}$$

where $\boldsymbol{\theta}_i$ is a fixed vector (unknown to the learner) that depends on the played arm $a_i \in \mathcal{A}$. The contextual-bandit problem then reduces to estimate accurately these vectors $\boldsymbol{\theta}_i$.

LinUCB is one such algorithm [**73, 66**]. Whereas the k-armed UCB builds a confidence interval for the average reward of each arm r_i, LinUCB does so for the whole vector $\boldsymbol{\theta}_i$. The confidence intervals are thus replaced by confidence regions $C_i(n) \subset \Theta$, where Θ is the set of all possible reward vectors $\boldsymbol{\theta}$. These confidence regions $C_i(n)$ are built so that the probability that $\boldsymbol{\theta}_i \in C_i(n)$ is high. The chosen arm is then:

$$x(n) \in \arg\max_{\mathbf{a}_i \in \mathcal{A}} \left\{ \max_{\hat{\boldsymbol{\theta}}_i \in C_i(n)} \mathbf{c}(n)^T\, \hat{\boldsymbol{\theta}}_i \right\}. \tag{2.3.7}$$

The most complex part of the algorithm is updating these confidence regions $C_i(n)$ based on the newly acquired knowledge. One way of updating them is explained in Section 2.4.3. With probability $\delta \in (0,1)$, LinUCB is guaranteed to have a regret bounded by [**74**, Theorem 1]

$$\mathbb{E}\{R(n)\} \leq C \sqrt{T\, d\, \log^3 \frac{k\, T\, \log T}{\delta}} \tag{2.3.8}$$

where $C > 0$ is a universal constant.

2.4. Structured bandits

So far, we only considered bandits with a finite and quite low number of arms. However, this situation is not very generic. For instance, if the bandit is used to determine which path in a network a packet should follow, the number of arms would be the number of paths. However, if two paths are not disjoint, having information about the time the packet needs to travel on one path might bring some information on the other path. With k-armed bandits, this cross-information cannot be exploited [**75**]. These bandits are structured because the rewards are not independent from each other: the rewards are composed of basic elements (like an edge in a graph), and only these basic elements are considered to be independent.

For these structured bandits, algorithms based on the same basic 'optimism in the face of uncertainty' principle do not work as well as for k-armed bandits (Section 2.2). Indeed, it has been proved that these algorithms all have special cases where they cannot be optimum [**76**, Theorem 1], [**?**, Proposition 1]. Other techniques must therefore be pursued in the quest for the optimum regret bound, like those based on information theory.

2.4.1. Common structures. A very generic structure for bandit problems, introduced in [**77**], is as follows. The action space $\mathcal{A}$ is finite, and contains $k \in \mathbb{N}_0$ arms. Each arm $a_i \in \mathcal{A}$ is associated to an unknown value, $\theta(a_i)$. Each time the bandit policy plays an arm a_i, it gets a reward whose average is $\theta(a_i)$, with some unknown distribution.

2.4.1.1. *Linear bandits.* In the linear-bandit setting, actions are considered to be vectors ($\mathcal{A} \subset \mathbb{R}^d$ for some dimension d), even though the action space is still finite (i.e. $|\mathcal{A}| = k \in \mathbb{N}_0$, like in the standard k-armed-bandit setting). The reward is assumed to be a linear function of the action vector.

The reward vector $\boldsymbol{\theta}(n)$ follows some probability distribution with means $\boldsymbol{\theta}$. Moreover, the individual rewards $\theta_i(n)$ are assumed to be independently and identically distributed. The optimum arm to play is denoted by $\mathbf{x}^\star \in \mathcal{A}$, and its reward is $r^\star = \mathbf{x}^{\star T}\, \boldsymbol{\theta}$. The goal of the bandit is to estimate these means [**78**]. Therefore, the reward received is written as:

$$r(n) = \mathbf{x}(n)^T\ \boldsymbol{\theta}(n)\,. \tag{2.4.1}$$

The regret is still defined in the same way as for k-armed bandits:

$$R(n) = \mathbb{E}\left\{\sum_{t=1}^{n} \Delta(n)\right\}, \qquad \text{with} \qquad \Delta(n) = r^\star - r(n)\,. \tag{2.4.2}$$

Most algorithms for linear bandits rely on the same principles as UCB, i.e. find high-probability confidence regions for the weights of $\boldsymbol{\theta}(n)$, use the upper bound of the confidence interval of the reward as the estimated reward, and play the arm that maximises this estimated upper bound ('optimism in face of uncertainty' principle) [**79, 72, 73**]. Other algorithms use statistical tests as a part of their derivation [**78, 80**].

The regret bounds for these algorithms typically depend on the dimension of $\mathcal{A}$, not its number of elements. For instance, if all arms in $\mathcal{A}$ are described by vectors of $\{0,1\}^d$, the maximum number of arms is 2^d: by having a dependency on d instead of k, it is possible to replace an exponential by a linear factor in the regret bound. However, typically, the computational complexity still depends linearly on k.

2.4.1.2. *Combinatorial linear bandits.* A special case of linear bandits is to consider that the action space $\mathcal{A}$ itself has a strong structure. In combinatorial bandits, it is considered to be described by a combinatorial set $\mathcal{X}$ **[81, 79]**. This implies that $\mathcal{A} = \mathcal{X} \subset \{0,1\}^d$ for some dimension d (unlike linear bandits, defined with $\mathcal{A} \subset \mathbb{R}^d$). The size of $\mathcal{A}$ can be large: unlike linear bandits, the number of arms k is not assumed to be small with respect to 2^d.

The optimum reward $r^\star$ is related to the optimum solution with the weights corresponding to the average rewards $\boldsymbol{\theta}$:

$$r^\star = \mathbf{x}^{\star T}\,\boldsymbol{\theta}, \qquad \mathbf{x}^\star \in \arg\max_{\mathbf{x}\in\mathcal{X}} \mathbf{x}^T\,\boldsymbol{\theta}. \tag{2.4.3}$$

Similarly, the gap and the regret are defined based on the average rewards:

$$\Delta(n) = \boldsymbol{\theta}^T\,\left[\mathbf{x}^\star - \mathbf{x}(n)\right], \qquad R(T) = \sum_{t=1}^{T} \Delta(t)\,. \tag{2.4.4}$$

This kind of bandits is studied in detail in Section 2.5.

2.4.1.3. *Lipschitz bandits.* Lipschitz bandits are defined on a vector space $\mathcal{A}$ of dimension d, for instance $\mathbb{R}^d$. Similarly to linear bandits, each arm k provides a stochastic reward with mean θ_k. The Lipschitz hypothesis is made between the average rewards and the distance between the arms in $\mathcal{X}$: for any two arms $\mathbf{x}_k$ and $\mathbf{x}_l$ in $\mathcal{A}$,

$$|\theta_k - \theta_l| \leq L\,\|\mathbf{x}_k - \mathbf{x}_l\|\,, \tag{2.4.5}$$

where the bandit knows the constant $L > 0$. This setting was introduced in **[82]**. L is a Lipschitz constant of the function f defined as $\theta_k = f(\mathbf{x}_k)$.

The problem is harder if the horizon T (i.e. the number of rounds where the bandit policy is used) is not known. In the continuous case, zooming algorithms can be used **[83]**. In the discrete case, some algorithms are based on the optimism principle, like UCBC **[84]**, but unrelated techniques like OSLB can be used **[85]**; OSLB is a restricted version of OSSB **[80]**. Having to know a good Lipschitz constant can be limiting, as these constants cannot be easily estimated for all practical problems. **[86]** proposes a two-phase mechanism for continuous Lipschitz bandits: first, a pure-exploration stage provides a crude approximation of L (the number of rounds of this step is a parameter of the algorithm) and performs the discretisation of the action space; second, an exploration-exploitation period uses a classic k-armed-bandit algorithm over the discretised space.

2.4.1.4. *Convex bandits.* Convex bandits are highly similar to Lipschitz ones, except that the rewards are supposed to follow a noisy convex function of the played arm $\mathbf{x}(n) \in \mathcal{A}$ **[87]**. The reward is supposed to have the following structure, with f being a convex function:

$$r(n) = f[\mathbf{x}(n)] + \eta(n)\,, \tag{2.4.6}$$

where $\eta(n)$ is a zero-mean noise term. Convex bandits are neither a special case of Lipschitz bandits nor a superclass, as a convex function does not necessarily have a Lipschitz constant, and Lipschitz functions are not necessarily convex. However, better convergence can be ensured if f is both convex and Lipschitz.

Algorithms in this class may be based on a 'pyramidal construction' [**87**] to approximate the noisy evaluations of f and determine if the (noisy) gradient at an arm $\mathbf{x}$ should be followed to minimise the regret, or if the opposite direction must be followed.

2.4.1.5. *Unimodal bandits.* Unimodal bandits continue in the same vein, but with a very different structure for f: a function defined on $\mathcal{A} = [0, 1]$ is said to be unimodal if it has one maximum at $x^\star$, is increasing on $[0, x^\star]$, and is decreasing on $[x^\star, 1]$ [**88**].

With this very peculiar structure, simple algorithms can be applied. For instance, LSE [**89**] works iteratively, by keeping an interval for $x^\star$, and sampling within this interval to reduce its size in the same way as the golden-search method. The classical UCB algorithm can also be extended to this case [**90**]. An optimum strategy is OSUB [**90**], an instance of the more generic OSSB. Thompson sampling (Section 2.2.5) can also be adapted to this setting, and the obtained algorithm is optimum [**91**].

2.4.2. Graves-Lai bound. Similarly to the Lai-Robbins bound for k-armed bandits (Section 2.2.4), implementable structured-bandit algorithms have a lower bound in terms of regret. This result is obtained through controlled Markov chains [**92**], of which structured bandits are a particular case.

For a consistently good policy, for any distribution of rewards $\boldsymbol{\theta} \in \Theta$ (where Θ is a compact set of reward distributions) such that the optimum arm is $\mathbf{x}^\star(\boldsymbol{\theta})$, this policy must be such that [**80, 76**]:

$$\liminf_{n \to +\infty} \frac{R(n)}{\log n} \geq C(\boldsymbol{\theta}) \tag{2.4.7}$$

where $C(\boldsymbol{\theta})$ is the value of the following optimisation problem:

$$\begin{array}{ll} \min & \sum_{\mathbf{x} \in \mathcal{A}} \eta_{\mathbf{x}} \, \Delta_{\mathbf{x}} \\ \text{s.t.} & \sum_{\mathbf{x} \in \mathcal{A}} \eta_{\mathbf{x}} \, \mathrm{kl}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{x}) \geq 1 \quad \forall \boldsymbol{\lambda} \in \Lambda(\boldsymbol{\theta}) \\ & \eta_{\mathbf{x}} \geq 0. \end{array} \tag{2.4.8}$$

$\mathrm{kl}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{x})$ is the Kullback-Leibler divergence between the distributions of the reward of $\mathbf{x} \in \mathcal{X}$ if the rewards follow the distributions $\boldsymbol{\theta}$ and $\boldsymbol{\lambda}$. In particular, if the rewards are linear in $\mathbf{x}$ and in the parameters,

$$\mathrm{kl}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{x}) = \mathrm{kl}\left[\boldsymbol{\theta}^T \mathbf{x}, \boldsymbol{\lambda}^T \mathbf{x}\right]. \tag{2.4.9}$$

$\Lambda(\boldsymbol{\theta})$ is the set of *confusing reward distributions*, i.e. those that are close to $\boldsymbol{\theta}$, but such that only sampling the optimum solution $\mathbf{x}^\star(\boldsymbol{\theta})$ for the parameters $\boldsymbol{\theta}$ could not allow to distinguish $\boldsymbol{\theta}$ from $\boldsymbol{\lambda}$. To be able to distinguish one distribution from the other, suboptimum arms must be played; this condition is required to ensure that the best solution is identified with high probability (otherwise, the agent would still play other arms to prove that $\mathbf{x}^\star(\boldsymbol{\theta})$ is optimum). Formally, $\Lambda(\boldsymbol{\theta})$ is defined as:

$$\Lambda(\boldsymbol{\theta}) = \left\{ \boldsymbol{\lambda} \in \Theta \,\middle|\, \mathrm{kl}[\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{x}^\star(\boldsymbol{\theta})] = 0, \quad \mathbf{x}^\star(\boldsymbol{\theta}) \neq \mathbf{x}^\star(\boldsymbol{\lambda}) \right\}. \tag{2.4.10}$$

This bound generalises the one of Lai-Robbins for k-armed bandits (Section 2.2.4).

The optimisation problem $C(\boldsymbol{\theta})$ can be interpreted in bandit terms. The variable $\eta_{\mathbf{x}}$ indicates how often the arm $\mathbf{x}$ should be played (even though the $\eta_{\mathbf{x}}$ are not a probability distribution over the solutions $\mathbf{x}$): over n rounds, the arm $\mathbf{x}$ is played $\lceil \eta_{\mathbf{x}} \log n \rceil$ times. The objective function, $\sum_{\mathbf{x} \in \mathcal{A}} \eta_{\mathbf{x}} \, \Delta_{\mathbf{x}}$, is a regret. The constraint $\sum_{\mathbf{x} \in \mathcal{A}} \eta_{\mathbf{x}} \, \mathrm{kl}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{x}) \geq 1$ imposes that the bandit performs enough exploration to ensure that it can distinguish both distributions.

This formulation is generic and applies to all kinds of structured bandits. Special cases were found before this formulation, for instance in the linear case **[76]**|**[90]**. Most importantly, these specific formulations have a polynomial size, unlike the generic one presented in this section: it is semi-infinite, with a finite number of variables (one per arm $\mathbf{x} \in \mathcal{X}$) and an infinite number of constraints (one per probability distribution in $\Lambda(\boldsymbol{\theta})$, which is a compact set).

2.4.3. Efficient algorithms for linear bandits. A major representative of the 'optimism in the face of uncertainty' family of algorithms **[49]** (to which UCB pertains, Section (2.2.5)) is LinUCB **[72]** (we follow the presentation of **[47]**). Most of the work of this algorithm consists of estimating the reward vector $\boldsymbol{\theta}$ and to define a sensible confidence set $\mathcal{C}(n)$. The reward estimation is performed as a regularised least-square problem based on all the previous played arms (with the regularisation parameter $\lambda > 0$):

$$\hat{\boldsymbol{\theta}}(n) \in \underset{\hat{\boldsymbol{\theta}} \in \mathbb{R}^d}{\arg\min} \sum_{t=1}^{n} \left[r(n) - \hat{\boldsymbol{\theta}}^T \mathbf{x}(t) \right]^2 + \lambda \left\| \hat{\boldsymbol{\theta}} \right\|_2^2. \tag{2.4.11}$$

This can be computed in time $\mathcal{O}\big(n\, d^3\big)$. The confidence set $\mathcal{C}(n)$ can be defined as centred on $\boldsymbol{\theta}(n-1)$:

$$\mathcal{C}(n) = \left\{ \hat{\boldsymbol{\theta}} \in \mathbb{R}^d \;\middle|\; \left\| \hat{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}(n-1) \right\|_2^2 \leq \underbrace{d + 2\sqrt{d\, \log \delta^{-1}} + 2\, \log \delta^{-1}}_{\beta} \right\}. \tag{2.4.12}$$

This definition ensures that the true vector $\boldsymbol{\theta}$ is in $\mathcal{C}(n)$ with probability $1 - \delta$ **[93]**. Finding the solution to play amounts to solving the following optimisation program:

$$\max_{\substack{\mathbf{a} \in \mathcal{A} \\ \hat{\boldsymbol{\theta}} \in \mathcal{C}(n)}} \hat{\boldsymbol{\theta}}^T \mathbf{a} \tag{2.4.13}$$

Written for a specific arm $\mathbf{a}_i \in \mathcal{A}$, it corresponds to a convex optimisation program, and it can be solved in time $\mathcal{O}\big(d^3\big)$. The complete algorithm has the following regret bound, which depends on the regularisation parameter λ:

$$\mathbb{P}\left\{ R(n) \leq \sqrt{8\, d\, n\, \beta\, \log \frac{d\,\lambda + n\, L^2}{d\,\lambda}} \right\} \geq 1 - \delta \tag{2.4.14}$$

with $L = \max_{\mathbf{x} \in \mathcal{A}} \|\mathbf{x}\|$. Usually, δ is chosen as $1/n$. This bound does not depend on the number of actions, but only on the number of dimensions of the feature vectors. This algorithm has a complexity $\mathcal{O}\big[d^3\ (k+n)\big]$. The algorithm is formally stated in Algorithm 6.

$$\mathbf{x}(n) \in \underset{\mathbf{x} \in \mathcal{A}}{\arg\max} \left\{ \begin{array}{lc} \max & \hat{\boldsymbol{\theta}}^T \mathbf{x} \\ \text{s.t.} & \left\| \hat{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}(n-1) \right\|_2^2 \leq \beta \\ & \hat{\boldsymbol{\theta}} \in \Theta \end{array} \right\}. \tag{2.4.15}$$

2.4.4. Efficient and generic algorithms. OSSB **[80, 94]** is based on the Graves-Lai bound (Section 2.4.2): actually, this algorithm is a direct application of the bound.

If the average rewards for each arm $\boldsymbol{\theta}$ were known, it would be sufficient to solve the optimisation program behind $C(\boldsymbol{\theta})$ and to play the arms accordingly: the arm $\mathbf{x} \in \mathcal{A}$ should be played $\eta_{\mathbf{x}} \log T$ times over T rounds. Once this exploration phase is performed, only the best arm $\mathbf{x}^\star$ should still be played. This mandates a minimum level of exploration.

However, in practice, $\boldsymbol{\theta}$ is not available: it can only be estimated. Let $T_{\mathbf{x}}(n)$ be the number of times the arm $\mathbf{x}$ has been played up to round n:

$$T_{\mathbf{x}}(n) = \sum_{t=1}^{n} \mathbb{1}_{\mathbf{x}(n)=\mathbf{x}} \tag{2.4.16}$$

and $\hat{\theta}_{\mathbf{x}}(n)$ the estimated reward of the arm $\mathbf{x}$ at round n:

$$\hat{\theta}_{\mathbf{x}}(n) = \frac{\sum_{t=1}^{n} \mathbb{1}_{\mathbf{x}(n)=\mathbf{x}}\, r(n)}{T_{\mathbf{x}}(n)}. \tag{2.4.17}$$

OSSB uses $\hat{\boldsymbol{\theta}}(n)$ as an estimator for $\boldsymbol{\theta}$ and explores accordingly to $C\left(\hat{\boldsymbol{\theta}}\right)$, this optimisation program being solved to exact optimality (no approximation factor or term). This technique is intuitively ensured to work if $\hat{\boldsymbol{\theta}}(n)$ converges towards $\boldsymbol{\theta}$ sufficiently fast.

In practice, OSSB works in three phases; its exploration is controlled by two parameters: $\gamma > 0$ and $\varepsilon > 0$. At round n:

exploitation: if all arms have been sufficiently explored, i.e. if

$$\eta_{\mathbf{x}}(n)\ (1+\gamma)\ \log n \leq T_{\mathbf{x}}(n)\,, \qquad \forall \mathbf{x} \in \mathcal{A}, \tag{2.4.18}$$

then the arm with the higher empirical reward is played: $\mathbf{x}^\star\left[\hat{\boldsymbol{\theta}}(n)\right]$ is very likely to be equal to $\mathbf{x}^\star\left(\hat{\boldsymbol{\theta}}\right)$.

estimation: let $\underline{\mathbf{x}}(n) \in \arg\min_{\mathbf{x}\in\mathcal{A}} T_{\mathbf{x}}(n)$ be the least played arm. If $T_{\underline{\mathbf{x}}(n)}(n) \leq \varepsilon\, s(n)$ where $s(n)$ is the number of rounds where the bandit has *not* entered the exploitation phase, then play $\underline{\mathbf{x}}(n)$.

exploration: play $\overline{\mathbf{x}}(n) \in \arg\min_{\mathbf{x}\in\mathcal{A}} T_{\mathbf{x}}(n) \,\big/\, \eta_{\mathbf{x}}(n)$, the arm that is furthest away from the goal of playing each arm approximately $\eta_{\mathbf{x}} \log n$ times.

The algorithm is formally stated in Algorithm 7.

2.5. Combinatorial bandits

Combinatorial bandits are a special case of linear bandits, but with a stronger structure behind the arm set $\mathcal{A}$. An arm is a combination of several unit decisions. For instance, the bandit can be used to determine a path for a packet in a computer network: this path is decomposed in a series of edges to follow, each of them being a unit decision. Each arm is a solution to some combinatorial problem (the shortest-path problem, to continue this example). Let $\mathcal{X}$ denote the set of solutions to this combinatorial problem. This set is fully known to the bandit, which means that $\mathcal{A} = \mathcal{X}$.

Optimising a linear function on $\mathcal{X}$ is not always simple. For instance, it might be the travelling salesperson problem (TSP), a well-known $\mathcal{NP}$-hard problem **[95]**. Usually, bandit algorithms using the 'optimism in the face of uncertainty' principle require optimising a *nonlinear* objective function (typically, the sum of an average reward and a standard deviation: in this case, the objective is a concave function). Even though efficient algorithms are known for many combinatorial problems in the linear case, they do not always generalise to the nonlinear case, even if it is concave. This indicates that computationally efficient algorithms for combinatorial bandits must take into account the intrinsic structure of the combinatorial set $\mathcal{X}$.

We consider the linear case of combinatorial bandits, where the reward is a (random) linear function of the arm $\mathbf{x}(n)$:

$$r(n) = \mathbf{x}^T(n)\ \boldsymbol{\theta}(n)\,, \qquad \text{where} \qquad \mathbb{E}\{\boldsymbol{\theta}(n)\} = \boldsymbol{\theta}. \tag{2.5.1}$$

Our bandit gets semibandit feedback, i.e. a vector giving the reward for each subarm that is played:

$$\mathbf{y}(n) = [x_1(n)\ \theta_1(n)\,,\ x_2(n)\ \theta_2(n) \dots\ x_d(n)\ \theta_d(n)]\,. \tag{2.5.2}$$

It will be useful to have the maximum number of elements in a solution, m:

$$m = \max_{\mathbf{x}\in\mathcal{X}} \sum_{i=1}^{d} x_i \leq d. \tag{2.5.3}$$

2.5.1. Applications. Combinatorial bandits generalise the individual online combinatorial problems (like the online shortest-path or the online matching problems) with a single algorithmic framework. This allows for a generic study of regret and convergence properties, for instance.

The *online matching problem* corresponds to the case where the combinatorial set $\mathcal{X}$ is the set of matchings. This technique has been used for more than a decade by Google AdWords (now Google Ads) [**96**], used to monetise services like Google Search or Google Maps. Online advertisement is ruled by auctions: several companies want to display ads for specific keywords and provide a daily budget; each time a user performs a query with a relevant ads, all the corresponding businesses enter in an auction. The enterprise only pays for the ad when a user clicks on it, and this is the only source of uncertainty in the model. Each query must be assigned to some bidder, i.e. an advertising customer, while the global objective is to maximise Google's revenues. Nevertheless, not all bidders have the same interest in all keywords, and are not ready to pay the same price for a given query (a pharmaceutical company is probably not interested in advertising on keywords like 'resin figurines', and would not be ready to pay anything to appear in the search results of such a query). This auction is modelled as a bipartite matching: queries are on one side, bidders on the other, and edges indicate an interest in advertising for a given keyword. In practice, the problem is slightly more complex, as some keywords are often requested and some bidders only advertise on a few keywords, so that the mapping must be balanced [**96, 97**].

Network routing is an application of the *online shortest-path problem*. The basic goal of network routing is to ensure that packets can cross a network as fast as possible, from a user point of view (i.e. minimise latency and maximise throughput). However, depending on the current load of the network, a path that is usually very good may become poor, because of network congestion. The online shortest-path problem involves measuring parameters of previously sent packets to take better decisions for the next ones [**75**].

Another application of the online shortest-path problem is that of Blotto games. A Blotto game is a resource allocation problem: the two players must simultaneously distribute a limited amount of resources (military troops) on battlefields; on each operation theatre, the player who invested more than the other wins the battle. Finding a strategy is equivalent to optimising a path in a 'layered graph': each path in this graph corresponds to one action; each layer in the graph corresponds to a battlefield, and there are as many nodes in a given layer as there are possible troop allocations for this battlefield (based on the already allocated troops in the previous layers) [**98**].

Combinatorial bandits can also be used to create computer-driven agents, thereby improving most existing script-based techniques like behaviour trees [**99**]. The actions that this agent should played are decided based on the exploration of a Monte-Carlo tree (an algorithm call MCTS, *Monte-Carlo tree search* [**100**]). In this tree, each edge corresponds to an action sequentially played, the

nodes encode the expected reward obtained by playing this action. Good strategies for exploring this tree are very important for the performance of the agent: exploring a large part of the tree ensures good actions, but at a prohibitive computational cost. The problem of exploring this tree can be formalised as a k-armed bandit problem (Section 2.2), where each outgoing branch out of a given node corresponds to an arm [**101**]. When complex actions are allowed in the environment, i.e. actions that can be decomposed (e.g., press two buttons at the same time), combinatorial bandits can be used to decide the next action to play. In this case, the decisions to take at each node of the Monte-Carlo tree to continue exploration are a combination of subarms, hence the combinatorial structure [**102, 103, 104**].

2.5.2. Semibandit and full-bandit feedbacks. Combinatorial bandits exist in three flavours: with full-bandit, semibandit, and full-information feedback.

- In a **full-bandit** setting, the agent only gets the reward for the arm that is played. For instance, with a linear bandit playing the arm $\mathbf{x}(n)$, the only feedback is $r(n) = \boldsymbol{\theta}(n)^T \mathbf{x}(n)$, where $\boldsymbol{\theta}(n)$ are the random weights drawn for round n such that $\mathbb{E}\{\boldsymbol{\theta}(n)\} = \boldsymbol{\theta}$. Learning in this case is more complex than in the two following situations: when playing an arm $\mathbf{x} \in \mathcal{A}$, the bandit does not know which part of the arm was rewarded.
- With **semibandit** feedback, the agent has the information for each element that is being played. More specifically, the feedback is given to the bandit as a vector $\mathbf{y}(n)$ giving access to the individual components of $\boldsymbol{\theta}(n)$, but only if these subarms are played:

$$\mathbf{y}(n) = [x_1(n)\ \theta_1(n)\,,\ x_2(n)\ \theta_2(n) \dots\ x_d(n)\ \theta_d(n)]\,. \tag{2.5.4}$$

  With this kind of feedback, the bandit knows which parts of the arm were successful, and this helps to take better decisions in the future.
- **Full information** corresponds to the case where the bandit gets the randomly drawn reward for each subarm at each round: the feedback is the full vector $\boldsymbol{\theta}(n)$. In this case, there is no more an exploration-exploitation trade-off, as any decision that is taken reveals information on all possible actions.

The two first scenarios are the most commonly found in practice. Full-bandit feedback is the scarcest possible input for bandit problems. For instance, when using bandit algorithms for online marketing, the decisions taken by the agent might be the different channels through which ads and other marketing materials are sent: if the client eventually buys something, the specific part that convinced the user to buy is not always obvious (was it the last email that redirected them to the Web store or rather the catalogue?). On the contrary, when a user clicks on a specific ad on a Web page, the ad server knows which one was clicked among all those that were displayed: this is a good example of semibandit feedback.

In each setting, different algorithms must be used. In the semibandit or full-information settings, the bandit can store an estimation of the average reward for each subarm θ_i directly, whereas full-bandit feedback requires an estimation of the vector $\boldsymbol{\theta}$ at once (for instance, using linear regression, as in Section 2.4.3).

2.5.3. Generic, computationally efficient algorithms. The paradigm of combinatorial bandits has attracted many researchers over time, and several polynomial-time algorithms have been designed. However, none of them has the optimum regret bound for this problem. Some of them can be optimum for special cases like matroids. (In Chapters 3 and 4, we propose two such algorithms.)

2.5.3.1. *Thompson sampling for combinatorial bandits.* Thompson sampling (Section 2.2.5) can be extended to combinatorial problems, with one estimated probability distribution per subarm [**105**]. When the rewards follow Bernoulli distributions, the subarm posterior distributions can be chosen to be Beta, the conjugate distribution to Bernoulli. At each round, the subarm distributions are chosen to be:

$$t_i(n) \sim \beta\left[\sum_{\mathbf{x}\in\mathcal{X}} x_i\, T_{\mathbf{x}}(n)\; \overline{r}_i(n) + 1, \quad \sum_{\mathbf{x}\in\mathcal{X}} x_i\, T_{\mathbf{x}}(n)\; [1 - \overline{r}_i(n)] + 1\right], \tag{2.5.5}$$

where $\overline{r}_i(n)$ is the average reward for the subarm i at round n ($\mathbf{y}$ being the reward vector for semibandits, as in Section 2.5.2):

$$\overline{r}_i(n) = \frac{\sum_{t=1}^{n} x_i\, y_i(n)}{\sum_{t=1}^{n} x_i}. \tag{2.5.6}$$

In order to choose an arm to play, the algorithm first samples the posterior probabilities to get the vector $\mathbf{t}(n)$ and then computes the optimum solution for these rewards:

$$\mathbf{x}(n) \in \arg\max_{\mathbf{x}\in\mathcal{X}} \mathbf{t}^T(n)\; \mathbf{x}. \tag{2.5.7}$$

The algorithm is formally stated in Algorithm 8.

Per round, Thompson sampling therefore only performs a very limited number of operations, apart from optimising a linear function over the combinatorial set $\mathcal{X}$: $\mathcal{O}(d)$ operations to update the posterior distributions (in the case of the Beta prior for Bernoulli rewards) and $\mathcal{O}(d)$ sampling operations (they can be performed in constant time). If a polynomial-time algorithms is known to optimise linear functions on $\mathcal{X}$, then Thompson sampling can be implemented in polynomial time.

However, this simple and efficient algorithm does not have a good regret bound [**106**, Theorem 4]:

$$R(n) \leq \sum_{i=1}^{d} \max_{\substack{\mathbf{x}\in\mathcal{X}:\\ x_i=1}} \frac{4\sqrt{|\mathcal{X}|}\,\log(d\,n)}{\Delta_{\mathbf{x}} - (k^\star+3)\;\varepsilon} + 9\,\alpha_3\,\frac{\Delta_{\max}}{\varepsilon^2}\left(\frac{3}{\varepsilon}+1\right)^{k^\star} + \Delta_{\max}\left(d + d\,m + \frac{d\,m^2}{\varepsilon^2} + 1\right) \tag{2.5.8}$$

where $k^\star$ is the minimum size of an optimum solution

$$k^\star = \min_{\substack{\mathbf{x}\in\mathcal{X}:\\ \Delta_{\mathbf{x}}=0}} \sum_{i=1}^{d} x_i \tag{2.5.9}$$

and $\varepsilon > 0$ is a parameter chosen such that

$$\Delta_{\mathbf{x}} > 2\;(k^\star+3)\;\varepsilon, \qquad \forall \mathbf{x}\in\mathcal{X}: \quad \Delta_{\mathbf{x}} > 0. \tag{2.5.10}$$

In practice, though, Thompson sampling is one of the best-performing algorithms (Section 3.4), indicating that this upper bound is probably not tight.

2.5.3.2. *CUCB.* CUCB [**107, 108**] is an application of the 'optimism in the face of uncertainty' principle [**49**]. It is simple to implement, but not as easy as Thompson sampling. It relies on maximising an *index function* on the set of arms. This index function is linear, and each subarm is assigned an estimated reward computed as the upper-confidence bound for this subarm (independently of the others):

$$\mathbf{x}(n) \in \arg\max_{\mathbf{x}\in\mathcal{X}} \sum_{i=1}^{d} x_i \left[\hat{\theta}_i + \sqrt{1.5\,\frac{\log n}{\sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1}}}\right]. \tag{2.5.11}$$

The algorithm is formally stated in Algorithm 9. Computationally speaking, CUCB is as efficient as Thompson sampling, with $\mathcal{O}(d)$ operations to perform before solving one linear program.

As expected for a UCB-like algorithm in a structured bandit [**76**], this algorithm does not have the tightest upper bound on its regret [**107**, Theorem 5]:

$$R(n) \leq m \sum_{i=1}^{d} \frac{534}{\Delta_{i,\min}} + \left(\frac{\pi^2}{3} + 1\right) d\, m. \tag{2.5.12}$$

$\Delta_{e,\min}$ denotes the minimum gap of suboptimum solutions containing the subarm i:

$$\Delta_{i,\min} = \min_{\substack{\mathbf{x}\in\mathcal{X}:\\ x_i=1\\ \Delta_{\mathbf{x}}>0}} \Delta_{\mathbf{x}}. \tag{2.5.13}$$

The major source of regret of CUCB is that the upper-confidence term does not consider all arms at once.

2.5.4. Generic, statistically efficient algorithms.

2.5.4.1. *ESCB.* ESCB [**109**] is the first statistically efficient algorithm for combinatorial bandits, meaning that its regret bound is a factor depending only on the size of the combinatorial problem away from the lower bound for this problem. This algorithm is not considerably more complex to state than CUCB (Section 2.5.3.2), and is also based on the 'optimism in the face of uncertainty' principle: it is a natural extension to UCB-1 (Section (2.2.5)). If $f(n) = \log n + 4\, m \log\log n$, at each round ESCB chooses an arm to play according to:

$$\mathbf{x}(n) \in \arg\max_{\mathbf{x}\in\mathcal{X}} \mathbf{x}^T \hat{\boldsymbol{\theta}}(n) + \sqrt{\frac{f(n)}{2} \sum_{i=1}^{d} \frac{x_i}{T_i(n)}}. \tag{2.5.14}$$

The algorithm is formally stated in Algorithm 10. While CUCB uses an index linear in $\mathbf{x}$, ESCB does not. ESCB's index is still a concave function, though. Having an efficient oracle to optimise a linear function on $\mathcal{X}$ does not imply that such an oracle exists for concave objective functions. The goal of Chapter 3 is to propose a polynomial-time approximation for this index computation. The regret of ESCB satisfies [**110**, Theorem 2]:

$$R(T) \leq C_1 \frac{d \log^2 m \log T}{\Delta_{\min}} + C_2(\theta, \mathcal{X}), \tag{2.5.15}$$

where $C_1 > 0$ is a universal constant and $C_2(\theta, \mathcal{X})$ is a positive number which does not depend on T.

2.5.4.2. *OSSB.* OSSB (Section 2.4.4) is a very generic algorithm that also applies to combinatorial bandits. However, as it uses an optimisation program having one variable per arm, it cannot be efficiently implemented for combinatorial bandits (typically having $\mathcal{O}\big(2^d\big)$ arms for d subarms), unless an appropriate reformulation is applied. This reformulation and its impacts are the topic of Chapter 4.

2.5.4.3. *OLS-UCB.* OLS-UCB is a generalisation of ESCB that considers that subarm rewards can be correlated [**110**]: the subarm-reward distributions are no more independent. More precisely, at each round n, the vector of rewards $\boldsymbol{\theta}(n)$ is drawn from an unknown distribution in an identical independent and identically-distributed fashion. The components of $\boldsymbol{\theta}(n)$ are not considered to be independent from each other, as the distribution is no more univariate. OLS-UCB can exploit this fact to improve its regret. OLS-UCB starts with a positive semidefinite estimation of the correlation

Algorithm	Regret bound	Complexity
Thompson sampling	$\mathcal{O}\Big(\frac{d\,m\,\log T}{\Delta_{\min}}\Big)$	$\mathcal{O}[\pi(d)]$
CUCB	$\mathcal{O}\Big(\frac{d\,m\,\log T}{\Delta_{\min}}\Big)$	$\mathcal{O}[\pi(d)]$
ESCB	$\mathcal{O}\Big(\frac{d\,\log^2 m\,\log T}{\Delta_{\min}}\Big)$	$\mathcal{O}(\lvert\mathcal{X}\rvert) \subset \mathcal{O}\big(2^d\big)$
AESCB	$\mathcal{O}\Big(\frac{d\,\log^2 m\,\log T}{\Delta_{\min}}\Big)$	$\mathcal{O}\big[\delta_T^{-1}(T)\ \mathrm{poly}(d)\big]$

TABLE 1. Regret and complexity of several combinatorial-bandit algorithms. $\pi(d)$ denotes the complexity of optimising one linear function on the combinatorial set $\mathcal{X}$.

matrix, $\mathbf{\Gamma}$, and a parameter $\lambda > 0$ weighing the update of the correlation matrix. The arm to play is chosen as follows:

$$\mathbf{x}(n) \in \arg\max_{\mathbf{x}\in\mathcal{X}} \mathbf{x}^T\,\hat{\boldsymbol{\theta}}(n) + \sqrt{2\,f(n)\ \mathbf{x}^T\,\hat{\mathbf{D}}^{-1}(n)\ \left[\lambda\,\mathbf{\Gamma}\,\hat{\mathbf{D}}(n) + \sum_{t=1}^{n-1} \mathbf{x}^T(t)\ \mathbf{\Gamma}\,\mathbf{x}(t)\right]\ \hat{\mathbf{D}}^{-1}(n)\ \mathbf{x}} \tag{2.5.16}$$

where $f(n) = \log n + (m+2)\,\log\log n + m/2\,\log\left(1 + e/\lambda\right)$ and:

$$\hat{\mathbf{D}}(n) = \sum_{t=1}^{n-1} \mathbf{x}^T(t)\ \mathbf{x}(t)\,, \tag{2.5.17}$$

$$\hat{\theta}_i(n) = \frac{1}{T_i(n)} \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1} r(n)\,. \tag{2.5.18}$$

2.5.5. Computationally and statistically efficient algorithms for specific cases. Some combinatorial problems are very easy to solve, including in the nonlinear case. Matroids are in this class (they are explained in greater detail in Section 2.6.3): a greedy algorithm can maximise any submodular function on a matroid (this is the matroid equivalent of maximising a concave function).

OMM (*optimistic matroid maximisation*) is an application of the optimism principle [**49**] for matroid-constrained bandits [**111**]. It is also a special case of CUCB [**107**] (Section 2.5.3.2) when the combinatorial $\mathcal{X}$ is a matroid. ESCB (Section 2.5.4) can also be efficiently implemented on matroids [**112**]. However, these algorithms use very specific properties of matroids to implement these algorithms: they are very far from being general.

2.6. Interesting combinatorial sets

As previously mentioned (Section 2.5.1), several kinds of combinatorial sets $\mathcal{X}$ might be relevant to study. We focus on four families of sets. They all correspond to 'easy' combinatorial problems: for most of them, it is possible to optimise exactly a linear function in polynomial time. The exception is the knapsack, an $\mathcal{NP}$-hard problem [**113**], but it can be approximated in polynomial time (for instance, a greedy solution sorting items based on their value-to-weight ratio might provide a 2-approximation, i.e. an objective function that is within a factor 2 of the true optimum).

2.6.1. Knapsack-like sets. The easiest combinatorial structure is the m-set: out of d elements, only m may be chosen. The combinatorial set is written as:

$$\mathcal{X}_{\mathrm{m}} = \left\{ \mathbf{x} \in \{0,1\}^d \;\middle|\; \sum_{i=1}^{d} x_i \leq m \right\}. \tag{2.6.1}$$

It corresponds to any situation where a subset of items may be chosen, without the items being different one from the other. A linear function can be optimised on this set in linear time, for instance using a greedy algorithm.

A direct generalisation of the m-sets is the knapsack problem, where each item i has a specific weight $w_i \in \mathbb{N}$. This problem only allows for a maximum capacity $\Omega \in \mathbb{N}$. The combinatorial set is written as:

$$\mathcal{X}_{\mathrm{kp}} = \left\{ \mathbf{x} \in \{0,1\}^d \;\middle|\; \mathbf{w}^T \mathbf{x} \leq \Omega \right\}. \tag{2.6.2}$$

Such a simple extension of weights is sufficient to make the problem $\mathcal{NP}$-hard [**113**].

