ANNÉE 2019

N° d'ordre : ?? ?? ??

Thèse / CentraleSupélec
sous le sceau de l'Université Bretagne Loire

pour le grade de

Docteur de CentraleSupélec

Mention : Télécommunications

Ecole doctorale 601 « Mathématiques et Sciences et Technologies de l'Information et de la Communication – (MathSTIC) »

présentée par

Lilian Besson

Algorithmes de bandits multi-joueurs pour l'Internet des Objets

Préparée à l'UMR 6164 - IETR (Équipe SCEE)

Institut d'Électronique et de Télécommunications de Rennes

Thèse soutenue à Rennes, le ?? octobre 2019, devant le jury composé de :

Vianney Perchet	Professeur à l'École Normale Supérieure de Paris Saclay / rapporteur
Nathalie Mitton	Directrice de Recherche à Inria, Lille / rapportrice
Richard Combes	Professeur Assistant à CentraleSupélec, Gif-sur-Yvette / évaluateur
Patrick Maillé	Chargé de Recherche à IMT Atlantique, Rennes / évaluateur
Raphaël Féraud	Chercheur à Orange Labs, Lannion / évaluateur
Christophe Moy	Professeur à Université Rennes 1 / *directeur de thèse*
Émilie Kaufmann	Chargée de Recherche au CNRS, Lille / encadrante

À mes très chers frères, mes deux plus grands modèles.

Acknowledgements

I would like to acknowledge and thank the following people, and also acknowledge the various fundings who supported our works during the last three years and made this thesis possible.

My first thoughts go to my two PhD advisors.

Firstly, thanks to Christophe Moy who welcomed me in the SCEE team, in IETR lab and in CentraleSupélec campus of Rennes, a very nice and warm environment for research and day-to-day life. Christophe was very supportive on all aspects of the PhD work, and he was also helpful during difficult moments in the last three years. Many thanks for being so nice and dynamic[1], and always comprehensive. Christophe was helpful for his expertise on wireless communications and engineering, and especially on cognitive radio. Regardless of what we did together, whether it was debugging a difficult GNU Radio block [BBM18, BBM19], writing a paper by night while Christophe was in a plane coming back from India [MB19], chatting at a barbecue or playing "palets" together along the ocean in Vannes, I was lucky to collaborate with and be around Christophe.

Secondly, thanks to Émilie Kaufmann, who accepted to co-supervised my thesis, and did an amazing job in the last four years[2]. Émilie was highly motivated by our collaboration, and she always pushed me further, to explore the mathematical aspects of my PhD work, to keep faith in the research world, and to keep the highest possible rigour in all parts of our exchanges and productions. I was lucky to be able to visit the SequeL team at Inria Lille on regular occasions, and to work in close collaboration with Émilie during these visits[4]. Intensive white board sessions proved to be fruitful, giving the research work that I am the most proud of [BK18a], and rich research papers that join her rigorous expertise in statistical learning and mathematical analysis of bandit algorithms with my passion for numerical experiments, simulations and algorithm designs [BK18a, BK18b, BK19b].

Then I would like to thank the dear members of my PhD committee. I was lucky to be able to discuss with them on regular occasions, and these discussions have been very useful for

[1] We had a nice first contact when we met in Spring 2015 in Mahindra Ecole Centrale, in Hyderabad, India, where I taught for one year, and I am honored to have pursued my PhD under his supervision and to have worked with him, and I am also happy to have lived great human moments alongside Christophe since 2016.

[2] Since our first discussions, in Autumn 2015 when she was giving practical sessions for the Reinforcement Learning course of Alessandro Lazaric at MVA[3], to the last repetitions of my PhD presentation in October 2019.

[4] Merci à Romain et toi pour votre accueil chaleureux, et aussi merci à Florian, Nicolas et Édouard.

some of our works. Many thanks to Vianney Perchet and Nathalie Mitton for accepting to review my thesis during the summer 2019, and for their very valuable comments. Thanks to Patrick Maillé and Raphaël Féraud for following our work since Spring 2017, and to Richard Combes as well for being part of my jury. Some of their articles were strong inspirations for our own works, and I wish to also thank their co-authors, and in particular the Masters or PhD students, and useful discussions we had whenever we met, especially thanks to Claire Vernade, Réda Alami, Étienne Boursier, Frédéric Loge-Munerel and Pratik Gajane, and others I could not meet (*e.g.*, Viktor Toldov or Robin Allesiardo).

Of course on a more personal level, my thoughts are strongly dedicated to my family. I first want to thank my older brother Florian, who has always been my main model and who has been of a constant support during all my studies. His solid and serious work kept pushing me further, some of our board games nights were brain teasers comparable with some maths problems solved for this thesis. I also thank my younger brother Fabian, for being an inspiration for other (highly important) directions of my life. Very warm thanks then go to Marine, to my grand-parents, Jeannine, Jean-Marc and Jacqueline, my uncle Jacques and aunt Bernadette, and my cousins. Finally, I thank sincerely my parents, Christophe and Patricia, for being the best parents since the last 30 years, and for their vital moral support. *Merci infiniement de nous avoir inculqué cette curiosité et d'être si gentils, et merci de votre soutien constant et de votre amour !*

I would like to thank my closest friends, sorted in chronological order of first encounter in the tall grass[5]: Florian, Fabian, Guillaume, Florian, Marian, Hélène, Lucie, Lætitia and Julia in Briançon, Mayotte and Pierre in Marseille, Laurent, Simon, Tristan, Ludovic, Jessica, Romain, Benjamin, Angèle, Clément in Cachan, Jill-Jênn, Alain, Claire, Édouard, Loïc and Damien also in Cachan, Kumudham and Priscilla in India, Thibault, Sébastien, Sélim, Valentin again in Cachan, and Rémi, Marine, Adrien, Bastien, Claire, Corentin and Lola in Rennes. *J'ai une pensée particulière pour Hélène, merci pour ton énergie et ta détermination depuis 2018.*

Thanks to my wonderful colleagues during these three years: Rémi*, Pascal*, Yves, Jacques, Ali, Haïfa, Marwa, Navik, Quentin*, Vincent*, Rami*, Muhammad, Cristo*, Adrien*, Esteban*, Ali, Éloïse*, Nabil*, Bastien*, Corentin*, Georgios, Morgane* in Rennes ; and Alessandro, Ronan, Odalric-Ambrym, Philippe, Olivier, Florian, Xuedong, Guillaume, Julien, Mathieu, Matteo, Mahsa, Nicolas, Édouard in Lille. I want to thank especially the "coffee friends*" in Rennes, for the daily interesting discussions, the best moments of our life at the office. Thanks to everybody who played *futsal* on Wednesday lunch, I still have to progress immensely before I could pretend to be useful, but it's been a lot of fun to play with you guys!

Thanks to some people for their administrative support regarding research, travelling or teaching: Karine, Jeannine, Grégory, Anne, Cécile (and others) at CentraleSupélec campus of

[5] Sorry if I forgot you, it's much harder to keep track of wild encounters in the tall grass in real wife, without a Pokédex! And due to space constraints, I had to chose quite arbitrarily who to include…

Rennes, Amélie at Inria Lille and Maryline (and others) at CRIStAL in Lille. Thanks to the nice people at Sodexo in the CentraleSupélec campus of Rennes, especially Martine and J-B, your daily smile is priceless! I warmly thank David P. for his trust, and François, David P. and David C. at ENS de Rennes, and Romaric at ENSAI : I loved every single of the 3×64 hours I spent teaching with you in the last three years. I am grateful to Amélie at Lycée Chateaubriand and Gilbert at Lycée Joliot-Curie for useful exchanges. I am thankful to the great professors who taught me while I studied at ENS de Cachan, between 2011 and 2016, and for some who pushed me into teaching, and studying statistical learning and computer science, including Frédéric, Alain, Claudine and Nicolas in the maths department, Hubert, Jean, Sylvain, Paul, Serge in the computer science department, and thanks to Carine, Delphine and the late Nicolas for their support.

My PhD was funded by the French Ministry of Higher Education and Research (MENESR), thanks to a "contrat doctoral spécifique normalien" obtained while I was a "normalien" student at École Normale Supérieure de Paris-Saclay. This work has been supported by the French National Research Agency (ANR), under the projects BADASS, SOGREEN and EPHYL (grants *N ANR-16-CE40-0002*, *N ANR-14-CE28-0025-02* and *N ANR-16-CE25-0002-03*), by Région Bretagne, France, by CNRS under the PEPS project BIO (in 2017), by European Union, through the European Regional Development Fund (ERDF), and by Ministry of Higher Education and Research, Brittany and Rennes Métropole, through the CPER Project *SOPHIE / STIC & Ondes*.

I would also like to thank some unusual things. Some people thank God in their thesis, I feel that I prefer to thank the Earth, and apologize for the too many times I took a plane for a conference abroad, and in general apologize for the environmental impact of this research work. I'll try hard to do better from now on! I also thank my bike which I enjoyed riding everyday despite the too many accidents and rain, my enthusiastic daily cooking which gave me smiles and energy, and finally my favorite technological tools: git, GitHub and Bitbucket ; LaTeX, Bash, Julia and Python ; GNU/Linux and XUbuntu, Firefox, Konsole and Visual Studio Code, and many other great free and open-source tools. . .

In the last few years, some friends thanked me in their own PhD thesis, and it is a pleasure to do the same, and cross-link to their own PhD thesis. In a chronological order, I thank Jill-Jênn Vie (thesis), Navikkumar Modi (thesis), Quentin Bodinier (thesis), Florian Besson[6], Jessica Guérand (thesis), Romain Ducasse (thesis), Simon Abelard (thesis), Ludovic Sacchelli (thesis), Claire Brécheteau (thesis), Damien Allonsius (thesis), Rémi Bonnefoi, and Rami Othman.

Deseo terminar este agradecimiento con un pensamiento especial para Lola, quien me apoyó diariamente, me sorprendió con su entusiasmo y me consoló con sus risas y su amor, durante el año pasado.

[6] Que je remercie deux fois plus que ce qu'il me remerciant dans sa propre thèse ! Pour répondre à la question, deux ans plus tard, tu remerciais Tom et moi un nombre non nul $x \in \mathbb{N} \cup \{+\infty\}$ satisfaisant $x = 2x$, soit $x = +\infty$, et je t'en remercie $2x$ fois !