The multiple-knapsack problem combines several knapsack constraints, for instance due to using an item requiring the use of several resources. In this case, each item i has a vector of resource utilisation $\mathbf{w}_i$, with $w_{i,j} \in \mathbb{N}$ being the quantity of resource j that is used when item i is taken; the total usage of each resource j has an upper bound $\Omega_j \in \mathbb{N}$. The quantity of each resource is indicated in the matrix $\mathbf{W} \in \mathbb{N}^{c \times d}$ and the upper bounds by a vector $\mathbf{\Omega} \in \mathbb{N}^c$. The combinatorial set is written as:

$$\mathcal{X}_{\mathrm{mkp}} = \left\{ \mathbf{x} \in \{0,1\}^d \;\middle|\; \mathbf{W}\,\mathbf{x} \leq \mathbf{\Omega} \right\}. \tag{2.6.3}$$

In all three cases, the dimension d of $\mathcal{X}$ corresponds to the number of items. In the case of the m-set, the maximum number of items in the solution is m, by definition. This parameter has no specific value for knapsacks: it heavily depends on the weights $\mathbf{w}$ and on the budget Ω.

A typical application is online-ad display. A given Webpage has several places where ads are shown. Depending on their position on the Webpage, they are sold at different prices: a very visible spot is very expensive; at most m ads can be served at once. Several advertisers have a given budget. The Webmaster tries to place the various ads on their Website, and they can do so using a multiple knapsack: one knapsack for the m spots, so that there are not too many ads; one knapsack per advertiser and their budget per page view [**114**].

2.6.2. Source-destination path. Consider $G = (V, E)$ a directed acyclic graph with vertices V and edges E. The path-finding problem amounts to finding a path through this directed graph from a given source $s \in V$ to a fixed destination $t \in V$. The combinatorial set is formally written as:

$$\mathcal{X}_{\mathrm{path}} = \left\{ \mathbf{x} \in \{0,1\}^d \;\middle|\; \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = \mathbb{1}_{v=t} - \mathbb{1}_{v=s}, \quad \forall v \in V \right\}. \tag{2.6.4}$$

$\delta^+(v)$ is the set of incoming edges at v and $\delta^-(v)$ the set of outgoing edges:

$$\delta^+(v) = \{ e = (e_1, e_2) \in E \,|\, e_2 = v \}, \tag{2.6.5}$$

$$\delta^-(v) = \{ e = (e_1, e_2) \in E \,|\, e_1 = v \}. \tag{2.6.6}$$

The dimension of $\mathcal{X}$ is the number of edges in the graph, $d = |E|$. m is the maximum path length (measured as the number of crossed edges).

Applications are those of the online shortest-path problem, which have been described in Section 2.5.1.

2.6.3. Matroids. A matroid $(\mathcal{E}, \mathcal{I})$ **[115]** is defined over a set with d elements of $\mathcal{E}$, called *ground set*, and where $\mathcal{I}$ is a family of *independent subsets* of $\mathcal{E}$, and must respect those three properties:

the non-emptiness property: formally, $\emptyset \in \mathcal{I}$.
the inclusion property: every subset of an independent set must be independent: if $\mathcal{A}' \subset \mathcal{A} \subset \mathcal{E}$ and $\mathcal{A} \in \mathcal{I}$, then $\mathcal{A}' \subset \mathcal{I}$.
the exchange property: if $\mathcal{A} \in \mathcal{I}$ and $\mathcal{B} \in \mathcal{I}$ are two independent subsets of $\mathcal{E}$ with $|\mathcal{A}| > |\mathcal{B}|$, then there is some element $e \in \mathcal{A} \backslash \mathcal{B}$ such that $B \cup \{e\} \in \mathcal{I}$.

Any independent set $\mathcal{A} \in \mathcal{I}$ can also be represented as a vector $\mathbf{x} \in \{0,1\}^d$: each element $e_i \in \mathcal{C}$ has an index $i \in \mathbb{N}_0$ (and vice-versa: the index uniquely determines the element); if $e_i \in \mathcal{A}$, then $x_i = 1$, otherwise $x_i = 0$.

Matroids are an important part of combinatorial optimisation, as a greedy algorithm can solve optimally all problems with a matroid structure and a linear objective function **[116]**. In particular, the m-set has a matroid structure (unlike the knapsack): the empty set of elements is a solution, a subset of a solution is necessarily a solution, and the exchange property is also satisfied as only the number of elements is important.

The spanning tree is another example of matroid structure. Let $G = (V, E)$ be a undirected graph. A spanning tree T is a subset of edges E forming a tree that covers each vertex in V at least once. As such, an empty set of edges is a spanning tree, any subset of a spanning tree also corresponds to this definition; for the exchange property, any edge in $\mathcal{A}$ that is not in $\mathcal{B}$ and that keeps $\mathcal{B}$ a tree works. The combinatorial set is formally written as:

$$\mathcal{X}_{\text{st}} = \left\{ \mathbf{x} \in \{0,1\}^d \;\middle|\; \begin{array}{cc} s \in V & \\ \sum_{e \in \delta^-(v)} f_e = 0 & \\ \sum_{e \in \delta^+(v)} f_e = |V| - 1 & \\ \sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} f_e = \mathbb{1}_{v=t} - \mathbb{1}_{v=s} & \forall v \in V \backslash \{s\} \\ f_e \leq (|V| - 1)\, x_e & \forall e \in E \\ \mathbf{f} \in \mathbb{R}_+^d & \end{array} \right\}. \tag{2.6.7}$$

The dimension is the number of edges, $d = |E|$, and the maximum solution length is one less than the number of vertices, $m = |V| - 1$.

Computer networks are a typical application of spanning tree: a spanning tree is a subset of links that can be used to reach any node in the network; the other links can be reserved for redundancy. In the online spanning-tree problem, vertices are revealed one at a time, when they are being reached for the first time **[117]**.

Other applications of matroids include job sequencing, assignments, the Japanese logic puzzle kakuro (also known as cross sums), heuristics for the travelling-salesperson problem, and recognition of totally-unimodular matrices**[118]**.

2.6.4. Intersection of two matroids. If $\mathcal{I}$ and $\mathcal{I}'$ are two matroids defined on the same ground set $\mathcal{E}$, the combinatorial set defined by their intersection is:

$$\mathcal{X}_{\cap} = \left\{ \mathbf{x} \in \{0,1\}^d \;\middle|\; \mathbf{x} \in \mathcal{I}, \quad \mathbf{x} \in \mathcal{I}' \right\}. \tag{2.6.8}$$

Unlike simple matroids, a greedy algorithm is no more ensured to work on intersections of matroids [**119**]. Starting with three matroids, finding the intersection is an $\mathcal{NP}$-hard problem, because the problem can reduce from the Hamiltonian-path or the travelling salesperson problems [**120**].

Bipartite matchings are an application of the intersection of two matroids. Let $G = (V_1, V_2, E)$ be a bipartite graph, i.e. a graph whose vertices are partitioned into two disjoint sets ($V = V_1 \cup V_2$) and whose edges have one end in one partition and the other in the other partition ($E \subset V_1 \times V_2$). A matching can be seen as the intersection of two matroids, as a matching must have all its edges adjacent to both sets of vertices. Let $\delta(v)$ be the set of edges adjacent to v, the two matroids are:

$$\mathcal{I}_1 = \left\{ F \subset E \;\middle|\; |F \cap \delta(v)| \leq 1, \quad \forall v \in V_1 \right\}, \tag{2.6.9}$$

$$\mathcal{I}_2 = \left\{ F \subset E \;\middle|\; |F \cap \delta(v)| \leq 1, \quad \forall v \in V_2 \right\}. \tag{2.6.10}$$

Similar results exist for nonbipartite matchings [**121**]. For a matching, the dimension is the number of edges, $d = |E|$, and the maximum solution length is the minimum between the number of vertices in V_1 and in V_2, $m = \min\{|V_1|, |V_2|\}$.

Applications of the online matching problem have already been discussed in Section 2.5.1.

CHAPTER 3

Approximation Algorithms for Optimum Combinatorial Bandits

We consider the setting of combinatorial semibandits (Sections 2.5 and 2.5.2). Furthermore, we only consider the case where rewards pertain to $[0, 1]$ and are independent across items. We focus on the combinatorial sets defined in Section 2.6.

ESCB is a state-of-the-art algorithm for combinatorial bandits (Section 2.5.4): its regret is, up to a constant factor, optimum. However, its complexity makes it unappealing: in the worst case, it might require an exponential time in the dimension d of the polytope $\mathcal{X}$ per round. Other algorithms have a worse regret, but a better computational complexity (Section 2.5.3). This antithesis seems to indicate that there is a trade-off between regret and computational complexity.

This chapter introduces an approximation of ESCB, called AESCB [**?**]: by carefully introducing approximation in the algorithm, it can keep its optimum regret, but can be implemented in polynomial time. The approximation appears in the regret bound, however. There is still a trade-off between regret and computational complexity, but a polynomial-time algorithm can still achieve the lower bound for combinatorial bandits up to some constant factor.

3.1. AESCB

What makes ESCB hard to implement in polynomial time is the following optimisation program (introduced in (2.5.14), Section 2.5.4), one instance of which must be solved for each bandit round:

$$\mathbf{x}(n) \in \operatorname*{arg\,max}_{\mathbf{x}\in\mathcal{X}} \left\{ \mathbf{x}^T \hat{\boldsymbol{\theta}}(n) + \sqrt{\frac{f(n)}{2} \sum_{i=1}^{d} \frac{x_i}{T_i(n)}} \right\}. \tag{3.1.1}$$

As before, $f(n) = \log n + 4\,m \log\log n$. For ease of notations, let $\boldsymbol{\sigma}^2(n)$ be the following vector:

$$\sigma_i^2(n) = \frac{f(n)}{2\,T_i(n)}. \tag{3.1.2}$$

Definition 2. The ESCB algorithm is the policy which at any time $n \in \mathbb{N}_0$ selects its decision as:

$$\mathbf{x}(n) \in \operatorname*{arg\,max}_{\mathbf{x}\in\mathcal{X}} \left\{ \mathbf{x}^T \hat{\boldsymbol{\theta}}(n) + \sqrt{\mathbf{x}^T\, \boldsymbol{\sigma}^2(n)} \right\} \tag{3.1.3}$$

where ties are broken arbitrarily.

3.1.1. NP-hardness behind ESCB . Solving this optimisation program exactly and efficiently does not seem to be a viable path. Indeed, in general, (2.5.14) is a $\mathcal{NP}$-hard problem: formally, no polynomial-time algorithm can exist, unless $\mathcal{P} = \mathcal{NP}$. In layman's terms, it is very

unlikely that such a theoretically efficient algorithm exists. Even restricting to a subset of combinatorial sets does not look promising, apart from very simple combinatorial sets like matroids (Section (2.5.5)).

For instance, if $\mathcal{X}$ is the set of paths in a graph (allowing for loops in the paths), then it is easy to find a shortest path between two nodes with nonnegative weights (or no negative-weight loop, depending on the choice of algorithm). However, slight generalisations of this problem quickly become $\mathcal{NP}$-hard. For instance, finding a longest path is $\mathcal{NP}$-hard: the main complexity is that such a path must avoid loops (i.e. only *simple* paths are allowed), as adding any positive-weight loop increases the length of the path. Similarly, finding a simple path that minimises any nonlinear function is $\mathcal{NP}$-hard: relaxing the simple-path constraint or using an objective function that exhibits a global minimum still yields an $\mathcal{NP}$-hard problem (like convex functions) [**122**]. The same property holds for maximum-cardinality bipartite matchings: minimising a linear objective function is easy (the Hopcroft-Karp algorithm has complexity $\mathcal{O}\Big(|E|\ \sqrt{|V|}\Big)$, for instance [**123**]), but not a convex function (the problem becomes $\mathcal{NP}$-hard [**124**]).

[**125**] proves a more general result (the following theorem corresponds to [**125**]'s Propositions 1, 3, and 4):

THEOREM 3. *If g is a strictly concave function, for any vectors $\mathbf{a}$ and $\mathbf{b}$ of $\mathbb{R}_+^d$, the problem $\max_{\mathbf{x}\in\mathcal{X}} \mathbf{a}^T\,\mathbf{x} + g\big(\mathbf{b}^T\,\mathbf{x}\big)$ is $\mathcal{NP}$-hard when $\mathcal{X}$ is the set of m-sets* [**125**, Proposition 1]*, the set of paths in a directed acyclic graph* [**125**, Proposition 3]*, or the set of perfect matchings in a bipartite graph* [**125**, Proposition 4]*.*

This theorem applies to ESCB, as the square root is a strictly concave function. Solving (2.5.14) exactly in polynomial time seems to be impossible, unless $\mathcal{P} = \mathcal{NP}$.

Therefore, we settle on solving (2.5.14) approximately. The AESCB algorithm requires two sequences, $\{\varepsilon_n\}$ and $\{\delta_n\}$, quantifying the level of approximation at each time step. δ_n is an additive approximation error on the total objective, while ε_n is a multiplicative approximation error for the nonlinear term.

DEFINITION 4. The AESCB algorithm, with the sequences $\{\varepsilon_n\}$ and $\{\delta_n\}$, is the policy that, at any time $n \in \mathbb{N}_0$, selects a decision $\mathbf{x}(n)$ satisfying:

$$\arg\max_{\mathbf{x}\in\mathcal{X}} \left\{\mathbf{x}^T\,\hat{\boldsymbol{\theta}}(n) + \sqrt{\mathbf{x}^{\star T}\,\boldsymbol{\sigma}^2(n)}\right\} \leq \delta_n + \mathbf{x}^T(n)\ \hat{\boldsymbol{\theta}}(n) + \frac{1}{\varepsilon_n}\sqrt{\mathbf{x}^T(n)\ \boldsymbol{\sigma}^2(n)} \tag{3.1.4}$$

where ties are broken arbitrarily.

3.1.2. Regret bound. AESCB's regret bound is highly similar to that of ESCB. Both approximation parameters ε_n and δ_n play a role in this bound.

THEOREM 5. *The regret of AESCB with parameters $(\varepsilon_n, \delta_n)$ admits the following upper bound for all $T \in \mathbb{N}_0$:*

$$R(T) \leq C_4(m) + \frac{2\,d\,m^3}{\Delta_{\min}^2} + \frac{24\,d\,f(T)}{\left(\min_{n\leq T}\varepsilon_n\right)^2\Delta_{\min}}\left\lceil\frac{\log m}{1.61}\right\rceil^2 + 4\sum_{n=1}^{T}\delta_n\,\mathbb{1}_{\Delta_{\min}\leq 4\,\delta_n} \tag{3.1.5}$$

with $f(n) = \log n + 4\,m\,\log\log n$ and $C_4(m)$ a positive number that solely depends on m.

This bound is highly similar to that of ESCB. In particular, it has the same asymptotic behaviour as ESCB with fixed ε (the most common case in this thesis, as our approximation algorithms have a fixed ratio, as detailed in Section 3.2) and vanishing δ_n, as highlighted in the next corollary.

COROLLARY 6. *For $\varepsilon_n = \varepsilon > 0$ and $\lim_{n\to+\infty} \delta_n = 0$, we have:*

$$R(T) \in \mathcal{O}\left(d \log^2 m \, \frac{1}{\Delta_{\min}} \log T \right) \quad as \quad T \to \infty. \tag{3.1.6}$$

Similarly, with $\varepsilon_n = \varepsilon$ and $\delta_n < \frac{1}{4}\Delta_{\min}$, we have, for all $T \in \mathbb{N}_0$:

$$R(T) \leq C_4(m) + \frac{2\,d\,m^3}{\Delta_{\min}^2} + \frac{24\,d\,f\,(T)}{\varepsilon^2 \Delta_{\min}} \left\lceil \frac{\log m}{1.61} \right\rceil^2. \tag{3.1.7}$$

We discuss the choice of sequences $\{\varepsilon_n\}$ and $\{\delta_n\}$ to obtain both low regret and low complexity in Section 3.1.6.

3.1.3. Sketch of proof. The regret analysis of AESCB involves upper bounding the reward gap of the decision chosen at round n, $\Delta_{\mathbf{x}(n)}$, by considering three cases. Define the two following events:

$$\mathcal{A}(n) = \left\{ \exists \mathbf{x} \in \mathcal{X} \;\middle|\; \left| \left[\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}(n) \right]^T \mathbf{x} \right| \geq \sqrt{\mathbf{x}^T \, \boldsymbol{\sigma}^2(n)} \right\}, \tag{3.1.8}$$

$$\mathcal{B}(n) = \left\{ \Delta_{\mathbf{x}(n)} \leq 4\,\delta_n \right\}. \tag{3.1.9}$$

If $\mathcal{A}(n)$ occurs, since $\boldsymbol{\theta} \in [0,1]^d$ and $\sum_{i=1}^d x_i^\star \leq m$:

$$\Delta_{\mathbf{x}(n)} \leq \boldsymbol{\theta}^T \, \mathbf{x}^\star \leq m. \tag{3.1.10}$$

If $\mathcal{B}(n)$ occurs, by definition:

$$\Delta_{\min} \leq \Delta_{\mathbf{x}(n)} \leq 4\,\delta_n. \tag{3.1.11}$$

Let $\mathcal{C}(n)$ be the union of these two events:

$$\mathcal{C}(n) = \overline{\mathcal{A}(n)} \cup \overline{\mathcal{B}(n)}. \tag{3.1.12}$$

If $\mathcal{C}(n)$ occurs, the index of the optimum decision is greater than the optimum reward $\boldsymbol{\theta}^T \, \mathbf{x}^\star$:

$$\begin{aligned}
\boldsymbol{\theta}^T \, \mathbf{x}^\star &\leq \hat{\boldsymbol{\theta}}^T(n) \; \mathbf{x}^\star + \sqrt{\mathbf{x}^{\star T} \, \boldsymbol{\sigma}^2(n)}, \text{ as } \overline{\mathcal{A}(n)} \text{ occurs} \\
&\leq \delta_n + \hat{\boldsymbol{\theta}}^T(n) \; \mathbf{x}(n) + \frac{1}{\varepsilon_n} \sqrt{\mathbf{x}^T(n) \, \boldsymbol{\sigma}^2(n)}, \text{ due to AESCB's approximation} \\
&\leq \delta_n + \boldsymbol{\theta}^T(n) \; \mathbf{x}(n) + \frac{2}{\varepsilon_t} \sqrt{\mathbf{x}^T(n) \, \boldsymbol{\sigma}^2(n)}, \text{ using the true parameters } \boldsymbol{\theta} \text{ instead of } \hat{\boldsymbol{\theta}} \\
&\leq \frac{1}{4} \, \Delta_{\mathbf{x}(n)} + \boldsymbol{\theta}^T(n) \; \mathbf{x}(n) + \frac{2}{\varepsilon_t} \sqrt{\mathbf{x}^T(n) \, \boldsymbol{\sigma}^2(n)}, \text{ as } \overline{\mathcal{B}(n)} \text{ occurs.}
\end{aligned}$$

Therefore, if $\mathcal{C}(n)$ occurs, using the fact that $\Delta_{\mathbf{x}(n)} = \boldsymbol{\theta}^T \, \mathbf{x}^\star - \boldsymbol{\theta}^T(n) \; \mathbf{x}(n)$:

$$\Delta_{\mathbf{x}(n)} \leq \frac{8}{3} \frac{1}{\varepsilon_n} \sqrt{\mathbf{x}^T(n) \, \boldsymbol{\sigma}^2(n)} \leq \frac{4}{\varepsilon_n} \sqrt{\mathbf{x}^T(n) \, \boldsymbol{\sigma}^2(n)}. \tag{3.1.13}$$

As $\mathcal{A}(n)$, $\mathcal{B}(n)$, and $\mathcal{C}(n)$ cover all possible cases, we get:

$$\begin{aligned}
\Delta_{\mathbf{x}(n)} &= \Delta_{\mathbf{x}(n)} \left(\mathbb{1}_{\mathcal{A}(n)} + \mathbb{1}_{\mathcal{B}(n)} + \mathbb{1}_{\mathcal{C}(n)} \right) \\
&\leq m \, \mathbb{1}_{\mathcal{A}(n)} + 4\,\delta_t \, \mathbb{1}_{\Delta_{\mathbf{x}(n)} \leq 4\,\delta_n} + \Delta_{\mathbf{x}(n)} \, \mathbb{1}_{\Delta_{\mathbf{x}(n)} \leq \frac{4}{\varepsilon_n} \sqrt{\mathbf{x}^T(n) \boldsymbol{\sigma}^2(n)}}.
\end{aligned}$$

Taking expectations and summing over n:

$$\begin{aligned} R(T) &= \sum_{n=1}^{T} \mathbb{E}\big\{\Delta_{\mathbf{x}(n)}\big\} \\ &\leq \sum_{n=1}^{T} m\,\mathbb{P}\{\mathcal{A}(n)\} + 4\sum_{n=1}^{T} \delta_n\,\mathbb{P}\big\{\Delta_{\mathbf{x}(n)} \leq 4\,\delta_n\big\} + \sum_{n=1}^{T} \mathbb{E}\left\{\Delta_{\mathbf{x}(n)}\,\mathbb{1}_{\Delta_{\mathbf{x}(n)} \leq \frac{4}{\varepsilon_n}\sqrt{\boldsymbol{\sigma}^2(n)\,\mathbf{x}(n)}}\right\}. \end{aligned}$$

The first term is bounded by a constant, since, using a concentration inequality, we may show that $\mathcal{A}(n)$ occurs with small probability. The second term creates a regret that only depends on the discretisation step δ_n. The last term can be bounded using (rather intricate) counting arguments as in the analysis of ESCB. The complete proof is presented in Section 3.5.

3.1.4. Reduction to budgeted problems. We now show a technique to implement AESCB that ensures polynomial time complexity. While our methodology is generic, the precise value of the computational complexity depends on the combinatorial set $\mathcal{X}$, and will be explained in detail in Section 3.2. Briefly, our technique involves moving the linear part of the objective function to a constraint with a minimum level, to optimise the nonlinear term, and to test several values for the minimum value of the linear term of the objective function.

In greater detail, our approach involves three steps:

- **rounding and scaling** to ensure that the weights are integer-valued. To this end, we define two vectors, $\mathbf{a}(n)$ and $\mathbf{b}(n)$:

$$\xi(n) = \left\lceil \frac{m}{\delta_n} \right\rceil, \tag{3.1.14}$$

$$a_i(n) = \Big\lceil \xi(n)\;\hat{\theta}_i(n) \Big\rceil, \quad \forall i \in \{1, 2 \dots d\}, \tag{3.1.15}$$

$$b_i(n) = \xi^2(n)\;\sigma_i^2(n), \quad \forall i \in \{1, 2 \dots d\}. \tag{3.1.16}$$

- **solving a budgeted linear maximisation** several times. For all budgets $s \in \mathcal{S}(n) = \{0, 1 \dots m\,\xi(n)\}$, we define a budgeted optimisation program:

$$\begin{aligned} &\max \mathbf{x}^T\,\mathbf{b}(n) \\ &\text{s.t.}\; \mathbf{x}^T\,\mathbf{a}(n) \geq s \\ &\quad \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{3.1.17}$$

  Let $\mathbf{x}(n, s)$ be any optimum solution to this budgeted program. We compute an ε_n-approximate solution $\overline{\mathbf{x}}(n, s)$ to this problem: the approximation is a multiplicative error on the objective function, not on the budget constraint. Formally, $\overline{\mathbf{x}}(n, s)$ respects the following conditions:

$$\overline{\mathbf{x}}^T(n, s)\;\mathbf{b}(n) \geq \varepsilon_n\,\mathbf{x}_s^T(n)\;\mathbf{b}(n), \tag{3.1.18}$$

$$\overline{\mathbf{x}}^T(n, s)\;\mathbf{b}(n) \geq s, \tag{3.1.19}$$

$$\overline{\mathbf{x}}^T(n, s) \in \mathcal{X}. \tag{3.1.20}$$

- **maximising over the budget** to obtain the final result. We return the decision $\mathbf{x}(n) = \overline{\mathbf{x}}[n, s^\star(n)]$ where

$$s^\star(n) \in \arg\max_{s \in \mathcal{S}(n)} \left\{ s + \frac{1}{\varepsilon_n} \sqrt{\overline{\mathbf{x}}^T(n,s)\ \mathbf{b}(n)} \right\}. \tag{3.1.21}$$

$\mathbf{a}(n)$ is defined using a ceiling operation in order to ensure that $\mathbf{x}^T\,\mathbf{a}(n)$ has an integer value for all $\mathbf{x} \in \mathcal{X}$. This property is crucial to have a limited set of budget values $\mathcal{S}(n)$ in the second step: we only need to cover integer values between zero and an upper bound on the value of $\mathbf{x}^T\,\mathbf{a}(n)$, which is $m\,\xi(n)$. Conversely, $\mathbf{b}(n)$ does not need to have integer entries, as no such property is required.

THEOREM 7. *The above algorithm returns a decision $\mathbf{x}(n)$ verifying the AESCB definition (Definition 4). It does so by approximately solving the budgeted optimisation problem 3.1.17 at most $m\,\xi(n)+1$ times with input parameters $\mathbf{a}(n)$ and $\mathbf{b}(n)$, where $\mathbf{a}(n) \in \{1, 2 \dots \xi(t)\}^d$ and $\mathbf{b}(n) \in \mathbb{R}^d$.*

3.1.5. Proof of Theorem 7. The first step is to upper bound the value of the original ESCB problem (2.5.14) with inputs $\mathbf{a}(n)$ and $\mathbf{b}(n)$ as a function of $\mathbf{x}(n)$.

We have defined the set of budget values $\mathcal{S}(n)$ as $\{0, 1 \dots m\,\xi(n)\}$. By definition, $\mathbf{a}(n) \in \{1, 2 \dots \xi(t)\}^d$ and $m = \max_{\mathbf{x} \in \mathcal{X}} \sum_{i=1}^d x_i$. This has the direct consequence that:

$$\mathbf{a}^T(n)\ \mathbf{x} \in \mathcal{S}(n)\,, \qquad \forall \mathbf{x} \in \mathcal{X}. \tag{3.1.22}$$

Therefore:

$$\begin{aligned}
&\max_{\mathbf{x} \in \mathcal{X}} \left\{ \mathbf{a}^T(n)\ \mathbf{x} + \sqrt{\mathbf{b}^T(n)\ \mathbf{x}} \right\} \\
&\quad = \max_{s \in \mathcal{S}(n)} \max_{\substack{\mathbf{x} \in \mathcal{X} \\ \mathbf{a}^T(n)\,\mathbf{x} = s}} \left\{ s + \sqrt{\mathbf{b}^T(n)\ \mathbf{x}} \right\} \\
&\quad \leq \max_{s \in \mathcal{S}(n)} \max_{\substack{\mathbf{x} \in \mathcal{X} \\ \mathbf{a}^T(n)\,\mathbf{x} \geq s}} \left\{ s + \sqrt{\mathbf{b}^T(n)\ \mathbf{x}} \right\}, \text{ with a minimum budget instead of equality} \\
&\quad \leq \max_{s \in \mathcal{S}(n)} \left\{ s + \frac{1}{\varepsilon_n} \sqrt{\mathbf{b}^T(n)\ \overline{\mathbf{x}}(n,s)} \right\}, \text{ using the approximate budgeted oracle} \\
&\quad = s^\star(t) + \frac{1}{\varepsilon_t} \sqrt{\mathbf{b}^T(n)\ \overline{\mathbf{x}}[n, s^\star(n)]}, \text{ where } s^\star(n) \in \arg\max_{s \in \mathcal{S}(n)} \left\{ s + \frac{1}{\varepsilon_t} \sqrt{\mathbf{b}^T(n)\ \overline{\mathbf{x}}_s(n)} \right\} \\
&\quad \leq \mathbf{a}^T(n)\ \overline{\mathbf{x}}_{s^\star(n)}(n) + \frac{1}{\varepsilon_t} \sqrt{\mathbf{b}^T(n)\ \overline{\mathbf{x}}[n, s^\star(n)]} \\
&\quad = \mathbf{a}^T(n)\ \mathbf{x}(n) + \frac{1}{\varepsilon_t} \sqrt{\mathbf{b}^T(n)\ \mathbf{x}(n)}, \text{ by choosing } \mathbf{x}(n) = \overline{\mathbf{x}}[n, s^\star(n)]\,.
\end{aligned} \tag{3.1.23}$$

The second step is to link the value of problem (2.5.14) to the rounded or scaled inputs $\mathbf{a}(n)$ and $\mathbf{b}(n)$ to the value of the same problem (2.5.14) with inputs $\hat{\boldsymbol{\theta}}(n)$ and $\boldsymbol{\sigma}^2(n)$. By definition of AESCB (3.1.15) and (3.1.16), these two sets of vectors are related by:

$$\xi(n) = \left\lceil \frac{m}{\delta_n} \right\rceil, \tag{3.1.24}$$

$$a_i(n) = \left\lceil \xi(n)\ \hat{\theta}_i(n) \right\rceil, \quad \forall i \in \{1, 2 \dots d\}\,, \tag{3.1.25}$$

$$b_i(n) = \xi^2(n)\ \sigma_i^2(n)\,, \quad \forall i \in \{1, 2 \dots d\}\,. \tag{3.1.26}$$

As $\mathbf{a}(n) \in \{1, 2 \dots \xi(t)\}^d$, the following inequalities hold:

$$\hat{\theta}_i(n) \leq \frac{1}{\xi(n)}\, a_i(n) \leq \frac{1}{\xi(n)} + \hat{\theta}_i(n)\,, \quad \forall i \in \{1, 2 \dots d\}\,. \tag{3.1.27}$$

As a consequence, for any $\mathbf{x} \in \mathcal{X}$:

$$\hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x} \leq \frac{1}{\xi(n)}\, \mathbf{a}^T(n)\ \mathbf{x} \leq \frac{\sum_{i=1}^d x_i}{\xi(n)} + \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x} \leq \delta_n + \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x}. \tag{3.1.28}$$

Merging the two results (3.1.23) and (3.1.28),

$$\begin{aligned}
&\max_{\mathbf{x} \in \mathcal{X}} \left\{ \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x} + \sqrt{\mathbf{x}^T\, \boldsymbol{\sigma}^2(n)} \right\} \\
&\quad = \frac{1}{\xi(n)}\ \max_{\mathbf{x} \in \mathcal{X}} \left\{ \xi(n)\ \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x} + \sqrt{\xi^2(n)\ \mathbf{x}^T\, \boldsymbol{\sigma}^2(n)} \right\} \\
&\quad \leq \frac{1}{\xi(n)}\ \max_{\mathbf{x} \in \mathcal{X}} \left\{ \mathbf{a}^T(n)\ \mathbf{x} + \sqrt{\mathbf{b}^T(n)\ \mathbf{x}} \right\}, \text{ by (3.1.15) and (3.1.16)} \\
&\quad \leq \frac{1}{\xi(n)}\, \mathbf{a}^T(n)\ \mathbf{x}(n) + \frac{1}{\xi(n)\ \varepsilon_n} \sqrt{\mathbf{b}^T(n)\ \mathbf{x}(n)}, \text{ by (3.1.23)} \\
&\quad \leq \delta_n + \hat{\boldsymbol{\theta}}(n)\ \mathbf{x}(n) + \frac{1}{\varepsilon_n} \sqrt{\mathbf{x}^T(n)\ \boldsymbol{\sigma}^2(n)}, \text{ by (3.1.28)}.
\end{aligned}$$

This is the announced result.

3.1.6. Choice of approximation parameters. The choice of sequences $\{\varepsilon_n\}$ and $\{\delta_n\}$ is paramount to obtain good performance with AESCB, both in terms of regret and computational time. If the chosen values of ε_n and δ_n are too large, solutions to the budgeted problems can be computed quickly, but they will probably be of little use for the bandit problem: the regret that AESCB would then endure would also be very high. On the other hand, if the chosen values are too low, it may be impossible to solve the budgeted problems on some polytopes.

When $\varepsilon_n = 1$ and $\delta_n = 0$ for all $n \in \{1, 2 \dots T\}$, AESCB reduces to ESCB. With an appropriate choice of parameters, AESCB can be implemented in polynomial time. The choice of parameters often depends on the combinatorial set $\mathcal{X}$, and and specifically on the approximation ratio of the available algorithms for the budgeted subproblem. In particular:

- For m-sets, knapsack-like sets, and paths, we choose $\varepsilon_n = 1$ for $n \in \{1, 2 \dots T\}$.
- For matroids (including spanning trees) and matroid intersections (like matchings), we choose $\varepsilon_n = \frac{1}{2}$ for $n \in \{1, 2 \dots T\}$.

The value of δ_n is not mandated by the available algorithms for the combinatorial set. This choice of parameters does not require any knowledge about the time horizon T, nor about the unknown problem parameters $\boldsymbol{\theta}$, nor about the minimal gap $\Delta_{\min}$.

Usually, ε_n cannot be chosen freely: there must be an approximation algorithm with this ratio available for the combinatorial set at hand. The situation for δ_n is opposite: this discretisation parameter can be chosen freely, to balance the regret of the bandit and the computational time for each round. The regret has a contribution that is linear in the sum of δ_n (as long as these terms are over $\Delta_{\min}/4$), but the complexity of AESCB typically depends on the inverse of the minimum value of δ_n (the exact dependency might vary based on the algorithm used to approximately solve the budgeted program 3.1.17).

The parameter δ_n does not appear in the regret bound of Theorem 5 as long as $\Delta_{\min} \leq 4\,\delta_n$. One way of choosing this parameter is therefore to estimate $\Delta_{\min}$ based on $\hat{\boldsymbol{\theta}}(n)$ and to choose δ_n accordingly. Estimating $\Delta_{\min}$ can be done in polynomial time, based on the fact that $\Delta_{\min}$ is realised between the optimum arm and the arm that has the largest reward after the optimum. Intuitively, if $\Delta_{\min}$ is realised between two other arms, the convergence of the bandit algorithm is not hindered, as the root cause of regret is often playing a slightly suboptimum arm. Let $\mathbf{x}^\star(n)$ be the optimum arm for $\hat{\boldsymbol{\theta}}(n)$ and $\mathbf{x}_i^\star\,(n)$ be the optimum arm with the subarm i different from that of $\mathbf{x}^\star(n)$:

$$\mathbf{x}^\star(n) \in \arg\min_{\mathbf{x}\in\mathcal{X}} \left\{\mathbf{x}^T\,\hat{\boldsymbol{\theta}}(n)\right\}, \tag{3.1.29}$$

$$\mathbf{x}_i^\star\,(n) \in \arg\max_{\substack{\mathbf{x}\in\mathcal{X},\\ x_i\neq x_i^\star}} \left\{\mathbf{x}^T\,\hat{\boldsymbol{\theta}}(n)\right\}. \tag{3.1.30}$$

$\Delta_{\min}$ is estimated as the minimum difference in reward between the optimum solution $\mathbf{x}^\star(n)$ and the solution basis formed by the $\mathbf{x}_i^\star\,(n)$ for $i \in \{1, 2 \dots d\}$. Computing $\mathbf{x}_i^\star\,(n)$ is often as easy as computing $\mathbf{x}^\star(n)$: for instance, if $\mathbf{x}^\star(n)$ is calculated using a totally unimodular matrix, adding a constraint like $x_i = 0$ (if $x_i^\star = 1$) or $x_i = 1$ (if $x_i^\star = 0$) keeps the total unimodularity [**126**]. This procedure is formalised in Algorithm 12. However, in this case, the complexity of AESCB depends (albeit polynomially) on $\Delta_{\min}^{-1}$.

REMARK 8. If the bandit is used with a horizon of T rounds, it cannot distinguish between two solutions with $\Delta_{\min} < T^{-1/2}$, due to the fact that $\beth^{-2}$ samples are required to tell two distributions whose means differ by $\beth$. Furthermore, the regret due to playing arms with a gap less than T^{-1} for T rounds is only constant (by definition, at most 1). Therefore, in the worst case, this choice of δ_n leads to an algorithm with a complexity linear in T.

Another way of choosing δ_n is taking any decreasing sequence. With n sufficiently large, $\delta_n < \Delta_{\min}/4$ and no more regret is incurred due to the choice of δ_n. The total contribution of δ_n is then constant. For instance,

$$\delta_n = \frac{1}{\log n}. \tag{3.1.31}$$

In this scenario, the complexity of AESCB depends logarithmically on T. δ_n can also be chosen to decrease slower than this, for instance $\delta_n = \log^{-2} n$: the dependency of the complexity on n can be made arbitrarily mild in Table 1.

3.2. Optimising budgeted programs

The difficult part of implementing AESCB in polynomial time is to provide algorithms to solve the budgeted optimisation programs of Section 3.1.4. The literature provides few algorithms for this problem, but instead focuses on the reverse constraint budget $\mathbf{b}^T\,\mathbf{x} \leq s$ [**127**] (for matroids and spanning trees [**128**], paths [**127**], matching and matroid intersection [**127, 129, 130, 131**]).

We now describe the algorithms we developed for the budgeted problems that arise when implementing AESCB for the combinatorial problems presented in Section 2.6. We also state their complexities, summarised in Table 1. The pseudocodes are provided in Appendix B.

	CUCB, TS	AESCB
m-set	$\mathcal{O}(d)$	$\mathcal{O}\big(d\,m\,\delta_n^{-2}\big)$
Path	$\mathcal{O}(d\log d)$	$\mathcal{O}\big(d\log d + d\,m\,\delta_n^{-1}\big)$
Spanning tree	$\mathcal{O}(d\log d)$	$\mathcal{O}\big(m\,d\,\log^3 d\,\delta_n^{-1}\big)$
Matching	$\mathcal{O}\big(m^3\big)$	$\mathcal{O}\big(m\,d^{10}\,\delta_n^{-1}\big)$

TABLE 1. Complexity of bandit policies (at round n) as a function of the chosen discretisation δ_n.

	Complexity (linear maximisation)	Complexity (budgeted linear maximisation) with budget s	Approximation ratio ε
m-set	$\mathcal{O}(d)$	$\mathcal{O}(s\,d)$	1
Path	$\mathcal{O}(\lvert E\rvert + \lvert V\rvert\,\log\lvert V\rvert)$	$\mathcal{O}(s\,\lvert E\rvert + \lvert V\rvert\,\log\lvert V\rvert)$	1
Spanning tree	$\mathcal{O}(\lvert E\rvert\,\log\lvert V\rvert)$	$\mathcal{O}\big(\lvert E\rvert\,\log^2\lvert V\rvert + \lvert V\rvert\,\log^3\lvert E\rvert\big)$	1/2
Matching	$\mathcal{O}\Big(\lvert V\rvert^2\,\lvert E\rvert\Big)$	$\mathcal{O}\Big(\lvert V\rvert^3\,\lvert E\rvert^4\Big)$	1/2

TABLE 2. Complexity of the budgeted combinatorial algorithms and their approximation ratio.

3.2.1. m-set and knapsack-like sets. For k knapsack constraints, we can solve the *rounded* problem exactly 3.1.17 (i.e. $\varepsilon = 1$) in time $\mathcal{O}\Big(m\,d\,\xi\,\prod_{\ell=1}^{k} c_\ell\Big)$. This complexity is highly similar to the classic dynamic-programming algorithm for single binary knapsacks **[132]**, except that the maximum capacity of the knapsack is known to be a polynomial in the dimension d and the inverse of the discretisation step $\delta^{-1} \in \mathcal{O}(\xi)$.

As the m-set problem is a special case of the k knapsack problem with $k = 1$, we can solve the rounded problem 3.1.17 exactly (i.e. $\varepsilon = 1$) with the same technique. To perform the reduction, we use a weight matrix $\mathbf{A} = \begin{pmatrix}1 & 1 & \cdots & 1\end{pmatrix}^T$ and a capacity vector $\mathbf{c} = (m)$. In this special case, the same algorithm (formalised in Algorithm 13) has a complexity $\mathcal{O}(m\,d\,\xi)$.

CLAIM 9. Optimisation problem 3.1.17 with $\mathcal{X}$ the set of m-sets can be solved exactly (i.e. $\varepsilon = 1$) in time $\mathcal{O}(m\,d\,\xi)$ using the algorithm below.

CLAIM 10. Optimisation problem 3.1.17 with $\mathcal{X}$ a k-knapsack set with the weights $\mathbf{A} \in \mathbb{N}^{d\times k}$ and the capacities $\mathbf{c} \in \mathbb{N}^k$ can be solved exactly (i.e. $\varepsilon = 1$) in time $\mathcal{O}\Big(m\,d\,\xi\,\prod_{\ell=1}^{k} c_\ell\Big)$ using the algorithm below.

The generic algorithm for k knapsack constraints is as follows. We denote the optimisation problem by $P(s, \mathbf{c}, i)$, which is a variation of the budgeted knapsack-like problem: the minimum budget is set to s, objects up to and including i are not used in the solution, the capacities are fixed to $\mathbf{c}$.

$$\begin{aligned} &\max \mathbf{b}^T\,\mathbf{x} && (P(s,\mathbf{c},i))\\ &\text{s.t. } \mathbf{A}\,\mathbf{x} \leq \mathbf{c}\\ &\quad \mathbf{a}^T\,\mathbf{x} \geq s \end{aligned}$$

$$\sum_{j=1}^{i} x_j = 0$$
$$\mathbf{x} \in \{0,1\}^d .$$

Let $V(s, \mathbf{c}, i)$ be the optimum value of $P(s, \mathbf{c}, i)$.

Solving the original budgeted problem reduces to $P(s, \mathbf{c}, 0)$. We can compute the solution to $P(s, \mathbf{c}, 0)$ using dynamic programming. To this end, it is sufficient to solve $P(s, \mathbf{c}, i)$ for i in $\{0, 1 \dots d\}$.

Let $\mathbf{x}^\star$ be an optimum solution to $P(s, \mathbf{c}, i)$. If $x^\star_{i+1} = 1$, then

$$\mathbf{A}\ (\mathbf{x}^\star - \mathbf{e}_i) = \mathbf{A}\,\mathbf{x}^\star - \mathbf{A}\,\mathbf{e}_i \leq \mathbf{c} - \mathbf{A}\,\mathbf{e}_i, \tag{3.2.1}$$

because $\mathbf{x}^\star$ is a feasible solution such that $\mathbf{A}\,\mathbf{x}^\star \leq \mathbf{c}$. Hence, $\mathbf{x}^\star - \mathbf{e}_i$ is an optimum solution to $P\left(\max\left\{s - a_i, 0\right\}, \mathbf{c} - \mathbf{A}\,\mathbf{e}_i, i+1\right)$. On the contrary, if $x^\star_i = 0$, then $\mathbf{x}^\star$ is an optimum solution to $P(s, \mathbf{c}, i+1)$. Therefore,

$$V\left(\max\left\{s - a_i, 0\right\}, \mathbf{c} - \mathbf{A}\,\mathbf{e}_i, i+1\right) = \max\left\{\begin{matrix} V\left(s, \mathbf{c}, i+1\right), \\ a_i + V\left(\max\left\{s - a_i, 0\right\}, \mathbf{c} - \mathbf{A}\,\mathbf{e}_i, i+1\right)\end{matrix}\right\}. \tag{3.2.2}$$

By recursion over i, $\mathbf{c}$, and s, we can compute the value $V\left(s, \mathbf{c}', i\right)$ for $s \in \{0, 1 \dots m\,\xi\}$, $\mathbf{c}' \in \mathbb{N}^k$ with $\mathbf{c}' \leq \mathbf{c}$, and $i \in \{0, 1 \dots d\}$ in time $\mathcal{O}\Big(m\,d\,\xi\,\prod_{\ell=1}^{k} c_\ell\Big)$. The solution to the original budgeted problem, denoted by $\mathbf{x}^\star$, is then:

$$x^\star_i = \begin{cases} 0 & \text{if } V\left(s, \mathbf{c}, i\right) = V\left(s, \mathbf{c}, i+1\right) \\ 1 & \text{otherwise.}\end{cases} \tag{3.2.3}$$

With the same technique, we can solve the budgeted problem for all $s \leq m\,\xi$ in time $\mathcal{O}\Big(m\,d\,\xi\,\prod_{\ell=1}^{k} c_\ell\Big)$. In particular, for m-sets, we can do so with a time complexity of $\mathcal{O}\big(m^2\,d\,\xi\big)$.

3.2.2. Simple path. We can solve the *rounded* budgeted simple-path problem 3.1.17 exactly (i.e. $\varepsilon = 1$), using a technique that generalises Dijkstra's algorithm [**133**]. We can solve this problem using Algorithm 14 with a complexity $\mathcal{O}(m\,\xi\ |E| + |V|\,\log|V|)$.

We only consider the case of a directed acyclic graph. Indeed, ignoring the budget constraint, we are trying to solve a variant of the weighted longest-simple-path problem (i.e. the path that maximises the average reward). This problem is notoriously $\mathcal{NP}$-hard [**134**]. However, when restricted to directed acyclic graphs, the problem becomes polynomially solvable [**135, 136**].

We also make the assumption that the weights $\mathbf{a}$ are never zero to simplify the design of the algorithm. This hypothesis is restrictive, but can be lifted at the cost of an increased computational complexity.

Claim 11. Optimisation problem 3.1.17 with $\mathcal{X}$ the set of paths from $u \in V$ to $v \in V$ in the directed acyclic graph $G = (V, E)$ can be solved exactly (i.e. $\varepsilon = 1$) in time $\mathcal{O}(m\,\xi\ |E| + |V|\,\log|V|)$ using the algorithm below.

We consider that the budget value v is fixed throughout and denote by $P(u, s)$ the optimisation problem

$$\max \mathbf{x}^T\,\mathbf{b}$$
$$\text{s.t.}\ \mathbf{x}^T\,\mathbf{a} \geq s$$

$$\mathbf{x} \text{ is a path starting at } u.$$

and $V(u,s)$ its optimum value.

- If $s \leq 0$, $P(u,s)$ is simply the problem of finding the path $\mathbf{x}$ from u to v maximising $\mathbf{x}^T\,\mathbf{b}$. Indeed, since $\mathbf{a}$ has only positive entries, $\mathbf{x}^T\,\mathbf{a} \geq 0 \geq s$, for all $\mathbf{x} \in \mathcal{X}$. Hence, we can compute $V(u,s)$ for all u and for any $s \leq 0$ by Dijkstra's algorithm in time $\mathcal{O}(|E| + |V| \log |V|)$, when Dijkstra's algorithm is implemented with Fibonacci heaps [**137**], because we consider that the graph is acyclic.
- Otherwise, $s > 0$. Let $\mathbf{x}^\star$ an optimum solution to $P(u,s)$. Since $\mathbf{x}^\star$ is a path from u to v, there exists a unique vertex $w \in V$ such that $x^\star_{(u,w)} = 1$, and that $\mathbf{x}^\star - \mathbf{e}_{(u,w)}$ is a path from w to v. In turn, $\mathbf{x}^\star - \mathbf{e}_{(u,w)}$ is an optimum solution to $P\big(w, \max\big\{s - a_{(u,v)}, 0\big\}\big)$. This is a simple extension of the optimum substructure of the path-finding problem [**138**, Section 15.3].

Therefore, we have the following dynamic programming equation:

$$V(u,s) = \max_{w:(u,w)\in E} \big\{ b_{(u,w)} + V\big(w, \max\big\{s - a_{(u,w)}, 0\big\}\big)\big\}\,. \tag{3.2.4}$$

As $\mathbf{a} \in \{1, 2 \dots \xi\}^d$, if $V(u,s')$ is known for all $u \in V$ and all $s' \in \{0, 1 \dots s-1\}$, applying the above relationship enables us to compute $V(u,s)$ for all $u \in V$. By recursion, we can compute $V(u,s)$ for all $s \in \{0, 1 \dots m\,\xi\}$ in time $\mathcal{O}(m\,\xi\;|E| + |V| \log |V|)$.

The solution to $P(u,s)$, denoted by $\mathbf{x}^\star$, can also be computed by recursion. Using the same dynamic programming principle, if $\mathbf{x}^\star(u,s)$ denotes the solution of $P(u,s)$, we have:

$$\mathbf{x}^\star(u,s) = \mathbf{e}_{(u,w^\star(u,s))} + \mathbf{x}^\star\big(w^\star\,(u,s)\,,\; s - a_{(u,w^\star(u,s))}\big) \tag{3.2.5}$$

where the optimum next vertex is given by $w^\star(u,s)$ as:

$$w^\star(u,s) \in \operatorname*{arg\,max}_{w:(u,w)\in E} \big\{ b_{(u,w)} + V\big(w, \max\big(s - a_{(u,w)}, 0\big)\big)\big\}\,. \tag{3.2.6}$$

By recursion, we can compute the solution to $P(u,s)$ for all $s \in \{0, 1 \dots m\,\xi\}$ in time $\mathcal{O}(m\,\xi\;|E|)$ once $V(u,s)$ is known.

Remark 12. If $a_i = 0$ is allowed, then, in some iterations, the budget may remain constant in the lookup phase. This value makes the problem more difficult to solve. In that case, the algorithm must be modified to loop several times for each budget value until reaching a fixed point.

3.2.3. Spanning tree and matroid. Matroid-structured combinatorial problems are ubiquitous (they have been introduced in Section 2.6.3). However, optimising a linear function on a matroid with a budget constraint is not as easy to perform as in the the previous cases. We use a particular case of the approximation algorithm proposed in [**128**]. The main inconvenience is that the solution is provided only with an approximation ratio of $\varepsilon = 1/2$. Unlike knapsack sets and simple paths, when computing the solution for a particular budget, this algorithm is unable to provide optimum (or even approximate) solutions for lower budgets. In this section, we focus on spanning trees, but the generalisation to any matroid is straightforward. For one value of the budget, this approximation algorithm runs in time $\mathcal{O}\big(m\,\xi\;|E|\,\log^2|V| + |V|\,\log^3|E|\big)$.

Claim 13. Optimisation problem 3.1.17 with $\mathcal{X}$ the set of spanning tree paths in the undirected graph $G = (V,E)$ can be solved approximately with $\varepsilon = 1/2$ in time $\mathcal{O}\big(m\,\xi\;|E| + |V|\,\log^3|E|\big)$ using the algorithm below.

The main idea behind this technique is to use Lagrangian relaxation **[139]** to transform the budget constraint into a penalisation in the objective function. Therefore, the budgeted spanning-tree problem $P(s)$ writes:

$$\begin{aligned} &\max \mathbf{x}^T \mathbf{b} && (P(s)) \\ &\text{s.t. } \mathbf{x}^T \mathbf{a} \geq s \\ &\quad \mathbf{x} \text{ is a spanning tree.} \end{aligned}$$

If the budget constraint is assigned the Lagrange multiplier λ, its Lagrange dual is $M(\lambda, s)$, defined as:

$$\begin{aligned} &\max \mathbf{x}^T \mathbf{b} + \lambda \left(\mathbf{x}^T \mathbf{a} - s\right) && (M(\lambda, s)) \\ &\text{s.t. } \mathbf{x} \text{ is a spanning tree.} \end{aligned}$$

As the objective is a piecewise-linear function of λ, the optimum value $\lambda^\star(s)$ can be found using Meggido's search technique **[140]** while solving the linear matroid problem **[116]**, without incurring a higher computational complexity:

$$\lambda^\star(s) \in \arg\min_{\lambda \geq 0} M(\lambda, s)\,. \tag{3.2.7}$$

If $\lambda^\star(s)$ corresponds to a spanning tree that satisfies the budget constraint, with a slightly lower value of λ, the constraint is no more satisfied. Conversely, if $\lambda^\star(s)$ yields a solution that does not satisfy the budget constraint, a higher value of λ will. Therefore, we use two solutions: $\mathbf{x}^+(s)$, computed from $M[\lambda^\star(s) + \varepsilon, s]$, and $\mathbf{x}^-(s)$, calculated from $M[\lambda^\star(s) - \varepsilon, s]$. Iteratively, we then 'refine' these solutions to bring them closer together, so that there is only one edge that differs between them, while still having the same objective value: this yields an additive-approximation algorithm. We finally apply this procedure on all pairs of distinct edges, so that we can build a multiplicative-approximation algorithm.

More formally, the algorithm is made of four steps. These are formalised in Algorithm 15.

1. **Lagrangian relaxation:** Let $\mathcal{L}(\lambda, s)$ be the set of optimum solutions to the Lagrange dual problem $M(\lambda, s)$. One evaluation of $M(\lambda, s)$ can be performed in polynomial time: it is equivalent to maximizing a linear function over a matroid, for instance using a greedy algorithm **[116]**. Furthermore, $\lambda^\star(s)$ can be found using Meggido's search technique **[140]**, as it involves minimizing a piecewise linear function.
2. **Candidate solutions:** For an arbitrarily small $\varepsilon > 0$, if $|\lambda - \lambda^\star(s)| < \varepsilon$, we must have that $\mathcal{L}(\lambda, s) \subset \mathcal{L}[\lambda^\star(s)\,, s]$. Therefore, by solving the Lagrangian relaxation of the problem for $\lambda^+(s) = \lambda^\star(s) + \varepsilon$ and $\lambda^-(s) = \lambda^\star(s) - \varepsilon$, we obtain two solutions $\mathbf{x}^+$ and $\mathbf{x}^-$ in $\mathcal{L}[\lambda^\star(s)\,, s]$ with $\mathbf{a}^T \mathbf{x}^+ \geq s$ and $\mathbf{a}^T \mathbf{x}^- \leq s$.
3. **Solution refining:** We now use an iterative procedure in order to find a good solution using candidates $\mathbf{x}^+$ and $\mathbf{x}^-$. We consider two distinct edges e and e' in E such that $x_e^+ = x_{e'}^- = 1$ and $x_{e'}^+ = x_e^- = 0$. We define a new solution $\mathbf{x} = \mathbf{x}^+ - \mathbf{e} + \mathbf{e}'$. For the next iteration, if $\mathbf{a}^T \mathbf{x} \geq s$, then we replace $\mathbf{x}^+$ by $\mathbf{x}$ and otherwise we replace $\mathbf{x}^-$ by $\mathbf{x}$. We repeat this procedure until $\mathbf{x}^+$ and $\mathbf{x}^-$ differ by exactly one element. Finally, $\mathbf{x}^+$ is the refined solution.

   At each step of this procedure, the following equalities always hold:

$$(\mathbf{b} + \lambda\,\mathbf{a})^T\ \mathbf{x} = (\mathbf{b} + \lambda\,\mathbf{a})^T\ \mathbf{x}^+ = (\mathbf{b} + \lambda\,\mathbf{a})^T\ \mathbf{x}^-. \tag{3.2.8}$$

Therefore, at each iteration, $\mathbf{x} \in \mathcal{L}(\lambda, s)$. Since both $\mathbf{x}^+$ and $\mathbf{x}^-$ are in $\mathcal{L}(\lambda, s)$, we have:

$$\mathbf{b}^T \mathbf{x}^- + \lambda \left(\mathbf{a}^T \mathbf{x}^- - s\right) \geq \mathbf{b}^T \mathbf{x}^\star + \lambda \left(\mathbf{a}^T \mathbf{x}^\star - s\right). \tag{3.2.9}$$

As $\mathbf{x}^-$ is slightly infeasible with respect to the budget constraint, i.e. $\mathbf{a}^T \mathbf{x}^- < s \leq \mathbf{a}^T \mathbf{x}^\star$, its objective value $\mathbf{b}^T \mathbf{x}^-$ is slightly higher than that of the optimum solution $\mathbf{x}^\star$:

$$\mathbf{b}^T \mathbf{x}^- \geq \mathbf{b}^T \mathbf{x}^\star. \tag{3.2.10}$$

Because, at the end of the refinement procedure, $\mathbf{x}^+$ and $\mathbf{x}^-$ differ by at most two elements, the following inequalities hold:

$$\mathbf{b}^T \mathbf{x}^+ \geq \mathbf{b}^T \mathbf{x}^- - \max_{e \in E} b_e \geq \mathbf{b}^T \mathbf{x}^\star - \max_{e \in E} b_e. \tag{3.2.11}$$

At the end of the first three steps, $\mathbf{x}$ is a solution such that $\mathbf{a}^T \mathbf{x} \geq s$ (i.e. feasible) and $\mathbf{b}^T \mathbf{x} \geq \mathbf{b}^T \mathbf{x}^\star - \max_{e \in E} b_e$ (i.e. close to the optimum solution, with an additive approximation term).

4. **A $1/2$-optimum solution:** Finally, we search over the two edges with the largest weight to obtain a constant multiplicative approximation factor. For all sets of two edges $E'' \subset E$, $|E''| = 2$, we define a reduced graph $G'(E'') = [V, E'(E'')]$ where

$$E'(E'') = \left\{ e \in E \setminus E'' \;\middle|\; b_e \leq \min_{e'' \in E''} b_{e''} \right\}. \tag{3.2.12}$$

For each such pair of edges, we apply the first three steps where G and s are replaced by $G'(E'')$ and $s'(E'') = s - \sum_{e'' \in E''} b_{e''}$ (i.e. consider that those two edges are always part of the solution); let $\mathbf{x}'[G'(E''), s'(E'')]$ denote the solution found at the end of the third step, if it exists.
The solution that is returned corresponds to the pair $E''^\star$ that maximises $\mathbf{a}^T \mathbf{x}'[G'(E''), s'(E'')]$:

$$E''^\star \in \operatorname*{arg\,max}_{\substack{E'' \subset E, \\ |E''|=2}} \left\{ \mathbf{a}^T \mathbf{x}'[G'(E''), s'(E'')] \right\}. \tag{3.2.13}$$

More precisely, return the following solution:

$$\mathbf{x}(E''^\star) + \sum_{e \in E''^\star} \mathbf{x}_e. \tag{3.2.14}$$

This final loop yields a $1/2$ optimum solution in time $\mathcal{O}\big(|E| \log^2 |V| + |V| \log^3 |E|\big)$ by the same arguments as those used in **[128, 129]**.

3.2.4. Matching and matroid intersection. One of the major applications of combinatorial bandits is for the online matching problem (Section 2.5.1). Matching is an instance of matroid intersection (Section 2.6.4). The previous algorithmic technique can be generalised to intersections of two matroids, while still having a $1/2$-approximation algorithm. The previous drawback of inability to solve for several budget values is still present.

The algorithm we propose is made of four steps and is very similar to that of **[129]**, which itself is inspired by the algorithm for matroids of **[128]**, exactly like our algorithm for matroids. Intuitively, it works in the same way as previously, with the exception that four edges must be fixed in order to guarantee a constant approximation ratio. The solution-refinement step is implemented using augmenting paths instead of edge switching. Our algorithm is formalised in Algorithm 16.

CLAIM 14. Optimisation problem 3.1.17 with $\mathcal{X}$ the set of bipartite matchings in the bipartite graph $G = (V_1, V_2, E)$ can be solved approximately with $\varepsilon = 1/2$ in time $\mathcal{O}\Big(|V|^3\ |E|^4\Big)$ using the algorithm below.

1. **Lagrangian relaxation:** We define the budgeted optimisation problem $P(s)$

$$\begin{aligned} &\max \mathbf{x}^T \mathbf{b} && (P(s)) \\ &\text{s.t. } \mathbf{x}^T \mathbf{a} \geq s \\ &\quad \mathbf{x} \text{ is a matching.} \end{aligned}$$

   If the budget constraint is assigned the Lagrange multiplier λ, its Lagrange dual is $M(\lambda, s)$:

$$\begin{aligned} &\max \mathbf{x}^T \mathbf{b} + \lambda \left(\mathbf{x}^T \mathbf{a} - s\right) && (M(\lambda, s)) \\ &\text{s.t. } \mathbf{x} \text{ is a matching.} \end{aligned}$$

   Let $\mathcal{L}(\lambda, s)$ be the set of optimum solutions to the Lagrange dual problem $M(\lambda, s)$. One evaluation of $M(\lambda, s)$ can be performed in polynomial time: it is equivalent to maximizing a linear function over an intersection of matroids [**119**] (for bipartite matchings, the Hungarian algorithm can be used [**141**]). Furthermore, $\lambda^\star(s)$ can be found using Meggido's search technique [**140**], as it involves minimizing a piecewise linear function.
2. **Candidate solutions:** For an arbitrarily small $\varepsilon > 0$, if $|\lambda - \lambda^\star(s)| < \varepsilon$, we must have that $\mathcal{L}(\lambda, s) \subset \mathcal{L}(\lambda^\star(s)\,, s)$. Therefore, by solving the Lagrangian relaxation of the problem for $\lambda^+(s) = \lambda^\star(s) + \varepsilon$ and $\lambda^-(s) = \lambda^\star(s) - \varepsilon$, we obtain two solutions $\mathbf{x}^+$ and $\mathbf{x}^-$ in $\mathcal{L}[\lambda^\star(s)\,, s]$ with $\mathbf{a}^T\,\mathbf{x}^+ \geq s$ and $\mathbf{a}^T\,\mathbf{x}^- \leq s$.
3. **Solution refining:** We now use an iterative procedure in order to find a good solution using candidates $\mathbf{x}^+$ and $\mathbf{x}^-$. Define their symmetric difference $\mathbf{x}' = \mathbf{x}^+ \oplus \mathbf{x}^-$. This symmetric difference $\mathbf{x}'$ is made of a disjoint union of paths and cycles. We take $\mathbf{x}''$ as one of such paths or cycles, and define the new solution $\mathbf{x} = \mathbf{x}^- \oplus \mathbf{x}''$. If $\mathbf{a}^T\,\mathbf{x} \geq s$, we then replace $\mathbf{x}^+$ by $\mathbf{x}$ and otherwise we replace $\mathbf{x}^-$ by $\mathbf{x}$. We repeat this procedure until $\mathbf{x}^+$ and $\mathbf{x}^-$ differ by at most two elements (the symmetric difference $\mathbf{x}^+ \oplus \mathbf{x}^-$ decreases at each step). Finally, $\mathbf{x}^+$ is the refined solution.
   If $\mathbf{a}^T\,\mathbf{x} \geq s$, we then replace $\mathbf{x}^+$ by $\mathbf{x}$ and otherwise we replace $\mathbf{x}^-$ by $\mathbf{x}$. We epeat this procedure until $\mathbf{x}^+$ and $\mathbf{x}^-$ differ by exactly one element. Finally, we return $\mathbf{x}^+$.
   At each step of this procedure, the following equalities always hold:

$$(\mathbf{b} + \lambda\,\mathbf{a})^T\ \mathbf{x} = (\mathbf{b} + \lambda\,\mathbf{a})^T\ \mathbf{x}^+ = (\mathbf{b} + \lambda\,\mathbf{a})^T\ \mathbf{x}^-. \tag{3.2.15}$$

   Therefore, at each iteration, $\mathbf{x} \in \mathcal{L}(\lambda, s)$. Since both $\mathbf{x}^+$ and $\mathbf{x}^-$ are in $\mathcal{L}(\lambda, s)$, we have:

$$\mathbf{b}^T\mathbf{x}^- + \lambda\,\left(\mathbf{a}^T\,\mathbf{x}^- - s\right) \geq \mathbf{b}^T\,\mathbf{x}^\star + \lambda\,\left(\mathbf{a}^T\,\mathbf{x}^\star - s\right). \tag{3.2.16}$$

   As $\mathbf{x}^-$ is slightly infeasible with respect to the budget constraint, i.e. $\mathbf{a}^T\,\mathbf{x}^- \leq s \leq \mathbf{a}^T\,\mathbf{x}^\star$, its objective value $\mathbf{b}^T\,\mathbf{x}^-$ is slightly higher than that of the optimum solution $\mathbf{x}^\star$:

$$\mathbf{b}^T\,\mathbf{x}^- \geq \mathbf{b}^T\,\mathbf{x}^\star. \tag{3.2.17}$$

   Because, at the end of the refinement procedure, $\mathbf{x}^+$ and $\mathbf{x}^-$ differ by at most one element, we have:

$$\mathbf{b}^T\,\mathbf{x}^+ \geq \mathbf{b}^T\,\mathbf{x}^- - 2\,\max_{e \in E} b_e \geq \mathbf{b}^T\,\mathbf{x}^\star - 2\,\max_{e \in E} b_e. \tag{3.2.18}$$

At the end of the first three steps, $\mathbf{x}$ is a solution such that $\mathbf{a}^T\,\mathbf{x} \geq s$ (i.e. feasible) and $\mathbf{b}^T\,\mathbf{x} \geq \mathbf{b}^T\,\mathbf{x}^\star - 2\,\max_{e\in E} b_e$ (i.e. close to the optimum solution, with an additive approximation term).

4. **A $1/2$-optimum solution:** Finally, we search over the four edges with the largest weight to obtain a constant multiplicative approximation factor. For all sets of four edges $E'' \subset E$, $|E''| = 4$, we define a reduced graph $G' = (V, E')$ where

$$E' = \left\{ e \in E \setminus E'' \;\middle|\; b_e \leq \min_{e''\in E''} b_{e''} \right\}. \tag{3.2.19}$$

For each such 4-tuple of edges, we apply the first three steps where G and s are replaced by G' and $s - \sum_{e''\in E''} b_{e''}$ (i.e. consider that those four edges are always part of the solution); let $\mathbf{x}'(G', s')$ denote the solution found at the end of the third step, if it exists. The solution that is returned corresponds to the pair $E''^\star$ that maximises $\mathbf{a}^T\,\mathbf{x}'(G', s')$:

$$E''^\star \in \operatorname*{arg\,max}_{\substack{E''\subset E,\\ |E''|=4}} \left\{ \mathbf{a}^T\,\mathbf{x}'(G', s') \right\}. \tag{3.2.20}$$

More precisely, we return the following solution:

$$\mathbf{x}(E''^\star) \cup E''^\star. \tag{3.2.21}$$

This final loop yields a $1/2$ optimum solution in time $\mathcal{O}\Big(|V|^3\;|E|^4\Big)$ by the same arguments as that used in [**128, 129**].

3.3. Exact implementation of ESCB

To provide a baseline to compare our algorithms, we use an MISOCP (*mixed-integer second-order cone program*) formulation of the optimisation program behind ESCB. Existing optimisation solvers like CPLEX [**142**], Gurobi [**143**], Mosek [**144**], Parajito [**145**], SCIP [**146**], or Xpress [**147**] can fully exploit this formulation. In practice, they can already solve ESCB's problem at each round very efficiently, even in large dimension, although their worst-case complexity is exponential, $\mathcal{O}\big(2^d\big)$.

At each iteration, ESCB solves one such problem:

$$\max_{\mathbf{x}\in\mathcal{X}} \left\{ \mathbf{x}^T\,\hat{\boldsymbol{\theta}}(n) + \sqrt{\frac{f(n)}{2}\sum_{i=1}^{d}\frac{x_i}{\mathbf{T}_i(n)}} \right\}. \tag{3.3.1}$$

The square root in the objective function can be thought of as a geometric mean, which is a special case of hyperbolic constraint [**148**]. The first step of the reformulation is to replace the square root by a single variable, t, and to add the right constraint:

$$\begin{array}{ll} \max\limits_{\mathbf{x},t} & \mathbf{x}^T\,\hat{\boldsymbol{\theta}}(n) + \sqrt{\frac{f(n)}{2}}\,t \\ \text{s.t.} & t^2 \leq \sum_{i=1}^{d}\frac{x_i}{\mathbf{T}_i(n)} \\ & \mathbf{x}\in\mathcal{X}. \end{array} \tag{3.3.2}$$

Applying the transformation proposed in [**148**, Section 2.3], the hyperbolic constraint can be written as a SOCP:

$$\begin{aligned} \max_{\mathbf{x},t} \quad & \mathbf{x}^T \hat{\boldsymbol{\theta}}(n) + \sqrt{\tfrac{f(n)}{2}}\, t \\ \text{s.t.} \quad & \left\| \begin{bmatrix} 2\,t \\ \sum_{i=1}^d \frac{x_i}{\mathbf{T}_i(n)} - 1 \end{bmatrix} \right\| \leq \sum_{i=1}^d \frac{x_i}{\mathbf{T}_i(n)} + 1 \\ & \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{3.3.3}$$

Even though the linear constraints defining $\mathcal{X}$ ensure that optimising a linear objective over $\mathcal{X}$ yields an integer solution, this is no more the case with the new formulation. Hence, integrality constraints must be added for the relevant variables. ESCB's program (3.3.1) is therefore:

$$\begin{aligned} \max_{\mathbf{x},t} \quad & \mathbf{x}^T \hat{\boldsymbol{\theta}}(n) + \sqrt{\tfrac{f(n)}{2}}\, t \\ \text{s.t.} \quad & \left\| \begin{bmatrix} 2\,t \\ \sum_{i=1}^d \frac{x_i}{\mathbf{T}_i(n)} - 1 \end{bmatrix} \right\| \leq \sum_{i=1}^d \frac{x_i}{\mathbf{T}_i(n)} + 1 \\ & \mathbf{x} \in \mathcal{X} \\ & \mathbf{x} \in \{0,1\}^d. \end{aligned} \tag{3.3.4}$$

3.4. Numerical results

We evaluate the performance of TS (Section 2.5.3.1), CUCB (Section 2.5.3.2), ESCB (Section 2.5.4), and AESCB (Section 3.1) through numerical experiments in order to compare their regret and the computation time. ESCB is implemented by casting the optimisation problem as an MISOCP (Section 3.3) and using CPLEX as MISOCP solver [**142**].

All experiments are repeated 10 times, only averages are reported. In the plots, the error bars correspond to the estimated 95% confidence intervals based on the measurements (under the standard hypothesis of Gaussian distribution). As done in most prior work [**56, 109**], we simulate ESCB and AESCB using $f(n) = \log n$, neglecting the $4\,m \log\log n$ term, as this choice gives better performance in practice. This issue is discussed in [**109**]; intuitively, a lower value of $f(n)$ decreases the upper bound on the reward for each solution, which then only holds with a lower probability than with a higher $f(n)$, but allows more aggressive exploitation in many cases. This phenomenon is already known for classical bandits [**56**]. The Julia [**37**] implementations of the four algorithms (ESCB is implemented using JuMP [**149**]) as well as the code to run the experiments are made available online[1].

3.4.1. Experimental setting. We run experiments on four different combinatorial sets, for various problem sizes (indicated by d). All rewards follow a Bernoulli distribution.

- For m-sets, we choose $m = \lfloor d/3 \rfloor$. Regarding the rewards, $\theta_i = 0.55$ for $i \leq d/2$ and $\theta_i = 0.4$ for $i > d/2$. We use a time horizon of $T = 1000$. Any optimum solution takes $\lfloor d/3 \rfloor$ elements among the $\lfloor d/2 \rfloor$ first ones.
- For simple paths, we consider the graph $G = (V, E)$ a complete directed acyclic graph: $(i, j) \in E$ if and only if $i < j$. The source is 1 and the destination is $|V|$. The rewards are defined as $\theta_{(i,j)} = 0.4$ for $(i, j) \neq (1, |V|)$ and $\theta_{(1,|V|)} = 0.55$. We have $d = |V|\,(|V| - 1)\,/2$ and $m = |V| - 1$. We choose $T = 5000$. The optimum path is $\{(i, i+1)\,, \forall i \in \{1, 2 \ldots |V| - 1\}\}$.

[1] CombinatorialBandits.jl

- For spanning trees, we consider $G = (V, E)$ a complete undirected graph. The rewards are set so that $\theta_{(i,j)} = 0.4$ for all edges $(i, j) \in E$ with $i \neq 1$ and $\theta_{(1,j)} = 0.55$ for each vertex $j \in V \setminus \{1\}$. We have $d = |V|(|V| - 1)/2$ and $m = |V| - 1$. We use $T = 1000$. The optimum decision is a star network centred on 1, i.e. $\{(1, j) \mid j \in V \setminus \{1\}\}$.
- For matchings, we consider a complete bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2|$. Regarding the rewards, $\theta_{(i,j)} = 0.4$ for all edges $(i, j) \in E$ where $i \neq j$ and $\theta_{(i,i)} = 0.55$ for each vertex $i \in V$. We have $d = |V_1|\ |V_2|$ and $m = |V_1| = |V_2|$. We use $T = 1000$. The optimum decision is $\mathbf{x}^\star = \{(i, j) \mid i \in V_1, j \in V_2, i = j\}$.

The choice of average rewards is such that the Bernoulli random variables have a significant variance: if the components of $\boldsymbol{\theta}$ were too close to either zero or one, their variance would be very low, and the bandit problem would almost reduce to deterministic combinatorial optimisation. Indeed, the variance of a Bernoulli random variable with a success probability p has variance $p\,(1 - p)$. For instance, with a success probability of $p = 0.95$, the variance of the reward is only $0.95 \times 0.05 = 0.0475$; on the other hand, if $p = 0.55$, the variance is $0.55 \times 0.45 = 0.2475$.

A low variance would unfairly advantage TS, as this algorithm is very greedy. We checked its performance in this case, and it clearly outperforms all other bandit algorithms in terms of regret.

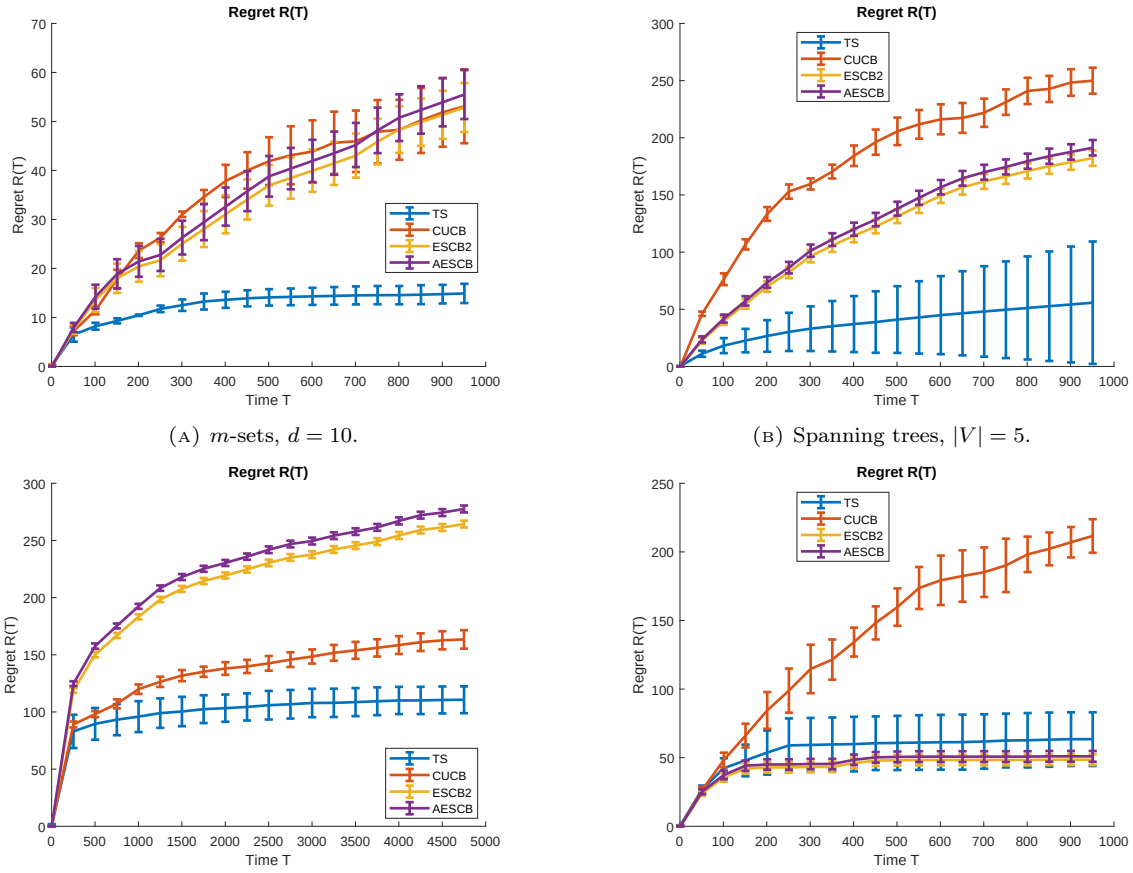
3.4.2. Conservative choice of discretisation parameters. In our first set of experiments, we choose a discretisation parameter that should not significantly contribute to the regret of AESCB, from a theoretical point of view. We decide to take $\delta_n = 1/T$, where T is the time horizon. Due to the rather large time horizons in our experiments, δ_n is ensured to be lower than $\Delta_{\min}/4$, and is thus never be a source of regret for AESCB.

Regret. In Figure 3.4.1, we present the expected regret of all four algorithms (with 95% confidence intervals) averaged over 10 sample paths. The regret of AESCB is very close to that of ESCB, for all the tested combinatorial sets: the approximation comes at virtually no cost in terms of regret. Both ESCB and AESCB outperform CUCB for matchings and spanning trees, whereas CUCB performs better than ESCB and AESCB for paths. TS performs well on average, even though it does not provide the best average regret for matchings; however, its regret has a lot of variability across sample paths, performing quite poorly on some of them. Therefore, it is a 'risky' algorithm to use, unlike the others.

Computation time. AESCB's computation times (Table 3) do not compare well to an off-the-shelf state-of-the-art MISOCP solver, especially when the problem size increases. This was expected from the choice of discretisation parameter: $\delta_n = 1/T$ ensures that the regret due to the approximation algorithm is very small, albeit at a high computational cost.

3.4.3. Practical choice of parameters. In our second set of experiments, we use a more aggressive choice of discretisation parameters: $\delta_n = 1/\log T$. With our time horizons ($T = 1000$ or $T = 5000$, depending on the combinatorial set), this choice is no more ensured to have no impact on the regret, as per our theoretical analysis. However, in practice, the regret does not significantly change between the two sets of experiments, and the plots in Figure 3.4.1 still accurately depict the situation with this new choice of parameters.

In Table 4, we present the computation times required to select an arm at round $n = 1000$ for ESCB and AESCB (again, with 95% confidence intervals) averaged over several sample paths, as a function of the problem dimension d. For most experiments, we average over 10 samples; however, for ESCB and AESCB in the case of paths, 100 samples were required to have disjoint confidence intervals. We observe that the computation time for AESCB indeed seems to grow slowly



(A) m-sets, $d = 10$.

(B) Spanning trees, $|V| = 5$.

(C) Paths, $|V| = 10$.

(D) Matchings, $|V| = 10$.

FIGURE 3.4.1. Expected regret of the algorithms.

in d. Moreover, the computation times for all algorithms appear of the same magnitude, except for matchings.

3.4.4. Parameter tuning. The function $f(n)$ for ESCB and AESCB had been tuned in previous experiments to increase its performance. There is no way to replicate this modification for CUCB. However, we can define a new parameter $\alpha \in [0, 1/2]$ to allow tuning the confidence radius for all algorithms. $\alpha = 1/2$ corresponds to the original algorithms, while $\alpha = 0$ makes them all behave as pure-exploration policies (i.e. no confidence radius).