Résumé

Dans cette thèse de doctorat, nous étudions les réseaux sans fil et les appareils reconfigurables qui peuvent accéder à des réseaux de radio intelligente, dans des bandes non licenciées et sans supervision centrale. Nous considérons des réseaux de l'Internet des Objets (IoT), avec l'objectif d'augmenter la durée de vie de la batterie des appareils, en les équipant d'algorithmes d'apprentissage machine peu coûteux mais efficaces, qui leur permettent d'améliorer automatiquement l'efficacité de leurs communications sans fil. Nous proposons différents modèles de réseaux IoT, et nous montrons empiriquement, par des simulations numériques et une validation expérimentale réaliste, le gain que peuvent apporter nos méthodes, qui utilisent l'apprentissage par renforcement. Les différents problèmes d'accès au réseau sont modélisés avec des Bandits Multi-Bras (MAB), mais leur analyse est difficile à réaliser, car il est délicat de prouver la convergence de nombreux appareils jouant à un jeu collaboratif sans communication ni aucune coordination, lorsque les appareils suivent tous un modèle d'activation aléatoire. Le reste de ce manuscrit étudie donc deux modèles restreints, d'abord des bandits multi-joueurs dans des problèmes stationnaires, puis des bandits mono-joueur non stationnaires. Nous détaillons également une autre contribution, la bibliothèque Python open-source SMPyBandits pour des simulations numériques de problèmes MAB, qui couvre les modèles étudiés et d'autres.

Abstract

In this PhD thesis, we study wireless networks and reconfigurable end-devices that can access large-scale Cognitive Radio networks, in unlicensed bands and without central control. We focus on Internet of Things networks (IoT), with the objective of extending the devices' battery life, by equipping them with low-cost but efficient machine learning algorithms, to let them automatically improve the efficiency of their wireless communications. We propose different models of IoT networks, and we show empirically on both numerical simulations and real-world validation the possible gain of our methods, that use Reinforcement Learning. The different network access problems are modeled as Multi-Armed Bandits (MAB), but we found that analyzing the realistic models was intractable, because proving the convergence of many end-devices playing a collaborative game without communication nor coordination is hard, when end-devices all follow random active pattern. The rest of this manuscript thus studies two restricted models, first multi-players bandits in stationary problems, then non-stationary single-player bandits. We also detail another contribution, SMPyBandits, an open-source Python library for numerical MAB simulations, covering all the studied models and more.

Resumé des Travaux de Thèse

Dans cette thèse de doctorat, nous étudions les réseaux sans fil et les appareils reconfigurables qui peuvent accéder à des réseaux de radio intelligente, dans des bandes non licenciées et sans supervision centrale. Plus spécifiquement, nous considérons des réseaux de l'Internet des Objets (IoT), avec l'objectif d'augmenter la durée de vie de la batterie des appareils, en les équipant d'algorithmes d'apprentissage machine peu coûteux mais efficaces, qui leur permettent d'améliorer automatiquement l'efficacité de leurs communications sans fil (Chapitre 1). Nous proposons différents modèles de réseaux IoT, et nous montrons empiriquement, par des simulations numériques et une validation expérimentale réaliste, le gain que peuvent apporter nos méthodes, qui utilisent l'apprentissage par renforcement (Chapitre 5). Les différents problèmes d'accès au réseau sont modélisés avec des Bandits Multi-Bras (MAB, Chapitre 2), mais leur analyse est difficile à réaliser, car il est difficile de prouver la convergence de nombreux appareils jouant à un jeu collaboratif sans communication aucune coordination, lorsque les appareils suivent tous un modèle d'activation aléatoire. Le reste de ce manuscrit étudie donc deux modèles restreints, d'abord des bandits multi-joueurs dans des problèmes stationnaires (Chapitre 6), puis des bandits mono-joueurs non stationnaires (Chapitre 7). Nous détaillons également une autre contribution, la bibliothèque Python open-source SMPyBandits pour des simulations numériques de problèmes MAB, qui couvre tous les modèles étudiés et d'autres (Chapitre 3).

Contributions

Nous pouvons énumérer les points suivants pour résumer les principales contributions de cette thèse. Nous signalons les conférences internationales auxquelles ont été publiées certains résultats.

- Nous avons écrit la plus complète bibliothèque de simulation pour les problèmes MAB, appelée SMPyBandits, qui est écrite en Python et publiée en ligne sous une licence open-source [Bes19, Bes18]. Nous présentons en détail son architecture et ses fonctionnalités dans le Chapitre 3, ainsi que différents exemples de son utilisation. Une documentation complète est disponible en ligne, ainsi que des instructions exhaustives pour reproduire les expériences utilisées dans la suite de cette thèse.

- Nous présentons le problème du choix de l'algorithme qu'un praticien devrait utiliser, ou de la sélection d'algorithme parmi la riche collection de différents algorithms MAB disponibles, dans le Chapitre 4. Nous présentons un algorithme appelé Aggregator pour l'agrégation d'algorithmes, comme une solution en ligne au problème de sélection d'algorithmes, et nous montrons par quelques simulations numériques qu'il atteint de bonnes performances empiriques (WCNC 2018) [BKM18].

- Nous proposons différents modèles pour les réseaux IdO (IoT), dans le Chapitre 5, où les appareils dotés de capacités de radio intelligente peuvent implémenter de leur côté des algorithmes MAB, pour augmenter automatiquement la durée de vie de leur batterie. Cela permet à davantage de périphériques d'utiliser le même réseau tout en maintenant une haute qualité de service (CROWNCOM 2017, ICT demo 2018, WCNC 2019 et MOTIoN 2019) [BBM+17, BBM19, BBMVM19].

- Nous avons implémenté une preuve de concept du modèle susmentionné [BBM18], et nous la présentons en détail en Section 5.3. Nous avons aussi réalisé une courte vidéo présentant notre démonstration, hébergée sur `youtu.be/HospLNQhcMk`.

- Nous formalisons le modèle du bandit multi-joueur, pour lequel nous avons introduit trois variantes, dans le Chapitre 6. Pour le cas avec information de détection ("sensing"), nous proposons deux nouveaux algorithmes, et nous donnons une analyse pour notre algorithme MCTopM, qui prouve qu'il est asymptotiquement optimal. Nous présentons aussi des expériences numériques approfondies pour montrer qu'il est bien plus efficace que les autres algorithmes de la littérature précédente. Notre travail [BK18a] (ALT 2018) a également contribué à une nouvelle impulsion à la recherche sur les bandits multi-joueurs, car certains travaux de recherche récents se sont construits sur nos résultats.

- Nous donnons également une revue détaillée de la littérature sur les différentes extensions du modèle MAB multi-joueurs, qui, selon nous, n'a jamais été écrit auparavant.

- Nous présentons ensuite le modèle de bandits multi-bras stationnaire par morceaux, dans le chapitre 7, et une vue détaillée de l'état de l'art de la recherche sur ce modèle [BK19b, BK19a] (GRETSI 2019). Suite à deux travaux récents, nous proposons un nouvel algorithme activement adaptatif pour ce problème stationnaire par morceaux, GLR-klUCB, qui atteint des performances comparables à l'état de l'art.

- La dernière contribution de cette thèse est une revue de la littérature sur les cas d'utilisation possibles de la technique du doublement successif de l'horizon ("doubling trick") pour les problèmes de bandits, ainsi qu'une analyse unifiée et plus générique de deux familles de doublements. Ceci a conduit à l'article [BK18b], qui est rapidement présenté en Annexe A.

Organisation du manuscrit

Ce manuscrit est organisé comme suit.

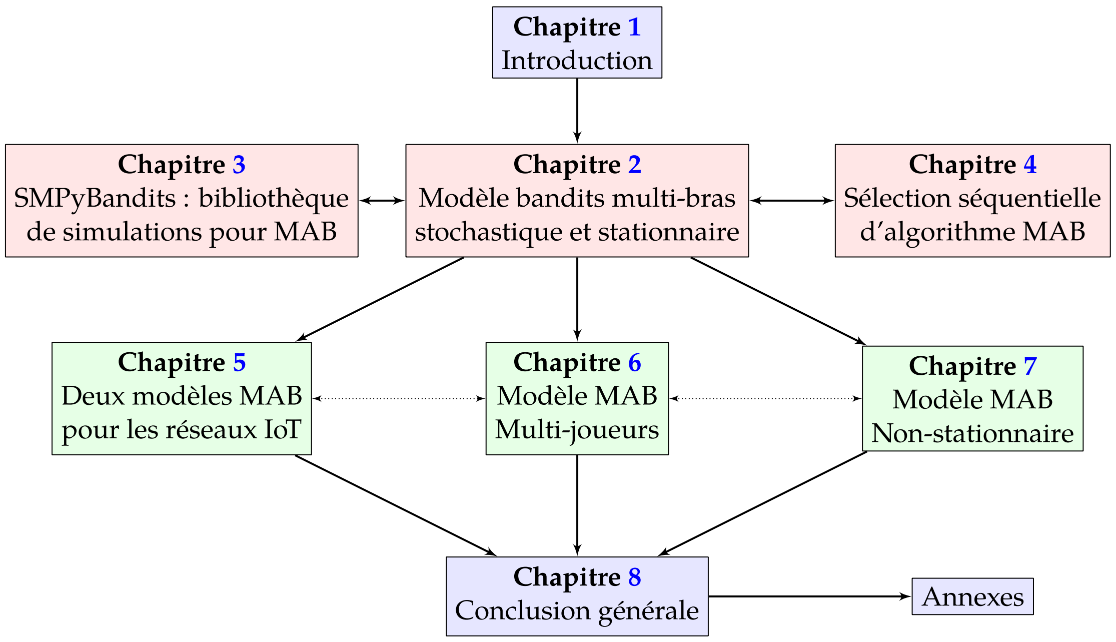

Chapitre 1
Introduction
Chapitre 3
SMPyBandits : bibliothèque de simulations pour MAB
Chapitre 2
Modèle bandits multi-bras stochastique et stationnaire
Chapitre 4
Sélection séquentielle d'algorithme MAB
Chapitre 5
Deux modèles MAB pour les réseaux IoT
Chapitre 6
Modèle MAB Multi-joueurs
Chapitre 7
Modèle MAB Non-stationnaire
Chapitre 8
Conclusion générale
Annexes

Figure 1 – Organisation de la thèse. Cette thèse peut se lire en suivant n'importe quel chemin contenant le Chapitre 1, le Chapitre 2, au moins un des trois Chapitres 5, 6 et 7, et la Conclusion.