This variant of CUCB selects its decision as:

$$\mathbf{x}(n) \in \arg\max_{\mathbf{x} \in \mathcal{X}} \left\{ \hat{\boldsymbol{\theta}}^T(n)\, \mathbf{x} + \alpha \sum_{i=1}^{d} x_i \frac{\log n}{t_i(n)} \right\}. \tag{3.4.1}$$

Problem	Algorithm	Time	[s]	
m-sets	ESCB	0.01 ± 0.00	0.02 ± 0.01	1.24 ± 0.05
	AESCB	0.01 ± 0.00	0.04 ± 0.00	0.99 ± 0.16
		$(d = 10)$	$(d = 20)$	$(d = 50)$
Paths	ESCB	0.00 ± 0.00	0.02 ± 0.00	0.10 ± 0.18
	AESCB	0.05 ± 0.00	0.12 ± 0.01	1.24 ± 0.04
		$(d = 10)$	$(d = 45)$	$(d = 190)$
Trees	ESCB	0.02 ± 0.00	0.06 ± 0.08	0.20 ± 0.05
	AESCB	0.01 ± 0.00	0.06 ± 0.01	0.40 ± 0.05
		$(d = 10)$	$(d = 45)$	$(d = 190)$
Matchings	ESCB	0.01 ± 0.00	0.26 ± 0.11	
	AESCB	0.01 ± 0.00	3.57 ± 1.29	
		$(d = 4)$	$(d = 25)$	

TABLE 3. Computing times of ESCB (above) and AESCB (below) using a conservative choice of parameters.

Problem	Algorithm	Time	[s]	
m-sets	ESCB	0.01 ± 0.00	0.02 ± 0.01	1.24 ± 0.03
	AESCB	0.00 ± 0.00	0.01 ± 0.00	0.10 ± 0.10
	CUCB	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	TS	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
		$(d = 10)$	$(d = 20)$	$(d = 50)$
Paths	ESCB	0.00 ± 0.00	0.02 ± 0.00	0.11 ± 0.04
	AESCB	0.00 ± 0.00	0.00 ± 0.00	0.05 ± 0.00
	CUCB	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
	TS	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
		$(d = 10)$	$(d = 45)$	$(d = 190)$
Trees	ESCB	0.02 ± 0.00	0.06 ± 0.05	0.20 ± 0.03
	AESCB	0.00 ± 0.00	0.02 ± 0.00	0.04 ± 0.01
	CUCB	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
	TS	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
		$(d = 10)$	$(d = 45)$	$(d = 190)$
Matchings	ESCB	0.01 ± 0.00	0.26 ± 0.06	
	AESCB	0.00 ± 0.00	0.18 ± 0.01	
	CUCB	0.00 ± 0.00	0.00 ± 0.00	
	TS	0.00 ± 0.00	0.00 ± 0.00	
		$(d = 4)$	$(d = 25)$	

TABLE 4. Computing times of ESCB, AESCB (below), CUCB, and TS. AESCB's discretisation parameter has been chosen in an aggressive way ($\delta_n = 1/\log T$).

ESCB uses the following decision rule:

$$\mathbf{x}(n) \in \operatorname*{arg\,max}_{\mathbf{x}\in\mathcal{X}} \left\{ \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x} + \sqrt{\alpha\, f(n) \sum_{i=1}^{d} \frac{x_i}{t_i(n)}} \right\}. \tag{3.4.2}$$

Finally, AESCB chooses a solution $\mathbf{x}(n)$ such that:

$$\operatorname*{arg\,max}_{\mathbf{x}\in\mathcal{X}} \left\{ \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x} + \sqrt{\alpha\, f(n) \sum_{i=1}^{d} \frac{x_i}{t_i(n)}} \right\} \leq \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x}(n) + \frac{1}{\varepsilon_n} \sqrt{\alpha\, f(n) \sum_{i=1}^{d} \frac{x_i(n)}{t_i(n)}}. \tag{3.4.3}$$

The results are shown in Table 5. As previously, AESCB's performance is extremely close to that of ESCB. Even with the best choice of parameter α, CUCB cannot outperform ESCB or AESCB in situations where it did not with the standard version of the algorithm.

TABLE 5. Regret of ESCB, AESCB, and CUCB with tuning of the confidence radius.

Problem	Algorithm	Regret				
		$(\alpha = 0.1)$	$(\alpha = 0.2)$	$(\alpha = 0.3)$	$(\alpha = 0.4)$	$(\alpha = 0.5)$
m-sets $(d = 50)$	ESCB	31.6 ± 24.8	27.9 ± 17.9	24.9 ± 14.3	25.1 ± 13.5	$\mathbf{24.9 \pm 13.0}$
	AESCB	36.1 ± 25.7	32.5 ± 15.6	29.6 ± 15.1	28.7 ± 15.4	$\mathbf{27.4 \pm 14.1}$
	CUCB	164.8 ± 129.9	73.6 ± 86.8	27.5 ± 10.3	$\mathbf{16.6 \pm 5.1}$	24.8 ± 13.9
Paths $(d = 190)$	ESCB	276.4 ± 3.6	275.0 ± 4.5	275.0 ± 4.5	$\mathbf{275.0 \pm 4.4}$	$\mathbf{275.0 \pm 4.4}$
	AESCB	284.0 ± 3.8	283.2 ± 6.1	283.2 ± 7.1	$\mathbf{282.3 \pm 4.6}$	$\mathbf{282.3 \pm 4.6}$
	CUCB	95.9 ± 43.0	74.4 ± 13.4	$\mathbf{71.7 \pm 2.1}$	84.5 ± 8.9	102.2 ± 5.7
Trees $(d = 190)$	ESCB	138.4 ± 54.3	137.1 ± 51.6	$\mathbf{128.1 \pm 37.9}$	148.1 ± 33.3	130.1 ± 33.8
	AESCB	166.0 ± 57.9	166.0 ± 52.9	$\mathbf{155.2 \pm 38.9}$	179.2 ± 35.2	136.6 ± 34.7
	CUCB	544.1 ± 30.6	225.8 ± 79.4	$\mathbf{205.5 \pm 73.9}$	290.5 ± 38.7	327.8 ± 47.2
Matchings $(d = 25)$	ESCB	108.9 ± 119.4	108.1 ± 114.2	108.1 ± 114.4	$\mathbf{325.1 \pm 107.9}$	$\mathbf{325.1 \pm 107.9}$
	AESCB	360.2 ± 123.3	360.1 ± 123.7	357.9 ± 117.4	357.3 ± 113.9	$\mathbf{357.0 \pm 113.3}$
	CUCB	838.2 ± 423.4	$\mathbf{365.4 \pm 104.2}$	431.1 ± 60.6	477.9 ± 106.6	566.9 ± 76.7

3.5. Regret upper bound for AESCB

Finally, we provide a complete proof of Theorem 5.

THEOREM 4. *The regret of AESCB with parameters $(\varepsilon_t, \delta_t)$ admits the following upper bound for all $T \in \mathbb{N}_0$:*

$$R(T) \leq C_4(m) + \frac{2\, d\, m^3}{\Delta_{\min}^2} + \frac{24\, d\, f(T)}{\left(\min_{t\leq T} \varepsilon_t\right)^2 \Delta_{\min}} \left\lceil \frac{\log m}{1.61} \right\rceil^2 + 4 \sum_{t=1}^{T} \delta_t\, \mathbb{1}_{\Delta_{\min} < 4\,\delta_t} \tag{3.5.1}$$

with $f(t) = \log t + 4\, m \log\log t$ and $C_4(m)$ a positive number that solely depends on m.

3.5.1. Generic regret bound. We decompose the regret based on three events:

- $\mathcal{G}(n)$: the estimate $\hat{\boldsymbol{\theta}}(n)$ deviates abnormally from $\boldsymbol{\theta}$. Formally:

$$\mathcal{G}(n) = \left\{ \boldsymbol{\theta}^T \, \mathbf{x}^\star \geq \hat{\boldsymbol{\theta}}^T(n) \, \mathbf{x}^\star + \sqrt{\mathbf{x}^{\star T} \, \boldsymbol{\sigma}^2(n)} \right\}. \tag{3.5.2}$$

- $\mathcal{H}(n)$: the reward of the decision chosen at round n is poorly estimated. This event is decomposed along all subarms $i \in \{1, 2 \dots d\}$ as $\mathcal{H}_i(n)$:

$$\mathcal{H}_i(n) = \left\{ x_i(n) = 1, \quad \left| \hat{\theta}_i(n) - \theta_i \right| \geq \frac{\Delta_{\min}}{2\, m} \right\}, \tag{3.5.3}$$

$$\mathcal{H}_n = \bigcup_{i=1}^{d} \mathcal{H}_i(n) \, . \tag{3.5.4}$$

- $\mathcal{I}(n)$: the reward gap of the decision chosen at round n is small:

$$\mathcal{I}(n) = \left\{ \Delta_{\mathbf{x}(n)} \leq 4 \, \delta_n \right\}. \tag{3.5.5}$$

Of course, most of the time, $\overline{\mathcal{G}(n)} \cap \overline{\mathcal{H}(n)}$ occurs, since both $\mathcal{G}(n)$ and $\mathcal{H}(n)$ have a small probability of occurring. Therefore, $\mathcal{G}(n)$ and $\mathcal{H}(n)$ only cause a constant regret. Also, $\mathcal{I}(n)$ causes a regret that is at most $4\,\delta_n$, by definition. For all $\mathbf{x} \in \mathcal{X}$ and $n \in \mathbb{N}_0$, we define the exploration bonus $E(\mathbf{x}, n)$ of arm $\mathbf{x}$ at round n:

$$E(\mathbf{x}, n) = \sqrt{\mathbf{x}^T \, \boldsymbol{\sigma}^2(n)}. \tag{3.5.6}$$

By definition (2.2.4), the regret is decomposed along rounds as:

$$\begin{aligned} R(T) &= \mathbb{E}\left\{ \sum_{n=1}^{T} \Delta(n) \right\} \\ &= \mathbb{E}\left\{ \sum_{n=1}^{T} \Delta_{\mathbf{x}(n)} \, \mathbb{1}_{\mathbf{x}(n) \neq \mathbf{x}^\star} \right\}. \end{aligned}$$

Decomposing according to the occurrence of $\mathcal{G}(n)$, $\mathcal{H}(n)$, and $\mathcal{I}(n)$, we get:

$$\begin{aligned} R(T) \leq \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{G}(n)\} \; \Delta_{\mathbf{x}(n)} \right\} + \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{H}(n)\} \; \Delta_{\mathbf{x}(n)} \right\} + \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{I}(n)\} \; \Delta_{\mathbf{x}(n)} \right\} \\ + \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\left\{ \overline{\mathcal{G}}(n) \, , \overline{\mathcal{H}}(n) \, , \overline{\mathcal{I}}(n) \, , \mathbf{x}(n) \neq \mathbf{x}^\star \right\} \; \Delta_{\mathbf{x}(n)} \right\}. \end{aligned}$$

Let $\overline{\varepsilon}_T = \min_{n \in \{1,2 \dots T\}} \varepsilon_n$. The last term can then be rewritten based on the following event:

$$\mathcal{F}(n) = \left\{ \Delta_{\mathbf{x}(n)} \leq \frac{4}{\overline{\varepsilon}_T} \, E[\mathbf{x}(n) \, , n] \right\}. \tag{3.5.7}$$

The last term of the regret bound depends on $\left(\overline{\mathcal{G}}(n) \cap \overline{\mathcal{H}}(n) \cap \overline{\mathcal{I}}(n) \cap \{\mathbf{x}(n) \neq \mathbf{x}^\star\} \right)$, and this event is implied by $\mathcal{F}(n)$, i.e. $\left(\overline{\mathcal{G}}(n) \cap \overline{\mathcal{H}}(n) \cap \overline{\mathcal{I}}(n) \cap \{\mathbf{x}(n) \neq \mathbf{x}^\star\} \right) \subset \mathcal{F}(n)$. Indeed, assuming that $\overline{\mathcal{G}}(n) \cap \overline{\mathcal{H}}(n) \cap \overline{\mathcal{I}}(n) \cap \{\mathbf{x}(n) \neq \mathbf{x}^\star\}$ occurs,

$$\boldsymbol{\theta}^T \, \mathbf{x}^\star \leq \hat{\boldsymbol{\theta}}^T(n) \, \mathbf{x}^\star + E(\mathbf{x}^\star, n) \, , \text{ as } \overline{\mathcal{G}}(n) \text{ occurs}$$

$$
\begin{aligned}
&\leq \max_{\mathbf{x}\in\mathcal{X}} \left\{ \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x} + E(\mathbf{x}, n) \right\} \\
&\leq \delta_t + \hat{\boldsymbol{\theta}}^T(n)\ \mathbf{x}(n) + \frac{1}{\varepsilon_n} E[\mathbf{x}(n)\,, n]\,, \text{ due to AESCB's approximation} \\
&\leq \delta_t + \boldsymbol{\theta}^T(n)\ \mathbf{x}(n) + \frac{\Delta_{\mathbf{x}(n)}}{2} + \frac{1}{\varepsilon_n} E[\mathbf{x}(n)\,, n]\,, \text{ as } \overline{\mathcal{H}}(n) \text{ occurs} \\
&\leq \boldsymbol{\theta}^T(n)\ \mathbf{x}(n) + \frac{3}{4}\,\Delta_{\mathbf{x}(n)} + \frac{1}{\varepsilon_n} E[\mathbf{x}(n)\,, n]\,, \text{ as } \overline{\mathcal{I}}(n) \text{ occurs.}
\end{aligned}
$$

Therefore, reorganising the terms yields:

$$
\frac{\Delta_{\mathbf{x}(n)}}{4} \leq \frac{1}{\varepsilon_n} E[\mathbf{x}(n)\,, n] \leq \frac{1}{\overline{\varepsilon}_T} E[\mathbf{x}(n)\,, n]\,, \tag{3.5.8}
$$

which corresponds to $\mathcal{F}(n)$.

As a consequence, the regret is upper bounded by the sum of four terms:

$$
\begin{aligned}
R(T) \leq\ & \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{G}(n)\}\ \Delta_{\mathbf{x}(n)} \right\} + \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{H}(n)\}\ \Delta_{\mathbf{x}(n)} \right\} \\
&+ \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{I}(n)\}\ \Delta_{\mathbf{x}(n)} \right\} + \mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{F}(n)\}\ \Delta_{\mathbf{x}(n)} \right\}.
\end{aligned}
$$

3.5.2. First term: poor reward estimation. For any $\mathbf{x} \in \mathcal{X}$, we have $\Delta_{\mathbf{x}} \leq \boldsymbol{\theta}^\top \mathbf{x}^\star \leq m$, since $\theta \in [0,1]^d$ and $\max_{x\in\mathcal{X}} \sum_{i=1}^{d} x_i = m$. Therefore, by applying [**109**, Theorem 3]:

$$
\begin{aligned}
\mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{G}(n)\}\ \Delta_{\mathbf{x}(n)} \right\} &\leq m\,\mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{G}(n)\} \right\} \\
&= m \sum_{t=1}^{\infty} \mathbb{P}[\mathcal{G}(n)] \\
&\leq C_4(m)\,.
\end{aligned}
$$

where $C_4(m)$ is a positive number which only depends on m.

3.5.3. Second term: poor choice of subarm. We turn to the second term, using a union bound:

$$
\mathbb{P}[\mathcal{H}(n)] = \mathbb{P}\left[\bigcup_{i=1}^{d} \mathcal{H}_i(n) \right] \leq \sum_{i=1}^{d} \mathbb{P}[\mathcal{H}_i(n)]\,. \tag{3.5.9}
$$

Using once again the fact that $\Delta_{\mathbf{x}} \leq m$, the regret due to $\mathcal{H}(n)$ is bounded as:

$$
\begin{aligned}
\mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{H}_n\}\ \Delta_{\mathbf{x}(n)} \right\} &\leq m\,\mathbb{E}\left\{ \sum_{n=1}^{T} \mathbb{1}\{\mathcal{H}_n\} \right\} \\
&= m \sum_{n=1}^{T} \mathbb{P}[\mathcal{H}(n)] \\
&\leq m \sum_{n=1}^{T} \sum_{i=1}^{d} \mathbb{P}[\mathcal{H}_i(n)]\,.
\end{aligned}
$$

By definition of $\mathcal{H}_i(n)$,

$$\mathbb{P}[\mathcal{H}_i(n)] = \mathbb{P}\bigg(x_i(n) = 1, \quad \Big|\hat{\theta}_i(n) - \theta_i\Big| \geq \frac{\Delta_{\min}}{2\,m}\bigg)\,. \tag{3.5.10}$$

Using Hoeffding's inequality, this probability can be bounded by:

$$\mathbb{P}[\mathcal{H}_i(n)] \leq \exp\left[-n\left(\frac{\Delta_{\min}}{m}\right)^2\right]. \tag{3.5.11}$$

Injecting this result in the regret component,

$$\begin{aligned}
\mathbb{E}\left\{\sum_{n=1}^{T} \mathbb{1}\{\mathcal{H}(n)\}\ \Delta_{\mathbf{x}(n)}\right\} &\leq m\ \sum_{n=1}^{T}\sum_{i=1}^{d} \mathbb{P}[\mathcal{H}_i(n)] \\
&\leq m\ \sum_{n=1}^{T}\sum_{i=1}^{d} \exp\left[-n\left(\frac{\Delta_{\min}}{m}\right)^2\right] \\
&\leq m\,d\ \sum_{n=1}^{T} \exp\left[-n\left(\frac{\Delta_{\min}}{m}\right)^2\right] \\
&\leq \frac{md}{1-e^{-\left(\frac{\Delta_{\min}}{m}\right)^2}}, \text{ recognising a geometric series} \\
&\leq \frac{m^3\,d}{\Delta_{\min}^2}\left(1+\frac{\Delta_{\min}^2}{m^2}\right), \text{ as } e^z \geq 1+z,\ \forall z>0 \\
&\leq \frac{2\,m^3\,d}{\Delta_{\min}^2}, \text{ as } \Delta_{\min} \leq m.
\end{aligned}$$

3.5.4. Third term: small reward gap. By definition, the third term is:

$$\begin{aligned}
\mathbb{E}\left\{\sum_{n=1}^{T} \mathbb{1}\{\mathcal{I}(n)\}\ \Delta_{\mathbf{x}(n)}\right\} &= \mathbb{E}\left\{\sum_{n=1}^{T} \mathbb{1}\big\{\Delta_{\mathbf{x}(n)} \leq 4\,\delta_n\big\}\ \Delta_{\mathbf{x}(n)}\right\} \\
&\leq 4\ \sum_{t=n}^{T} \delta_n\ \mathbb{1}\{\Delta_{\min} \leq 4\delta_t\}\,.
\end{aligned}$$

3.5.5. Fourth term: dominant term. We now consider the event $\mathcal{F}(n)$. By definition, if $\mathcal{F}(n)$ occurs,

$$\Delta_{\mathbf{x}(n)} \leq \frac{4}{\bar{\varepsilon}_T}\,E[\mathbf{x}(n)\,,n]\,. \tag{3.5.12}$$

Squaring this definition,

$$\begin{aligned}
\Delta_{\mathbf{x}(n)}^2 &\leq \frac{16}{\bar{\varepsilon}_T}\,E^2[\mathbf{x}(n)\,,n] \\
&= \frac{16}{\bar{\varepsilon}_T}\,\mathbf{x}^T(n)\ \boldsymbol{\sigma}^2(n)\,, \text{ by definition of } E(\mathbf{x},n) \\
&= \frac{8}{\bar{\varepsilon}_T^2}\,f(n)\ \sum_{i=1}^{d} \frac{x_i(n)}{T_i(n)}, \text{ by definition of } \boldsymbol{\sigma}^2(n)\,.
\end{aligned}$$

If this event happens, it means that there exists a subset of subarms such that the number of samples for each subarm $T_i(n)$ is small. We further decompose this event as follows.

We consider (α_j) and (β_j), two positive, non-increasing sequences verifying the following properties:

$$\lim_{j\to+\infty} \alpha_j = \lim_{j\to+\infty} \beta_j = 0, \tag{3.5.13}$$

$$\lim_{j\to+\infty} \frac{\beta_j}{\sqrt{\alpha_j}} = 0, \tag{3.5.14}$$

$$\beta_0 = 1. \tag{3.5.15}$$

We define j_0 as the first integer j such that $\beta_j \leq \frac{1}{m}$ and we let ℓ as the sum

$$\ell = \frac{\beta_{j_0}}{\alpha_{j_0}} + \sum_{j=1}^{j_0} \frac{\beta_{j-1} - \beta_j}{\alpha_{j-1}}. \tag{3.5.16}$$

These two sequences $\{\alpha_j\}$ and $\{\beta_j\}$ are fixed, and their exact value will be specified later.

For all $j \in \mathbb{N}$, we define the following sets:

$$S_j(n) = \begin{cases} \left\{ i \in \{1, 2 \dots d\} \,\middle|\, x_i(n) = 1, \quad T_i(t) \leq \alpha_j \, \frac{2\, f(n)\; g(m)}{\Delta^2_{\mathbf{x}(n)}} \right\} & \text{if } j \geq 1 \\ \{ i \in \{1, 2 \dots d\} \,|\, x_i(n) = 1 \} & \text{if } j = 0 \end{cases} \tag{3.5.17}$$

where $g(m) = 4\, m\, \ell / \bar{\varepsilon}_T$.

Remark 15. The sets $S_j(n)$ used in this proof are unrelated to the set $\mathcal{S}(n)$ used when dealing with budgeted optimisation problems.

Since the function $j \mapsto \alpha_j$ is decreasing and $\lim_{j\to\infty} \alpha_j = 0$, the sequence of sets $\{S_j(n)\}$ is decreasing for set inclusion when j increases. Moreover, there is an index $j_\emptyset$ such that $S_{j_\emptyset}(n) = \emptyset$:

$$\emptyset = S_{j_\emptyset}(n) \subset S_{j_\emptyset - 1}(n) \subset \dots \subset S_1(n) \subset S_0(n)\,. \tag{3.5.18}$$

We define the event $\mathcal{A}_j(n)$ as:

$$\mathcal{A}_j(n) = \{ |S_j(n)| \geq m\, \beta_j \quad \text{and} \quad \forall k < j, \quad |S_k(n)| < m\, \beta_j \}\,. \tag{3.5.19}$$

By assumption on the sequence $\{\beta_j\}$, we have:

$$|S_0(n)| = m\, \beta_0 = m. \tag{3.5.20}$$

Finally, we also define the events $\mathcal{A}(n)$ as the following unions:

$$\mathcal{A}(n) = \bigcup_{j=1}^{+\infty} \mathcal{A}_j(n)\,. \tag{3.5.21}$$

$\mathcal{A}(n)$ is a finite union of the events $\mathcal{A}_j(n)$ for $j \in \mathbb{N}_0$, as $\mathcal{A}_j(n)$ cannot occur if $j \geq j_\emptyset$:

$$\mathcal{A}(n) = \bigcup_{j=1}^{j_\emptyset} \mathcal{A}_j(n)\,. \tag{3.5.22}$$

We formally prove this fact by *reductio ad absurdum*. Initially, suppose that the event $\mathcal{A}_j(n)$ happens for some $j > j_\emptyset$. For all $j > j_\emptyset$, due to $\beta_{j_\emptyset} \leq 1/m$ and the fact that $\{\beta_j\}$ is a decreasing sequence,

$$m\,\beta_j \leq m\,\frac{1}{m} = 1. \tag{3.5.23}$$

Thus, by definition of the event $\mathcal{A}_j(n)$, we have that:

$$\begin{aligned}
\mathcal{A}_j(n) &= \left\{ \underbrace{m\,\beta_j}_{\leq 1} \leq |S_j(n)| \quad \text{and} \quad \forall k < j, \quad |S_k(n)| < m\,\beta_j \right\} \\
&= \left\{ \begin{array}{c} |S_j(n)| \geq 1 \quad \text{and} \quad \forall k < j_\emptyset,\ |S_k(n)| < m\,\beta_j \quad \text{and} \\ \underbrace{\forall k \in [j_\emptyset, j-1]\,,\ |S_k(n)| = 0}_{\text{by definition of } j_\emptyset} \end{array} \right\}.
\end{aligned}$$

However, the same set $S_j(n)$ cannot be both empty and have at least one element. In other words, the event $\mathcal{A}_j(n)$ cannot happen for $j > j_\emptyset$, and $\mathcal{A}(n)$ is a finite union of events.

Under the event $\overline{\mathcal{A}}(n)$, the sum $\sum_{i=1}^{d} \frac{x_i(n)}{T_i(n)}$ can be bounded. The event $\overline{\mathcal{A}}(n)$ is, by De Morgan's law (recall that j_0 is finite):

$$\begin{aligned}
\overline{\mathcal{A}}(n) &= \bigcap_{j=1}^{j_\emptyset} \overline{\mathcal{A}_j}(n)\,, \text{ as } \mathcal{A}(n) \text{ is a finite union of } \mathcal{A}_j(n) \\
&= \bigcap_{j=1}^{j_\emptyset} \{|S_j(n)| < m\,\beta_j \quad \text{or} \quad \exists k < j, \quad |S_k(n)| \geq m\,\beta_j\} \\
&= \bigcap_{j=1}^{j_\emptyset} \left[\{|S_j(n)| < m\,\beta_j\} \cup \left\{ \bigcup_{k=1}^{j-1} |S_k(n)| \geq m\,\beta_j \right\} \right] \\
&= \bigcap_{j=1}^{j_\emptyset} \{|S_j(n)| < m\,\beta_j\}\,, \text{ as the two events are incompatible} \\
&= \bigcap_{j=1}^{j_\emptyset - 1} \{|S_j(n)| < m\,\beta_j\} \cap \left\{ \left|S_{j_\emptyset}(n)\right| < m\,\beta_{j_\emptyset} \right\}.
\end{aligned}$$

Since $\beta_{j_\emptyset} \leq 1/m$, the last event can be written as $\left|S_{j_\emptyset}(n)\right| < \frac{m}{m} = 1$. A set whose cardinality is strictly less than one must be empty, thus:

$$\overline{\mathcal{A}(n)} = \bigcap_{j=1}^{j_\emptyset - 1} \{|S_j(n)| < m\,\beta_j\} \cap \left\{ \left|S_{j_\emptyset}(n)\right| = 0 \right\}. \tag{3.5.24}$$

If the event $\overline{\mathcal{A}}(n)$ happens, then:

$$\begin{aligned}
\overline{S_j}(n) &= \{i\,\{\in 1, 2 \ldots d\} \,|\, x_i(n) = 1, i \notin S_j(n)\} \\
&= \left\{ i\,\{\in 1, 2 \ldots d\} \,\middle|\, x_i(n) = 1, \quad T_i(t) > \alpha_j\,\frac{2\,f(n)\,g(m)}{\Delta^2_{x(n)}} \right\},
\end{aligned}$$

$$\overline{S_{j_\emptyset}}(n) = \{i \{\in 1, 2 \dots d\} \,|\, x_i(n) = 1\}\,.$$

Indeed, due to the fact that $\{S_j(n)\}$ is a decreasing sequence for set inclusion when j increases, the complement $\overline{S_j}(n)$ must be an increasing sequence for set inclusion when j increases. This implies that:

$$\left\{i \in \{1, 2 \dots d\} \;\middle|\; x_i(t) = 1\right\} = \bigcup_{j=1}^{j_\emptyset} \left(\overline{S_j}(n) \setminus \overline{S_{j-1}}(n)\right). \tag{3.5.25}$$

Thus,

$$\sum_{i=1}^{d} \frac{x_i(n)}{T_i(n)} = \sum_{j=1}^{j_\emptyset} \sum_{i \in \overline{S_j}(n) \setminus \overline{S_{j-1}}(n)} \frac{x_i(n)}{T_i(n)}. \tag{3.5.26}$$

Using the definition of $S_j(n)$, one might write that, if the event $\overline{S_j}(n)$ occurs, then:

$$\begin{aligned}\sum_{i \in \overline{S_j}(n) \setminus \overline{S_{j-1}}(n)} \frac{x_i(n)}{T_i(n)} &< \frac{\Delta^2_{\mathbf{x}(n)}}{2\, f(n)\; g\,(m)\; \alpha_j} \sum_{i \in \overline{S_j}(n) \setminus \overline{S_{j-1}}(n)} x_i(n) \\ &= \frac{\Delta^2_{\mathbf{x}(n)}}{2\, f\,(n)\; g\,(m)} \frac{\left|\overline{S_j}(n) \setminus \overline{S_{j-1}}(n)\right|}{\alpha_j}.\end{aligned}$$

This implies that the previous sum is bounded as:

$$\begin{aligned}\sum_{i=1}^{d} \frac{x_i(n)}{T_i(n)} &= \sum_{j=1}^{j_\emptyset} \sum_{i \in \overline{S_j(n) \setminus \overline{S_{j-1}}(n)}} \frac{x_i(n)}{T_i(n)} \\ &\leq \frac{\Delta^2_{\mathbf{x}(n)}}{2\, f\,(n)\; g\,(m)} \sum_{j=1}^{j_\emptyset} \frac{\left|\overline{S_j}\,(n) \setminus \overline{S_{j-1}}\,(n)\right|}{\alpha_j}.\end{aligned}$$

The inner sum on j can be decomposed as follows, by definition of $S_j(n)$ and $\overline{S_j(n)}$:

$$\begin{aligned}\sum_{j=1}^{j_\emptyset} \frac{\left|\overline{S_j}\,(n) \setminus \overline{S_{j-1}}\,(n)\right|}{\alpha_j} &= \sum_{j=1}^{j_\emptyset} \frac{|S_j(n) \setminus S_{j-1}(n)|}{\alpha_j}, \text{ dropping the complements} \\ &= \sum_{j=1}^{j_\emptyset} \frac{|S_j(n)| - |S_{j-1}(n)|}{\alpha_j} \\ &= \frac{\left|S_{j_\emptyset}(n)\right|}{\alpha_0} + \sum_{j=1}^{j_\emptyset} \left[|S_j(n)| \left(\frac{1}{\alpha_{j-1}} - \frac{1}{\alpha_j}\right)\right], \text{ factoring the } j_\emptyset \text{ term} \\ &< \frac{m\, \beta_{j_\emptyset}}{\alpha_0} + \sum_{j=1}^{j_0} \left[m\, \beta_j \left(\frac{1}{\alpha_{j-1}} - \frac{1}{\alpha_j}\right)\right], \text{ as } \overline{\mathcal{A}(n)} \text{ holds.}\end{aligned}$$

Finally, replacing g and ℓ by their definition,

$$\sum_{i=1}^{d} \frac{x_i(n)}{T_i(n)} < \frac{m\, \Delta^2_{\mathbf{x}(n)}}{2\, f(t)\; g(m)} \left(\frac{\beta_{j_\emptyset}}{\alpha_{j_\emptyset}} + \sum_{j=1}^{j_\emptyset} \frac{\beta_{j-1} - \beta_j}{\alpha_{j-1}}\right)$$

$$= \frac{\Delta^2_{\mathbf{x}(n)} \overline{\varepsilon}^2_T}{8\, f(n)}$$

Next, we prove that the event $\Delta_{\mathbf{x}(n)} \leq \frac{4}{\overline{\varepsilon}_T}\, E[\mathbf{x}(n)\,, n]$ implies $\mathcal{A}(n)$ by *reductio ad absurdum*. Indeed, if $\Delta_{\mathbf{x}(n)} \leq \frac{4}{\overline{\varepsilon}_T}\, E[\mathbf{x}(n)\,, n]$ and $\overline{\mathcal{A}}(n)$, then:

$$\begin{aligned}
\Delta^2_{\mathbf{x}(n)} &\leq \frac{16}{\overline{\varepsilon}^2_T} E^2[\mathbf{x}(n)\,, n] \\
&= \frac{16}{\overline{\varepsilon}^2_T}\, \mathbf{x}^T(n)\; \boldsymbol{\sigma}^2(n) \\
&= \frac{8\, f(n)}{\overline{\varepsilon}^2_T} \sum_{i=1}^{d} \frac{x_i(n)}{T_i(n)} \\
&< \Delta^2_{\mathbf{x}(n)}.
\end{aligned}$$

which is a contradiction.

We now bound the regret due to the event $\mathcal{F}(n)$. We further decompose $\mathcal{A}_j(n)$ into a sequence of $\mathcal{A}_{j,i}(n)$ to include the fact that a specific item i is included among the (at least) $m\,\beta_j$ items that have not yet been selected often enough (i.e., the arm rewards are not yet well estimated):

$$\mathcal{A}_{j,i}(n) = \mathcal{A}_j(n) \cap \left\{ x_i(t) = 1\,,\, T_i(n) \leq \frac{\alpha_j\, 2\, f(T)\; g(m)}{\Delta^2_{\mathbf{x}(n)}} \right\}. \tag{3.5.27}$$

Of course, the union over all i yields back $\mathcal{A}_j(n)$:

$$\bigcup_{i=1}^{d} \mathcal{A}_{j,i}(n) = \mathcal{A}_j(n)\,. \tag{3.5.28}$$

Since $\mathcal{A}_j(n)$ implies that at least $m\,\beta_j$ items have not yet been selected often enough,

$$\mathbb{1}_{\mathcal{A}_j(n)} \leq \frac{1}{m\,\beta_j} \sum_{i=1}^{d} \mathbb{1}_{\mathcal{A}_{j,i}(n)}. \tag{3.5.29}$$

The contribution to the regret of the event $\mathcal{F}(n)$ is bounded by the items that are not selected frequently enough to ensure a good reward estimate:

$$\begin{aligned}
\sum_{t=1}^{T} \Delta_{\mathbf{x}(n)}\, \mathbb{1}_{\mathcal{F}(n)} &\leq \sum_{t=1}^{T} \Delta_{\mathbf{x}(n)}\, \mathbb{1}_{\mathcal{A}(n)} \\
&\leq \sum_{t=1}^{T} \sum_{j=1}^{+\infty} \Delta_{\mathbf{x}(n)}\, \mathbb{1}_{\mathcal{A}_j(n)} \\
&\leq \sum_{t=1}^{T} \sum_{j=1}^{+\infty} \sum_{i=1}^{d} \frac{\Delta_{\mathbf{x}(n)}}{m\,\beta_j}\, \mathbb{1}_{\mathcal{A}_{j,i}(n)}.
\end{aligned}$$

For any $i \in \{1, 2 \ldots k\}$, define the K_i possible values of the gaps $\Delta_{\mathbf{x}}$ for $\mathbf{x} \in \mathcal{X}$ where $x_i = 1$, namely:

$$\left\{ \Delta_{\mathbf{x}} \,\middle|\, x \in \mathcal{X},\; x_i = 1 \right\} = \{\Delta_{i,1}, ..., \Delta_{i,K_i}\}\,, \qquad \forall i \in \{1, 2 \ldots k\}\,. \tag{3.5.30}$$

where K_i is the number of possible values for the gap and we assume that the gaps are sorted in decreasing order:

$$\Delta_{i,1} > \Delta_{i,2} > ... > \Delta_{i,K_i}, \qquad \forall i \in \{1, 2 \dots k\} \tag{3.5.31}$$

with the convention that $\Delta_{i,0} = +\infty$. We can then decompose the previous sum according to the values of the gap:

$$\begin{aligned}
\sum_{t=1}^{T} \mathbb{1}_{\mathcal{F}(n)} \Delta_{\mathbf{x}(n)} &\leq \sum_{t=1}^{T} \sum_{j=1}^{+\infty} \sum_{i=1}^{d} \frac{\Delta_{\mathbf{x}(n)}}{m\,\beta_j} \mathbb{1}\{\mathcal{A}_{j,i}(n)\} \\
&\leq \sum_{t=1}^{T} \sum_{j=1}^{+\infty} \sum_{i=1}^{d} \sum_{k=1}^{K_i} \frac{\Delta_{i,k}}{m\,\beta_j} \mathbb{1}\left\{\mathcal{A}_{j,i}(n)\,, \quad \Delta_{\mathbf{x}(n)} = \Delta_{i,k}\right\} \\
&\leq \sum_{t=1}^{T} \sum_{j=1}^{+\infty} \sum_{i=1}^{d} \sum_{k=1}^{K_i} \frac{\Delta_{i,k}}{m\,\beta_j} \mathbb{1}\left\{\begin{array}{c} \mathcal{A}_j(n)\,, \quad x_i(t) = 1, \\ T_i(n) \leq \dfrac{\alpha_j f(T)\; g(m)}{2\,\Delta_{i,k}^2}, \\ \Delta_{\mathbf{x}(n)} = \Delta_{i,k} \end{array}\right\},
\end{aligned}$$

by definition of $\mathcal{A}_{j,i}(n)$. To simplify notation, let

$$\tau_j = \frac{1}{2}\,\alpha_j\, f(n)\; g(m)\,. \tag{3.5.32}$$

Thus, the previous bound can be written as:

$$\sum_{t=1}^{T} \mathbb{1}_{\mathcal{F}(n)} \Delta_{\mathbf{x}(n)} \leq \sum_{t=1}^{T} \sum_{j=1}^{+\infty} \sum_{i=1}^{d} \sum_{k=1}^{K_i} \frac{\Delta_{i,k}}{m\,\beta_j} \mathbb{1}\left\{x_i(n) = 1, \quad T_i(n) \leq \frac{\tau_j}{\Delta_{i,k}^2}, \quad \Delta_{\mathbf{x}(n)} = \Delta_{i,k}\right\}. \tag{3.5.33}$$

To simplify the developments, focus on the two sums, the one on the rounds t and the one on the gap values k:

$$\sum_{t=1}^{T} \sum_{k=1}^{K_i} \frac{\Delta_{i,k}}{m\,\beta_j} \mathbb{1}\left\{x_i(n) = 1, \quad T_i(n) \leq \frac{\tau_j}{\Delta_{i,k}^2}, \quad \Delta_{\mathbf{x}(n)} = \Delta_{i,k}\right\}. \tag{3.5.34}$$

As the values of $\Delta_{i,k}$ are ordered, we can decompose this sum as follows:

$$\begin{aligned}
&\sum_{t=1}^{T} \sum_{k=1}^{K_i} \frac{\Delta_{i,k}}{m\,\beta_j} \mathbb{1}\left\{x_i(n) = 1, \quad T_i(n) \leq \frac{\tau_j}{\Delta_{i,k}^2}, \quad \Delta_{\mathbf{x}(n)} = \Delta_{i,k}\right\} \\
&\quad = \sum_{t=1}^{T} \sum_{k=1}^{K_i} \sum_{p=1}^{K_i} \frac{\Delta_{i,k}}{m\,\beta_j} \mathbb{1}\left\{x_i(n) = 1, \quad T_i(n) \in \left(\frac{\tau_j}{\Delta_{i,p-1}^2}, \frac{\tau_j}{\Delta_{i,p}^2}\right], \quad \Delta_{\mathbf{x}(n)} = \Delta_{i,k}\right\}.
\end{aligned}$$

The factor $\Delta_{i,k}$ can become $\Delta_{i,p}$, as it will be counted only once, when the step function is nonzero (when $j = k$):

$$\begin{aligned}
&\sum_{t=1}^{T} \sum_{k=1}^{K_i} \frac{\Delta_{i,k}}{m\,\beta_j} \mathbb{1}\left\{x_i(n) = 1, \quad T_i(n) \leq \frac{\tau_j}{\Delta_{i,k}^2}, \quad \Delta_{\mathbf{x}(n)} = \Delta_{i,k}\right\} \\
&\quad \leq \sum_{t=1}^{T} \sum_{k=1}^{K_i} \sum_{p=1}^{K_i} \frac{\Delta_{i,p}}{m\,\beta_j} \mathbb{1}\left\{x_i(n) = 1, \quad T_i(n) \in \left(\frac{\tau_j}{\Delta_{i,p-1}^2}, \frac{\tau_j}{\Delta_{i,p}^2}\right], \quad \Delta_{\mathbf{x}(n)} = \Delta_{i,k}\right\}.
\end{aligned}$$

Again, if the solution $\mathbf{x}(n)$ is not taken to be exactly k, but any suboptimum solution, many new terms now count in the sum. With this change, the sum over k becomes irrelevant, as all gaps that may contribute to the regret are still counted in the sum.