L'ordre de lecture du manuscrit peut être n'importe quel chemin, descendant entre l'introduction donnée dans Chapitre 1, et la conclusion générale qui constitue le dernier Chapitre 8. Comme le montre le graphique de la Figure 1 ci-dessus, la thèse est organisée en deux parties :

- Dans la Partie I, nous commençons par le Chapitre 2 où nous présentons les modèles MAB, les concepts et les notations utilisés dans tout ce document. Ce premier chapitre est nécessaire à la lecture du reste du manuscrit. Par contre le Chapitre 3 présente notre bibliothèque de simulations SMPyBandits, et même si les Chapitres 2, 4, 6 et 7 utilisent la bibliothèque pour leurs simulations numériques, lire ce chapitre n'est pas obligatoire pour comprendre la suite du document. Nous terminons cette première partie par le Chapitre 4, qui n'est pas non plus nécessaire à la compréhension de la suite du manuscrit, et qui détaille la première contribution : un nouvel algorithme pour la sélection séquentielle d'algorithmes MAB.

- La deuxième Partie II contient ensuite trois chapitres, qui sont inclus à la fois dans l'ordre logique et chronologique, mais peuvent être lus presque indépendamment. Le Chapitre 5 commence par présenter différents modèles de réseaux IdO où les algorithmes MAB ont été utilisés avec succès. Nos deux modèles sont intéressants et proches de la réalité, mais ils se sont révélés trop généraux pour proposer une analyse mathématique de la bonne performance empirique des solutions envisagées. Pour cette raison, nous affaiblissons les modèles pour le reste du document, et les deux Chapitres 6 et 7 étudient un modèle intermédiaire, situé entre le modèle MAB stationnaire à un joueur du chapitre 2 et les modèles de réseaux IdO du chapitre 5.

Table of Contents

Chapter 1

Introduction

Where and when? This manuscript concludes my doctoral thesis, which started in October 2016 and finished in October 2019. My research took place at the IETR laboratory in Rennes (France), in the SCEE team hosted on the Rennes campus of the engineering college CentraleSupélec. In Rennes I was supervised by Professor Christophe Moy, and I was also co-supervised by Doctor Emilie Kaufmann, whom I visited many times at Inria Lille Nord Europe in Lille (France).

1.1 Context of this thesis

At the root of the problems that motivate this thesis are the two questions of global warming, and the increase of the world population. During the last 150 years, mankind has developed a wide range of different communication technologies, and since the late 1890s wireless communications between manufactured devices have been made possible, and more and more frequent in our lives. With the avent of Internet of Things networks (IoT), billions of autonomous low-power devices are expected to be deployed worldwide, for a large range of different applications. It is now well-known that with the current trend of population increase and with the energy crisis, any newly deployed technology should be cheap and energy efficient, as well as adapted to serve a large number of people and devices.

That is why we are interested in this thesis about the possible applications of embedding a certain kind of Machine Learning algorithms (Multi-Armed Bandit algorithms), directly into the future IoT devices, in order to improve their battery life and reduce the energy cost of IoT networks, by optimizing the wireless communication thanks to embedded decentralized decision making.

From old TV to the IoT standards. Historically, three families of wireless communication systems have been deployed: first, centralized broadcasting (*e.g.*, old TV), then centralized bi-directional systems (*e.g.*, 4G or WiFi), and nowadays decentralized broadcasting for the Internet of Things (IoT) networks.

Starting from FM radio music and news, then old black-and-white TV and color TV, the first systems were purely made for centralized broadcasting of information. In a large area, one antenna well located emits continuously in a fixed Radio Frequency (RF) band, and many devices can listen to this RF traffic and decode it, for instance to listen to music. Different standards have been developed, and some of them are still in use today, and their nature requires a central authority that allocates RF bands to different usages (*e.g.*, TV channels etc). For example, in the city of Cesson-Sévigné where the campus of CentraleSupélec is located, a high RF tower emits on many standards, and in particular the 92.3 MHz band is allocated to a French classical music radio (*Radio Classique*). Broadcast-based systems are not only used for TV or radio, and for instance two other famous and worldwide deployed examples are the long-range global navigation satellite systems (such as USA's GPS or European Union's Galileo), and short-range infrared remote controllers.

However, this kind of mono-directional wireless communications were soon found to not be satisfactory for many applications, and thus bi-directional standards were defined from the 1910s. First developed by and for the military, and the navy (*e.g.*, the Titanic sent the first SOS signal over radio in 1912), such applications have reached general public usage since the 1950s. The greatest achievement of this long-running R&D domain was the avent of mobile telephony in the 1990s, which has continued to be more and more present in people lives in the richer countries in the world, with the booming of cellphones since the 2000s and smartphones since the 2010s. In most systems, there is still an antenna in charge of a large area (or cell), and many devices are able to receive and transmit data to the antenna. Nowadays, the most well-known systems include WiFi, and systems deployed for mobile phone communications, with a long list of standards from 1G, 2G, 3G, 4G and now 5G, still in development in 2019. Even if both systems initially differed in usage, as 1G was designed to exchange only voice, and WiFi to link wireless devices at home to an Internet-connected box, they are conceptually very close. As the systems are cellular, any device knows the antenna (or base station) it is associated with, and the bi-directional nature of their exchanges makes a centralized control possible. In order to optimize the efficiency of the network, the base station is equipped with decision making algorithms that affects the devices in the cell to the available resources (time slots, RF bands, power etc). To be able to communicate with the base station without interfering with each others, such affectation should be orthogonal, and led to systems based on Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA), or Non-Orthogonal techniques only started to emerge recently (*e.g.*, NOMA). Traditionally, the base station aimed only at maximizing the Quality of Service (QoS), that englobes the data rate, latency, availability and other measures of performance of the wireless communications to and from the devices. More recently, green radio has emerged as a solution to the tradeoff between maximizing QoS while minimizing the power consumption of base stations and devices.

Finally, a third kind of systems are of the opposite kind, and can be designed as decentralized broadcasting: an antenna is still in charge of many devices, but such devices can only send up-link packets, and the only down-link data they can receive are short acknowledgements sent from the base-station to indicate success or failure of every up-link packet. This family of wireless systems are referred to as Internet of Things, and a typical example of application is for sensor networks.

For the future development of smart grids, smart cities, smart homes, or smart agriculture, sensor networks are promised to be widely deployed. Two examples of future applications that are already in deployment, in France or other countries, are connected buildings and connected agriculture. For buildings, the main goal is to reduce the cost of heating empty buildings and use temperature sensor networks to get accurate and regular data about the temperature in each room and each floor, and let the centralized heat control optimize its cost and energy consumption. For agriculture, one example can be to equipped every cow (in large farms) with a sensor that emits biological information, such as body temperature or stress level etc, in order to optimize the time of milking and to monitor the health of the animals.

This third kind of wireless systems are characterized by their decentralized nature: due to the really limited information that the base station can send to the devices, it is no longer possible to optimize the efficiency of the network from the centralized point-of-view of the base station. In the present and future IoT networks, many devices of heterogeneous nature are using the same antenna for different applications. A common hypothesis is that such IoT devices have a strong constraint on their power consumption, as most of them will be deployed without a direct power access and run on a battery, which duration should be maximized (typically more than 10 years). Another common constraint for IoT devices is their low duty cycle, as most applications target one or a few messages to send every day, in strike opposition to the high data-rate pursued for centralized systems such as 4G/5G and WiFi. Many different standards for IoT networks have been proposed, and they consist in a specification for both the *PHY*sical and the *M*edium *A*ccess *C*ontrol (*MAC*) layer. To quote some examples of standards, ZigBee, Z-Wave or Bluetooth are targeting for short-range communications, while LoRaWAN, SIGFOX, Ingenu or Weightless are designed for long-range communications. We refer to the survey [CVZZ16] for more details, and references in [AC18] or our recent article [MB19].

The spectrum scarcity issue. A major problem of current wireless technologies is the issue of spectrum scarcity: in most kinds of frequency bands, the entire RF spectrum is now allocated and free bands no longer exist, limiting the possibility of adding new usage. As illustrated in Figure 1.1 below, only a very small portion of the RF spectrum in the USA is not yet allocated, from 0 to 9 kHz, while the rest of the displayed portion of the spectrum, from 9 kHz to 300 GHz, is allocated to various usages, that goes from maritime radionavigation (historically the first usage of radio telecommunication) to space research, inter-satellite, mobile telephony and a large number of other applications. Regulatory bodies in the world, like the Federal Communications Commission in the USA (see `FCC.gov`) or TDF in France (see `TDF.fr`, previously known as *"TéléDiffusion de France"*), as well as different independent measurement campaigns, found that most radio frequency spectrum was inefficiently utilized, meaning that while a band can be allocated to a certain unique usage, it can be free from any user in certain times and/or places. We refer to [PPS11] for a survey on worldwide spectrum utilization, and to [VMB+10] for the situation in Europe.

Cellular network bands are overloaded in most parts of the world, but other frequency bands (such as military, amateur radio and paging frequencies) are insufficiently utilized. Independent studies performed in some countries confirmed that observation, and concluded that spectrum usage highly depends on time and place. Moreover, the fixed spectrum allocation prevents rarely used frequencies, like those assigned to specific services, from being used, even when any unlicensed users would not cause noticeable interference to the assigned service. Therefore, in the last 15 years, regulatory bodies in the world have been considering whether to allow unlicensed users in licensed bands if they would not cause any interference to licensed users. These initiatives have initiated the research on **Cognitive Radio**, and especially on one subfield that we detail below, **Dynamic Spectrum Access (DSA)**. To highlight that the spectrum scarcity is not only an issue in the USA, we quote the abstract of the "Dynamic Spectrum Management For Innovation And Growth" report by J. Toledano [Tol14a, Tol14b], who was commissioned by the French government in 2014. Different important problems are mentioned:

- *"In France, digital economy represents 5% of GDP and affects 80% of the French economy."* And the usage of wireless communications is nowadays worldwide and omnipresent: *"Radio frequencies are essential to many sectors: communications, broadcasting, transportation, satellite networks, energy networks and smart grids, public or private security, defense, etc."* Thus it is of highest interest to set up efficient, green, low-cost and durable new technologies for the future wireless standards, networks and devices.