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{k=1}^{K_i}\frac{\Delta_{i,k}}{m\,\beta_j}\mathbb{1}\left\{x_i(n)=1,\quad T_i(n)\leq\frac{\tau_j}{\Delta_{i,k}^2},\quad \Delta_{\mathbf{x}(n)}=\Delta_{i,k}\right\}\\
&\quad\leq\sum_{t=1}^{T}\sum_{p=1}^{K_i}\frac{\Delta_{i,p}}{m\,\beta_j}\mathbb{1}\left\{x_i(n)=1,\quad T_i(n)\in\left(\frac{\tau_j}{\Delta_{i,p-1}^2},\frac{\tau_j}{\Delta_{i,p}^2}\right],\quad \Delta_{\mathbf{x}(n)}=\Delta_{i,k}\right\}\\
&\quad\leq\frac{\tau_j}{m\,\beta_j}\left(\frac{1}{\Delta_{i,1}}+\sum_{p=2}^{K_i}\Delta_{i,p}\left(\frac{1}{\Delta_{i,p}^2}-\frac{1}{\Delta_{i,p-1}^2}\right)\right)\\
&\quad\leq\frac{2\tau_j}{m\,\beta_j\Delta_{\min}}
\end{aligned}$$

where we used the following algebra, since the $\Delta_{i,p}$ are increasing when p increases:

$$\begin{aligned}
\frac{1}{\Delta_{i,1}}+\sum_{p=2}^{K_i}\Delta_{i,p}\left(\frac{1}{\Delta_{i,p}^2}-\frac{1}{\Delta_{i,p-1}^2}\right)&=\frac{1}{\Delta_{i,K_i}}+\sum_{p=1}^{K_i-1}\frac{\Delta_{i,p}-\Delta_{i,p+1}}{\Delta_{i,p}^2}\\
&\leq\frac{1}{\Delta_{i,K_i}}+\sum_{p=1}^{K_i-1}\frac{\Delta_{i,p}-\Delta_{i,p+1}}{\Delta_{i,p+1}\Delta_{i,p}}\\
&=\frac{1}{\Delta_{i,K_i}}+\sum_{p=1}^{K_i-1}\frac{1}{\Delta_{i,p+1}}-\frac{1}{\Delta_{i,p}}\\
&=\frac{2}{\Delta_{i,K_i}}-\frac{1}{\Delta_{i,1}}\\
&\leq\frac{2}{\Delta_{\min}}.
\end{aligned}$$

Injecting this result into the regret term bound,

$$\begin{aligned}
\sum_{t=1}^{T}\mathbb{1}_{\mathcal{F}(n)}\,\Delta_{\mathbf{x}(n)}&\leq\sum_{t=1}^{T}\sum_{j=1}^{+\infty}\sum_{i=1}^{d}\sum_{k=1}^{K_i}\frac{\Delta_{i,k}}{m\,\beta_j}\mathbb{1}\left\{x_i(n)=1,\quad T_i(n)\leq\frac{\tau_j}{\Delta_{i,k}^2},\quad \Delta_{\mathbf{x}(n)}=\Delta_{i,k}\right\}\\
&\leq\sum_{j=1}^{+\infty}\sum_{i=1}^{d}\frac{2\,\tau_j}{m\,\beta_j\,\Delta_{\min}}\\
&=\frac{f(T)\;d\,g(m)}{m\Delta_{\min}}\left[\sum_{j=1}^{j_0}\frac{\alpha_j}{\beta_j}\right],\text{ by definition of }\tau_j\\
&=\frac{4\,\ell\,d\,f(T)}{\bar{\varepsilon}_T^2\Delta_{\min}}\left[\sum_{j=1}^{j_0}\frac{\alpha_j}{\beta_j}\right],\text{ by definition of }g.
\end{aligned}$$

Now, set $\alpha_i = \beta_i = \beta^i$, for some $\beta \in (0,1)$. This choice satisfies the previous assumptions. Since $j_\emptyset$ is the first integer j such that $\beta_j \leq m^{-1}$, we have $j_\emptyset = \left\lceil \frac{\log m}{\log \beta^{-1}} \right\rceil$. Also,

$$\begin{aligned}
\ell \sum_{j=1}^{j_\emptyset} \frac{\alpha_j}{\beta_j} &= \ell\, j_\emptyset, \text{ as } \alpha_i = \beta_i \\
&= j_\emptyset \left(\frac{\beta_{j_0}}{\alpha_{j_0}} + \sum_{j=1}^{j_\emptyset} \frac{\beta_{j-1} - \beta_j}{\alpha_{j-1}} \right), \text{ by definition of } \ell \\
&= j_\emptyset \left(1 + \sum_{j=1}^{j_\emptyset} \frac{\beta^{j-1} - \beta^j}{\beta^j} \right) \\
&= j_\emptyset \left(1 + \sum_{j=1}^{j_\emptyset} \frac{1-\beta}{\beta} \right) \\
&= j_\emptyset \left(1 + \frac{j_\emptyset}{\beta} - j_\emptyset \right) \\
&\leq j_\emptyset \left(1 + \frac{j_\emptyset}{\beta} \right).
\end{aligned}$$

Taking $\beta = 1/5$,

$$j_\emptyset = \left\lceil \frac{\log m}{\log \beta^{-1}} \right\rceil \leq \left\lceil \frac{\log m}{1.61} \right\rceil. \tag{3.5.35}$$

Injecting these into the regret term, we get:

$$\begin{aligned}
\sum_{t=1}^{T} \Delta_{\mathbf{x}(n)} \mathbb{1}_{\mathcal{F}(n)} &\leq \frac{4\,\ell\, d\, f(T)}{\overline{\varepsilon}_T^2\, \Delta_{\min}} \left[\sum_{j=1}^{j_\emptyset} \frac{\alpha_j}{\beta_j} \right] \\
&\leq \frac{4\, d\, f(T)}{\overline{\varepsilon}_T^2 \Delta_{\min}} \left(\left\lceil \frac{\log m}{1.61} \right\rceil + 5 \left\lceil \frac{\log m}{1.61} \right\rceil^2 \right) \\
&\leq \frac{24\, d\, f(T)}{\overline{\varepsilon}_T^2 \Delta_{\min}} \left\lceil \frac{\log m}{1.61} \right\rceil^2.
\end{aligned}$$

3.5.6. Complete regret bound. Gathering the results about the three terms of the regret decomposition, the regret can be bounded by:

$$R(T) \leq C_4(m) + \frac{2\, d\, m^3}{\Delta_{\min}^2} + \frac{24\, d\, f(T)}{\overline{\varepsilon}_T^2\, \Delta_{\min}} \left\lceil \frac{\log m}{1.61} \right\rceil^2 + 4 \sum_{n=1}^{T} \delta_t \mathbb{1}\{\Delta_{\min} \leq 4\, \delta_n\}. \tag{3.5.36}$$

CHAPTER 4

Nonsmooth Optimisation for Optimum Combinatorial Bandits

ESCB is a state-of-the-art algorithm for combinatorial bandits (Chapter 3). We now turn our attention to OSSB (Section 2.4.4), another state-of-the-art algorithm for combinatorial bandits (Section 2.5.4), but provably asymptotically optimum: when implemented exactly, this algorithm yields the lowest asymptotical regret. On the contrary, ESCB is part of a family of algorithms that has been proved to suffer large suboptimality in special cases [**76**, Theorem 1], [**?**, Proposition 1].

OSSB uses the current estimates of the bandit-problem parameters $\boldsymbol{\theta}$: at each bandit round, it computes the current estimate for the subarm rewards, and chooses the next arm to play based on the optimum solution to (2.4.8).

However, to the best of our knowledge, like ESCB, this algorithm cannot be straightforwardly implemented in polynomial time for combinatorial bandits. The problem lies in the formulation of the Graves-Lai bound, which is not amenable to direct optimisation. Indeed, this formulation is semi-infinite, due to the infinite number of constraints. Moreover, in the worst case, the number of variables is exponential: $\mathcal{O}(|\mathcal{X}|) \subset \mathcal{O}\big(2^d\big)$. Nevertheless, this semi-infinite formulation is still convex [**150**]: the objective function is linear, and the set of constraints $\sum_{\mathbf{x}\in\mathcal{A}} \eta_{\mathbf{x}} \operatorname{kl}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{x}) \geq 1$ is convex $\forall \boldsymbol{\lambda} \in \Lambda(\boldsymbol{\theta})$ (the left-hand side is a convex combination of Kullback-Leibler divergences, which are convex [**151**, P148], and the right-hand one is constant).

We propose two ways of solving the Graves-Lai problem in the special case of combinatorial bandits. Both are based on a reformulation of the problem (Section 4.3.3). This reformulation is reversible in the sense that it is possible to compute a solution to the original formulation based on the solution from the reformulation (Section 4.3.6).

- The simplest technique is based on standard constraint-generation techniques: it does not guarantee a polynomial complexity, but uses existing optimisation solvers whose practical efficiency has already been proved (Section 4.4).
- The second technique is guaranteed to have a polynomial complexity, and is based on nonsmooth convex optimisation instead of constraint generation (Section 4.3).

In this chapter, we only consider Gaussian combinatorial bandits, i.e. the rewards are drawn from Gaussian distributions.

4.1. Graves-Lai bound for combinatorial bandits

The Graves-Lai formulation (2.4.8) is the following in the specific case of Gaussian combinatorial linear bandits, as shown in Section (4.1.2):

$$\begin{array}{lll} \min & \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}} \, \Delta_{\mathbf{x}} & \\ \text{s.t.} & \sum_{i\in\mathcal{I}} \frac{x_i}{\sum_{\mathbf{y}\in\mathcal{X}} y_i \, \alpha_{\mathbf{y}}} \leq \Delta_{\mathbf{x}}^2 & \forall \mathbf{x} \in \mathcal{X} \\ & \eta_{\mathbf{x}} \geq 0 & \forall \mathbf{x} \in \mathcal{X} \end{array} \tag{4.1.1}$$

where $\mathcal{I}$ is the set of subarms that do not appear in any optimum decision:

$$\mathcal{I} = \left\{ i \in \{1, 2 \dots d\} \;\middle|\; \left(\max_{\substack{\mathbf{x} \in \mathcal{X}: \\ x_i = 1}} \boldsymbol{\theta}^T \mathbf{x} \right) < \boldsymbol{\theta}^T \mathbf{x}^\star \right\}. \tag{4.1.2}$$

4.1.1. Interpretation of the formulation. Lower bounds typically consider consistently good algorithms (Lai-Robbins in Section 2.2.4 and Graves-Lai in Section 2.4.2). Consider such an algorithm that selects $t_{\mathbf{x}} = \eta_{\mathbf{x}} \log T$ times the solution $\mathbf{x} \in \mathcal{X}$ over a horizon of T rounds. Its regret has the following decomposition:

$$\begin{aligned} R(T) &= \sum_{n=1}^{T} \Delta_{\mathbf{x}(n)} \\ &= \sum_{\mathbf{x} \in \mathcal{X}} t_{\mathbf{x}} \, \Delta_{\mathbf{x}} \\ &= \log T \sum_{\mathbf{x} \in \mathcal{X}} \eta_{\mathbf{x}} \, \Delta_{\mathbf{x}}. \end{aligned}$$

This quantity is a multiple of the objective function of (4.1.1). Similarly, the subarm i is observed $\log T \sum_{\mathbf{x} \in \mathcal{X}} x_i \, \eta_{\mathbf{x}}$ times over T rounds.

In order to guarantee with high probability that a solution $\mathbf{x} \in \mathcal{X}$ is not an optimum solution, the agent must gather enough statistical information about $\boldsymbol{\theta}^T \mathbf{x}^\star$. Indeed, these subarms are not shared by any optimum solution and still contained in $\mathbf{x}$. Estimating these suboptimum θ_i incurs some regret, as no optimum solution plays these subarms. To the contrary, estimating θ_i where i is such that $x_i^\star = 1$ can be done without regret, because $\mathbf{x}^\star$ is optimum.

Formalising this reasoning, one can show that the number of observations of any $\mathbf{x} \in \mathcal{X}$ must satisfy

$$\sum_{i \in \mathcal{I}} \frac{x_i}{\sum_{\mathbf{y} \in \mathcal{X}} y_i \, \alpha_{\mathbf{y}}} \leq \Delta_{\mathbf{x}}^2. \tag{4.1.3}$$

If this constraint is not satisfied, then it is not possible to distinguish between a suboptimum decision $\mathbf{x} \in \mathcal{X}$ and an optimum decision $\mathbf{x}^\star$ with high probability.

The Graves-Lai problem (4.1.1) amounts to minimising the regret under the constraint that the agent must be able to distinguish between suboptimum and optimum decisions.

4.1.2. Formal proof. The formulation (4.1.1) can be derived directly from (2.4.7) and (2.4.8).

From [**109**, Theorem 1], the result holds when $C(\boldsymbol{\theta})$ is the value of the following optimisation problem, recalled from (2.4.8):

$$\begin{aligned} \min \quad & \sum_{\mathbf{x} \in \mathcal{A}} \eta_{\mathbf{x}} \, \Delta_{\mathbf{x}} \\ \text{s.t.} \quad & \min_{\boldsymbol{\lambda} \in \Lambda(\boldsymbol{\theta})} \left\{ \sum_{i=1}^{d} \sum_{\mathbf{x} \in \mathcal{X}} \eta_{\mathbf{x}} \, x_i \, \mathrm{kl}(\theta_i, \lambda_i) \right\} \geq 1 \\ & \eta_{\mathbf{x}} \geq 0, \quad \forall \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{4.1.4}$$

where $\Lambda(\boldsymbol{\theta})$ is the set of confusing probability distributions (2.4.10):

$$\Lambda(\boldsymbol{\theta}) = \left\{ \boldsymbol{\lambda} \in \mathbb{R}^d \;\middle|\; \boldsymbol{\lambda}^T \mathbf{x}^\star < \max_{\mathbf{x} \in \mathcal{X}} \left\{ \boldsymbol{\lambda}^T \mathbf{x} \right\} \quad \text{and} \quad \theta_i = \lambda_i \quad \forall i \notin \mathcal{I} \right\}. \tag{4.1.5}$$

All the parameters $\boldsymbol{\lambda} \in \Lambda(\boldsymbol{\theta})$ are such that $\mathbf{x}^\star$ is not the optimum decision, and such that $\boldsymbol{\lambda}$ cannot be distinguished from $\boldsymbol{\theta}$ when selecting only optimum decisions under $\boldsymbol{\theta}$.

$\mathrm{kl}(\theta_i, \lambda_i)$ is the Kullback-Leibler divergence between the distribution of the rewards for i with respective means θ_i and λ_i (this notion was first defined in Section (2.4.2)). Since the reward distributions are assumed to be Gaussian with variance $1/2$, the divergence is given by $\mathrm{kl}(\theta_i, \lambda_i) = (\theta_i - \lambda_i)^2$. Furthermore, if $i \notin \mathcal{I}$, then $\theta_i = \lambda_i$ by definition of $\Lambda(\boldsymbol{\theta})$, so that $\mathrm{kl}(\theta_i, \lambda_i) = 0$.

Therefore, the optimisation program (2.4.8) simplifies to:

$$\begin{array}{ll} \min & \sum_{\mathbf{x}\in\mathcal{A}} \eta_{\mathbf{x}}\, \Delta_{\mathbf{x}} \\ \text{s.t.} & \min_{\boldsymbol{\lambda}\in\Lambda(\boldsymbol{\theta})} \left\{ \sum_{i=1}^d \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, x_i\, (\theta_i - \lambda_i)^2 \right\} \geq 1 \\ & \eta_{\mathbf{x}} \geq 0, \quad \forall \mathbf{x} \in \mathcal{X}. \end{array} \tag{4.1.6}$$

We then decompose $\Lambda(\boldsymbol{\theta})$ according to the optimum decisions and their values:

$$\Lambda_{\mathbf{x},v}(\boldsymbol{\theta}) = \left\{ \boldsymbol{\lambda} \in \Lambda(\boldsymbol{\theta}) \,\middle|\, \boldsymbol{\lambda}^T \mathbf{x} = \boldsymbol{\theta}^T \mathbf{x}^\star + v \right\}, \tag{4.1.7}$$

$$\Lambda(\boldsymbol{\theta}) = \bigcup_{v\in\mathbb{R}^+} \bigcup_{\substack{\mathbf{x}\in\mathcal{X}: \\ \mathbf{x}\neq\mathbf{x}^\star}} \Lambda_{\mathbf{x},v}(\boldsymbol{\theta})\,. \tag{4.1.8}$$

The optimum solution to $\min_{\Lambda_{\mathbf{x},v}(\boldsymbol{\theta})} \left\{ \sum_{i=1}^d \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, x_i\, (\theta_i - \lambda_i)^2 \right\}$, which corresponds to the constraint of (4.1.6) written for a single $\Lambda_{\mathbf{x},v}(\boldsymbol{\theta})$, can then be written as:

$$\min_{\Lambda_{\mathbf{x},v}(\boldsymbol{\theta})} \left\{ \sum_{i=1}^d \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, x_i\, (\theta_i - \lambda_i)^2 \right\} \quad = \quad \begin{array}{ll} \min & \sum_{i=1}^d \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, x_i\, (\theta_i - \lambda_i)^2 \\ \text{s.t.} & \boldsymbol{\lambda}^T \mathbf{x} = \boldsymbol{\theta}^T \mathbf{x}^\star + v. \end{array} \tag{4.1.9}$$

Using the Karush-Kuhn-Tucker optimality conditions [**152**, Section 3.3.1] and solving, the optimum value can be computed analytically as:

$$\min_{\Lambda_{\mathbf{x},v}(\boldsymbol{\theta})} \left\{ \sum_{i=1}^d \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, x_i\, (\theta_i - \lambda_i)^2 \right\} \quad = \quad \frac{(\Delta_{\mathbf{x}} + v)^2}{\sum_{i\in\mathcal{I}} x_i \left(\sum_{\mathbf{y}\in\mathcal{X}} y_i\, \alpha_{\mathbf{y}} \right)^{-1}}. \tag{4.1.10}$$

The constraint written for $\Lambda(\boldsymbol{\theta})$ is satisfied only if it is valid for all $\mathbf{x} \in \mathcal{X}$ and all $v \in \mathbb{R}^+$, the most constraining values of v being the smallest ones. If if it satisfied for $v = 0$, then it will be satisfied for all $v \in \mathbb{R}^+$. Therefore, the constraint can be written as:

$$\sum_{i\in\mathcal{I}} \frac{x_i}{\sum_{\mathbf{y}\in\mathcal{X}} y_i\, \alpha_{\mathbf{y}}} \leq \Delta_{\mathbf{x}}^2, \qquad \forall \mathbf{x} \in \mathcal{X}. \tag{4.1.11}$$

As a consequence, the optimisation program (2.4.8) is equivalent to the claimed formulation (4.1.1) for combinatorial bandits over the set $\mathcal{X}$:

$$\begin{array}{lll} \min & \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, \Delta_{\mathbf{x}} & \\ \text{s.t.} & \sum_{i\in\mathcal{I}} \frac{x_i}{\sum_{\mathbf{y}\in\mathcal{X}} y_i\, \alpha_{\mathbf{y}}} \leq \Delta_{\mathbf{x}}^2 & \forall \mathbf{x} \in \mathcal{X} \\ & \eta_{\mathbf{x}} \geq 0 & \forall \mathbf{x} \in \mathcal{X}. \end{array} \tag{4.1.12}$$

4.2. Elements of nonsmooth convex optimisation

Typical algorithms for convex optimisation include gradient descent [**153**], and interior-point method [**154**], especially with self-concordant barrier functions [**155**]. They are very efficient, but limited to smooth problems, i.e. optimisation programs whose objective and constraint functions are differentiable. However, once the differentiability is not ensured, the convergence proofs of these

methods fail: while the algorithms themselves may still work, their convergence is usually far from the guaranteed one in the smooth case.

4.2.1. Smooth convex optimisation. A very common assumption in the domain of smooth convex optimisation is the μ-strong convexity of the function $f : \mathbb{R}^d \mapsto \mathbb{R}$ to minimise, where $\mu \in \mathbb{R}^+$is the largest constant such that:

$$f(\mathbf{v}) \geq f(\mathbf{u}) + \nabla f^T(\mathbf{u})\ (\mathbf{v} - \mathbf{u}) + \frac{\mu}{2}\ \|\mathbf{v} - \mathbf{u}\|_2^2\,, \qquad \forall \mathbf{v}, \mathbf{u} \in \mathbb{R}^d. \tag{4.2.1}$$

In particular, linear functions are convex, but their strong-convexity constant μ is zero. A second common assumption is L-smoothness of the function:

$$\|\nabla f(\mathbf{u}) - \nabla f(\mathbf{v})\| \leq L\ \|\mathbf{u} - \mathbf{v}\|\,, \qquad \forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^d. \tag{4.2.2}$$

A *nonsmooth* function is not differentiable everywhere: at some points, its gradient does not exist. For instance, at $u = 0$, $f(u) = |u|$ does not have a derivative. However, nonsmooth functions can still be μ-strongly convex and L-smooth. A *subgradient* of f at $\mathbf{u} \in \mathbb{R}^d$ is a vector $\mathbf{g} \in \mathbb{R}^d$ such that:

$$f(\mathbf{v}) \geq f(\mathbf{u}) + \mathbf{g}^T\ (\mathbf{v} - \mathbf{u})\,, \qquad \forall \mathbf{y} \in \mathbb{R}^d. \tag{4.2.3}$$

Therefore, a subgradient still gives a lower linear approximation of the function, exactly like a gradient. The major difference with the gradient is that subgradients are typically nonunique where the function is not differentiable. For instance, with $f(u) = |u|$, at $u = 0$, the set of subgradients (called the *subdifferential* and denoted by ∂f) is $[-1, 1]$.

The most basic method in smooth convex optimisation is the gradient descent **[153]**. Considering a function $f : \mathbb{R}^d \mapsto \mathbb{R}$, it starts from a point $\mathbf{u}^{(0)} \in \mathbb{R}^0$ and iteratively follows the gradient of f:

$$\mathbf{u}^{(k)} = \mathbf{u}^{(k-1)} + \eta\, \nabla f\Big(\mathbf{u}^{(k-1)}\Big)\,, \tag{4.2.4}$$

where $\eta > 0$ is a learning rate. This method has a linear convergence rate **[156**, Section 9.3.1]: after k iterations,

$$f\Big(\mathbf{u}^{(k)}\Big) - f(\mathbf{u}^\star) \leq 2\, L\, \frac{\big\|\mathbf{u}^{(0)} - \mathbf{u}^{(k)}\big\|}{k}. \tag{4.2.5}$$

However, gradient descent can only be applied on unconstrained problems.

Interior-point methods **[154]** are also based on the gradient, but handle constraints. These methods are based on a reformulation of the problem. If $f(\mathbf{u})$ is the objective function and $f_1(\mathbf{u}) \geq 0$ the only constraint, the reformulated problem is:

$$B(\mathbf{u}, \nu) = f(\mathbf{u}) - \nu\, \log f_1(\mathbf{u})\,. \tag{4.2.6}$$

$\nu \geq 0$ is the barrier parameter: the higher ν, the higher the constraint penalisation. For a high value of ν, the value of $f(\mathbf{u})$ becomes very low compared to $\nu \log f_1(\mathbf{x})$: minimising $B(\mathbf{x}, \nu)$ gives a point that is ensured to be within the feasible set defined by $f_1(\mathbf{u}) \geq 0$. Interior-point methods start with a high penalty ν; at each iteration, these algorithms approximately minimise $B(\mathbf{u}, \nu)$, then lower ν. As $B(\mathbf{u}, \nu)$ is only approximately minimised, the new iterate is close to the previous one, and thus stays within the feasible set.

These two techniques (gradient descent and interior-point method) are very popular and very effective for a very large class of problems. However, when it comes to nonsmooth optimisation, these methods may fail.

4.2.2. Subgradient method. Gradient descent is often a method of choice, even for nonsmooth problems. By replacing the use of the gradient by *any* subgradient, the algorithm becomes the *subgradient method*. This technique is more recent than gradient descent [**157**]. Formally, the iteration rule is:

$$\mathbf{u}_k = \mathbf{u}_{k-1} - \eta\,\mathbf{g}, \qquad \text{where} \qquad \mathbf{g} \in \partial f(\mathbf{u}_{k-1})\,. \tag{4.2.7}$$

In order to minimise the function f, let $\mathbf{u}_{\text{best}}^{(k)}$ denote the best iterate:

$$\mathbf{u}_{\text{best}}^{(k)} \in \operatorname*{arg\,min}_{i\in\{0,1\ldots k\}} f\Big(\mathbf{u}^{(k)}\Big)\,. \tag{4.2.8}$$

After k iterations, supposing that the optimum $\mathbf{u}^\star$ is unique [**158**, Theorem 3],

$$f\Big(\mathbf{u}_{\text{best}}^{(k)}\Big) - f(\mathbf{u}^\star) \leq \frac{F^2}{2\,k\,\eta} + \frac{G^2\,\eta}{2}, \tag{4.2.9}$$

where F is an upper bound on the value of f and G is an upper bound on the norm of its subgradient:

$$f(\mathbf{u}) \leq F, \qquad \forall \mathbf{u} \in \mathbb{R}^d, \tag{4.2.10}$$

$$\|\mathbf{g}\|_2^2 \leq G, \qquad \forall \mathbf{g} \in \partial f(\mathbf{u})\,, \qquad \forall \mathbf{u} \in \mathbb{R}^d. \tag{4.2.11}$$

4.2.3. Bundle method. The bundle method is a very different kind of algorithm to solve smooth and nonsmooth problems [**159**]. The crux of the method is a piecewise, lower linear approximation $\check{f}$ of the function to minimise. Along the way, this algorithm collects a series of *test points* where the function f and one of its subgradients is evaluated: all these points form the bundle $\mathcal{B}^{(k)}$ after k iterations.

$$\mathcal{B}^{(k)} = \left\{ \left[\mathbf{u}^{(i)}, \mathbf{g}^{(i)}, f\Big(\mathbf{u}^{(i)}\Big) \right], \quad i \in \{1, 2 \ldots k\} \right\}. \tag{4.2.12}$$

Based on this bundle $\mathcal{B}^{(k)}$, the approximation $\check{f}$ is given by:

$$\check{f}^{(k)}(\mathbf{u}) = \max_{i\in\{1,2\ldots k\}} \left\{ f\left(\mathbf{u}^{(i)}\right) + \mathbf{g}^{(i)T}\,\left(\mathbf{u} - \mathbf{u}^{(i)}\right) \right\}. \tag{4.2.13}$$

Given a scale factor γ_k, the proximal function is defined as:

$$\operatorname{prox}_{\gamma_k}\Big(\mathbf{u}^{(k)}\Big) = \arg\min_{\mathbf{u}\in\mathbb{R}^d} \left\{ \check{f}^{(k)}(\mathbf{u}) + \frac{\gamma_k}{2}\,\left\|\mathbf{u} - \mathbf{u}^{(k)}\right\|_2^2 \right\}. \tag{4.2.14}$$

Test points are obtained by solving this proximal problem with a sequence of scale factors γ_k. The scale factors are supposed to remain within some bounds: $0 < \gamma_{\min} \leq \gamma_k \leq \gamma_{\max} < +\infty$ for all iterations k. The other parameter of the method is $K > 0$, whose exact role is explained later.

At each iteration of the bundle method, the test point is obtained through the bundle approximation and a proximal problem with the penalty parameter γ_k:

$$\mathbf{v}^{(k+1)} \in \operatorname{prox}_{\gamma_k}\Big(\mathbf{u}^{(k)}\Big)\,. \tag{4.2.15}$$

Compute the multiplier π_k of the newly obtained point $\mathbf{v}^{(k+1)}$:

$$\pi_k = \check{f}^{(k)}\Big(\mathbf{v}^{(k+1)}\Big) - f\Big(\mathbf{u}^{(k)}\Big)\,. \tag{4.2.16}$$

If $\pi_k \geq 0$, the current iterate $\mathbf{x}^{(k)}$ is optimum. Otherwise, add the test point to the bundle. Depending on the decrease in f, perform either a short or a long step:

- If $f\big(\mathbf{v}^{(k+1)}\big) \leq f\big(\mathbf{u}^{(k)}\big) + K\,\pi_k$, the function f has significantly decreased (as $\pi_k < 0$), perform a descent step:

$$\mathbf{u}^{(k+1)} = \mathbf{v}^{(k+1)}. \tag{4.2.17}$$

  Choose γ_{k+1} freely between $\gamma_{\min}$ and $\gamma_{\max}$.
- Otherwise, perform a short step:

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)}. \tag{4.2.18}$$

  The iterate might not change, but the bundle approximation is slightly more precise: $\mathbf{v}^{(k+1)}$ will be different from $\mathbf{v}^{(k+2)}$. Choose γ_{k+1} between γ_k and $\gamma_{\max}$.

This method converges to a precision of ε with the following number of iterations [**159**, Theorem 3.1]:

$$k_{\max} = \begin{cases} \left\lceil \frac{2^{11}\, u_{\max}^2\, F^4\, G^2}{K\,(1-K^2)\, u_{\min}\, \varepsilon^3} \right\rceil & \text{if } \varepsilon \leq 4\, u_{\max}\, F^2 \\ \left\lceil \frac{2^7\, G^2}{K\,(1-K^2)\, u_{\min}\, \varepsilon} \right\rceil & \text{if } \varepsilon > 4\, u_{\max}\, F^2 \end{cases}. \tag{4.2.19}$$

4.3. AOSSB and GLPG

At first glance, the Graves-Lai optimisation program (4.1.1) seems impossible to solve in polynomial time: it has $\mathcal{O}(|\mathcal{X}|)$ variables and constraints (in the worst case, exponentially-many), with one variable and one convex constraint per solution $\mathbf{x}$ in the combinatorial set $\mathcal{X}$. To reformulate this program, we need a few supplementary hypotheses on the combinatorial set.

In this section, we present GLPG (*Graves-Lai projected gradient*), a polynomial-time algorithm to compute the Graves-Lai bound (4.1.1). We call AOSSB the variant of OSSB (Section (4.1)) where the Graves-Lai subproblem is solved approximately, with an accuracy $\delta > 0$.

4.3.1. Assumptions. Our results rely on a series of technical assumptions that do not restrict the generality of our results.

Assumption 16. *For each subarm $i \in \{1, 2 \dots d\}$, there exists a solution $\mathbf{x}^{(i)} \in \mathcal{X}$ such that $x_i^{(i)} = 1$. There is no restriction on the other components of the $\mathbf{x}^{(i)}$, i.e. $\mathbf{x}_j^{(i)} \in \{0, 1\}$ for all $j \in \{1, 2 \dots d\} \setminus \{i\}$.*

This assumption simply indicates that all subarms can be played and that their average reward θ_i can be estimated by sampling at least one solution, $\mathbf{x}^{(i)}$. If Assumption (16) is not satisfied, the violating subarms can simply be removed, as they do not play any role in the bandit problem. Therefore, this assumption does not lead to any loss of generality.

Assumption 17. *The combinatorial set $\mathcal{X}$ can be represented as the set of extreme points of a polytope in the following canonical form:*

$$\mathcal{X} = \Big\{ \mathbf{x} \in \{0,1\}^d \,\Big|\, \mathbf{A}\,\mathbf{x} = \mathbf{b} \Big\} \tag{4.3.1}$$

where $\mathbf{A} \in \mathbb{R}^{c \times d}$, with a number of constraints $c \in \mathcal{O}(\mathrm{poly}(d))$. The vector $\mathbf{b}$ is an element of $\mathbb{R}^c$.

This canonical form is the one required by the simplex algorithm [**160**], and all polytopes can be cast into this form by adding slack variables [**152**, Section 5.6.1]. This assumption thus reduces to a description by the means of a polynomial number of linear constraints.

Assumption (17) is equivalent to a hypothesis on the convex hull of $\mathcal{X}$:

$$\operatorname{conv}(\mathcal{X}) = \left\{\mathbf{x} \in \mathbb{R}_+^d \;\middle|\; \mathbf{A}\,\mathbf{x} = \mathbf{b}\right\}. \tag{4.3.2}$$

In particular, all the considered combinatorial sets satisfy this hypothesis: matroids, spanning trees, paths, matchings.

ASSUMPTION 18. *The average-reward vector $\boldsymbol{\theta}$ only has positive integer components: $\boldsymbol{\theta} \in \mathbb{N}^d$.*

Assumption 18 has implications on the minimum and maximum rewards, and therefore gaps:

$$\boldsymbol{\theta}^T \mathbf{x} \in \{0, 1 \dots m \; \|\boldsymbol{\theta}\|_\infty\}, \tag{4.3.3}$$

$$1 \leq \Delta_{\min} \leq \Delta_{\max} \leq m \; \|\boldsymbol{\theta}\|_\infty, \tag{4.3.4}$$

where the largest entry in $\boldsymbol{\theta}$ is denoted by $\|\boldsymbol{\theta}\|_\infty$.

Again, this assumption does not lead to any loss of generality. While it makes stating our results simpler, we can easily generalise them to the case where $\boldsymbol{\theta}$ has continuous values. Indeed, if Assumption 18 is violated and is in some interval like $[0,1]^d$, we can discretise its values in a new vector $\tilde{\boldsymbol{\theta}}$:

$$\tilde{\theta}_i = \varepsilon \left\lceil \frac{\theta_1}{\varepsilon} \right\rceil, \tag{4.3.5}$$

then use our technique on $\tilde{\boldsymbol{\theta}}/\varepsilon$. This vector has integer entries and, as a consequence, verifies Assumption 18.

This transformation can be performed without changing the optimum value of 4.1.1, thanks to the homogeneity of 18. More precisely, the Graves-Lai bound satisfies $C\left(\tilde{\boldsymbol{\theta}}\right) = C\left(\tilde{\boldsymbol{\theta}}/\varepsilon\right)/\varepsilon$ and its optimum solution is such that $\boldsymbol{\alpha}^\star\left(\tilde{\boldsymbol{\theta}}\right) = \boldsymbol{\alpha}^\star\left(\tilde{\boldsymbol{\theta}}/\varepsilon\right)/\varepsilon^2$.

One can therefore solve 4.1.1 up to any fixed accuracy in polynomial time using our results.

4.3.2. Polynomial-time solvability of the Graves-Lai program. The main result of this chapter is GLPG (*Graves-Lai projected gradient*), a technique to solve (4.1.1) in polynomial time, using a projected subgradient method. We first formally state the result before delving into the algorithmic details. The complexity is polynomial in the dimension d, the required accuracy level $\delta > 0$, and the largest entry in $\boldsymbol{\theta}$.

ε is the approximation ratio of the algorithm used for budgeted linear maximisation (as in Section 3.2). If ε is not 1, using the approximate solution to the Graves-Lai problem in order to guide the exploration in OSSB (Section 4.1) does not yield an asymptotically optimum algorithm, but instead an algorithm whose regret scales like the asymptotic optimum with a ratio ε^{-1}. This is typically better than what existing algorithms can achieve. In this case, improving the regret bound amounts to finding an exact budgeted-maximisation algorithm.

Our algorithm is ensured to find solutions that are either exact or approximate, with an approximation ratio ε and a rounding factor δ. This approximation does not compromise the feasibility of the solution, only its optimality. Specifically, GLPG outputs an (ε, δ)-solution $\boldsymbol{\alpha}$ such that:

- $\boldsymbol{\alpha}$ is feasible:

$$\sum_{i \in \mathcal{I}} \frac{x_i}{\sum_{\mathbf{y} \in \mathcal{X}} y_i \, \alpha_{\mathbf{y}}} \leq \Delta_{\mathbf{x}}^2, \qquad \forall \mathbf{x} \in \mathcal{X}, \tag{4.3.6}$$

$$\eta_{\mathbf{x}} \geq 0, \qquad \forall \mathbf{x} \in \mathcal{X}. \tag{4.3.7}$$

- $\boldsymbol{\alpha}$ is almost optimum, i.e. up to a multiplicative error ε^{-1} and an additive error δ [i.e. $\boldsymbol{\alpha}$ is an (ε, δ)-optimum solution]:

$$\sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, \Delta_{\mathbf{x}} \leq \frac{C(\boldsymbol{\theta})}{\varepsilon} + \delta. \tag{4.3.8}$$

  In particular, if the underlying budgeted optimisation algorithm is exact, $\varepsilon = 1$ and the optimality then only depends on δ, a parameter that can be freely tuned:

$$\sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, \Delta_{\mathbf{x}} \leq C(\boldsymbol{\theta}) + \delta. \tag{4.3.9}$$

  However, the choice of value of δ has a direct impact on the computational complexity of our method: the lower the value of δ, the higher the complexity of the algorithm.

THEOREM 19. *Consider $\delta > 0$. Let Assumptions 16, 17, and 18 hold. GLPG outputs $\boldsymbol{\alpha}$, an (ε, δ)-optimum solution to (4.1.1) in time polynomial in d, δ^{-1}, and $\|\boldsymbol{\theta}\|_\infty$.*

The proof of this theorem is made in three steps, detailed in the next three sections (4.3.3, 4.3.5, and 4.3.6). The corresponding algorithm is shown in full detail in Algorithm 17.

4.3.3. Reformulation to lower dimensionality. The first step to prove Theorem 19 is to show that a solution to (4.1.1), with $|\mathcal{X}|$ variables, can be derived from a solution to another, smaller optimisation program, with a number of variables that only depends on a polynomial of d (whereas $|\mathcal{X}|$ is typically an exponential of d). The number of constraints remains exponential so far.

The major idea of the reformulation is to forgo optimising over the variables $\eta_{\mathbf{x}}$ that indicate how often each solution $\mathbf{x} \in \mathcal{X}$ should be played. Instead, the new formulation uses the variables $t_i = \sum_{\mathbf{x}\in\mathcal{X}} x_i\, \eta_{\mathbf{x}}$. These variables are proportional to the number of samples that should be obtained to correctly estimate the value of $\boldsymbol{\theta}$. Inverting this transformation is not trivial, and the third step of the proof is devoted to it (Section 4.3.6).

PROPOSITION 20. *If $\boldsymbol{\alpha}^\star$ is the optimum solution to (4.1.1) and $\mathbf{w}^\star$ the optimum solution to the following optimisation program*

$$\begin{array}{lcl} \min & \mathbf{q}^T\, \mathbf{w} & \\ s.t. & \mathbf{M}\, \mathbf{w} = \mathbf{0} & \\ & \sum_{i\in\mathcal{I}} \frac{x_i}{w_i} \leq \Delta_{\mathbf{x}}^2 & \forall \mathbf{x} \in \mathcal{X} \\ & w_i \geq w_{\min} & \forall i \in \mathcal{I} \\ & w_i \geq 0 & \forall i \in \{1, 2 \dots d\} \end{array} \tag{4.3.10}$$

where

$$\mathbf{M} = \left(\mathbf{I}_c - \frac{\mathbf{b}\,\mathbf{b}^{\mathbf{T}}}{\|\mathbf{b}\|_2^2}\right) \mathbf{A}, \quad \textit{where } \mathbf{I}_c \textit{ is the identity matrix of size } c, \tag{4.3.11}$$

$$\mathbf{q} = \left(\boldsymbol{\theta}^T\, \mathbf{x}^\star\right) \frac{\mathbf{b}^T\, \mathbf{A}}{\|\mathbf{b}\|^2} - \boldsymbol{\theta}, \tag{4.3.12}$$

$$w_{\min} = \frac{1}{m^2\, \|\boldsymbol{\theta}\|_\infty^2}, \tag{4.3.13}$$

then the optimum solutions are related by:

$$\mathbf{w}^\star = \sum_{\mathbf{x}\in\mathcal{X}} \mathbf{x}\,\eta^\star_{\mathbf{x}}. \tag{4.3.14}$$

Proof. Indeed, the original formulation (4.1.1) only depends on two combinations of the $\boldsymbol{\alpha}$:

$$v = \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}, \qquad \mathbf{w} = \sum_{\mathbf{x}\in\mathcal{X}} \mathbf{x}\,\eta_{\mathbf{x}}. \tag{4.3.15}$$

While this is obvious for the constraints, it is not for the objective function. It can be rewritten as follows:

$$\begin{aligned}
\sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\,\Delta_{\mathbf{x}} &= \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\,\boldsymbol{\theta}^{\mathbf{T}}\,(\mathbf{x}^\star - \mathbf{x}), \qquad \text{by definition of the gap } \Delta_{\mathbf{x}} \\
&= \underbrace{\left(\sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\right)}_{v}\boldsymbol{\theta}^{\mathbf{T}}\,\mathbf{x}^\star - \boldsymbol{\theta}^{\mathbf{T}}\,\underbrace{\sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\,\mathbf{x}}_{\mathbf{w}} \\
&= v\,\left(\boldsymbol{\theta}^{\mathbf{T}}\,\mathbf{x}^\star\right) - \boldsymbol{\theta}^{\mathbf{T}}\,\mathbf{w}.
\end{aligned}$$

$\mathbf{w}$ is a conic combination of solutions of $\mathcal{X}$, because $\boldsymbol{\alpha} \geq \mathbf{0}$. Thus, $\mathbf{w}/v$ must be a convex combination of such solutions. Using Assumption (17), the set of values for $\mathbf{w}$ and v can be described as:

$$\begin{aligned}
\left\{(\mathbf{w}, v) \;\middle|\; \begin{array}{l} v = \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}} \in \mathbb{R}^+, \\ \mathbf{w} = \sum_{\mathbf{x}\in\mathcal{X}} \mathbf{x}\,\eta_{\mathbf{x}} \in \mathbb{R}^d_+ \end{array}\right\} &= \left\{(\mathbf{w}, v) \;\middle|\; \frac{\mathbf{w}}{v} \in \operatorname{conv}(\mathcal{X}),\ v \geq 0\right\} \\
&= \left\{(\mathbf{w}, v) \;\middle|\; \frac{\mathbf{A}\,\mathbf{w}}{v} = \mathbf{b},\ \mathbf{w} \geq \mathbf{0},\ v \geq 0\right\} \\
&= \{(\mathbf{w}, v) \,|\, \mathbf{A}\,\mathbf{w} = v\,\mathbf{b},\ \mathbf{w} \geq \mathbf{0},\ v \geq 0\}.
\end{aligned}$$

If $\mathbf{b} = \mathbf{0}$, the set reduces to $\{(\mathbf{w}, v) \,|\, \mathbf{A}\,\mathbf{w} = \mathbf{0},\ \mathbf{w} \geq \mathbf{0},\ v \geq 0\}$, i.e. the value of v only has a lower bound of zero. Otherwise, the value of v can be determined as follows:

$$\begin{aligned}
\mathbf{A}\,\mathbf{w} = v\,\mathbf{b} \quad &\iff \quad \mathbf{b}^T\,\mathbf{A}\,\mathbf{w} = v\,\mathbf{b}^T\,\mathbf{b} = v\,\|\mathbf{b}\|_2^2 \\
&\iff \quad v = \frac{\mathbf{b}^T\,\mathbf{A}\,\mathbf{w}}{\|\mathbf{b}\|_2^2}.
\end{aligned}$$

Injecting this back into the linear constraints for $\mathbf{w}$ and v,

$$\begin{aligned}
\mathbf{A}\,\mathbf{w} = v\,\mathbf{b} \quad &\iff \quad \mathbf{A}\,\mathbf{w} - \frac{\mathbf{b}^T\,\mathbf{A}\,\mathbf{w}}{\|\mathbf{b}\|_2^2}\,\mathbf{b} = \mathbf{0}. \\
&\iff \quad \mathbf{M}\,\mathbf{w} = \mathbf{0},
\end{aligned}$$

by definition of $\mathbf{M}$.

Finally, the objective function can be rewritten as:

$$\begin{aligned}
\sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\,\Delta_{\mathbf{x}} &= v\,\left(\boldsymbol{\theta}^{\mathbf{T}}\,\mathbf{x}^\star\right) - \boldsymbol{\theta}^{\mathbf{T}}\,\mathbf{w} \\
&= \mathbf{w}
\end{aligned}$$

$$= \underbrace{\left[\left(\boldsymbol{\theta}^{\mathbf{T}} \mathbf{x}^{\star} \right) \frac{\mathbf{b}^T \mathbf{A}}{\|\mathbf{b}\|_2^2} - \boldsymbol{\theta}^{\mathbf{T}} \right]}_{\mathbf{q}} \mathbf{w},$$

by definition of $\mathbf{q}$.

Lemma 21 concludes the proof with the lower bound on $\mathbf{w}$. □

4.3.4. Technical lemmas. We require several technical results in order to complete the proof of Theorem 19. In this section, $\boldsymbol{\alpha}^{\star}$ is an optimum solution to (4.1.1) and $\mathbf{w}^{\star}$ a corresponding optimum solution to (4.3.10).

Let $\tilde{\mathbf{w}}$ be the following vector:

$$\tilde{\mathbf{w}} = \sum_{i=1}^{d} \frac{m}{\Delta_{\min}^2} \mathbf{x}^{(i)}, \tag{4.3.16}$$

where the $\mathbf{x}^{(i)}$ are the solutions mentioned in Assumption 16. $\tilde{\mathbf{w}}$ is intended to be a solution of the Graves-Lai problem, not necessarily optimum. Indeed, by construction, each component has a lower bound of:

$$\tilde{w}_i \geq \frac{m}{\Delta_{\min}^2}. \tag{4.3.17}$$

Furthermore, $\tilde{\mathbf{w}}$ is a feasible solution, because, for any $\mathbf{x} \in \mathcal{X}$,

$$\begin{aligned} \sum_{i \in \mathcal{I}} \frac{x_i}{\tilde{w}_i} &\leq \frac{\Delta_{\min}^2}{m} \sum_{i \in \mathcal{I}} x_i \\ &\leq \frac{\Delta_{\min}^2}{m} \boxed{\sum_{i=1}^{d}} x_i \\ &\leq \frac{\Delta_{\min}^2}{m} \times m \\ &= \Delta_{\min}^2 \leq \Delta_{\mathbf{x}}^2. \end{aligned}$$

Lemma 21. *Each component w_i of $\mathbf{w}$, with $i \in \{1, 2 \dots d\}$, has a lower bound of $w_{\min} = m^{-2} \|\boldsymbol{\theta}\|_{\infty}^{-2}$.*

Proof. The lower bound on $\mathbf{w}$ is obtained thanks to Assumption 16. Taking a feasible solution $\mathbf{w} \in \mathbb{R}_+^d$, the convex constraint writes:

$$\frac{1}{w_i} \leq \sum_{j \in \mathcal{I}} \frac{x_j^{(i)}}{w_j} \leq \Delta_{\mathbf{x}^{(i)}}^2 \leq \left(\boldsymbol{\theta}^T \mathbf{x}^{\star} \right)^2 \leq m^2 \|\boldsymbol{\theta}\|_{\infty}^2, \tag{4.3.18}$$

where $\mathbf{x}^{(i)}$ is a solution in $\mathcal{X}$ such that $x_i^{(i)} = 1$, per Assumption 16. As the previous inequality holds for any $i \in \{1, 2 \dots d\}$, we can impose the following lower bound on $\mathbf{w}$:

$$\mathbf{w} \geq w_{\min} = \frac{1}{m^2 \|\boldsymbol{\theta}\|_{\infty}^2}. \tag{4.3.19}$$

□

Lemma 22. *The optimum value of both (4.1.1) and (4.3.10), $\mathbf{q}^T \mathbf{w}^{\star}$, has an upper bound of $m^2 d \|\boldsymbol{\theta}\|_{\infty}$.*

PROOF. As $\mathbf{w}^\star$ is the optimum solution, the value of the objective function is less than or equal to that of any feasible $\mathbf{w}$ in (4.3.10).

$$\mathbf{q}^T \mathbf{w}^\star \leq \mathbf{q}^T \mathbf{w}. \tag{4.3.20}$$

In particular, consider the solution $\tilde{\mathbf{w}}$ defined earlier in (4.3.16):

$$\begin{aligned}
\mathbf{q}^T \mathbf{w}^\star &\leq \frac{m}{\Delta_{\min}^2} \sum_{i=1}^{d} \Delta_{\mathbf{x}^{(i)}}, \text{ by construction of } \tilde{\mathbf{w}} \\
&\leq m \sum_{i=1}^{d} \frac{\Delta_{\max}}{\Delta_{\min}^2}, \text{ by definition of } \Delta_{\max} \\
&\leq m \sum_{i=1}^{d} \Delta_{\max}, \text{ by Assumption (18)} \\
&\leq m^2 \, d \, \|\boldsymbol{\theta}\|_\infty .
\end{aligned}$$

□

LEMMA 23. *The Euclidean norm of* $\mathbf{w}^\star$ *has an upper bound of* $m^{5/2} \, d \, \|\boldsymbol{\theta}\|_\infty$.

PROOF. Due to Proposition (20),

$$\mathbf{w}^\star = \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x} \, \eta_{\mathbf{x}} \quad \text{and} \quad \mathbf{q}^T \mathbf{w}^\star = \sum_{\mathbf{x} \in \mathcal{X}} \eta_{\mathbf{x}} \, \Delta_{\mathbf{x}}. \tag{4.3.21}$$

Because $\Delta_{\mathbf{x}} \leq \Delta_{\min}$ by definition,

$$\mathbf{q}^T \mathbf{w}^\star \geq \Delta_{\min} \sum_{\mathbf{x} \in \mathcal{X}} \eta_{\mathbf{x}}, \quad \text{hence} \quad \sum_{\mathbf{x} \in \mathcal{X}} \eta_{\mathbf{x}} \leq \frac{\mathbf{q}^T \mathbf{w}^\star}{\Delta_{\min}}. \tag{4.3.22}$$

By Minkowski's inequality,

$$\|\mathbf{w}^\star\|_2 = \left\| \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x} \, \eta_{\mathbf{x}} \right\|_2 \leq \sum_{\mathbf{x} \in \mathcal{X}} \eta_{\mathbf{x}} \underbrace{\|\mathbf{x}\|_2}_{\leq \sqrt{m}}. \tag{4.3.23}$$

Merging with the previous result and Lemma (22),

$$\|\mathbf{w}^\star\|_2 \leq \sqrt{m} \, \frac{\mathbf{q}^T \mathbf{w}^\star}{\Delta_{\min}} \leq \sqrt{m} \, m^2 \, d \, \|\boldsymbol{\theta}\|_\infty = m^{5/2} \, d \, \|\boldsymbol{\theta}\|_\infty . \tag{4.3.24}$$

□

LEMMA 24. *Consider* $\mathcal{M}$ *a convex set with the orthogonal projection operator* $\Pi_{\mathcal{M}}$*, a scalar* $\eta > 0$*, arbitrary vectors* $\hat{\mathbf{w}}$*,* $\mathbf{g}^{(1)}$*,* $\mathbf{g}^{(2)} \ldots \ \mathbf{g}^{(M)}$ *of* $\mathbb{R}^d$*, and a sequence* $\{w_m\}$ *defined as:*

$$w^{(m+1)} = \Pi_{\mathcal{M}} \left\{ \mathbf{w}^{(m)} - \eta \, \mathbf{g}^{(m)} \right\}. \tag{4.3.25}$$

Then, the following equality holds:

$$\sum_{m=1}^{M} \left(\mathbf{w}^{(m)} - \hat{\mathbf{w}} \right)^T \mathbf{g}^{(m)} \leq \frac{\|\hat{\mathbf{w}}\|_2^2}{2 \, \eta} + \frac{\eta}{2} \sum_{m=1}^{M} \left\| \mathbf{g}^{(m)} \right\|^2. \tag{4.3.26}$$

[**161**, Lemma 14.1] first states this result without the projection step, and afterwards argue that their proof still holds when a projection step is added, as in Lemma 24.

4.3.5. Approximate subgradient descent. The next step is to solve the reduced form (4.3.10) using an iterative scheme. We split the constraints into two groups:

- the convex constraints: they are handled by penalisation. For each solution $\mathbf{x} \in \mathcal{X}$, define the function $h_{\mathbf{x}}(\mathbf{w})$ as the value of the convex constraint of (4.3.10):

$$h_{\mathbf{x}}(\mathbf{w}) = \sum_{i \in \mathcal{I}} \frac{x_i}{w_i} - \Delta_{\mathbf{x}}^2. \tag{4.3.27}$$

  The corresponding constraint of (4.3.10) is violated when $h_{\mathbf{x}}(\mathbf{w}) > 0$.
- the linear constraints: they are handled by projection onto the polytope defined by these constraints, $\mathcal{M}$.

$$\mathcal{M} = \left\{ \mathbf{w} \in \mathbb{R}_+^d \,\middle|\, \mathbf{M}\,\mathbf{w} = \mathbf{0}, \quad w_i \geq w_{\min} \quad \forall i \in \mathcal{I} \right\}. \tag{4.3.28}$$

Let $\bullet^+$ denote the positive part of $\bullet$, i.e. $\max\{\bullet, 0\}$. The optimisation program (4.3.10) where the convex constraints are penalised with a weight $\lambda > 0$ is:

$$\begin{array}{lll} \min & \mathbf{q}^T\,\mathbf{w} + \lambda \max_{\mathbf{x} \in \mathcal{X}} \left\{ [h_{\mathbf{x}}(\mathbf{w})]^+ \right\} & \\ \text{s.t.} & \mathbf{M}\,\mathbf{w} = \mathbf{0} & \\ & w_i \geq w_{\min} & \forall i \in \mathcal{I} \\ & w_i \geq 0 & \forall i \in \{1, 2 \dots d\}\,. \end{array} \tag{4.3.29}$$

This new formulation has several peculiarities.

- The objective function is not smooth, due to the maximum operator. Nevertheless, as the maximum convex functions, the objective function remains convex.
- The value of λ must be appropriately large to ensure that the constraints are satisfied.
- It only has d variables and $\mathcal{O}(c + d)$ constraints, where c is typically upper bounded by a polynomial in d.

We solve (4.3.29) by the means of an approximate projected subgradient method with a fixed step length $\eta > 0$. If $\Pi_{\mathcal{M}}$ is the projection operator onto $\mathcal{M}$, our technique uses the following iteration rules for $m \in \{0, 1, 2 \dots M-1\}$:

$$\mathbf{w}^{(0)} = (w_{\min}, w_{\min} \dots w_{\min})\,, \tag{4.3.30}$$

$$\mathbf{x}^{(m)} \text{ is chosen such that } \max_{\mathbf{x} \in \mathcal{X}} h_{\mathbf{x}}\left(\mathbf{w}^{(m)}\right) \leq h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right), \tag{4.3.31}$$

$$\mathbf{g}^{(m)} = \mathbf{q} + \left[\lambda\,\varepsilon\,\nabla h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right)\right] \mathbb{1}_{h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right) > 0}, \tag{4.3.32}$$

$$\mathbf{w}^{(m+1)} = \Pi_{\mathcal{M}}\left\{\mathbf{w}^{(m)} + \eta\,\mathbf{g}^{(m)}\right\}. \tag{4.3.33}$$

The result of our numerical procedure is the average iterate $\overline{\mathbf{w}}$, not the last iterate:

$$\overline{\mathbf{w}} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{w}^{(m)}. \tag{4.3.34}$$

Using the same techniques as in Section 3.1, we can compute $\mathbf{x}^{(m)}$ in time $\mathcal{O}\big[\mathrm{poly}\big(d, \delta^{-1}, \|\boldsymbol{\theta}\|_\infty\big)\big]$. All the other steps can be implemented in time $\mathcal{O}\big[\mathrm{poly}\big(d, \delta^{-1}, \|\boldsymbol{\theta}\|_\infty\big)\big]$ (the case of the projection step is detailed in Section 4.3.5.2). As the algorithm only makes M iterations, with M itself bounded by $\mathcal{O}\big[\mathrm{poly}\big(d, \delta^{-1}, \|\boldsymbol{\theta}\|_\infty\big)\big]$, the whole procedure has a computational complexity $\mathcal{O}\big[\mathrm{poly}\big(d, \delta^{-1}, \|\boldsymbol{\theta}\|_\infty\big)\big]$.

4.3.5.1. *Subgradient and approximate subgradient.* When the underlying budgeted-linear-maximisation algorithm is exact (i.e. $\varepsilon = 1$), $\mathbf{x}^{(m)}$ is an exact maximiser of $\max_{\mathbf{x}\in\mathcal{X}} h_{\mathbf{x}}\big(\mathbf{w}^{(m)}\big)$, and $\mathbf{g}^{(m)}$ is an exact subgradient of the objective function of (4.3.29): our method reduces to the standard subgradient method in this case. Otherwise, $\varepsilon < 1$ and our algorithm guarantees that, for any $\mathbf{x} \in \mathcal{X}$, $h_{\mathbf{x}}\big(\varepsilon\, \mathbf{w}^{(m)}\big)$ cannot become too large.

4.3.5.2. *Projection step.* The projection operator $\Pi_{\mathcal{M}}$ is not an inconsequent part of the global method. Evaluating $\Pi_{\mathcal{M}}$ amounts to solving the following optimisation program:

$$\Pi_{\mathcal{M}}\Big(\mathbf{w}^{(m)} + \eta\, \mathbf{g}^{(m)}\Big) \in \quad \underset{\mathbf{w}\in\mathbb{R}^d_+}{\arg\min} \left\| \Big[\mathbf{w}^{(m)} + \eta\, \mathbf{g}^{(m)}\Big] - \mathbf{w} \right\|_2^2$$
$$\text{s.t.} \quad \mathbf{M}\,\mathbf{w} = \mathbf{0}, \quad w_i \geq w_{\min} \quad \forall i \in \mathcal{I}.$$

It can easily be solved by techniques like interior-point methods [**156**, Section 11.4]. The initial iterate for these methods can be $\mathbf{w}^{(m)}$, a solution that is already known to be feasible by construction, although warm-starting techniques are known not to improve running times in practice [**162**].

4.3.5.3. *Choice of parameters.*

Proposition 25. *With a target accuracy $\delta > 0$ and the approximation ratio ε, defining*

$$\delta_2 = \frac{\delta\,\varepsilon}{m^2\, d\, \|\boldsymbol{\theta}\|_\infty}, \qquad \delta_1 = \frac{\delta}{2 + 2\,\delta_2}, \qquad q_1 = \|\mathbf{q}\|_2^2 + \frac{\lambda^2\, d\, m^8\, \|\boldsymbol{\theta}\|_\infty^8}{\varepsilon^2}, \tag{4.3.35}$$

if the parameters of the methods (i.e. the penalty parameter $\lambda > 0$, the step length $\eta > 0$, and the number of iterations $M \in \mathbb{N}$) are as follows

$$\lambda = \frac{\delta_1 + m^2\, d\, \|\boldsymbol{\theta}\|_\infty}{\delta_2}, \tag{4.3.36}$$

$$M = \left\lceil q_1\, \frac{m^5\, d^2\, \|\boldsymbol{\theta}\|_\infty^2}{\delta_1^2\, \varepsilon^2} \right\rceil, \tag{4.3.37}$$

$$\eta^2 = \frac{m^5\, d^2\, \|\boldsymbol{\theta}\|_\infty^2}{\varepsilon^2\, T\, q_1}, \tag{4.3.38}$$

then the numerical procedure (4.3.30)-(4.3.34) runs in time polynomial in d, δ^{-1}, and $\|\boldsymbol{\theta}\|_\infty$. Furthermore, the average iterate $\overline{\mathbf{w}}$ defined in (4.3.34) yields a solution $\overline{\mathbf{w}}' = (1 + \delta_2)\, \overline{\mathbf{w}}$ that is an (ε, δ)-optimum solution to (4.3.10).

4.3.5.4. *Convergence proof.* Let E be the error at the end of the projected subgradient algorithm:

$$E = \mathbf{q}^T\, \overline{\mathbf{w}} - \frac{\mathbf{q}^T\, \mathbf{w}^\star}{\varepsilon} + \lambda \max_{\mathbf{x}\in\mathcal{X}} \Big\{ [h_{\mathbf{x}}(\overline{\mathbf{w}})]^+ \Big\}. \tag{4.3.39}$$

As $\overline{\mathbf{w}}$ is defined as the average of the M iterates (4.3.34), because $\mathbf{w} \mapsto \max_{\mathbf{x}\in\mathcal{X}} \Big\{ [h_{\mathbf{x}}(\mathbf{w})]^+ \Big\}$ is a convex function and $\lambda > 0$, we can use Jensen's inequality:

$$E = \mathbf{q}^T \left(\frac{1}{M} \sum_{m=1}^{M} \mathbf{w}^{(m)} \right) - \frac{\mathbf{q}^T\, \mathbf{w}^\star}{\varepsilon} + \lambda \max_{\mathbf{x}\in\mathcal{X}} \left\{ \left[h_{\mathbf{x}} \left(\frac{1}{M} \sum_{m=1}^{M} \mathbf{w}^{(m)} \right) \right]^+ \right\}$$

$$\leq \frac{1}{M}\sum_{m=1}^{M}\left[\mathbf{q}^T\,\mathbf{w}^{(m)}\right] - \frac{\mathbf{q}^T\,\mathbf{w}^\star}{\varepsilon} + \frac{\lambda}{M}\,\sum_{m=1}^{M}\max_{\mathbf{x}\in\mathcal{X}}\left\{\left[h_{\mathbf{x}}\Big(\mathbf{w}^{(m)}\Big)\right]^+\right\}.$$

By choice of $\mathbf{x}^{(m)}$, we know that

$$\max_{\mathbf{x}\in\mathcal{X}}\left\{\left[h_{\mathbf{x}}\Big(\mathbf{w}^{(m)}\Big)\right]^+\right\} \leq \left[h_{\mathbf{x}^{(m)}}\Big(\varepsilon\,\mathbf{w}^{(m)}\Big)\right]^+. \tag{4.3.40}$$

$h_{\mathbf{x}}$ being a smooth convex function, $\mathbf{w}\mapsto[h_{\mathbf{x}}(\mathbf{w})]^+$ is a nonsmooth convex function and one of its subgradient is

$$\varepsilon\,\nabla h_{\mathbf{x}^{(m)}}(\varepsilon\,\mathbf{w})\;\mathbb{1}_{h_{\mathbf{x}^{(m)}}(\varepsilon\,\mathbf{w})>0}, \tag{4.3.41}$$

the following inequality holds, by definition of a subgradient, for all $\mathbf{w}\in\mathbb{R}^d_+$:

$$[h_{\mathbf{x}^{(m)}}(\varepsilon\,\mathbf{w})]^+ - [h_{\mathbf{x}^{(m)}}(\mathbf{w}^\star)]^+ \leq \left(\mathbf{w}-\frac{\mathbf{w}^\star}{\varepsilon}\right)^T\left(\varepsilon\,\nabla h_{\mathbf{x}^{(m)}}(\varepsilon\,\mathbf{w})\;\mathbb{1}_{h_{\mathbf{x}^{(m)}}(\varepsilon\,\mathbf{w})>0}\right). \tag{4.3.42}$$

As $\mathbf{w}^\star$ is a feasible solution, $h_{\mathbf{x}}(\mathbf{w}^\star)\leq 0$, and $[h_{\mathbf{x}^{(m)}}(\mathbf{w}^\star)]^+ = 0$. Using the definition of $\mathbf{x}^{(m)}$ in (4.3.40),

$$\max_{\mathbf{x}\in\mathcal{X}}\left\{\left[h_{\mathbf{x}}\Big(\mathbf{w}^{(m)}\Big)\right]^+\right\} \leq \left(\mathbf{w}-\frac{\mathbf{w}^\star}{\varepsilon}\right)^T\left(\varepsilon\,\nabla h_{\mathbf{x}^{(m)}}(\varepsilon\,\mathbf{w})\;\mathbb{1}_{h_{\mathbf{x}^{(m)}}(\varepsilon\,\mathbf{w})>0}\right). \tag{4.3.43}$$

Relating this to the error E, we get:

$$E \leq \frac{1}{M}\sum_{m=1}^{M}\left(\mathbf{w}^{(m)}-\frac{\mathbf{w}^\star}{\varepsilon}\right)^T\left(\mathbf{q}+\lambda\,\varepsilon\,\nabla h_{\mathbf{x}^{(m)}}\Big(\varepsilon\,\mathbf{w}^{(m)}\Big)\;\mathbb{1}_{h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right)>0}\right). \tag{4.3.44}$$

We now apply Lemma (24) with the definition of the algorithm (4.3.32), (4.3.33), and (4.3.34), then Minkowski's inequality:

$$\begin{aligned}
E &\leq \frac{1}{2\,M}\left[\frac{\left\|\mathbf{w}^{(0)}-\mathbf{w}^\star/\varepsilon\right\|_2^2}{\eta} + \eta\,\sum_{m=1}^{M}\left\|\mathbf{q}+\lambda\,\varepsilon\,\nabla h_{\mathbf{x}^{(m)}}\Big(\varepsilon\,\mathbf{w}^{(m)}\Big)\;\mathbb{1}_{h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right)>0}\right\|_2^2\right]\\
&\leq \frac{1}{2\,M}\left[\frac{\left\|\mathbf{w}^{(0)}-\mathbf{w}^\star/\varepsilon\right\|_2^2}{\eta} + \eta\,M\;\|\mathbf{q}\|_2^2 + \eta\,\lambda^2\,\varepsilon^2\,\sum_{m=1}^{M}\left\|\nabla h_{\mathbf{x}^{(m)}}\Big(\varepsilon\,\mathbf{w}^{(m)}\Big)\right\|_2^2\right].
\end{aligned}$$

We now determine upper bounds for each term.

Distance to $\mathbf{w}^{(0)}$.: Since $\mathbf{w}^{(0)} = (w_{\min}, w_{\min}\ldots w_{\min})$ and $\mathbf{w}^\star/\varepsilon \geq (w_{\min}, w_{\min}\ldots w_{\min})$, using Lemma (23),

$$\left\|\mathbf{w}^{(0)}-\mathbf{w}^\star/\varepsilon\right\|_2 \leq \frac{\|\mathbf{w}^\star\|}{\varepsilon} \leq \frac{m^{5/2}\,d\;\|\boldsymbol{\theta}\|_\infty}{\varepsilon}. \tag{4.3.45}$$

Subgradient norm.: By definition of $h_{\mathbf{x}}$,

$$\nabla h_{\mathbf{x}^{(m)}}\Big(\varepsilon\,\mathbf{w}^{(m)}\Big) = -\left(\frac{x_1^{(m)}}{\left[\varepsilon\,w_1^{(m)}\right]^2}, \frac{x_2^{(m)}}{\left[\varepsilon\,w_2^{(m)}\right]^2}\cdots\frac{x_d^{(m)}}{\left[\varepsilon\,w_d^{(m)}\right]^2}\right). \tag{4.3.46}$$

Therefore, the norm of the gradient is:

$$\begin{aligned}\left\|\nabla h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right)\right\|_2^2 &\leq \sum_{i=1}^{d}\frac{x_i^{(m)}}{\left[\varepsilon\,w_i^{(m)}\right]^4}\\ &\leq \frac{d}{\varepsilon^4\,w_{\min}^4}\\ &= \frac{d\,m^8\,\|\boldsymbol{\theta}\|_\infty^8}{\varepsilon^4}.\end{aligned}$$

The error bound now becomes:

$$E \leq \frac{1}{2\,M}\left[\frac{m^5\,d^2\,\|\boldsymbol{\theta}\|_\infty^2}{\varepsilon^2\,\eta} + \eta\,M\,\left(\|\mathbf{q}\|_2^2 + \frac{\lambda^2\,d\,m^8\,\|\boldsymbol{\theta}\|_\infty^8}{\varepsilon^2}\right)\right]. \tag{4.3.47}$$

The value of η that minimises the error equalises the two terms:

$$\eta^2 = \frac{m^5\,d^2\,\|\boldsymbol{\theta}\|_\infty^2}{\varepsilon^2\,M\,\left(\|\mathbf{q}\|_2^2 + \frac{\lambda^2\,d\,m^8\,\|\boldsymbol{\theta}\|_\infty^8}{\varepsilon^2}\right)}. \tag{4.3.48}$$

With this value of the step size, the error bound becomes:

$$E \leq \frac{m^5\,d^2\,\|\boldsymbol{\theta}\|_\infty^2}{\varepsilon^2\,\eta\,M}. \tag{4.3.49}$$

We now set M so that $E \leq \delta_1$:

$$M = \left\lceil \frac{m^5\,d^2\,\|\boldsymbol{\theta}\|_\infty^2}{\delta_1^2\,\varepsilon^2}\,\left(\|\mathbf{q}\|_2^2 + \frac{\lambda^2\,d\,m^8\,\|\boldsymbol{\theta}\|_\infty^8}{\varepsilon^2}\right)\right\rceil. \tag{4.3.50}$$

As $E \leq \delta_1$ with this value of M, the constraint violation can be bounded as follows:

$$E = \mathbf{q}^T\,\overline{\mathbf{w}} - \frac{\mathbf{q}^T\,\mathbf{w}^\star}{\varepsilon} + \lambda\,\max_{\mathbf{x}\in\mathcal{X}}\left\{[h_{\mathbf{x}}(\overline{\mathbf{w}})]^+\right\} \leq \delta_1, \tag{4.3.51}$$

$$\begin{aligned}\max_{\mathbf{x}\in\mathcal{X}}\left\{[h_{\mathbf{x}}(\overline{\mathbf{w}})]^+\right\} &\leq \frac{\delta_1 - \mathbf{q}^T\,\overline{\mathbf{w}} + \frac{\mathbf{q}^T\,\mathbf{w}^\star}{\varepsilon}}{\lambda}\\ &\leq \frac{\delta_1 + m^2\,d\,\|\boldsymbol{\theta}\|_\infty}{\lambda}, \text{ by Lemma (22).}\end{aligned}$$

To ensure that the constraint violation is bounded by δ_2, i.e. so that

$$\max_{\mathbf{x}\in\mathcal{X}}\left\{[h_{\mathbf{x}}(\overline{\mathbf{w}})]^+\right\} \leq \delta_2, \tag{4.3.52}$$

the parameter λ must take the following value:

$$\lambda = \frac{\delta_1 + m^2\,d\,\|\boldsymbol{\theta}\|_\infty}{\delta_2}. \tag{4.3.53}$$

The final result is defined as $\overline{\mathbf{w}}' = (1+\delta_2)\,\overline{\mathbf{w}}$ in Proposition (25). Since, for all $\mathbf{x} \in \mathcal{X}$, the constraint violation of $\overline{\mathbf{w}}$ is bounded as:

$$\sum_{i\in\mathcal{I}}\frac{x_i}{\overline{w}_i} \leq \Delta_{\mathbf{x}}^2 + \delta_2, \tag{4.3.54}$$

the constraint violation of $\overline{\mathbf{w}}'$ must be:

$$\sum_{i\in\mathcal{I}} \frac{x_i}{\overline{w}'_i} \leq \frac{\Delta_{\mathbf{x}}^2 + \delta_2}{1+\delta_2} = \Delta_{\mathbf{x}}^2 \frac{\Delta_{\mathbf{x}}^{-2} + \delta_2}{1+\delta_2} \leq \Delta_{\mathbf{x}}^2, \tag{4.3.55}$$

using the fact that $\Delta_{\mathbf{x}}^2 \geq 1$ by Assumption (18). Hence, $\max_{\mathbf{x}\in\mathcal{X}} \left\{ [h_{\mathbf{x}}(\overline{\mathbf{w}}')]^+ \right\} = 0$, indicating that $\overline{\mathbf{w}}'$ is feasible for all constraints.

The last step is to check the optimality gap of $\overline{\mathbf{w}}'$. For the average iterate $\overline{\mathbf{w}}$, we have:

$$\mathbf{q}^T \overline{\mathbf{w}} - \frac{\mathbf{q}^T \mathbf{w}^\star}{\varepsilon} \leq \mathbf{q}^T \overline{\mathbf{w}} - \frac{\mathbf{q}^T \mathbf{w}^\star}{\varepsilon} + \underbrace{\lambda \max_{\mathbf{x}\in\mathcal{X}} \left\{ [h_{\mathbf{x}}(\overline{\mathbf{w}})]^+ \right\}}_{\geq 0} \leq \delta_1. \tag{4.3.56}$$

This implies that, for the returned solution $\overline{\mathbf{w}}'$,

$$\begin{aligned} \mathbf{q}^T \overline{\mathbf{w}}' - \frac{\mathbf{q}^T \mathbf{w}^\star}{\varepsilon} &\leq (1+\delta_2)\ \delta_1 + \delta_2 \frac{\mathbf{q}^T \mathbf{w}^\star}{\varepsilon} \\ &\leq (1+\delta_2)\ \delta_1 + \frac{\delta_2\, m^2\, d\ \|\boldsymbol{\theta}\|_\infty}{\varepsilon} \\ &= \frac{\delta}{2} + \frac{\delta}{2} = \delta. \end{aligned}$$

4.3.6. Convex combination of vertices. The subgradient method generates a solution $\mathbf{w}^\star \in \mathbb{R}_+^d$ that is close to the true optimum to (4.3.10). The last step of GLPG is to retrieve an optimum solution $\boldsymbol{\alpha}^\star \in \mathbb{R}_+^{|\mathcal{X}|}$ to the original problem (4.1.1). We claim that we can do so with a computational complexity bounded by a polynomial in d, even though $\boldsymbol{\alpha}^\star$ has $|\mathcal{X}| \in \mathcal{O}(2^d)$ components. Indeed, there may be an infinite number of $\boldsymbol{\alpha}^\star$ corresponding to the same $\mathbf{w}^\star$ solution. All these solutions have the same objective value, by construction of the reformulation, and we choose an $\boldsymbol{\alpha}^\star$ with most of its entries being zero.

The two sets of variables are related by (Proposition 20):

$$\mathbf{w}^\star = \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}^\star\, \mathbf{x}. \tag{4.3.57}$$

To recover the initial variables $\boldsymbol{\alpha}$, we must decompose them into a conical combination of points within $\mathcal{X}$: the weights of this convex combination are the values for $\boldsymbol{\alpha}$. As an application of Carathéodory's theorem [**163**], there is a decomposition that uses at most $d+1$ points.

A constructive proof of Carathéodory's theorem provides a polynomial-time algorithm to provide this decomposition [**164**]. In our specific case, as we only deal with bounded polytopes, the algorithm can be largely simplified. The technique of [**164**] is summarised in Algorithm (18). Its complexity is bounded by a polynomial in d, δ^{-1}, and $\|\boldsymbol{\theta}\|_\infty$.

A common variant of this problem is the approximate Carathéodory problem, where fewer than $d+1$ points should be found; the decomposition is therefore no more ensured to be exact in all cases, and the maximum distance between the approximate decomposition and the original point is related to the number of points [**165**]. Such a technique could improve the practical run time of our algorithm (in practice, this parts often bottlenecks the performance of GLPG), at the cost of an additional approximation.

4.4. Exact computation of Graves-Lai bound for combinatorial bandits

A completely different approach to solving (4.1.1) is to use common tools from the operational-research practitioner on top of existing solver technology like CPLEX [**142**], Gurobi [**143**], Mosek [**144**], Parajito [**145**], SCIP [**146**], or Xpress [**147**].

We start from the reformulation of Proposition 20. Instead of instantiating the formulation with the exponential number of constraints (which would require first to enumerate all solutions of $\mathcal{X}$), we use traditional constraint generation [**166, 167, 168, 169**]. This technique has been in wide use for decades and yielded great performance, albeit without any complexity guarantee.

The algorithm works as follows. It first starts with a relaxed problem, without any of the complicating constraints:

$$\begin{array}{lll} \min & \mathbf{q}^T \mathbf{w} & \\ \text{s.t.} & \mathbf{M}\,\mathbf{w} = \mathbf{0} & \\ & w_i \geq w_{\min} & \forall i \in \mathcal{I} \\ & w_i \geq 0 & \forall i \in \{1, 2 \dots d\}\,. \end{array} \tag{4.4.1}$$

An off-the-shelf solver is used to find the optimum $\mathbf{w}^{(0)}$ for this optimisation program, called the *master program*. The solution is not ensured to be feasible. A separation procedure finds a decision $\mathbf{x} \in \mathcal{X}$ to add into the master program. It amounts to finding the constraint that is most violated with the solution $\mathbf{w}^{(0)}$:

$$\mathbf{x}^{(0)} \in \arg\max_{\mathbf{x} \in \mathcal{X}} \left\{ \sum_{i \in \mathcal{I}} \frac{x_i}{w_i^{(0)}} - \Delta_{\mathbf{x}}^2 \right\}. \tag{4.4.2}$$

This separation is performed in the same way as GLPG checks whether there is a decision $\mathbf{x} \in \mathcal{X}$ such that $h_{\mathbf{x}}(\mathbf{w}) < 0$ (Section (4.3.5)). Once this solution is found, it is added into the master program:

$$\begin{array}{lll} \min & \mathbf{q}^T \mathbf{w} & \\ \text{s.t.} & \mathbf{M}\,\mathbf{w} = \mathbf{0} & \\ & \sum_{i \in \mathcal{I}} \frac{x_i^{(0)}}{w_i} \leq \Delta_{\mathbf{x}^{(0)}}^2 & \\ & w_i \geq w_{\min} & \forall i \in \mathcal{I} \\ & w_i \geq 0 & \forall i \in \{1, 2 \dots d\}\,. \end{array} \tag{4.4.3}$$

This program can then be solved to optimality, before the separation procedure is called again. On the contrary, if the objective function of the separation problem is negative, no new constraint must be added, and the optimum solution has been found. Otherwise, the new constraint is added into the master program, and the algorithm loops.

Whereas GLPG (Section (4.3)) starts from a feasible solution and works toward optimality, this exact algorithm does the reverse: it starts with an initially infeasible solution, and progressively reaches feasibility.

4.5. Numerical results

We evaluate the performance of GLPG through numerical experiments, and compare the objective value at each iteration with the exact value. In nonsmooth optimisation, bundle methods converge faster than subgradient methods [**170**], notably due to the implicitly adaptive choice of step length; we also include a bundle-based implementation of GLPG where the linear constraints are implemented in the proximal step. The exact Graves-Lai bound is obtained by the technique described in Section (4.4) and using CPLEX as MISOCP solver [**142**].

The Julia [**37**] implementations of the algorithms as well as the code to run the experiments are made available online[1].

4.5.1. Experimental setting. We run experiments on four different combinatorial sets, for various problem sizes (indicated by d). All rewards follow a normal distribution with a $1/2$ variance and an integer average.