- The radio spectrum is not only widely used for all kinds of applications, it is nowadays intensely used for some of them. *"Everyone agrees on the growing need for spectrum. This growing need results from two phenomena. On the one hand, mobile traffic should be multiplied by 13 to 25 fold between 2011 and 2017. On the other hand, the development of new innovative services such as the Internet of Things and its multiple applications (smart cities, e-health...), could lead to a growing number of connected devices, up to 50 billion in 2020, according to estimations."* The first estimation was correct, and the mobile traffic is again estimated to be increased by 10 times between 2019 and 2025. If the second estimation was overestimating the growth of the IoT market, forecasts predict more than 64 billion IoT devices by 2025.

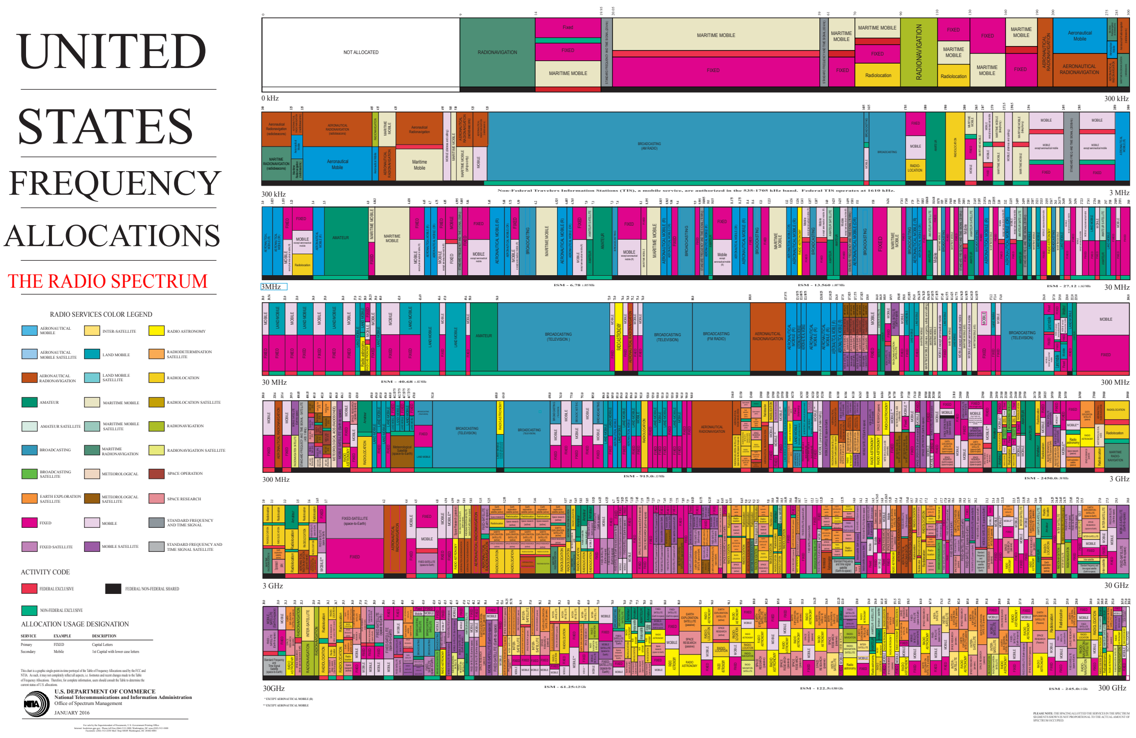


Figure 1.1 – A chart representing the allocation of radio spectrum in the USA in 2016. Only a small share of the spectrum is not yet allocated, the top-left corner in white (it is in logarithmic scale from 0 kHz to 300 GHz). © Wikimedia.

- *"Today, there are no more available frequencies in the easily exploitable frequency bands. Moreover, it is getting more and more difficult to resort to classical methods of frequency bands liberation."* As illustrated above in Figure 1.1, and discussed in [PPS11, VMB+10].

- *"Growing recourse to spectrum sharing, and in particular dynamic spectrum sharing, constitutes an important spectrum reserve. A particular sharing form, the use of unlicensed bands, open to all and free, has been intensely developed with the growth of Wi-Fi use."* This thesis focusses on spectrum sharing in unlicensed bands, for IoT networks that lie at the opposite direction as the one underlying Wi-Fi networks: the spectrum sharing will be learned automatically by the end-devices, and not centrally decided by the base station.

1.2 Cognitive Radio and Multi-Armed Bandits

In this thesis, we study Internet of Things networks, and these two constraints of low power consumption (or long battery life) and low duty cycle are essential. More precisely, we study the possible interconnections between **Cognitive Radio** and **Machine Learning** applied for IoT networks. Let us define and detail both concepts.

From Machine Learning to Multi-Armed Bandits. We can narrow down the first field of study of this thesis, steps by steps. Starting from the general concept of Machine Learning (ML), we focused our interest to statistical learning, then sequential learning, then Reinforcement Learning (RL), then RL with limited feedback and finally to Multi-Armed Bandits (MAB) learning. As illustrated in Figure 1.2, there are other types of ML, that we do not focus on in this thesis. The reference book on RL is [SB18], so let us quote the definition of RL given by R. Sutton and A. Barto: *"Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them"*.

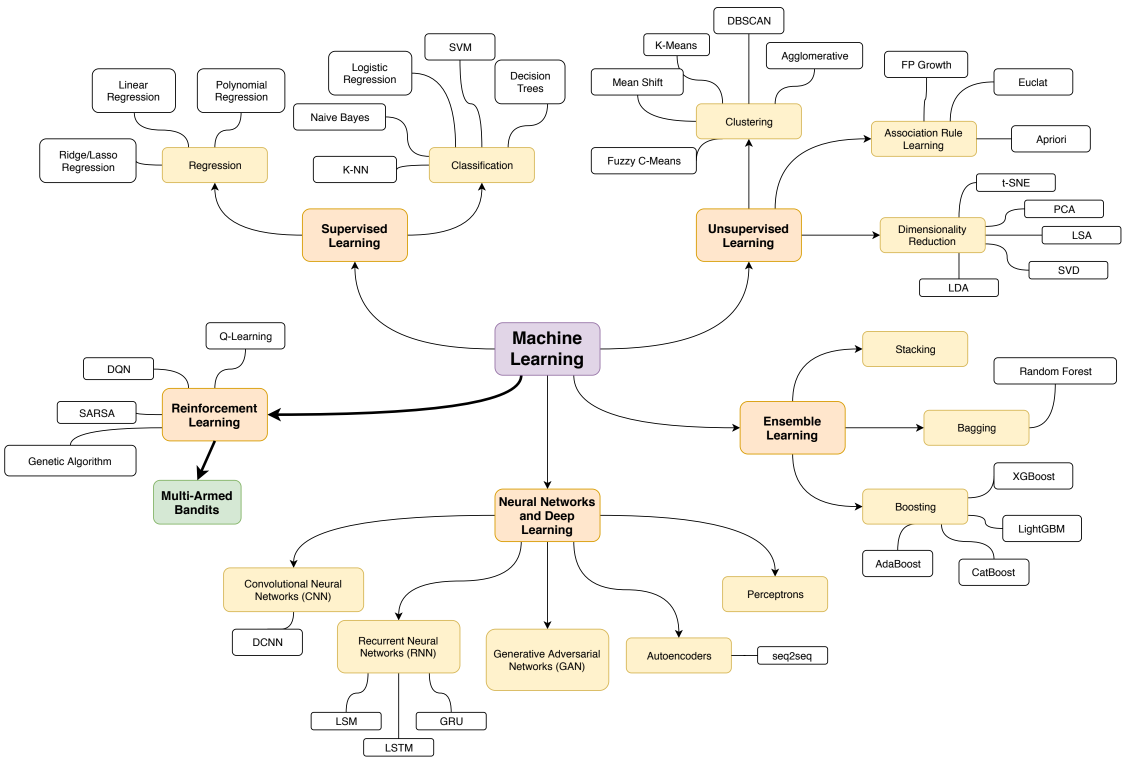


Figure 1.2 – Classification of the main types of Machine Learning: MAB is a subset of Reinforcement Learning and RL is a subset of ML. MAB is RL with partial information, and RL is ML with no supervision or data, just an environment to interact with.

We illustrate this idea of a *learning cycle* alternating between actions and feedback, in the following Figure 1.3. A player (or learner) interacts with its environment by taking an action $A(t)$ (*e.g.*, a choice in a finite set, $A(t) \in \{1, \dots, K\}$, or a vector $A(t) \in \mathbb{R}^d$), and then observing a reward $r(t)$, which is a certain measure of success from the environment (*e.g.*, $r(t) \in \{0, 1\}$ for binary failure/success, or $r(t) \in \mathbb{R}$). The goal of the player is to maximize its reward, by trials and errors (*i.e.*, actions and rewards). Many real-world problems can be framed as Reinforcement Learning problems, as illustrated by the survey [BR19] and Section 2.2, for instance learning to walk or to drive, learning to play a board or computer game, or discovering which treatment is efficient in healing a certain disease (clinical trial), etc.

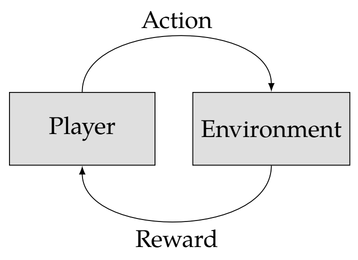


Figure 1.3 – Reinforcement learning cycle: a learner interacts with its environment through actions, and observes a reward, iteratively.

More precisely, in this thesis we focus on a certain kind of Reinforcement Learning models where the learner does not have access to the entire reaction of the environment after taking its action. In other words, we focus on RL with limited feedback, when the player only sees the reward given by its action at every round, and not the reward that would have been given if she chose any other action. This kind of limited feedback is called *bandit information*, and we discuss the history and the concept of *Multi-Armed Bandits* in details in the next Chapter 2. Clinical trials and treatment discovery were historically the first applications of MAB since 1930s, with the early work of W. Thompson [Tho33]. Multi-Armed Bandits (MAB) are a simple yet powerful example of the well-known exploration/exploitation dilemma: when facing a set of K actions whose effects on the environment are unknown, a learning must balance between exploration of the unknown actions, in order to collect more information about them, and exploitation of the best action according to its current knowledge. The MAB problem have been studied in both the machine learning and the statistics community, since the 1950s with pioneers like H. Robbins [Rob52], and more recently it is an active field of research, since the late 1990s [AVW87a, AVW87b, ACBFS95, Agr95]. Research on MAB has produced a vast literature in the 2000s [ACBF02, ACBFS02, AB09] and continues to be a topic of high interest, as illustrated by the surveys and books [BCB12, LS19, Sli19], and the wide range of applications of MAB in the recent years.