- For m-sets, we choose $m = \lfloor d/3 \rfloor$. Regarding the rewards, $\theta_i = 1$ for $i \leq d/2$ and $\theta_i = 2$ for $i > d/2$. Any optimum solution takes $\lfloor d/3 \rfloor$ elements among the $\lfloor d/2 \rfloor$ first ones.
- For simple paths, we consider the graph $G = (V, E)$ a complete directed acyclic graph: $(i, j) \in E$ if and only if $i < j$. The source is 1 and the destination is $|V|$. The rewards are defined as $\theta_{(i,j)} = 1$ for $(i, j) \neq (1, |V|)$ and $\theta_{(1,|V|)} = 2$. We have $d = |V|$ $(|V| - 1)/2$ and $m = |V| - 1$. The optimum path is $\{(i, i + 1), \forall i \in \{1, 2 \dots |V| - 1\}\}$.
- For spanning trees, we consider $G = (V, E)$ a complete undirected graph. The rewards are set so that $\theta_{(i,j)} = 0.4$ for all edges $(i, j) \in E$ with $i \neq 1$ and $\theta_{(1,j)} = 0.55$ for each vertex $j \in V \setminus \{1\}$. We have $d = |V|(|V| - 1)/2$ and $m = |V| - 1$. The optimum decision is a star network centred on 1, i.e. $\{(1, j) \mid j \in V \setminus \{1\}\}$.
- For matchings, we consider a complete bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2|$. Regarding the rewards, $\theta_{(i,j)} = 1$ for all edges $(i, j) \in E$ where $i \neq j$ and $\theta_{(i,i)} = 2$ for each vertex $i \in V$. We have $d = |V_1|$ $|V_2|$ and $m = |V_1| = |V_2|$. The optimum decision is $\mathbf{x}^\star = \{(i, j) \mid i \in V_1, j \in V_2, i = j\}$.

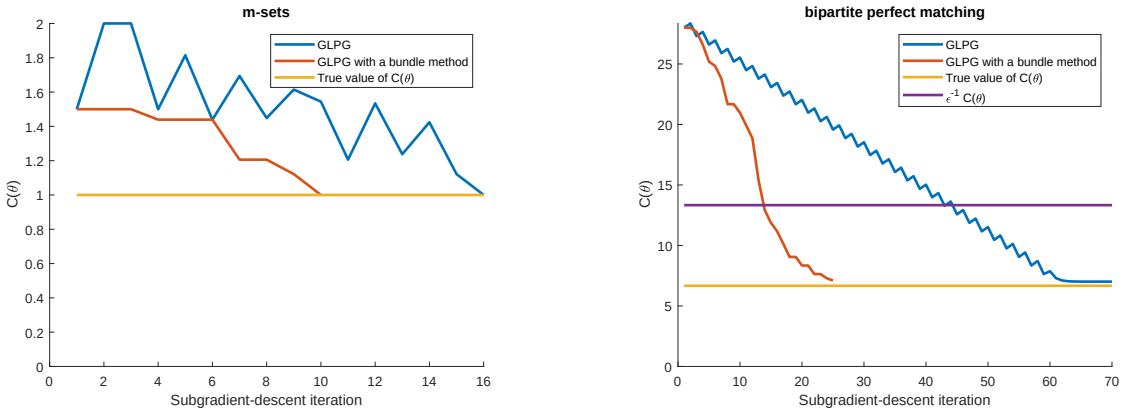
4.5.2. Convergence of GLPG. First, we study the convergence of GLPG, i.e. the value of the objective function with the iterations. The results are shown in Figure (4.5.1). The standard GLPG has a chaotic behaviour, with the objective function not decreasing monotonically at each iteration, which is due to the subgradient method: it is not a descent method, as following a subgradient never ensures that the objective function decreases. On the other hand, the bundle method never sees an increase of the objective function, although it may stagnate for a few iterations, thanks to its alternation of long and short steps.

Both methods converge to the true value of $C(\boldsymbol{\theta})$, but the bundle-based implementation of GLPG needs fewer iterations to do so. Although, for the matchings (Figure (4.5.1)b), the underlying budgeted-linear-maximisation solver has an approximation ratio of $1/2$, both methods reach a value that is very close to the true optimum, closer that the maximum theoretical difference: both techniques reach a value 0.34 higher than the true value, i.e. 5% of the true value.

4.5.3. Scaling of $C(\boldsymbol{\theta})$ with dimension. Thanks to our new algorithm, it is now possible to compute the value of $C(\boldsymbol{\theta})$ in polynomial time for a large class of combinatorial semibandit problems. In particular, this allows to determine the scaling of $C(\boldsymbol{\theta})$ when the dimension of the problem increases. Theoretical formulae are known in some cases, like matroids (including m-sets and spanning trees) [**171**], but not for more complex problems like simple paths and bipartite matchings.

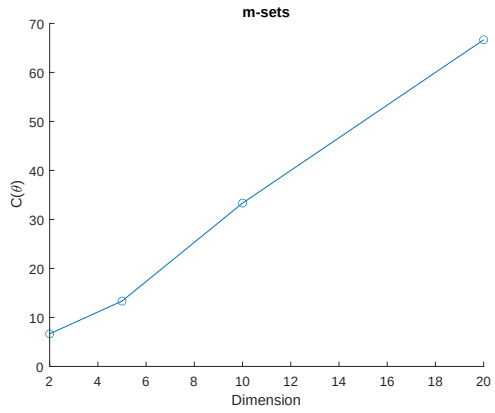
We can now compute the value of $C(\boldsymbol{\theta})$ for simple paths and bipartite matchings, as shown in Figure (4.5.2). We use the same parameters for $\boldsymbol{\theta}$ as before, which implies that $\Delta_{\min}$ linearly scales with the dimension.

[1] CombinatorialBandits.jl

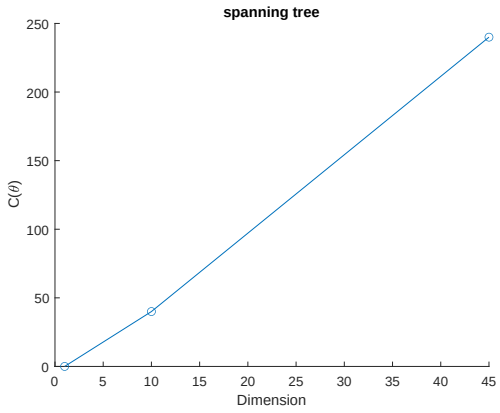


(A) m-sets, $d = 2$. (B) Matchings, $d = 4$.

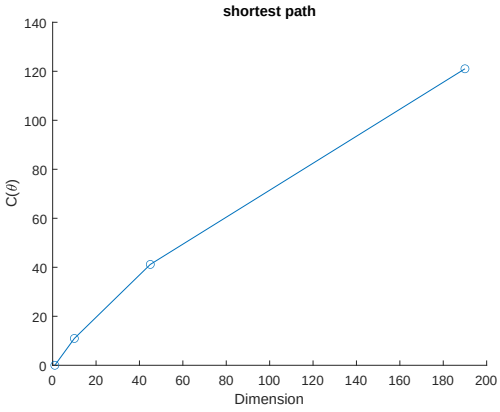
FIGURE 4.5.1. Convergence speed of GLPG and bundle-based GLPG.



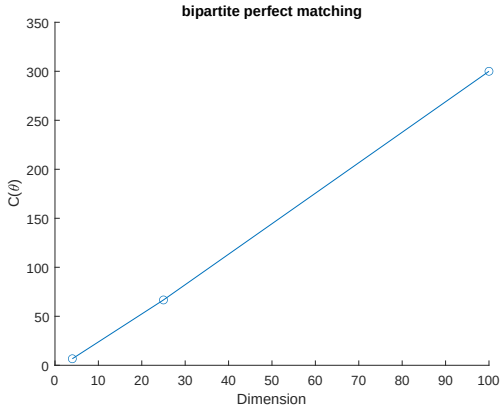
(A) m-sets.



(B) Spanning trees.



(C) Paths.



(D) Matchings.

FIGURE 4.5.2. Evolution of $C(\boldsymbol{\theta})$ with the dimension for the four studied combinatorial set.

APPENDIX A

Notations

Symbol	Meaning
$n \in \mathbb{N}_0$	Round, time step
$T \in \mathbb{N}_0$	Time horizon, last round
$k \in \mathbb{N}_0$	Number of arms
$\mathcal{A}$, $\lvert\mathcal{A}\rvert = k$	Set of arms, action space
$a_i \in \mathcal{A}$	ith arm
$r^\star \in [0,1]$	Optimum reward for one round
$r_i \in [0,1]$	Average reward of the ith arm
$\hat{r}_i(n) \in [0,1]$	Estimated reward of the ith arm at round n
$x(n) \in \mathcal{A}$	Arm played at round n
$T_i(n) \in \mathbb{N}$	Number of times the ith arm has been played up to round n
$r(n) \in [0,1]$	Reward obtained at round n
$\Delta(n) \in [0,1]$	Gap at round n
$\Delta_i \in [0,1]$	Gap of the ith arm
$\Delta_{\min} \in [0,1]$	Minimum gap for the bandit problem
$\Delta_{\max} \in [0,1]$	Maximum gap for the bandit problem
$R(n) \in [0,n]$	Regret up to round n

TABLE 1. Notations for classical bandits.

Symbol	Meaning
$\mathcal{C}$	Set of contexts
$c_i \in \mathcal{C}$	ith context (in **bold** if $\mathcal{C}$ is a vector space)
$c(n)$	Context at round n (in **bold** if $\mathcal{C}$ is a vector space)
$r_c^\star \in [0,1]$	Optimum reward for one round in context $c \in \mathcal{C}$
$r_{i,c} \in [0,1]$	Average reward of the ith arm in context $c \in \mathcal{C}$
$\hat{r}_{i,c}(n) \in [0,1]$	Estimated reward of the ith arm at round n in context $c \in \mathcal{C}$
$x_{c(n)}(n) \in \mathcal{A}$	Arm played at round n with the context $c(n) \in \mathcal{C}$
$T_{i,c}(n) \in \mathbb{N}$	Number of times the ith arm has been played up to round n in context $c \in \mathcal{C}$
$\Delta_{i,c} \in [0,1]$	Gap of the ith arm in context $c \in \mathcal{C}$

TABLE 2. Notations for contextual bandits.

APPENDIX B

Pseudocode for Algorithms

Algorithm 1 UCB-1 [**53**].

procedure UCB1
  $\hat{\theta}_i(1) = 0$ for $i \in \{1, 2 \dots k\}$
  $T_i(1) = 0$ for $i \in \{1, 2 \dots k\}$
  for $n = 1$ **to** k **do**
    $r(n) =$ reward when playing n
    $\hat{\theta}_n(n+1) = r(n)$
    $T_n(n+1) = 1$
  end for
  for $n = k+1$ **to** T **do**
    $x(n) \in \underset{i \in \{1,2 \dots k\}}{\arg\max} \hat{\theta}_i + \sqrt{\frac{2 \log n}{T_i(n)}}$
    $r(n) =$ reward when playing $x(n)$
    $\hat{\theta}_{x(n)}(n+1) = \frac{T_i(n)+1}{T_i(n)} \hat{\theta}_i(n) + \frac{r(n)}{T_i(n)+1}$
    $T_{x(n)}(n+1) = T_{x(n)}(n) + 1$
    $\hat{\theta}_i(n+1) = \hat{\theta}_i(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x\,(n)\}$
    $T_i(n+1) = T_i(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x\,(n)\}$
  end for
end procedure

Algorithm 2 KL-UCB [**55**].

procedure KL-UCB
  $\hat{\theta}_i(1) = 0$ for $i \in \{1, 2 \dots k\}$
  $T_i(1) = 0$ for $i \in \{1, 2 \dots k\}$
  for $n = 1$ **to** k **do**
    $r(n) =$ reward when playing n
    $\hat{\theta}_n(n+1) = r(n) + 1$
    $T_n(n+1) = 1$
  end for
  for $n = k+1$ **to** T **do**
    $x(n) \in \underset{i \in \{1,2 \dots k\}}{\arg\max} \left\{ q_i \in \Theta \,\middle|\, T_i(n)\ \mathrm{kl}\left[\hat{\theta}_i(n), q_i\right] \leq \log n \right\}$
    $r(n) =$ reward when playing $x(n)$
    $\hat{\theta}_{x(n)}(n+1) = \frac{T_i(n)+1}{T_i(n)}\, \hat{\theta}_i(n) + \frac{r(n)}{T_i(n)+1}$
    $T_{x(n)}(n+1) = T_{x(n)}(n) + 1$
    $\hat{\theta}_i(n+1) = \hat{\theta}_i(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x(n)\}$
    $T_i(n+1) = T_i(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x(n)\}$
  end for
end procedure

Algorithm 3 Thompson sampling for Bernoulli rewards with beta prior [**48**].

procedure ThompsonSampling
  $\hat{\theta}_i^+(1) = 1$ for $i \in \{1, 2 \dots k\}$
  $\hat{\theta}_i^-(1) = 1$ for $i \in \{1, 2 \dots k\}$
  for $n = 1$ **to** T **do**
    Draw $t_i(n) \sim \beta\left(\hat{\theta}_i^+(n), \hat{\theta}_i^-(n)\right)$
    $x(n) \in \underset{i \in \{1,2 \dots k\}}{\arg\max}\, t_k(n)$
    $r(n) =$ reward when playing $x(n)$
    $\hat{\theta}_{x(n)}^+(n+1) = \hat{\theta}_{x(n)}^+(n) + r(n)$
    $\hat{\theta}_{x(n)}^-(n+1) = \hat{\theta}_{x(n)}^-(n) + [1 - r(n)]$
    $\hat{\theta}_i^+(n+1) = \hat{\theta}_i^+(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x(n)\}$
    $\hat{\theta}_i^-(n+1) = \hat{\theta}_i^-(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x(n)\}$
  end for
end procedure

Algorithm 4 EXP3 [**50**].

procedure EXP3($\eta > 0$)
  $w_i(1) = 1$ for $i \in \{1, 2 \dots k\}$
  for $n = 1$ **to** T **do**
    $p_i(n) = (1-\eta)\ \frac{w_i(n)}{\sum_{a_j \in \mathcal{A}} w_j(n)} + \frac{\eta}{k}$ for $i \in \{1, 2 \dots k\}$
    Draw $x(n) \sim \mathbf{p}(n)$
    $r(n) =$ reward when playing $x(n)$
    $w_{x(n)}(n+1) = w_{x(n)}(n) \times \exp\left(\eta\ \frac{r(n)}{k\ p_i(n)}\right)$
    $w_i(n+1) = w_i(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x\,(n)\}$
  end for
end procedure

Algorithm 5 EXP4 [**53**].

procedure EXP4($\eta > 0$, E)
  $p_e(1) = |E|^{-1}$ for $e \in E$
  for $n = 1$ **to** T **do**
    $q_i(n) = \sum_{e \in E} p_e(n)\ E_{e,i}(n)$ for $i \in \{1, 2 \dots k\}$
    Draw $x(n) \sim \mathbf{p}(n)$
    $r(n) =$ reward when playing $x(n)$
    $\hat{r}_{x(n)}(n) = \frac{r(n)}{q_{x(n)}(n)}$
    $\hat{r}_i(n) = 0$ for $i \in \{1, 2 \dots k\} \setminus \{x\,(n)\}$
    $g_i(n) = \sum_{a_i \in \mathcal{A}} e_i(n)\ \hat{r}_i(n)$ for $i \in \{1, 2 \dots k\}$
    $p_e(n+1) = p_e(n) \times \exp(-\eta \sum_{t=1}^{n} g_i(t))$ for $e \in E$
  end for
end procedure

Algorithm 6 LinUCB [**72**].

procedure LinUCB($\beta > 0$, $\delta \in (0, 1)$, $\lambda > 0$)
  for $n = 1$ **to** k **do**
    $r(n) =$ reward when playing $\mathbf{a}_n$
    $\hat{\theta}_n(n+1) = r(n)$
  end for
  for $n = k+1$ **to** T **do**
    $\mathbf{x}(n) \in \arg\max_{\mathbf{x} \in \mathcal{A}} \left\{ \begin{array}{cc} \max & \hat{\boldsymbol{\theta}}^T \mathbf{x} \\ \text{s.t.} & \left\| \hat{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}(n-1) \right\|_2^2 \leq \beta \\ & \hat{\boldsymbol{\theta}} \in \Theta \end{array} \right\}$
    $\hat{\boldsymbol{\theta}}(n) \in \arg\min_{\hat{\boldsymbol{\theta}} \in \mathbb{R}^d} \sum_{t=1}^{n} \left[r(n) - \hat{\boldsymbol{\theta}}^T \mathbf{x}(t) \right]^2 + \lambda \left\| \hat{\boldsymbol{\theta}} \right\|_2^2$
  end for
end procedure

Algorithm 7 OSSB [**80**].

procedure OSSB($\gamma > 0$, $\varepsilon > 0$)
  $\hat{\theta}_i(1) = 0$ for $i \in \{1, 2 \dots k\}$
  $T_i(1) = 0$ for $i \in \{1, 2 \dots k\}$
  $S(1) = 0$
  for $n = 1$ **to** T **do**
    $\boldsymbol{\eta}(n)$ is the solution to the Graves-Lai optimisation program
    if $\eta_{\mathbf{x}}(n)\ (1+\gamma)\ \log n \leq T_{\mathbf{x}}(n)\,, \qquad \forall \mathbf{x} \in \mathcal{A}$ **then**
      $\mathbf{x}(n) \in \arg\max_{\mathbf{x} \in \mathcal{A}} \mathbf{x}^T\, \hat{\boldsymbol{\theta}}(n)$
      $S(n+1) = S(n)$
    else
      $S(n+1) = S(n) + 1$
      $\underline{\mathbf{x}}(n) \in \arg\min_{\mathbf{x} \in \mathcal{A}} T_{\mathbf{x}}(n)$
      if $T_{\underline{\mathbf{x}}(n)}(n) \leq \varepsilon\, S(n)$ **then**
        $\mathbf{x}(n) = \underline{\mathbf{x}}(n)$
      else
        $\mathbf{x}(n) \in \arg\min_{\mathbf{x} \in \mathcal{A}} T_{\mathbf{x}}(n) \Big/ \eta_{\mathbf{x}}(n)$
      end if
    end if
    $\mathbf{y}(n) =$ play $\mathbf{x}(n)$
    $\hat{\theta}_{x(n)}(n+1) = \frac{T_i(n)+1}{T_i(n)}\, \hat{\theta}_i(n) + \frac{r(n)}{T_i(n)+1}$
    $T_{x(n)}(n+1) = T_{x(n)}(n) + 1$
    $\hat{\theta}_i(n+1) = \hat{\theta}_i(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x\,(n)\}$
    $T_i(n+1) = T_i(n)$ for $i \in \{1, 2 \dots k\} \setminus \{x\,(n)\}$
  end for
end procedure

Algorithm 8 Combinatorial Thompson sampling for Bernoulli rewards with beta prior [**105**].

procedure CombinatorialThompsonSampling($\mathcal{X}$)
  $\hat{\theta}_i^+(1) = 1$ for $i \in \{1, 2 \dots d\}$
  $\hat{\theta}_i^-(1) = 1$ for $i \in \{1, 2 \dots d\}$
  for $n = 1$ **to** T **do**
    Draw $t_i(n) \sim \beta\left(\hat{\theta}_i^+(n)\,, \hat{\theta}_i^*(n)\right)$
    $\mathbf{x}(n) \in \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbf{x}^T\, \mathbf{t}(n)$
    $r(n)\,, \mathbf{y}(n) =$ total reward and subarm rewards when playing $\mathbf{x}(n)$
    $\hat{\theta}_i^+(n+1) = \hat{\theta}_i^+(n) + r(n)\, y_i(n)$ for $i \in \{1, 2 \dots d\}$
    $\hat{\theta}_i^-(n+1) = \hat{\theta}_i^+(n) + [1 - r(n)]\, y_i(n)$ for $i \in \{1, 2 \dots d\}$
  end for
end procedure

Algorithm 9 CUCB [**107**].

procedure CUCB($\mathcal{X}$)
  $\hat{\theta}_i(1) = 0$ for $i \in \{1, 2 \dots d\}$
  while there is a subarm that has not yet been pulled **do**
    $\mathbf{x}(n)$ is an arm with at least one subarm that has not yet been pulled
    $r(n)\,, \mathbf{y}(n) =$ total reward and subarm rewards when playing $\mathbf{x}(n)$
    $\hat{\theta}_i(n) = \frac{1}{T_i(n)} \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1} r(n)$
  end while
  for n **to** T **do**
    $T_{\mathbf{x}}(n) = \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1}$
    $w_i(n) = \hat{\theta}_i + \sqrt{1.5\, T_{\mathbf{x}}^{-1}(n)\, \log n}$ for $i \in \{1, 2 \dots d\}$
    $\mathbf{x}(n) \in \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbf{x}^T\, \mathbf{w}(n)$
    $r(n)\,, \mathbf{y}(n) =$ total reward and subarm rewards when playing $\mathbf{x}(n)$
    $\hat{\theta}_i(n) = \frac{1}{T_i(n)} \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1} r(n)$
  end for
end procedure

Algorithm 10 ESCB [**109**].

procedure ESCB($\mathcal{X}$)
  $\hat{\theta}_i(1) = 0$ for $i \in \{1, 2 \dots d\}$
  while there is a subarm that has not yet been pulled **do**
    $\mathbf{x}(n)$ is an arm with at least one subarm that has not yet been pulled
    $r(n)\,, \mathbf{y}(n) =$ total reward and subarm rewards when playing $\mathbf{x}(n)$
    $\hat{\theta}_i(n) = \frac{1}{T_i(n)} \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1} r(n)$
  end while
  for $n = 1$ **to** T **do**
    $f(n) = \log n + 4\, m\, \log \log n$
    $\mathbf{x}(n) \in \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbf{x}^T\, \hat{\boldsymbol{\theta}}(n) + \sqrt{\frac{f(n)}{2} \sum_{i=1}^{d} \frac{x_i}{\mathbf{T}_i(n)}}$
    $r(n)\,, \mathbf{y}(n) =$ total reward and subarm rewards when playing $\mathbf{x}(n)$
    $\hat{\theta}_i(n) = \frac{1}{T_i(n)} \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1} r(n)$
  end for
end procedure

Algorithm 11 OLS-UCB [**110**].

procedure OLS-UCB($\mathcal{X}$, $\mathbf{\Gamma} \in S_+^d$, $\lambda > 0$)
  $\hat{\theta}_i(1) = 0$ for $i \in \{1, 2 \dots d\}$
  while there is a subarm that has not yet been pulled **do**
    $\mathbf{x}(n)$ is an arm with at least one subarm that has not yet been pulled
    $r(n)\,, \mathbf{y}(n) =$ total reward and subarm rewards when playing $\mathbf{x}(n)$
    $\hat{\theta}_i(n) = \frac{1}{T_i(n)} \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1} r(n)$
  end while
  for $n = 1$ **to** T **do**
    $f(n) = \log n + (m+2) \log\log n + m/2 \log\left(1 + \frac{e}{\lambda}\right)$
    $\mathbf{E}(n) = \hat{\mathbf{D}}^{-1}(n) \left[\lambda \mathbf{\Gamma}\, \hat{\mathbf{D}}(n) + \sum_{t=1}^{n-1} \mathbf{x}^T(t)\ \mathbf{\Gamma}\, \mathbf{x}(t)\right] \hat{\mathbf{D}}^{-1}(n)$
    $\mathbf{x}(n) \in \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbf{x}^T\, \hat{\boldsymbol{\theta}}(n) + \sqrt{2\, f(n)\ \mathbf{x}^T\, \mathbf{E}(n)\ \mathbf{x}}$
    $r(n)\,, \mathbf{y}(n) =$ total reward and subarm rewards when playing $\mathbf{x}(n)$
    $\hat{\theta}_i(n) = \frac{1}{T_i(n)} \sum_{t=1}^{n} \mathbb{1}_{x_i(n)=1} r(n)$
  end for
end procedure

Algorithm 12 Estimation of $\Delta_{\min}$.

procedure ESTIMATE-$\Delta_{\min}$($\mathcal{X}$, $\hat{\boldsymbol{\theta}}(n)$)
  $\mathbf{x}^\star(n) \in \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbf{x}^T\, \hat{\boldsymbol{\theta}}(n)$
  $\hat{r}^\star(n) = \mathbf{x}^{\star T}(n)\ \hat{\boldsymbol{\theta}}(n)$
  for $i = 1$ **to** d **do**
    $\mathbf{x}_i^\star\,(n) \in \arg\max_{\substack{\mathbf{x} \in \mathcal{X}, \\ x_i \neq x_i^\star}} \mathbf{x}^T\, \hat{\boldsymbol{\theta}}(n)$
    $\hat{r}_i^\star(n) = \mathbf{x}_i^{\star T}(n)\ \hat{\boldsymbol{\theta}}(n)$
  end for
  return $\min_{\substack{i \in \{1,2 \dots d\} \\ r^\star(n) \neq \hat{r}_i^\star(n)}} r^\star(n) - \hat{r}_i^\star(n)$
end procedure

Algorithm 13 Budgeted m-set problem.

procedure BUDGETED-m-SET($\mathbf{b} \in \mathbb{R}^d$, $\mathbf{a} \in \mathbb{N}^d$)
  L = empty array of size $(m \max_i a_i + 1, m, d)$
  S = empty array of size $(m \max_i a_i + 1, m, d)$
  $S^\star$ = empty array of size $(m \max_i a_i + 1)$
  for $s = 1$ **to** $m \max_i a_i$ **do**
    for $\ell = 0$ **to** m **do**
      for $i = d$ **down to** 0 **do**
        if $i = d$ **then**
          if $s = 0$ **then**
            $L[s, \ell, i] = 0$
            $S[s, \ell, i] = \emptyset$
          else
            $L[s, \ell, i] = -\infty$
            $S[s, \ell, i] = \nexists$
          end if
        else
          if $\ell = 0$ **then**
            $L[s, \ell, i] = L[s, \ell, i + 1]$
            $S[s, \ell, i] = S[s, \ell, i + 1]$
          else
            $L[s, \ell, i] = \max\{b_i + L[\max(s - a_i, 0), \ell, i + 1], L[s, \ell, i + 1]\}$
            if $L[s, \ell, i] =, L[s, \ell, i + 1]$ **then**
              $S[s, \ell, i] = S[s, \ell, i + 1]$
            else
              $S[s, \ell, i] = S[s, \ell, i + 1] \cup \{i\}$
            end if
          end if
        end if
      end for
    end for
    $S^\star[s] = S[s, m, 0]$
  end for
  return $S^\star$
end procedure

Algorithm 14 Budgeted simple-path problem.

procedure BUDGETED-u-v-PATH($G = (V, E)$, $u \in V$, $v \in V$, $\mathbf{b} \in \mathbb{R}^d$, $\mathbf{a} \in \mathbb{N}^d$)
  $L =$ empty array of size $(m \max_i a_i + 1, |V|)$
  $S =$ empty array of size $(m \max_i a_i + 1, |V|)$
  $S^\star =$ empty array of size $(d \max_i a_i + 1)$
  for $s = 1$ **to** $d \max_i a_i$ **do**
    for $w \in V$ **do**
      if $s = 0$ **then**
        $L[s, w], S[s, w] = \texttt{Dijkstra}(G, b, u, w)$
      else
        $x^\star \in \arg\max_{x:(w,x)\in E}\{b_{(w,x)} + L[x, \max(s - a_{(w,x)}, 0)]\}$
        $L[s, w] = b_{(w,x^\star)} + L[x^\star, \max(s - a_{(w,x^\star)}, 0)]$
        $S[s, w] = (w, x) \cup S[x^\star, \max(s - a_{(w,x^\star)}, 0)]$
      end if
    end for
    $S^\star[s] = S[s, t]$
  end for
  return $S^\star$
end procedure

Algorithm 15 Budgeted spanning-tree problem.

procedure BUDGETED-SPANNING-TREE($G = (V, E)$, $\mathbf{b} \in \mathbb{R}^d$, $\mathbf{a} \in \mathbb{N}^d$, $s \in \mathbb{N}$)
  $x = \emptyset$
  for all unordered pairs (e_1, e_2) of distinct edges of E **do**
    $E' = \{e \in E \mid b_e \leq \min\{b_{e_1}, b_{e_2}\}\}$
    $G' = (V, E')$
    $x^\star, \lambda^\star = \texttt{Meggido}(\texttt{Greedy}, G', \mathbf{a} + \lambda\mathbf{b})$
    $\epsilon =$ arbitrary small value
    $\mathbf{x}^+ = \texttt{Greedy}(G', \mathbf{a} + (\lambda^\star + \epsilon)\,\mathbf{b})$
    $\mathbf{x}^- = \texttt{Greedy}(G', \mathbf{a} + (\lambda^\star - \epsilon)\,\mathbf{b})$
    while $|\mathbf{x}^+ \oplus \mathbf{x}^-| > 1$ **do**
      Find e, e' such that $x^+_e = x^-_{e'} = 1$ and $x^+_{e'} = x^-_e = 0$
      $\mathbf{x} = \mathbf{x}^+ \setminus \{e\} \cup \{e'\}$
      if $\mathbf{a}^T \tilde{\mathbf{x}} \geq s$ **then**
        $\mathbf{x}^+ = \tilde{\mathbf{x}}$
      else
        $\mathbf{x}^- = \tilde{\mathbf{x}}$
      end if
    end while
    if $x = \emptyset$ **or** $\mathbf{b}^T\tilde{\mathbf{x}} > \mathbf{b}^T\mathbf{x}$ **then**
      $x \leftarrow x^+$
    end if
  end for
  return x
end procedure

Algorithm 16 Budgeted bipartite-matching problem.

procedure BUDGETED-BIPARTITE-MATCHING($G = (V, E)$, $\mathbf{b} \in \mathbb{R}^d$, $\mathbf{a} \in \mathbb{N}^d$, $s \in \mathbb{N}$)
  $x = \emptyset$
  for all unordered 4-tuples (e_1, e_2, e_3, e_4) of distinct edges of E **do**
    $E' = \{e \in E \mid b_e \leq \min\{b_{e_1}, b_{e_2}, b_{e_3}, b_{e_4}\}\}$
    $G' = (V, E')$
    $x^\star, \lambda^\star = \texttt{Meggido}(\texttt{Hungarian}, G', \mathbf{a} + \lambda\mathbf{b})$
    $\epsilon =$ arbitrary small value
    $\mathbf{x}^+ = \texttt{Hungarian}(G', \mathbf{a} + (\lambda^\star + \epsilon)\,\mathbf{b})$
    $\mathbf{x}^- = \texttt{Hungarian}(G', \mathbf{a} + (\lambda^\star - \epsilon)\,\mathbf{b})$
    while $|\mathbf{x}^+ \oplus \mathbf{x}^-| > 2$ **do**
      $\mathbf{x}' = \mathbf{x}^+ \oplus \mathbf{x}^-$
      $\mathbf{x}'' =$ one path or one cycle from $\mathbf{x}'$
      $\tilde{\mathbf{x}} = \mathbf{x}^- \oplus \mathbf{x}''$
      if $\mathbf{a}^T\,\tilde{\mathbf{x}} \geq s$ **then**
        $\mathbf{x}^+ = \tilde{\mathbf{x}}$
      else
        $\mathbf{x}^- = \tilde{\mathbf{x}}$
      end if
    end while
    if $\mathbf{x} = \emptyset$ **or** $\mathbf{b}^T\mathbf{x}^+ > \mathbf{b}^T\mathbf{x}$ **then**
      $\mathbf{x} \leftarrow \mathbf{x}^+$
    end if
  end for
  return x
end procedure

Algorithm 17 Subgradient method applied on the reformulation of the Graves-Lai problem.

procedure GLPG($\boldsymbol{\theta} \in \mathbb{N}^d$, $\delta > 0$, $0 < \varepsilon \leq 1$, $\mathbf{A}$ and $\mathbf{b}$ that represent the convex hull of $\mathcal{X}$)
  Compute $\mathbf{M}$ such that $\mathbf{M}\,\mathbf{w} = \mathbf{A}\,\mathbf{w} - \frac{\mathbf{b^T}\,\mathbf{A}\,\mathbf{w}}{\|\mathbf{b}\|_2^2}\,\mathbf{b}$
  $\mathbf{q} = \left(\boldsymbol{\theta}^T\,\mathbf{x}^\star\right)\,\frac{\mathbf{b}^T\,\mathbf{A}}{\|\mathbf{b}\|^2} - \boldsymbol{\theta}$
  $w_{\min} = \frac{1}{m^2\,\|\boldsymbol{\theta}\|_\infty^2}$
  $\delta_2 = \frac{\delta\,\varepsilon}{m^2\,d\,\|\boldsymbol{\theta}\|_\infty}$
  $\delta_1 = \frac{\delta}{2+2\,\delta_2}$
  $\lambda = \frac{\delta_1 + m^2\,d\,\|\boldsymbol{\theta}\|_\infty}{\delta_2}$
  $q_1 = \|\mathbf{q}\|_2^2 + \frac{\lambda^2\,d\,m^8\,\|\boldsymbol{\theta}\|_\infty^8}{\varepsilon^2}$
  $M = \left\lceil q_1\,\frac{m^5\,d^2\,\|\boldsymbol{\theta}\|_\infty^2}{\delta_1^2\,\varepsilon^2} \right\rceil$
  $\eta = \sqrt{\frac{m^5\,d^2\,\|\boldsymbol{\theta}\|_\infty^2}{\varepsilon^2\,T\,q_1}}$ $\mathbf{w}^{(0)} = (w_{\min}, w_{\min} \dots w_{\min})$
  for $m \in \{0, 1 \dots M\}$ **do**
    Find $\mathbf{x}^{(m)}$ such that $\max_{\mathbf{x}\in\mathcal{X}}\, h_{\mathbf{x}}\left(\mathbf{w}^{(m)}\right) \leq h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right)$
    $\mathbf{g}^{(m)} = \mathbf{q} + \left[\lambda\,\varepsilon\,\nabla h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right)\right]\,\mathbb{1}_{h_{\mathbf{x}^{(m)}}\left(\varepsilon\,\mathbf{w}^{(m)}\right)>0}$
    $\mathbf{w}^{(m+1)} = \Pi_{\mathcal{M}}\left\{\mathbf{w}^{(m)} + \eta\,\mathbf{g}^{(m)}\right\}$
  end for
  $\overline{\mathbf{w}} = \frac{1}{M}\,\sum_{m=1}^{M}\,\mathbf{w}^{(m)}$
  $\overline{\mathbf{w}}' = (1+\delta_2)\,\overline{\mathbf{w}}$
  return Algorithm **??** applied on $\overline{\mathbf{w}}'$ and the polytope described by $\mathbf{A}$ and $\mathbf{b}$
end procedure

Algorithm 18 Convex combination of vertices [**164**].

procedure GLPG(A polytope $X = \left\{\mathbf{x} \in \mathbb{R}^d \,|\, \mathbf{A}\,\mathbf{x} \leq \mathbf{b}\right\}$ with $\mathbf{A} \in \mathbb{R}^{c \times d}$ and $\mathbf{b} \in \mathbb{R}^c$, $\mathbf{x} \in X$)
  $K = 1$
  $\mathbf{x}_1 = \mathbf{x}$
  $\overline{\delta}_1 = 1$
  Find the r_1 tight inequalities at $\mathbf{x}_1$, the corresponding submatrix $\mathbf{G}_1 \in \mathbb{R}^{r_1 \times d}$ of $\mathbf{A}$ and the corresponding vector $\mathbf{g}_1 \in \mathbb{R}^{r_1}$ of $\mathbf{b}$ such that $\mathbf{G}_1\,\mathbf{x}_1 = \mathbf{g}_1$
  Initialise the vector of scalars $\boldsymbol{\delta}$
  Initialise the vector of vertices $\mathbf{p}$
  while $\mathbf{G}_K$ does not have rank d **do**
    $i = 1$
    $\mathbf{G}_{K,0} = \mathbf{G}_K$
    $\mathbf{g}_{K,0} = \mathbf{g}_K$
    while $\mathbf{G}_{K,i}$ does not have rank d **do**
      Find any $\mathbf{d}_{K,i} \in \mathbb{R}^d$ such that $\mathbf{G}_{K,i}\,\mathbf{d}_{K,i} = 0$ and $\mathbf{d}_{K,i} \neq \mathbf{0}$
      $\mathbf{y}_{K,i} = \mathbf{x}_K - \mathbf{d}_{K,i}$
      $\mathbf{z}_{K,i} = \mathbf{x}_K + \mathbf{d}_{K,i}$
      $\gamma_{K,i,1} = \begin{array}{ll} \max & \alpha \\ \text{s.t.} & \mathbf{y}_{K,i} + \alpha\,(\mathbf{y}_{K,i} - \mathbf{z}_{K,i}) \in X \end{array}$
      $\overline{\mathbf{y}}_{\mathbf{K,i}} = \mathbf{y}_{K,i} + \gamma_{K,i,1}\,(\mathbf{y}_{K,i} - \mathbf{z}_{K,i})$
      Add the newly tight constraints at $\overline{\mathbf{y}}_{K,i}$ to form $\mathbf{G}_{K,i}$ and $\mathbf{g}_{K,i}$ from $\mathbf{G}_{K,i-1}$ and $\mathbf{g}_{K,i-1}$
      $\mathbf{y}_{K,i+1} = \overline{\mathbf{y}}_{K,i}$
      $i = i + 1$
    end while
    $\gamma_{K,2} = \begin{array}{ll} \max & \alpha \\ \text{s.t.} & \mathbf{x}_K + \alpha\,(\mathbf{x}_K - \overline{\mathbf{y}}_K) \in X \end{array}$
    $\delta_K = \frac{\gamma_{K,2}}{1+\gamma_{K,2}}\,\overline{\delta}_K$
    $\mathbf{p}_K = \overline{\mathbf{y}}_K$
    $\overline{\delta}_{K+1} = \left(1 - \frac{\gamma_{K,2}}{1+\gamma_{K,2}}\right)\,\overline{\delta}_K$
    $\mathbf{x}_{K+1} = \mathbf{z}_K$
    $K = K + 1$
    Find the r_K tight inequalities at $\mathbf{x}_K$, the corresponding submatrix $\mathbf{G}_K \in \mathbb{R}^{r_K \times d}$ of $\mathbf{A}$ and the corresponding vector $\mathbf{g}_K \in \mathbb{R}^{r_K}$ of $\mathbf{b}$ such that $\mathbf{G}_K\,\mathbf{x}_K = \mathbf{g}_K$
  end while
  $\delta_K = \overline{\delta}_K$
  $\mathbf{p}_k = \mathbf{x}_K$
  return $\boldsymbol{\delta}$ and $\mathbf{p}$
end procedure

Chapitre C

Résumé de la thèse

La prise de décision séquentielle est une composante essentielle de nombreuses applications, de la gestion des réseaux informatiques aux annonces en ligne. L'outil principal est l'apprentissage par renforcement : un agent prend une séquence de décisions afin d'atteindre son objectif, avec des mesures typiquement bruitées de son environnement. Par exemple, un agent peut contrôler une voiture autonome ; l'environnement est la ville dans laquelle la voiture se déplace. Les problèmes de bandits forment une classe d'apprentissage de renforcement pour laquelle on peut démontrer de très forts résultats théoriques. Les algorithmes de bandits se concentrent sur le dilemme exploration-exploitation : pour avoir une bonne performance, l'agent doit avoir une connaissance approfondie de son environnement (exploration) ; cependant, il doit aussi jouer des actions qui le rapprochent de son but (exploitation).