From IT to Cognitive Radio. Similarly, we can narrow down the second field of study of this thesis, steps by steps: from Information Technologies (IT), to telecommunications, to wireless communications, then to Software Defined Radio (SDR), and finally to Cognitive Radio (CR). The transition from the historical approach of hardware-based radio to SDR architectures and is a gradual process that started in the early 1990s and has accelerated in the 2000s. A SDR is a radio communication system where components that have been traditionally implemented in hardware (*e.g.*, mixers, filters, amplifiers, modulators/demodulators, detectors, etc) are instead implemented by means of software on a personal computer or embedded system, with the exception of a unique pair of receiver/transmitter antennas. Even though the SDR paradigm was initiated by the USA defense research, over the past years the industry has started to be interested by SDR, and CR have got significant attention from the academia and industry as well. CR is not a standard technology, and as such it does not have a single definition, so let us start by quoting the definition of two researchers whose works were the roots of the development of CR, in the last 20 years.

- Joseph Mittola in 1999 said that *"a really smart radio that would be self-, RF- and user-aware, and that would include software technology and machine learning capabilities along with a lot of high-fidelity knowledge of the radio environment"* [MM99].

- Then Simon Haykin in 2005 also said that *"a CR is an intelligent wireless communication system that is capable of being aware of its surroundings, learning, and adapting its operating parameters* (e.g., *transmit power and carrier frequency) on the fly with an objective of providing reliable anytime, anywhere, and spectrally efficient communication"* [Hay05]. He was the first to propose to use CR for dynamic spectrum access.

- The Wikipedia encyclopedia states *"A cognitive radio (CR) is a radio that can be programmed and configured dynamically to use the best wireless channels in its vicinity to avoid user interference and congestion. Such a radio automatically detects available channels in wireless spectrum, then accordingly changes its transmission or reception parameters to allow more concurrent wireless communications in a given spectrum band at one location"*.

In licensed bands, there are Primary Users (PU) paying to access the network, for instance anybody has to pay to have a mobile phone number and use the mobile telephony network. The PU have a strict priority over any other non-paying users, referred to as Secondary User (SU). So even though the RF bands are allocated, real world measurements often show that some bands are not densely used, and thus if a SU is equipped with an efficient sensing capacity, it can analyze its environment, and use a licensed band if it is free of any PU. This defines the concept of DSA was thus proposed in the early 2000s, and we refer to the survey [ZS07] for more details. We focus on one example, the **Opportunistic Spectrum Access** (OSA) problem.

Opportunistic Spectrum Access. In OSA, the focus is on one SU accessing a licensed spectrum, occupied by PU that have a strict priority over the SU, which has to follow a "listen-before-talk" access scheme. The following hypothesis are maid: the SU is equipped with sensing, and a consider a fixed and finite set of orthogonal channels, *i.e.*, different frequency bands in a licensed spectrum. For instance, it can be a set of three Wi-Fi channels at different frequency, emitted by the same Wi-Fi station. Another hypothesis is that both PU and SU are synchronized in times, by sub-dividing the time in discrete time steps. Thus if the SU spends a short time in the beginning of each time slot to do sensing, it can scan for the presence or absence of any PU before transmitting. If a SU was able to sense for all the K frequency bands, it could simply transmit in one of the free channels if any, or not transmit if all channels are used at a given time step. However, sensing is known to be error-prone and costly, especially for wide-band sensing, thus most works on OSA limit the sensing capacity of SU to sensing only one channel at a time. This assumption, along with the regulation that PU cannot be disturbed, enforces the SU to transmit in the channel that it sensed.

Focussing on one SU in an OSA network, it has to sequentially decide a channel to sense (in the set of $K \geq 2$ channels), then it performs sensing, and finally it transmits in this channel if it is free. The goal of the SU is to minimize its energy consumption (we remind that we focus on green radio) and to maximize its uplink data rate, or equivalently, to maximize its number of successful transmission. If the different channels are not uniformly used by the PU, and if we assume a stationary hypothesis on the PU traffic, then the goal of the SU boils down to explore the different channels and exploiting the best ones. This frames the Opportunistic Spectrum Access problem as an exploration/exploitation problem, with a finite set of actions (the channels, also called *arms*), in a sequential action-then-feedback cycle (time steps are $t \in \mathbb{N}^*$), under partial information feedback. These three hypotheses are the ones that restrict from the general RL framework to the specific Multi-Armed Bandit case (see Figure 1.3).

MAB for OSA, and a short presentation of the history of this research at the SCEE team. Previous works of our SCEE team showed that MAB can be used to model the OSA problem: orthogonal frequency bands (or channels) are models by arms $k \in \{1, \dots, K\}$, and the feedback obtained by the CR-equipped device after sensing the channel k at time t is modeled by a reward of $r(t) \in \{0, 1\}$. Indeed, $r(t) = 1$ indicates that no Primary User was sensed (and thus an uplink message can be sent), while a reward of $r(t) = 0$ indicates that the channel k is busy at time t and no uplink message should be sent. This model was first studied by W. Jouini during his PhD thesis with C. Moy, ten years ago, first with [JEMP09] and later with [JEMP10, JMP12].

Their works were among the first ones to propose to use Reinforcement Learning for Cognitive Radio and the OSA problem, especially the MAB model and the UCB algorithm, along the early works of Q. Zhao and her team, for instance with the works [LZ08, LZ10]. Shortly after in 2014, proof-of-concepts using real-world radio hardware and Software Defined Radio were developed by C. Moy and his student C. Robert [RMZ14, Moy14]. In a second PhD thesis [Mod17], N. Modi studied from 2014 to 2017 the impact of using MAB algorithms to optimize channel selection on the battery life of a wireless device. On the one hand, running a MAB algorithm such as UCB-like algorithms was proven to be useful and can bring significant improvement in terms of successful transmission rates, directly increasing the battery life of the device. On the other hand, classical MAB algorithms tend to switch arms a lot of times, especially in the beginning of the learning process, and this induces a lot of dynamical hardware reconfiguration for the wireless device, as selecting a different channel requires a change in the radio hardware used by the device. Each hardware reconfiguration costs energy for the device, and quickly switching algorithms will lead to a reduction of the battery life. The tradeoff between the two aspects is studied empirically in [DMNM16].

In 2017, C. Moy continued to work on this direction, with a post-doctoral student, S. Darak, leading to publications such as [DNMP16, DMP16]. For example, proof-of-concepts like [KDY+16] have proven the capability of such approaches on real radio signals for OSA, Some analysis on real radio measurements made for HF ionospheric channels have also proven that solutions based on MAB learning is appropriate and solves efficiently this kind of decision-making problems on real-world wireless signals [MGMM+15]. Since 2017, S. Darak and his team at IIIT Delhi have actively worked in the research on cognitive radio using multi-armed bandits, and some of their recent works are illustrated with real-world demo using USRP and the MATLAB/Simulink system [KYDH18, SKHD18, H. 18].

Limitations and specifities of IoT networks. The aforementioned previous works have shown that MAB algorithms can be applied with success for the OSA problem. But if we consider CR-ready devices that cannot perform sensing, such as low-cost and low energy-consumption end-devices designed for the future Internet of Things networks, the MAB model that uses sensing to detect PU in the OSA case can no longer be applied. Moreover, in most cases, the IoT networks use unlicensed bands, and as such there are no longer a distinction between PU and SU. The specificities of IoT networks can be listed as follows, and we refer to [CVZZ16] for more details.

- most IoT networks run in unlicensed bands (no more distinction between PU and SU),
- most IoT devices are very low-cost devices, with low material and computational capacities, meaning that they are not equipped with any sensing capabilities. However, they are equipped with a transceiver as well as with a receiver, and with low-complexity storage and computation capacities,
- a single IoT base station will have to handle a very large number of devices, and cannot send coordination orders to them in order to optimize the network,
- IoT devices have low duty cycles (only a few messages every minute) to very low duty cycles (only a few messages a day).

One could think that in the absence of sensing, Reinforcement Learning is no longer possible, but any real IoT device still receives some information about its environment after some or every transmissions. In most IoT standards an up-link message sent to a base station is usually followed by a down-link message sent back by the base station, to indicate if the up-link message was successfully received and understood. By using this feedback, that consists in an *acknowledgement* (*Ack*) message received shortly after every successful transmission, or in an absence of *Ack* avery a failed transmission, it is possible that an IoT end-device can also use RL algorithms to optimize its communications. To the best of our knowledge, this direction of research has never been studied before the year 2016 and the beginning of this PhD.

1.3 Our contributions

We start by formulating the problem studied in this thesis, and then we develop our approach.

Problematic. If we sum-up the studied problems and summarize them in one question, it could be the following: *"Can we adapt the decision making tools already successfully applied to Cognitive Radio for Opportunistic Spectrum Access to the specific needs of CR for the (future) Internet of Things networks?"* We answer partially to this problematic by the following research steps.

1.3.1 Exploring the literature of MAB algorithms, Part I

Chapter 2. We started by exploring the rich literature of multi-armed bandits, as many different algorithms exist, with lots of variants on the simple problem presented above (see [LS19, Sli19] for surveys). We thus start the first Part I of this thesis with Chapter 2, which presents the MAB model and gives a review of the most important algorithms designed to solve stochastic and stationary MAB problems. In order to clearly understand which algorithms could be adapted to the aforementioned constraints of CR for IoT networks, we were not only interested by the usual measure of performance of any MAB algorithm (its regret, see below in Section 2.3), but also by their empirical performances in terms of computational complexity and storage requirements, both from the point of view of the theoretical analyses and real-world measurements on time and memory footprints.

Chapter 3. Our exploration of the large number of MAB algorithms and models developed in the recent literature has given us the ambition to write a single piece of software allowing anyone to easily implement new models and algorithms, in order to compare the existing ones and empirically explores the performances of newly proposed algorithms. To answer this goal, we produced a Python library of simulation of MAB problems. We wrote the most exhaustive open-source simulation library for MAB problems, called SMPyBandits, which is published online under an open-source licence [Bes19, Bes18] and hosted on `GitHub.com/SMPyBandits`. We present in details its architecture and its features in Chapter 3, along with different examples of its usage. A full documentation is available online, as well as exhaustive instructions to reproduce the experiments used in the rest of this thesis.

Chapter 4. Because there are so many different MAB algorithms, we are also interested by the question of how practitioner can chose the one she will implement, in a given IoT object, in order to let this object adapts robustly in any environment. To answer this question, we present two approaches. The first one is an empirical comparison of the most efficient and well known existing algorithms, in Section 3.3 and 3.4. We confirm that widely used and not too sophisticated algorithms, such as UCB [ACBF02], Thompson sampling [Tho33] and kl-UCB [CGM+13], are the most efficient in terms of regret, and offer a good balance between regret and (time and memory) complexity. The second approach is an online algorithm selection, consisting in aggregating a (finite) set of algorithms and automatically discovering which one performs the most efficiently, against a given problem, with our contribution Aggregator that we detail in Chapter 4. This work on aggregating MAB algorithms was motivated for the OSA case of Cognitive Radio, where many previous research works only considered the UCB_1 algorithm without really justifying this choice. This contribution was presented at the IEEE WCNC conference in Barcelona, Spain, in April 2018.

1.3.2 Our model of IoT network and decentralized MAB, Part II

In the second Part II of this thesis, we start by proposing and studying different models of IoT networks, with simulations and a real-world proof-of-concept, and then we study interesting questions on two mathematical models of MAB that arise from our first contribution.

Chapter 5. We study two models of IoT networks in Chapter 5. The first model considers IoT devices that simply have data to send to a base station, at a regular frequency (but random times), and that use the acknowledgement as feedback, in order to optimize their uplink communications, by accessing more the best channels (*i.e.*, the less occupied by the surrounding traffic). The goal of this first model of applying decentralized RL on the device side is to increase the Quality of Service (QoS) of the application of such IoT network, by reducing the rate of failed transmission. The base station will thus receive more up-link packets from the devices it serves, if they can successfully learn an efficient spectrum access scheme. The second model we propose then considers packet retransmissions, and while the two concepts are similar, the application of an efficient decentralized learning algorithm now results in both an increased battery life for each of the devices, as well as an increase QoS for the entire network.

We present in Section 5.2 a first model of an IoT network, that is made of thousands (or more) of end-devices and a single base station. We consider a time-frequency slotted ALOHA-based protocol, by assuming a perfect synchronization between all devices and the base station. Our IoT network model contains two types of devices: *static devices* that use only one channel (fixed in time), and *dynamic devices* that can choose the channel for each of their transmissions. Static devices form an interfering traffic, which could be generated by devices using other standards as well. Dynamic devices can use RL, and especially MAB algorithms, to optimize their sequential choices of channels, in order to maximize the number of successful up-link transmissions, thanks to the success/failure feedback that is the presence/absence of acknowledgement (*Ack*) received from the base station. We first detail three base-lines, a naive approach and two centralized "oracle" approaches, that are used to evaluate the performance of the considered MAB algorithms (UCB and TS), in terms of the centralized system-wide successful communications rate. We show that our approach is interesting, as these low-cost MAB algorithms have near-optimal performance, even when the proportion of end-devices increases and the interfering traffic from the other devices becomes more and more fluctuating and unpredictable. Even without changing anything on the level of the considered IoT standard, our proposal is just an add-on capability, that can be set-up on each device, on a unit-per-unit basis. This model constitutes the first article written during this thesis, and initiated a collaboration with R. Bonnefoi, another PhD student of our team SCEE. We presented this first work at the EAI CROWNCOM conference in Lisboa, Portugal in September 2017, and obtained the "best paper award" [BBM+17].

Verifying the potential gain of performance brought by these MAB algorithms in numerical simulations was a first step, but it is satisfactory to also verify our results on a real-world proof-of-concept (PoC). In Section 5.2, we detail what we believe to be the first PoC of using decentralized MAB learning in a simplified "IoT" like wireless network. We were able to implement a network with a base station which receives incoming messages from different end-devices and replies with acknowledgements, that are used as feedback by the devices and let them learn an efficient spectrum access scheme. This is again the result of a fruitful collaboration with R. Bonnefoi, and this work was demonstrated during three days at the IEEE ICT conference in Saint-Malo, France, in June 2018 [BBM18], and later presented at the IEEE WCNC conference in Marrakech, Morocco, in April 2019 [BBM19].

We then consider an extension of our first model, following the idea of packet retransmissions, that is at the root of the famous ALOHA protocol [Abr70, Rob75]. The idea is that when a device sends an uplink message but does not receive its *Ack* (before a certain delay), instead of going back to sleep mode and waiting for its next activation (*e.g.*, following the Bernoulli random activation pattern, or waiting for the application to need to send some data), the device will try again to send the same packet, a few times (*e.g.*, $m \leq 10$ times), after some small random waiting times. We propose to also use decision making, that is to also use RL and MAB algorithms, for this other model, at the second layer of the MAC protocol, in order to learn the best channels to use for retransmitting the packets that suffered from a collision at their first transmissions. We present the model in Section 5.4, along with mathematical justifications of the potential of using this second-stage MAB learning for retransmissions, and numerical illustrations. This shows that using learning in a decentralized way on the devices' side is again a powerful idea under this model with retransmissions, and we show as well that the simplest of the different heuristics we compared is efficient, and allows the devices to learn to behave almost optimally. This third contribution concluded our collaboration with R. Bonnefoi, and was presented at the 1st MoTION workshop, during the IEEE WCNC 2019 conference [BBMVM19].

Chapter 6. In the two models presented above, the core concept is to let every dynamic device of a IoT network runs a fully decentralized learning algorithm, each running on its own (random) activation times, in order to optimize the complete system. It means that each of them target its own local objective, which is to maximize its cumulated reward (in a selfish way). It is well known in the game theory literature that playing selfishly can be disastrous for the centralized performance measure: for instance, one can think of the prisoner dilemma or other popular "dilemma" games. So it was quite surprising that our numerical simulations, as well as the realistic PoC, showed that decentralized selfish MAB learning lead to efficient coordination between devices, despite the fact that these IoT objects cannot communicate directly with each other and receive only a collision feedback from the base station they are associated to.

We first tried to analyze the performance of this decentralized algorithm, that we refer to as Selfish, in the model of Section 5.2, but due to the random number of active devices at each time step (*i.e.*, as soon as the probability p of activation is $p < 1$), we were unable to develop a clean analysis. That is why in Chapter 6, we relax the hypothesis of having $M \gg K$ (or even simply $M > K$) devices in a network with K orthogonal wireless channel, equivalent to having $p < 1$, and we consider the case of devices communicating at every time step (*i.e.*, $p = 1$), and so we restrict to only up-to $M \leq K$ devices. The model we studied comes with different variants, depending on the feedback level, and covers both the OSA case (*i.e.*, with sensing information) or the IoT case (*i.e.*, without sensing feedback). The goal was to understand the heuristic of Chapter 5, Selfish, in the simpler framework of multi-player MAB, which has been studied before for the case of OSA, as studied for instance by [LZ10, AMT10, AMTA11]. The OSA case covered by our model is actually slightly different as the one considered by [JEMP10] and other previous works, as our model requires that an *Ack* is sent back by the base station if the transmission was successful, even in the case where sensing information is available. It differs from the previously studied models of applying MAB for OSA, as they only considered synchronized devices, one SU and PU, and thus if the sensing indicates that one channel is free at one time slot, the up-link message sent by the SU device is sure to be successfully received by the base station (in the ideal model), so there is no risk of collision.

On the first hand, in the OSA setting, we failed to obtain positive result for the Selfish policy, as we proved that in some limited settings (*e.g.*, $K = 3$ and $M = 2$), Selfish-UCB can suffer from linear regret with a small probability, and thus suffers from linear mean regret. This result was later confirmed and analyzed by [BP18]. On the other hand, we were able to propose new algorithms for this multi-player bandit problem with sensing information, and we analyzed our proposal MCTopM-kl-UCB, to show that it achieves a finite-time logarithmic regret upper bound, improving over previous state-of-the-art results. Our algorithm also achieves a logarithmic number of collisions and arm switches, and allows a fixed group of M devices to efficiently learn to use the M best channels orthogonally for almost all their up-link communications. This strong theoretical result heavily depends on the presence of sensing feedback, and as such our results are not (yet) applicable to the IoT model without sensing.

Our work on multi-player bandit was presented at the ALT conference in Lanzarote, Spain, in April 2019 [BK18a], and it relaunched the research on this model, as a few independent teams started to work on different questions shortly after its publication. Since last year, it was proven that the "no sensing" case (*i.e.*, IoT) is essentially not harder that the "sensing" (*i.e.*, OSA) case, from a theoretical point-of-view, by E. Boursier and V. Perchet [BP18]. Even if their approach is costly empirically, the result is surprising and very interesting: they showed that even under the strong hypothesis of no direct communication between players and no central coordination, the players can use the collision feedback (*i.e.*, receiving or not an *Ack*) as an indirect way to communicate bits of information with each other. This idea requires that devices coordinate and split the time steps between "listening" and "talking" phases, and the authors denote it the "communication trick". As they showed, after a long enough initialization period, the M players have elected a "master" player and $M-1$ "followers" players (with high probability), and thanks to this communication trick, the problem can boil down to a centralized multiple-play (MP) MAB, which can be solved efficiently with the MP generalization of kl-UCB [LKC16] or Thompson sampling [KHN15].

We discuss more in details this interesting follow-up of our article [BP18], along with other recent research works such as [LM18], in Section 6.7. Moreover, the case of different means of arms for the different players was studied in [BL18, KM19], and this generalization of our model is very interested for CR applications, as the mean of an arm is the average quality (*e.g.*, availability) of a radio channel, and as such it has no reason to be uniform among end-devices (*i.e.*, players). Finally the adversarial case of multi-player bandit has started to be studied, independently by [BV19] and [ALK19].