C.1. Algorithmes de bandit combinatoires

Dans cette thèse, nous nous concentrons sur les bandits combinatoires, qui sont des bandits dont les décisions sont très structurées (une structure « combinatoire »). Il s'agit notamment des cas où l'agent détermine un chemin à suivre (sur une route, dans un réseau informatique, etc.) ou des publicités à afficher sur un site Web. De telles situations partagent leur complexité algorithmique : alors qu'il est souvent facile de déterminer la décision optimale lorsque les paramètres sont connus (comme le temps pour traverser une route ou le profit généré par l'affichage d'une publicité à un endroit donné), la variante bandit (lorsque les paramètres doivent être déterminés par des interactions avec l'environnement) est bien plus complexe.

Les algorithmes de bandit qui exploitent cette structure combinatoire se regroupent en deux catégories principales (en ne considérant que ceux connus avant cette thèse) :

- ceux qui résolvent un seul problème d'optimisation combinatoire linéaire, ce qui leur permet notamment d'atteindre une complexité algorithmique polynomiale pour une série de problèmes d'intérêt. Par exemple, l'échantillonnage de Thompson [**48, 105**] ou CUCB [**107, 108**] ;
- ceux qui résolvent un problème d'optimisation combinatoire non linéaire, ce qui leur permet d'atteindre un bien meilleur regret, au prix d'une complexité algorithmique qui n'est plus polynomiale, sauf dans de très rares cas particuliers peu intéressants en pratique. Par exemple, ESCB [**109**], OLS-UCB [**110**] ou OSSB [**80, 94**].

C.2. Contributions

Nous proposons deux nouveaux algorithmes pour aborder ces problèmes de bandit combinatoire par des techniques d'optimisation mathématique. Basés sur des hypothèses faibles (l'existence d'un algorithme efficace d'optimisation linéaire budgétée), ils présentent une complexité temporelle

polynomiale, tout en étant performants par rapport aux algorithmes de pointe pour les mêmes problèmes. Ils présentent également d'excellentes propriétés statistiques, ce qui signifie qu'ils trouvent un équilibre entre exploration et exploitation proche de l'optimum théorique. Les travaux précédents sur les bandits combinatoires ont dû faire un choix entre le temps de calcul et la performance statistique ; nos algorithmes montrent que ce dilemme n'a pas lieu d'être.

C.2.1. Décomposition d'un problème d'optimisation combinatoire non linéaire. Plus précisément, ces deux algorithmes se basent sur une décomposition particulière du problème d'optimisation combinatoire non linéaire d'origine. Par exemple, ESCB nécessite la solution optimale au problème suivant :

$$\begin{array}{ll} \max & \hat{\boldsymbol{\theta}}^T \mathbf{x} + \sqrt{\boldsymbol{\sigma}^T \mathbf{x}} \\ \text{t.q.} & \mathbf{x} \in \mathcal{X}, \end{array} \tag{C.2.1}$$

où $\mathcal{X} \subset \{0,1\}^d$ est l'ensemble combinatoire choisi et $\hat{\boldsymbol{\theta}}$ et $\boldsymbol{\sigma}$ sont des paramètres maintenus par ESCB (2.5.14). De manière générale, ce problème d'optimisation appartient à la classe de complexité $\mathcal{NPH}$ (Section 3.1.1), ce qui implique qu'il est très peu probable qu'il existe un algorithme exact pour trouver la solution optimale en un temps polynomial en la dimension de $\mathcal{X}$, d. Par conséquent, nous ne tentons pas de trouver une solution exacte à ce problème, mais bien une solution approchée. Notre technique de décomposition déplace le terme non linéaire de l'objectif en une contrainte de budget minimum s :

$$\begin{array}{ll} \max & \hat{\boldsymbol{\theta}}^T \mathbf{x} \\ \text{t.q.} & \mathbf{x} \in \mathcal{X} \\ & \boldsymbol{\sigma}^T \mathbf{x} \geq s. \end{array} \tag{C.2.2}$$

En résolvant ce nouveau problème pour une série de valeurs de s, on peut s'assurer de retrouver une solution assez proche de l'optimum de la formulation non linéaire d'origine. Pour que cette approche fonctionne, deux hypothèses principales doivent être vérifiées :

- résolution d'un sous-problème budgété en temps polynomial : pour ce faire, nous écrivons un algorithme spécifique à chaque ensemble combinatoire d'intérêt. Ils sont présentés à la Section 3.2. Certains de ces algorithmes sont exacts (ils donnent une solution dans l'ensemble combinatoire qui respecte la contrainte de budget minimum avec la valeur optimale de la fonction objectif), d'autres sont approchés avec un facteur d'approximation (ils donnent une solution dans l'ensemble combinatoire qui respecte la contrainte de budget minimum avec une valeur de la fonction objectif qui est éloignée de la valeur optimale d'un facteur donné) ;
- limitation du nombre de valeurs de budget minimum s à tester : avec une procédure de mise à l'échelle et d'arrondi, utilisant le même paramètre de discrétisation, nous pouvons nous assurer que l'ensemble des valeurs possibles de $\boldsymbol{\sigma}^T \mathbf{x}$ est discret (uniquement des nombres entiers). La qualité de l'approximation dépend du paramètre de discrétisation de manière additive.

C.2.2. Nouveaux algorithmes pour les bandits combinatoires. AESCB (Chapitre 3) est une implémentation approchée d'ESCB qui utilise cette décomposition. En choisissant de manière adéquate le paramètre de discrétisation (Sections 3.1.6 et 3.4), on peut s'assurer, de manière théorique, que la discrétisation effectuée n'aura aucun impact sur le regret de l'algorithme ; des choix moins conservatifs améliorent cependant les temps d'exécution.

GLPG (Chapitre 4) est une technique de calcul pour la borne de Graves-Lai (Section 2.4.2), qui caractérise le regret asymptotique minimum pour une grande classe de problèmes d'apprentissage par renforcement, notamment les bandits combinatoires. Cependant, cette borne (2.4.7) s'écrit avec une variable et une contrainte convexe par solution combinatoire de l'ensemble $\mathcal{X}$, c'est-à-dire que le programme d'optimisation a $\mathcal{O}(|\mathcal{X}|) \subset \mathcal{O}\big(2^d\big)$ variables et $\mathcal{O}(|\mathcal{X}|) \subset \mathcal{O}\big(2^d\big)$ contraintes.

$$\begin{array}{lll} \min & \sum_{\mathbf{x}\in\mathcal{X}} \eta_{\mathbf{x}}\, \Delta_{\mathbf{x}} & \\ \text{s.t.} & \sum_{i\in\mathcal{I}} \frac{x_i}{\sum_{\mathbf{y}\in\mathcal{X}} y_i\, \alpha_{\mathbf{y}}} \leq \Delta_{\mathbf{x}}^2 & \forall \mathbf{x} \in \mathcal{X} \\ & \eta_{\mathbf{x}} \geq 0 & \forall \mathbf{x} \in \mathcal{X} \end{array} \tag{C.2.3}$$

$\eta_{\mathbf{x}}$ indique la fraction de temps que la solution $\mathbf{x}$ devrait être jouée pour atteindre l'optimum asymptotique de regret ; $\Delta_{\mathbf{x}}$ est le regret de cette solution. Nous proposons une reformulation non dérivable (4.3.29) :

$$\begin{array}{lll} \min & \mathbf{q}^T\, \mathbf{w} + \lambda\, \max_{\mathbf{x}\in\mathcal{X}} \left\{ \left[\sum_{i\in\mathcal{I}} \frac{x_i}{w_i} - \Delta_{\mathbf{x}}^2 \right]^+ \right\} & \\ \text{s.t.} & \mathbf{M}\, \mathbf{w} = \mathbf{0} & \\ & w_i \geq w_{\min} & \forall i \in \mathcal{I} \\ & w_i \geq 0 & \forall i \in \{1, 2 \dots d\}\,. \end{array} \tag{C.2.4}$$

La variable w_i indique à quelle fréquence le bras i doit être joué, ce qui permet de n'avoir que d variables. $\lambda > 0$ est un paramètre de pénalisation des contraintes. Cette nouvelle formulation peut être résolue à l'aide d'une technique de projection de sous-gradient. L'évaluation de la fonction objectif et le calcul d'un sous-gradient peuvent se faire de la même manière que pour le calcul d'une solution pour AESCB. Lorsque l'algorithme d'optimisation budgétée n'est pas exact, les preuves de convergence habituelles de la méthode ne s'appliquent plus ; cependant, elles peuvent être étendues dans ce cas (Section 4.3.5). Cette reformulation ne permet pas d'obtenir directement une solution dans les variables d'origine ; cependant, la solution en $\mathbf{w}$ peut être réécrite comme une combinaison convexe de points extrêmes du polytope décrivant l'enveloppe convexe de $\mathcal{X}$, c'est-à-dire en $\boldsymbol{\alpha}$, en temps polynomial (Section 4.3.6).

GLPG peut ainsi être utilisé dans le cadre d'OSSB afin de fournir un algorithme de bandit combinatoire asymptotiquement optimal, pour autant que la constante de discrétisation soit suffisamment faible.

C.2.3. Résultats numériques et reproductibilité. Les résultats numériques montrent qu'AESCB donne un regret très proche de celui d'ESCB ; de plus, les temps de calcul sont compétitifs par rapport à l'utilisation de solveurs d'optimisation mathématique de pointe (Section 3.4). Pour GLPG, la valeur de la borne de Graves-Lai est extrêmement proche de la valeur réelle, même avec un algorithme d'optimisation budgétée approximé (Section 4.5).

Les algorithmes ont été implémentés en Julia [**37**], y compris ceux exploitant la programmation mathématique [**172, 149**]. Le code correspondant est disponible en ligne :

- Kombinator.jl contient les implémentations des algorithmes d'optimisation combinatoire, budgétée ou non ;
- NonsmoothOptim.jl se focalise sur l'optimisation non dérivable ;
- CombinatorialBandits.jl fournit une abstraction pour les bandits combinatoires, ainsi que diverses politiques d'exploration comme l'échantillonnage de Thompson, CUCB, ESCB ou AESCB.

Bibliography

[1] D. Bertsekas and R. Gallager, *Data Network*, 1st ed. Englewood Cliffs, NJ: Prentice Hall, Englewood Cliffs, NJ, 1987.

[2] J. M. Kleinberg, "Single-source unsplittable flow," in *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 1996, pp. 68–77.

[3] L. Gouveia, "Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints," *Computers & Operations Research*, vol. 22, no. 9, pp. 959–970, 1995.

[4] W. Ben-Ameur and H. Kerivin, "Routing of uncertain traffic demands," *Optimization and Engineering*, vol. 6, no. 3, pp. 283–313, 2005.

[5] H. Räcke, "Survey on oblivious routing strategies," in *Conference on Computability in Europe*. Springer, 2009, pp. 419–429.

[6] J. Boyan and M. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," *Advances in neural information processing systems*, vol. 6, pp. 671–678, 1993.

[7] N. Tao, J. Baxter, and L. Weaver, "A multi-agent, policy-gradient approach to network routing," in *In: Proc. of the 18th Int. Conf. on Machine Learning*. Citeseer, 2001.

[8] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[9] S. Wassermann, M. Seufert, P. Casas, L. Gang, and K. Li, "ViCrypt to the Rescue: Real-time, Machine-Learning-driven Video-QoE Monitoring for Encrypted Streaming Traffic," *IEEE Transactions on Network and Service Management*, 2020.

[10] A. Rashelbach, O. Rottenstreich, and M. Silberstein, "A Computational Approach to Packet Classification," in *SIGCOMM '20: Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, New York, NY, 2020.

[11] A. D. Lopez, A. P. Mohan, and S. Nair, "Network Traffic Behavioral Analytics for Detection of DDoS Attacks," *SMU Data Science Review*, vol. 2, no. 1, p. 14, 2019.

[12] K. Li, H. Zheng, and J. Wu, "Migration-based virtual machine placement in cloud systems," in *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*. IEEE, 2013, pp. 83–90.

[13] D. Castro-Silva and E. Gourdin, "A study on load-balanced variants of the bin packing problem," *Discrete Applied Mathematics*, vol. 264, pp. 4–14, 2019.

[14] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[15] P. Amani, S. Bastani, and B. Landfeldt, "Towards optimal content replication and request routing in content delivery networks," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 5733–5739.

[16] S. O. Somuyiwa, A. György, and D. Gündüz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1331–1344, 2018.

[17] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *International Conference on Machine Learning*, 2019, pp. 3050–3059.

[18] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: a pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 632–647.

[19] M. Masson, Z. Altman, and E. Altman, "Multi-User collaborative scheduling in 5G massive MIMO heterogeneous networks," in *IFIP*, 2020.

[20] Y. Li, Y. Wen, D. Tao, and K. Guan, "Transforming cooling optimization for green data center via deep reinforcement learning," *IEEE transactions on cybernetics*, vol. 50, no. 5, pp. 2002–2013, 2019.

[21] Y. Carlinet and N. Perrot, "Energy-efficient load balancing in a SDN-based Data-Center network," in *2016 17th international telecommunications network strategy and planning symposium (Networks)*. IEEE, 2016, pp. 138–143.

[22] K. Pilarska, B. Liau, and N. Perrot, "Estimates of the economic impact of energy savings in the E2E chain for Video On Demand service," in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. IEEE, 2016, pp. 75–80.

[23] D. Sanvito, I. Filippini, A. Capone, S. Paris, and J. Leguay, "Clustered robust routing for traffic engineering in software-defined networks," *Computer Communications*, vol. 144, pp. 175–187, 2019.

[24] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé, "Semi-oblivious traffic engineering: The road not taken," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 157–170.

[25] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[26] M. Brehob, S. Wagner, E. Torng, and R. Enbody, "Optimal replacement is NP-hard for nonstandard caches," *IEEE Transactions on computers*, vol. 53, no. 1, pp. 73–76, 2004.

[27] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *arXiv preprint arXiv:2003.03600*, 2020.

[28] Y. Bengio, A. Lodi, and A. Prouvost, "Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon," *arXiv preprint arXiv:1811.06128*, 2018.

[29] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[30] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 6348–6358.

[31] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *arXiv preprint arXiv:1803.08475*, 2018.

[32] A. Mittal, A. Dhawan, S. Manchanda, S. Medya, S. Ranu, and A. Singh, "Learning heuristics over large graphs via deep reinforcement learning," *arXiv preprint arXiv:1903.03332*, 2019.

[33] Y. Bengio, E. Frejinger, A. Lodi, R. Patel, and S. Sankaranarayanan, "A learning-based algorithm to quickly compute good primal solutions for Stochastic Integer Programs," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2020, pp. 99–111.

[34] N. Dupin and E.-G. Talbi, "Machine Learning-Guided Dual Heuristics and New Lower Bounds for the Refueling and Maintenance Planning Problem of Nuclear Power Plants," *Algorithms*, vol. 13, no. 8, p. 185, 2020.

[35] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. Cire, "Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization," *arXiv preprint arXiv:2006.01610*, 2020.

[36] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, and A. Lodi, "Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers," in *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. [Online]. Available: https://openreview.net/forum?id=IVc9hqgibyB

[37] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A Fresh Approach to Numerical Computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, nov 2017.

[38] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[39] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society*, vol. 58, no. 5, pp. 527–535, 1952.

[40] C. Jin, Z. Yang, Z. Wang, and M. I. Jordan, "Provably efficient reinforcement learning with linear function approximation," in *Proceedings of Thirty Third Conference on Learning Theory*, ser. Proceedings of Machine Learning Research, J. Abernethy and S. Agarwal, Eds., vol. 125. PMLR, 2020, pp. 2137–2143. [Online]. Available: http://proceedings.mlr.press/v125/jin20a.html

[41] O. Delalleau, "Deep Reinforcement Learning in Action: For Honor & Watch_Dogs 2 Case Studies," in *1st International Summer School on Artificial Intelligence and Games*, Chania, Greece, 2018. [Online]. Available: https://school.gameaibook.org/2018-school/

[42] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and Others, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[43] S. Padakandla, S. Bhatnagar, and Others, "Reinforcement learning in non-stationary environments," *arXiv preprint arXiv:1905.03970*, 2019.

[44] A. Krishnamurthy, J. Langford, A. Slivkins, and C. Zhang, "Contextual Bandits with Continuous Actions: Smoothing, Zooming, and Adapting," *Journal of Machine Learning Research*, vol. 21, no. 137, pp. 1–45, 2020.

[45] M. Majzoubi, C. Zhang, R. Chari, A. Krishnamurthy, J. Langford, and A. Slivkins, "Efficient Contextual Bandits with Continuous Actions," *arXiv preprint arXiv:2006.06040*, 2020.

[46] B. Li, T. Chen, and G. B. Giannakis, "Bandit online learning with unknown delays," in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 993–1002.

[47] T. Lattimore and C. Szepesvári, *Bandit Algorithms*. Cambridge University Press, jul 2020. [Online]. Available: https://www.cambridge.org/core/product/identifier/9781108571401/type/book

[48] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[49] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[50] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE, 1995, pp. 322–331.

[51] J. White, *Bandit algorithms for website optimization*. O'Reilly Media, Inc., 2012.

[52] S. Kullback and R. A. Leibler, "10.1214/aoms/1177729694," *Ann. Math. Stat*, vol. 22, pp. 79–86, 1951.

[53] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM journal on computing*, vol. 32, no. 1, pp. 48–77, 2002.

[54] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[55] A. Garivier and O. Cappé, "The KL-UCB algorithm for bounded stochastic bandits and beyond," in *Proceedings of the 24th annual Conference On Learning Theory*, 2011, pp. 359–376.

[56] O. Cappé, A. Garivier, O.-A. Maillard, R. Munos, G. Stoltz, and Others, "Kullback–leibler upper confidence bounds for optimal sequential allocation," *The Annals of Statistics*, vol. 41, no. 3, pp. 1516–1541, 2013.

[57] O.-A. Maillard, R. Munos, and G. Stoltz, "A finite-time analysis of multi-armed bandits problems with kullback-leibler divergences," in *Proceedings of the 24th annual Conference On Learning Theory*, 2011, pp. 497–514.

[58] T. Bayes, "An essay towards solving a problem in the doctrine of chances," *Philosophical transactions of the Royal Society of London*, no. 53, pp. 370–418, 1763.

[59] E. Kaufmann, N. Korda, and R. Munos, "Thompson sampling: An asymptotically optimal finite-time analysis," in *International conference on algorithmic learning theory*. Springer, 2012, pp. 199–213.

[60] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich, "Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine." Omnipress, 2010.

[61] O.-C. Granmo, "Solving two-armed Bernoulli bandit problems using a Bayesian learning automaton," *International Journal of Intelligent Computing and Cybernetics*, 2010.

[62] P. A. Ortega and D. A. Braun, "A minimum relative entropy principle for learning and acting," *Journal of Artificial Intelligence Research*, vol. 38, pp. 475–511, 2010.

[63] V. G. Vovk, "Aggregating strategies," *Proc. of Computational Learning Theory, 1990*, 1990.

[64] G. Stoltz, "Incomplete information and internal regret in prediction of individual sequences," Ph.D. dissertation, Université Paris Sud-Paris XI, 2005.

[65] A. Tewari and S. A. Murphy, "From ads to interventions: Contextual bandits in mobile health," in *Mobile Health*. Springer, 2017, pp. 495–517.

[66] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 661–670.

[67] X. Wang, Y. Wang, D. Hsu, and Y. Wang, "Exploration in interactive personalized music recommendation: a reinforcement learning approach," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 1, pp. 1–22, 2014.

[68] L. Tang, R. Rosales, A. Singh, and D. Agarwal, "Automatic ad format selection via contextual bandits," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 1587–1594.

[69] A. S. Lan and R. G. Baraniuk, "A Contextual Bandits Framework for Personalized Learning Action Selection." in *EDM*, 2016, pp. 424–429.

[70] S. Wassermann, T. Cuvelier, and P. Casas, "RAL: Improving Stream-Based Active Learning by Reinforcement Learning," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD) Workshop on Interactive Adaptive Learning (IAL)*, Würzburg, Germany, 2019.

[71] T. Lu, D. Pál, and M. Pál, "Showing relevant ads via context multi-armed bandits," in *Proceedings of AISTATS*, 2009.

[72] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[73] S. Filippi, O. Cappe, A. Garivier, and C. Szepesvári, "Parametric bandits: The generalized linear case," in *Advances in Neural Information Processing Systems*, 2010, pp. 586–594.

[74] W. Chu, L. Li, L. Reyzin, and R. Schapire, "Contextual bandits with linear payoff functions," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 208–214.

[75] M. S. Talebi, Z. Zou, R. Combes, A. Proutiere, and M. Johansson, "Stochastic online shortest path routing: The value of feedback," *IEEE Transactions on Automatic Control*, vol. 63, no. 4, pp. 915–930, 2017.

[76] T. Lattimore and C. Szepesvari, "The end of optimism? an asymptotic analysis of finite-armed linear bandits," in *Artificial Intelligence and Statistics*, 2017, pp. 728–737.

[77] T. Lattimore and R. Munos, "Bounded regret for finite-armed structured bandits," in *Advances in Neural Information Processing Systems*, 2014, pp. 550–558.

[78] N. Abe and P. M. Long, "Associative reinforcement learning using linear probabilistic concepts," in *ICML*. Citeseer, 1999, pp. 3–11.

[79] V. Dani, T. P. Hayes, and S. M. Kakade, "Stochastic linear optimization under bandit feedback," 2008.

[80] R. Combes, S. Magureanu, and A. Proutiere, "Minimal exploration in structured stochastic bandits," in *Advances in Neural Information Processing Systems*, 2017, pp. 1763–1771.

[81] N. Cesa-Bianchi and G. Lugosi, "Combinatorial bandits," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1404–1422, 2012.

[82] R. Agrawal, "The continuum-armed bandit problem," *SIAM journal on control and optimization*, vol. 33, no. 6, pp. 1926–1951, 1995.

[83] R. Kleinberg, A. Slivkins, and E. Upfal, "Multi-armed bandits in metric spaces," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008, pp. 681–690.

[84] P. Auer, R. Ortner, and C. Szepesvári, "Improved rates for the stochastic continuum-armed bandit problem," in *International Conference on Computational Learning Theory*. Springer, 2007, pp. 454–468.

[85] S. Magureanu, R. Combes, and A. Proutiere, "Lipschitz bandits: Regret lower bounds and optimal algorithms," in *Proceedings of COLT*, 2014, pp. 521–529.

[86] S. Bubeck, G. Stoltz, and J. Y. Yu, "Lipschitz bandits without the Lipschitz constant," in *International Conference on Algorithmic Learning Theory*. Springer, 2011, pp. 144–158.

[87] A. Agarwal, D. P. Foster, D. J. Hsu, S. M. Kakade, and A. Rakhlin, "Stochastic convex optimization with bandit feedback," in *Advances in Neural Information Processing Systems*, 2011, pp. 1035–1043.

[88] E. W. Cope, "Regret and convergence bounds for a class of continuum-armed bandit problems," *IEEE Transactions on Automatic Control*, vol. 54, no. 6, pp. 1243–1253, 2009.

[89] J. Y. Yu and S. Mannor, "Unimodal bandits," in *ICML*, 2011.

[90] R. Combes and A. Proutiere, "Unimodal bandits: Regret lower bounds and optimal algorithms," in *International Conference on Machine Learning*, 2014, pp. 521–529.

[91] C. Trinh, E. Kaufmann, C. Vernade, and R. Combes, "Solving Bernoulli rank-one bandits with unimodal Thompson sampling," in *Algorithmic Learning Theory*. PMLR, 2020, pp. 862–889.

[92] T. L. Graves and T. L. Lai, "Asymptotically efficient adaptive choice of control laws incontrolled markov chains," *SIAM journal on control and optimization*, vol. 35, no. 3, pp. 715–743, 1997.

[93] B. Laurent and P. Massart, "Adaptive estimation of a quadratic functional by model selection," *Annals of Statistics*, pp. 1302–1338, 2000.

[94] B. P. G. Van Parys and N. Golrezaei, "Optimal Learning for Structured Bandits," *arXiv preprint arXiv:2007.07302*, 2020.

[95] C. H. Papadimitriou and P. CH, "The Euclidean traveling salesman problem is NP-complete." 1977.

[96] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, "Adwords and generalized online matching," *Journal of the ACM (JACM)*, vol. 54, no. 5, pp. 22—-es, 2007.

[97] J. Feldman, A. Mehta, V. Mirrokni, and S. Muthukrishnan, "Online stochastic matching: Beating 1-1/e," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2009, pp. 117–126.

[98] D. Q. Vu, P. Loiseau, and A. Silva, "Combinatorial bandits for sequential learning in colonel blotto games," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 867–872.

[99] K. Winter, "Formalising behaviour trees with CSP," in *International Conference on Integrated Formal Methods*. Springer, 2004, pp. 148–167.

[100] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo Tree Search: A New Framework for Game AI." in *AIIDE*, 2008.

[101] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[102] S. Ontanón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013, pp. 58–64.

[103] A. Shleyfman, A. Komenda, and C. Domshlak, "On combinatorial actions and CMABs with linear side information," in *ECAI*, 2014, pp. 825–830.

[104] S. Ontanón, "Combinatorial multi-armed bandits for real-time strategy games," *Journal of Artificial Intelligence Research*, vol. 58, pp. 665–702, 2017.

[105] D. Russo and B. Van Roy, "An information-theoretic analysis of Thompson sampling," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2442–2471, 2016.

[106] S. Wang and W. Chen, "Thompson sampling for combinatorial semi-bandits," *arXiv preprint arXiv:1803.04623*, 2018.

[107] B. Kveton, Z. Wen, A. Ashkan, and C. Szepesvari, "Tight regret bounds for stochastic combinatorial semi-bandits," in *Artificial Intelligence and Statistics*, 2015, pp. 535–543.

[108] W. Chen, Y. Wang, and Y. Yuan, "Combinatorial multi-armed bandit: General framework and applications," in *International Conference on Machine Learning*, 2013, pp. 151–159.

[109] R. Combes, M. S. T. M. Shahi, A. Proutiere, and Others, "Combinatorial bandits revisited," in *Advances in Neural Information Processing Systems*, 2015, pp. 2116–2124.

[110] R. Degenne and V. Perchet, "Combinatorial semi-bandit with known covariance," in *Advances in Neural Information Processing Systems*, 2016, pp. 2972–2980.

[111] B. Kveton, Z. Wen, A. Ashkan, H. Eydgahi, and B. Eriksson, "Matroid bandits: Fast combinatorial optimization with learning," in *arXiv preprint arXiv:1403.5045*, ser. UAI'14. Arlington, Virginia, United States: AUAI Press, 2014, pp. 420–429. [Online]. Available: http://dl.acm.org/citation.cfm?id=3020751.3020795

[112] P. Perrault, V. Perchet, and M. Valko, "Exploiting structure of uncertainty for efficient matroid semi-bandits," *arXiv preprint arXiv:1902.03794*, 2019.

[113] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[114] Y. Zhou, D. Chakrabarty, and R. Lukose, "Budget constrained bidding in keyword auctions and online knapsack problems," in *International Workshop on Internet and Network Economics*. Springer, 2008, pp. 566–576.

[115] H. Whitney, "On the abstract properties of linear dependence," *American Journal of Mathematics*, vol. 57, no. 3, pp. 509–533, 1935.

[116] J. G. Oxley, *Matroid theory*. Oxford University Press, USA, 2006, vol. 3.

[117] M. P. Bianchi, H.-J. Böckenhauer, T. Brülisauer, D. Komm, and B. Palano, "Online minimum spanning tree with advice," *International Journal of Foundations of Computer Science*, vol. 29, no. 04, pp. 505–527, 2018.

[118] J. Lee and J. Ryan, "Matroid applications and algorithms," *ORSA Journal on Computing*, vol. 4, no. 1, pp. 70–98, 1992.

[119] E. L. Lawler, "Matroid intersection algorithms," *Mathematical programming*, vol. 9, no. 1, pp. 31–56, 1975.

[120] P. M. Camerini and F. Maffioli, "Bounds for 3-matroid intersection problems," *Information Processing Letters*, vol. 3, no. 3, pp. 81–83, 1975.

[121] S. P. Fekete, R. T. Firla, and B. Spille, "Characterizing matchings as the intersection of matroids," *Mathematical Methods of Operations Research*, vol. 58, no. 2, pp. 319–329, 2003.

[122] E. Nikolova, M. Brand, and D. R. Karger, "Optimal Route Planning under Uncertainty," in *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'06. AAAI Press, 2006, pp. 131–140.

[123] D. P. Williamson, *Network Flow Algorithms*. Cambridge University Press, 2019.

[124] Y. Berstein and S. Onn, "Nonlinear bipartite matching," *Discrete Optimization*, vol. 5, no. 1, pp. 53–65, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157252860700062X

[125] A. Atamtürk and A. Gómez, "Maximizing a class of utility functions over the vertices of a polytope," *Operations Research*, vol. 65, no. 2, pp. 433–445, 2017.

[126] L. A. Wolsey, *Integer programming*, R. L. Graham, J. K. Lenstra, and R. E. Tarjan, Eds. John Wiley and Sons, 1998. [Online]. Available: http://eprints.lse.ac.uk/31572/

[127] A. Jüttner, "On budgeted optimization problems," *SIAM Journal on Discrete Mathematics*, vol. 20, no. 4, pp. 880–892, 2006.

[128] R. Ravi and M. X. Goemans, "The constrained minimum spanning tree problem," in *Scandinavian Workshop on Algorithm Theory*. Springer, 1996, pp. 66–75.

[129] A. Berger, V. Bonifaci, F. Grandoni, and G. Schäfer, "Budgeted matching and budgeted matroid intersection via the gasoline puzzle," *Mathematical Programming*, vol. 128, no. 1-2, pp. 355–372, 2011.

[130] C. Chekuri, J. Vondrák, and R. Zenklusen, "Multi-budgeted matchings and matroid intersection via dependent rounding," in *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 2011, pp. 1080–1097.

[131] G. Amanatidis, G. Birmpas, and E. Markakis, "Coverage, matching, and beyond: new results on budgeted mechanism design," in *International Conference on Web and Internet Economics*. Springer, 2016, pp. 414–428.

[132] P. Toth, "Dynamic programming algorithms for the zero-one knapsack problem," *Computing*, vol. 25, no. 1, pp. 29–45, 1980.

[133] E. W. Dijkstra and Others, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[134] D. Karger, R. Motwani, and G. D. S. Ramkumar, "On approximating the longest path in a graph," in *Workshop on Algorithms and Data structures*. Springer, 1993, pp. 421–432.

[135] E. W. Dijkstra, "Some theorems on spanning subtrees of a graph," *Indag. math*, vol. 22, no. 2, pp. 196–199, 1960.

[136] E. T. A. Club, R. W. Bulterman, F. W. van der Sommen, G. Zwaan, T. Verhoeff, A. J. M. van Gasteren, and W. H. J. Feijen, "On computing a longest path in a tree," *Information Processing Letters*, vol. 81, no. 2, pp. 93–96, 2002.

[137] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[138] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[139] H. Everett III, "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources," *Operations research*, vol. 11, no. 3, pp. 399–417, 1963.

[140] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *Journal of the ACM (JACM)*, vol. 30, no. 4, pp. 852–865, 1983.

[141] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[142] IBM, "IBM ILOG CPLEX 12.10 User's Manual," 2020. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[143] Gurobi Optimization LLC, "Gurobi Optimizer Reference Manual," 2020. [Online]. Available: http://www.gurobi.com

[144] M. ApS, "The MOSEK optimization toolbox for MATLAB manual. Version 9.2." 2020. [Online]. Available: https://docs.mosek.com/9.2/cxxfusion/index.html

[145] C. Coey, M. Lubin, and J. P. Vielma, "Outer approximation with conic certificates for mixed-integer convex problems," *arXiv preprint arXiv:1808.05290*, 2018.

[146] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," Optimization Online, Technical Report, mar 2020. [Online]. Available: http://www.optimization-online.org/DB{_}HTML/2020/03/7705.html

[147] FICO, "FICO®Xpress Optimization Suite," 2020.

[148] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear algebra and its applications*, vol. 284, no. 1-3, pp. 193–228, 1998.

[149] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A Modeling Language for Mathematical Optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.
[150] R. Hettich and P. Zencke, *Numerische Methoden der Approximation und semi-infiniten Optimierung*. Springer-Verlag, 2014.
[151] J. Soch, T. J. Faulkenberry, K. Petrykowski, and C. Allefeld, "StatProofBook/StatProofBook.github.io: StatProofBook 2020," dec 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4305950
[152] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*. SIAM, 2019.
[153] C. Lemaréchal, "Cauchy and the gradient method," *Doc Math Extra*, vol. 251, p. 254, 2012.
[154] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984, pp. 302–311.
[155] Y. E. Nesterov and A. S. Nemirovskii, "Interior point methods in convex programming: theory and applications," *Society for Industrial and Applied Mathematics, Philadelphia*, 1994.
[156] S. S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004, vol. 25, no. 3.
[157] N. Z. Shor, "The Subgradient Method," in *Minimization methods for non-differentiable functions*. Springer, 1985, pp. 22–47.
[158] B. T. Polyak, "Minimization of unsmooth functionals," *USSR Computational Mathematics and Mathematical Physics*, vol. 9, no. 3, pp. 14–29, 1969.
[159] K. C. Kiwiel, "Efficiency of proximal bundle methods," *Journal of Optimization Theory and Applications*, vol. 104, no. 3, pp. 589–603, 2000.
[160] G. B. Dantzig, *Linear programming and extensions*. Princeton university press, 1998, vol. 48.
[161] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
[162] M. Glavic, "Interior point methods: A survey, short survey of applications to power systems, and research opportunities," University of Liege, Tech. Rep., 2004.
[163] C. Carathéodory, "Über den Variabilitätsbereich der Koeffizienten von Potenzreihen, die gegebene Werte nicht annehmen," *Mathematische Annalen*, vol. 64, no. 1, pp. 95–115, 1907.
[164] H. D. Sherali, "A constructive proof of the representation theorem for polyhedral sets based on fundamental definitions," *American Journal of Mathematical and Management Sciences*, vol. 7, no. 3-4, pp. 253–270, 1987.
[165] V. Mirrokni, R. P. Leme, A. Vladu, and S. C.-w. Wong, "Tight bounds for approximate Carathéodory and beyond," in *International Conference on Machine Learning*, 2017, pp. 2440–2448.
[166] J. Barcelo, E. Hallefjord Åand Fernandez, and K. Jörnsten, "Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing," *Operations-Research-Spektrum*, vol. 12, no. 2, pp. 79–88, 1990.
[167] M. A. Odijk, "A constraint generation algorithm for the construction of periodic railway timetables," *Transportation Research Part B: Methodological*, vol. 30, no. 6, pp. 455–464, 1996.
[168] B. Boots, G. J. Gordon, and S. M. Siddiqi, "A constraint generation approach to learning stable linear dynamical systems," in *Advances in neural information processing systems*, 2008, pp. 1329–1336.
[169] B. Zeng and L. Zhao, "Solving two-stage robust optimization problems using a column-and-constraint generation method," *Operations Research Letters*, vol. 41, no. 5, pp. 457–461, 2013.
[170] M. M. Mäkelä, N. Karmitsa, and A. Bagirov, "Subgradient and Bundle Methods for Nonsmooth Optimization," in *Numerical Methods for Differential Equations, Optimization, and Technological Problems: Dedicated to Professor P. Neittaanmäki on His 60th Birthday*, S. Repin, T. Tiihonen, and T. Tuovinen, Eds. Dordrecht: Springer Netherlands, 2013, pp. 275–304. [Online]. Available: https://doi.org/10.1007/978-94-007-5288-7{_}15
[171] M. S. Talebi and A. Proutiere, "An optimal algorithm for stochastic matroid bandit optimization," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 2016, pp. 548–556.
[172] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A Modeling Language for Mathematical Optimization," *arXiv:1508.01982 [math.OC]*, vol. 59, no. 2, pp. 295–320, aug 2015. [Online]. Available: http://arxiv.org/abs/1508.01982

Index

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Algorithmes en temps polynomial pour les semi-bandits combinatoires : apprentissage par renforcement efficace dans des environnements complexes

Mots clés : apprentissage par renforcement, bandit combinatoire, optimisation mathématique

Résumé : La prise de décision séquentielle est une composante essentielle de nombreuses applications, de la gestion des réseaux informatiques aux annonces en ligne. L'outil principal est l'apprentissage par renforcement : un agent prend une séquence de décisions afin d'atteindre son objectif, avec des mesures typiquement bruitées de son environnement. Par exemple, un agent peut contrôler une voiture autonome; l'environnement est la ville dans laquelle la voiture se déplace. Les problèmes de bandits forment une classe d'apprentissage de renforcement pour laquelle on peut démontrer de très forts résultats théoriques. Les algorithmes de bandits se concentrent sur le dilemme exploration-exploitation : pour avoir une bonne performance, l'agent doit avoir une connaissance approfondie de son environnement (exploration) ; cependant, il doit aussi jouer des actions qui le rapprochent de son but (exploitation).
Dans cette thèse, nous nous concentrons sur les bandits combinatoires, qui sont des bandits dont les décisions sont très structurées (une structure "combinatoire"). Il s'agit notamment des cas où l'agent détermine un chemin à suivre (sur une route, dans un réseau informatique, etc.) ou des publicités à afficher sur un site Web. De telles situations partagent leur complexité algorithmique : alors qu'il est souvent facile de déterminer la décision optimale lorsque les paramètres sont connus (le temps pour traverser une route, le profit généré par l'affichage d'une publicité à un endroit donné), la variante bandit (lorsque les paramètres doivent être déterminés par des interactions avec l'environnement) est bien plus complexe.
Nous proposons deux nouveaux algorithmes pour aborder ces problèmes par des techniques d'optimisation mathématique. Basés sur des hypothèses faibles, ils présentent une complexité temporelle polynomiale, tout en étant performants par rapport aux algorithmes de pointe pour les mêmes problèmes. Ils présentent également d'excellentes propriétés statistiques, ce qui signifie qu'ils trouvent un équilibre entre exploration et exploitation proche de l'optimum théorique. Les travaux précédents sur les bandits combinatoires ont dû faire un choix entre le temps de calcul et la performance statistique ; nos algorithmes montrent que ce dilemme n'a pas lieu d'être.

Title: Polynomial-Time Algorithms for Combinatorial Semibandits: Computationally Tractable Reinforcement Learning in Complex Environments

Keywords: reinforcement learning, combinatorial bandit, mathematical optimisation

Abstract: Sequential decision making is a core component of many real-world applications, from computer-network operations to online ads. The major tool for this use is reinforcement learning: an agent takes a sequence of decisions in order to achieve its goal, with typically noisy measurements of the evolution of the environment. For instance, a self-driving car can be controlled by such an agent; the environment is the city in which the car manœuvers. Bandit problems are a class of reinforcement learning for which very strong theoretical properties can be shown. The focus of bandit algorithms is on the exploration-exploitation dilemma: in order to have good performance, the agent must have a deep knowledge of its environment (exploration); however, it should also play actions that bring it closer to its goal (exploitation).
In this dissertation, we focus on combinatorial bandits, which are bandits whose decisions are highly structured (a "combinatorial" structure). These include cases where the learning agent determines a path to follow (on a road, in a computer network, etc.) or ads to display on a Website. Such situations share their computational complexity: while it is often easy to determine the optimum decision when the parameters are known (the time to cross a road, the monetary gain of displaying an ad at a given place), the bandit variant (when the parameters must be determined through interactions with the environment) is more complex.
We propose two new algorithms to tackle these problems by mathematical-optimisation techniques. Based on weak hypotheses, they have a polynomial time complexity, and yet perform well compared to state-of-the-art algorithms for the same problems. They also enjoy excellent statistical properties, meaning that they find a balance between exploration and exploitation that is close to the theoretical optimum. Previous work on combinatorial bandits had to make a choice between computational burden and statistical performance; our algorithms show that there is no need for such a quandary.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France