As explained above, the model of Chapter 5 was found intractable to analyze, mainly because we consider an IoT network with many end-devices, all following random activation patterns. The difficulty does not reside in the fact that we are trying to analyze MAB algorithms, designed to tackle stationary problems, on a non-stationary problem, but rather that we are analyzing algorithms which are all playing in different (random) subsets of the global synchronized time. Generalizing to different activation probabilities would lead to a model even harder to analyze, and enforcing at most $M \leq K$ devices is not really realistic for IoT networks, but leads to the model of Chapter 6.

On the first hand, if the activation pattern of all devices could be fixed, for instance based on a centralized affectation of the devices to different time slots, then the model of multi-player bandit from Chapter 6 can be used to let all groups of $M \leq K$ devices learn an optimal orthogonal affectation in the K channels (*e.g.*, groups can be the set of devices that all emit at the same time). However, even if we continue to assume a synchronized time in all the rest of the thesis, it is hard to argue that this hypothesis of a centralized time schedule for the end-devices can be realistic, because we study the case of decentralized learning for OSA and IoT precisely in order to avoid any central control of the devices by the base station, as we explained above. On the other hand, another way to relax the non-stationary hypothesis is to take the point-of-view of one device, like in the OSA case mentioned above, where the focus is on one SU surrounded by many PU having stationary behaviors. If we focus on one IoT device, its environment (*i.e.*, the surrounding devices) is non-stationary, meaning that its average properties can fluctuate with time, but we could assume a certain structure on this non-stationarity. For instance, enforcing stationarity on some time intervals leads to the last contribution and chapter of this thesis.

Chapter 7. In order to also understand precisely how MAB algorithms behave under non-stationary environment, we studied the literature on adversarial and on non-stationary MAB, for both cases of slowly-varying and abruptly-changing environments. We begin our last Chapter 7 by reviewing the existing works, and we chose to focus on the piece-wise stationary problem, meaning that the underlying bandit problem is stationary on some intervals, separated by change-points located at unknown times. Assuming the environment to be piece-wise stationary indeed makes sense for applications to wireless networks, where a change-point can for instance correspond to the arrival of a new group of end-devices in the network. Examples of such situation can be a new company arriving on the market in one city (*e.g.*, the Linky counters being set-up nowadays in France), or the neighbor farmer installing sensors on his own herd of cows, etc. Two main families of algorithms have been proposed for the piece-wise stationary problem, that usually consist in combining an efficient policy designed for the stationary MAB problem (*e.g.*, Thompson Sampling or kl-UCB) and a way to adapt to changes in the arms distributions. Passively adaptive policies use a window of a fixed or evolving size [GM11], or a discount factor, in order to forget about the past observations [KS06, GGCA11], while actively adaptive policies use a statistical test to detect the change-points [MS13, AF15]. As the recent literature showed that the later approach is usually more competitive, and usually obtains better results from both empirical and theoretical aspects, we chose to develop our own actively adaptive algorithm.

Following two recent previous works [LLS18, CZKX19], we combine an efficient index policy (kl-UCB) and an efficient change-point detection test, under the assumption of bounded rewards. We build on very recent results on the Generalized Likelihood Ratio test (GLRT) for Gaussian and sub-Gaussian variables [Mai19], and we instead focus on bounded rewards and Bernoulli distributions. Bounded rewards are indeed usually more appropriate for Cognitive Radio applications. Using the fact that bounded variables in $[0, 1]$ are not only $1/4$ sub-Gaussian but also sub-Bernoulli, we prove the first finite-time guarantees for the GLRT for bounded variables. We first show finite-time bounds on both the false alarm probability of our test, and its detection delay, under mild assumption on the lengths of the stationary sequences. Our algorithm, denoted GLR-klUCB, is then presented in two variants, whether change-points are local (*i.e.*, only one arm mean changes at each change-point) or global (*i.e.*, possibly all the arm means change at a time). We prove that by combining the asymptotically optimal policy kl-UCB, designed for stationary problems, and our new analysis of the GLRT for bounded rewards, we obtain state-of-the-art guarantees on the regret of our proposed algorithm. Like our competitors, we make the some hypothesis on the length of the stationary intervals, and the best regret upper-bound is obtained when the algorithm knows before-hand the horizon and the number of change-points, but our algorithm does not need to know any other knowledge of the problem difficulty to be tuned optimally. The performance of GLR-klUCB is also illustrated on numerical experiments on synthetic data, where it is shown to outperforms all the passively adaptive policies as well as the previous actively adaptive policies. This last chapter is based on our last work, since October 2018, and it first lead to a publication at the GRETSI conference in Lille, France, in August 2019 [BK19a]. Our work also lead to the long version article [BK19b], that will be partially rewritten and completed with more recent results and submitted soon to a journal (probably the Journal of Machine Learning Research) in fall 2019.

Conclusion. We conclude this thesis in Chapter 8, to give some perspectives and directions about possible future works. Most of the contributions that we present in this thesis call for more developments, on both directions of providing more mathematical analysis and implementing new proof-of-concepts, using real wireless hardware and realistic Internet of Things networks.

This manuscript ends with the appendixes, first with a short overview of another contribution, in Appendix A we discuss about our work on doubling-trick, in order to remove the need for an algorithm to know the horizon of the bandit game before starting to play it. The rest of the appendix contains a list of abbreviations and notations, then lists of figures, algorithms, code samples and tables, and finally the list of the bibliographical references.

1.3.3 Summary of the contributions

To summarize the main contributions of this thesis, we can list the following points:

- We developed SMPyBandits, the most exhaustive open-source simulation library for MAB problems, that we published on-line under an open-source licence [Bes19, Bes18]. We present it in details in Chapter 3.
- We present in Chapter 4 an algorithm called Aggregator for aggregation of algorithms as an online solution to the algorithm selection problem, and numerical simulations to illustrate that it achieves state-of-the-art empirical performances [BKM18].
- We propose different models for IoT networks, in Chapter 5, where end-devices with cognitive radio capabilities can implement MAB algorithms on their side, to automatically increase their battery life and allow more devices to use the same network while maintaining a high Quality of Service [BBM$^+$17, BBM19, BBMVM19, MB19].
- We implemented a proof-of-concept of the aforementioned model [BBM18], and we present it in details in Section 5.3. We made a video showcasing our demonstration, hosted at `youtu.be/HospLNQhcMk`.
- We present the multi-players bandit model, for which we introduce three variants, in Chapter 6. For the case with sensing information, we propose two new algorithms, and we give an analysis for our algorithm MCTopM to show it is order-optimal, as well as extensive numerical experiments to demonstrate its excellent performance in comparison with the rest of the literature. Our work [BK18a] also gave a new impulse on research on multi-players bandits, as some recent research works heavily built up on our results.
- We also present the piece-wise stationary MAB model, in Chapter 7, and a detailed literature review of the research on non-stationary MAB [BK19b, BK19a]. Following two recent works, we propose a new actively adaptive algorithm for the piece-wise stationary problem, GLR-klUCB, that achieves state-of-the-art performance.
- The last contribution of this thesis is a literature review of the possible use cases of the sequentially doubling horizon trick technique for MAB problems, and a unified and more generic analysis of two families of doubling tricks. This lead to the article [BK18b], which is quickly presented in Appendix A.

1.4 Organization of the thesis

The reading order of the manuscript can be any top-down path between the Introduction in the current Chapter 1, and the Conclusion in the last Chapter 8. As the graph in Figure 1.4 shows, the thesis is organized in two parts.

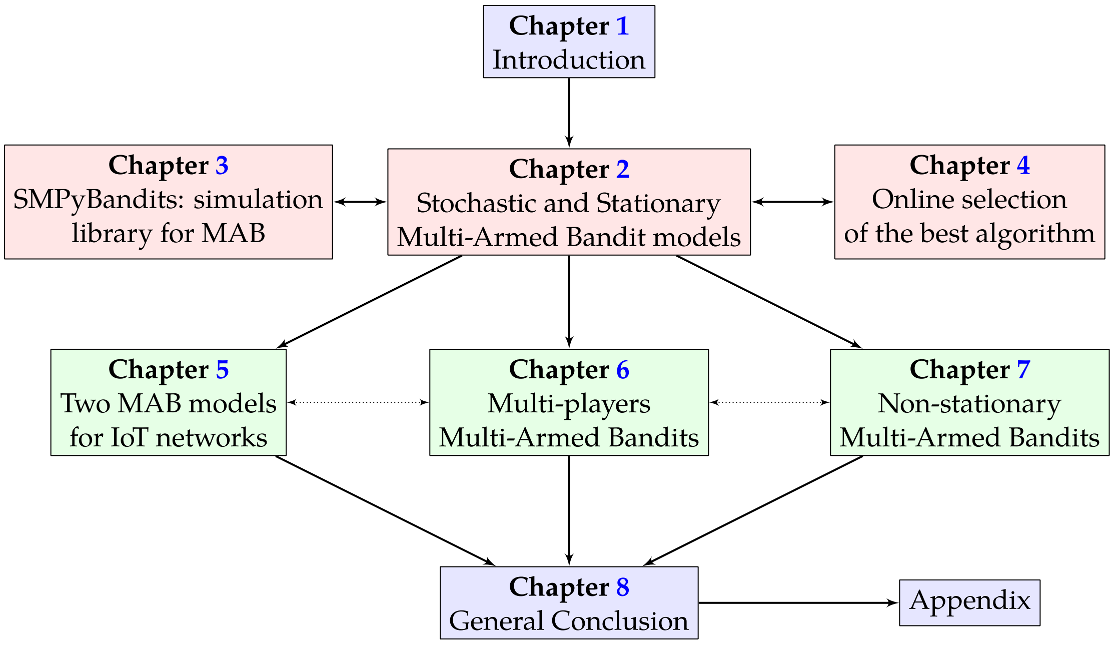

Chapter 1
Introduction
Chapter 3
SMPyBandits: simulation library for MAB
Chapter 2
Stochastic and Stationary Multi-Armed Bandit models
Chapter 4
Online selection of the best algorithm
Chapter 5
Two MAB models for IoT networks
Chapter 6
Multi-players Multi-Armed Bandits
Chapter 7
Non-stationary Multi-Armed Bandits
Chapter 8
General Conclusion
Appendix

Figure 1.4 – A reading map of the thesis. Any top-down path containing Chapter 1, Chapter 2, at least one of the three Chapters 5, 6 and 7, and the Conclusion is a self contained way to read this thesis.

First, in Part I (second line), we start by the next Chapter 2, required for the rest of the document, as we introduce the MAB models and the notations used in this thesis. Conversely, even if Chapters 2, 6 and 7 use numerical simulations based on our simulation library SMPyBandits, the Chapter 3 where we present it is not required to understand them. We conclude this part with Chapter 4, which is also not mandatory for the rest of this thesis, and which details one of the first contributions of this thesis, a new algorithm for online MAB algorithms selection.

Then, the second Part II contain three chapters (third line), that are included in both the logical and chronological orders, but can be read almost independently. Chapter 5 starts by presenting different models of IoT networks where we show that MAB algorithms can be used with success. Our two models are interesting and close to reality, but they appeared to be too general to propose a mathematical analysis of the good empirical performance of the considered solutions. For this reason, we weaken the models for the rest of the document, and both Chapters 6 and 7 studies an intermediate model, lying between the stationary single-player MAB model from Chapter 2 and the IoT networks models from Chapter 5.

1.5 List of publications

We conclude this chapter with a list of works published during this PhD. All the following works are published entirely and freely, on the HAL platform (see the `HAL.Archives-Ouvertes.fr` website). The complete list can be found on `CV.Archives-Ouvertes.fr/lilian-besson`. The following publications are ordered historically, from the most recent to the oldest.

Publications in international conferences with proceedings

- *Decentralized Spectrum Learning for IoT Wireless Networks Collision Mitigation*, by Christophe Moy & **Lilian Besson**. 1st International ISIoT workshop, at *Conference on Distributed Computing in Sensor Systems*, Santorini, Greece, May 2019. See Section 5.3. [MB19]

  ↪ *Note that we are already working on an extended version that will be submitted to a journal on Machine Learning for Wireless Communications, in July* 2019.

- *Upper-Confidence Bound for Channel Selection in LPWA Networks with Retransmissions*, by Rémi Bonnefoi, **Lilian Besson**, J. Manco-Vasquez & Christophe Moy. 1st International MOTIoN workshop, at *IEEE WCNC*, Marrakech, Morocco, April 2019. See Section 5.4. [BBMVM19]

- *GNU Radio Implementation of MALIN: "Multi-Armed bandits Learning for Internet-of-things Networks"*, by **Lilian Besson**, Rémi Bonnefoi & Christophe Moy. *Wireless Communication and Networks Conference*, Marrakech, Morocco, April 2019, See Section 5.3. [BBM19]

- *Multi-Player Bandits Revisited*, by **Lilian Besson** & Emilie Kaufmann. *Algorithmic Learning Theory*, Lanzarote, Spain, April 2018, See Chapter 6. [BK18a]

- *Aggregation of Multi-Armed Bandits learning algorithms for Opportunistic Spectrum Access*, by **Lilian Besson**, Emilie Kaufmann & Christophe Moy. *Wireless Communication and Networks Conference*, Barcelona, Spain, April 2018, See Chapter 4. [BKM18]

- *Multi-Armed Bandit Learning in IoT Networks and non-stationary settings*, by Rémi Bonnefoi, **Lilian Besson**, Christophe Moy, Emilie Kaufmann & J. Palicot. *Conference on Cognitive Radio Oriented Wireless Networks*, Lisboa, Portugal, Septembre 2017, **Best Paper Award**. See Section 5.2. [BBM$^+$17]

Demonstration in international conferences

- *MALIN: "Multi-Arm bandit Learning for Iot Networks" with GRC: A TestBed Implementation and Demonstration that Learning Helps*, by **Lilian Besson**, Rémi Bonnefoi, Christophe Moy. Demonstration presented at *International Conference on Communication*, Saint-Malo, France in June 2018. See YouTu.be/HospLNQhcMk for a 6-minutes presentation video. See Section 5.3. [BBM18]

French language conference with proceedings

- *Analyse non asymptotique d'un test séquentiel de détection de ruptures et application aux bandits non stationnaires* (in French), by **Lilian Besson** & Emilie Kaufmann, GRETSI 2019, August 2019, See Chapter 7. [BK19a]

In progress works waiting for a new submission

- *The Generalized Likelihood Ratio Test meets klUCB: an Improved Algorithm for Piece-Wise Non-Stationary Bandits*, by **Lilian Besson** & Emilie Kaufmann, February 2019.
  See Chapter 7. Preprint at `HAL.Inria.fr/hal-02006471`. [BK19b]

  $\hookrightarrow$ *Note that we are already working on an extended version that will be submitted to a journal on Machine Learning and Statistical Learning, in autumn 2019.*

- *SMPyBandits: an Open-Source Research Framework for Single and Multi-Players Multi-Arms Bandits (MAB) Algorithms in Python*, by **Lilian Besson**, active development between October 2016 and March 2019, `HAL.Inria.fr/hal-01840022`. It currently consists in about 40000 lines of code, hosted on `GitHub.com/SMPyBandits`, and a complete documentation on `SMPyBandits.rtfd.io` and `SMPyBandits.GitHub.io`. See Chapter 3. [Bes19, Bes18]

- *What Doubling-Trick Can and Can't Do for Multi-Armed Bandits*, by **Lilian Besson** & Emilie Kaufmann, September 2018.
  See Appendix A. Preprint at `HAL.Inria.fr/hal-01736357`. [BK18b]

Part I

Introduction to and Simulations of Multi-Armed Bandits Models

Chapter 2

The Stochastic Multi-Armed Bandits Model

In this chapter, we present the common base of all the more sophisticated models studied in this thesis: the stochastic multi-armed bandit (MAB) model, restricting to the single-player and stationary stochastic case. We motivate this model and discuss its application to and beyond cognitive radio, and we review existing MAB algorithms. Some of which are crucially used in the rest of this thesis, like UCB, kl-UCB and Thompson sampling. We focus on decision-making models with a finite number of resources, called arms, and on stochastic models, where an arm is associated with a one-dimensional distribution. We formalize this MAB model, then we quote some of its important applications. We define the notion of regret of an algorithm, and quote important results from the literature. A short review of the most important families of MAB algorithms is then given.

Contents

2.1 The stochastic Multi-Armed Bandit model

Multi-Armed Bandits (MAB) models were introduced by Thompson as early as in 1933 [Tho33], and later studied from the 1950s by Robbins [Rob52] and others. This family of models was first proposed for clinical trials, and later applied to a wide range of different problems. Their name refers to one-armed bandits found in casinos, as shown in Figure 2.2 below. We now formally introduce the model.

A MAB refers to a decision-making game, where a *player* has to sequentially select one action in a usually finite set of actions, and only receives a (random) feedback about the selected action, also called a *reward*. **We always consider** $K \in \mathbb{N}, K \geq 2$ **arms, and discrete times** $t \in \mathbb{N}^*$. We denote $[K]$ the set $\{1, \dots, K\}$. Each arm $k \in [K]$ is associated with a stream of rewards $(Y_{k,t})_{t \in \mathbb{N}^*}$, and when the player decides at time t to play (or pull) the arm $A(t) \in [K]$, she receives a reward $r(t) \doteq Y_{A(t),t}$. She keeps playing an arm and observing a reward iteratively, and so on for a finite number of steps t (or rounds), from $t = 1$ until $t = T$ for an *horizon* T. A commonly studied goal for the player is to maximize its sum of received rewards, $\sum_{t=1}^{T} r(t)$ (or its *mean* in stochastic and stationary models).

The difficulty of this decision-making game comes from balancing the trade-off between *exploration*, because the player has to observe as much as possible all the arms to get more knowledge about the distributions of their rewards, and *exploitation*, because she also has to select as much as possible the best arm. The MAB model is a famous example of a **reinforcement learning** model, where the decision-making process has to adapt to an unknown environment using noisy observations and a discrete time, alternating between decisions and observations. We illustrate this cycle in Figure 2.1 below. For more details on reinforcement learning, in particular about more generic models, we refer to the famous book [SB18].

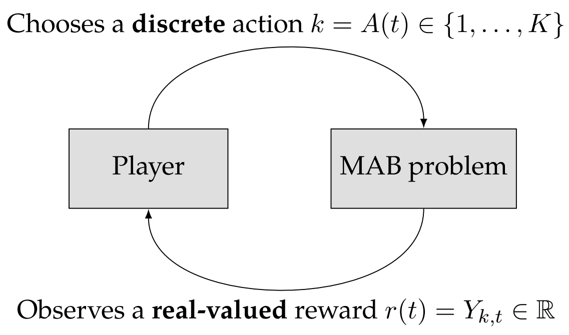

Chooses a **discrete** action $k = A(t) \in \{1, \dots, K\}$
Player
MAB problem
Observes a **real-valued** reward $r(t) = Y_{k,t} \in \mathbb{R}$

Figure 2.1 – Reinforcement learning cycle in a MAB model, for time steps $t = 1, \dots, T$.

This quantity $\sum_{t=1}^{T} r(t)$ can indeed be random, and it usually depends on two aspects. First, the rewards may depend on the randomness of the environment, *i.e.*, the unknown and unpredictable values $r(t) = Y_{A(t),t}$. Then this quantity also depends on the player's decision-making policy, *i.e.*, the choices $A(t)$, that are based on (all) the *past* observations, *i.e.*, $A(1), Y_{A(1),1}, \dots, A(t-1), Y_{A(t-1),t-1}$, and possibly on an external source of randomness. Without loss of generality, it can be given by a sequence $U_0, U_1, \dots$ of *i.i.d.* random values, uniformly distributed on $[0,1]$, and that has to be independent from the samples $Y_{k,t}$. External randomness is used in most MAB algorithms, *e.g.*, to break ties uniformly in an index policy.

About notations. We use in this thesis the usual notations from the MAB literature, and a summary is included in the Nomenclature, included in the Appendix in Part III (Page 274).

Main references for the curious reader. For more details and a more formal introduction to multi-armed bandits, we suggest to the interested reader to work on a very recent text-book by Slivkins [Sli19]. Another excellent but reasonably short survey is the book by Bubeck and Cesa-Bianchi [BCB12], while the more recent book by Lattimore and Szepesvári [LS19] is the most complete resource about bandit algorithms. Finally, we recommend [BR19] for a short but good survey on applications of MAB.

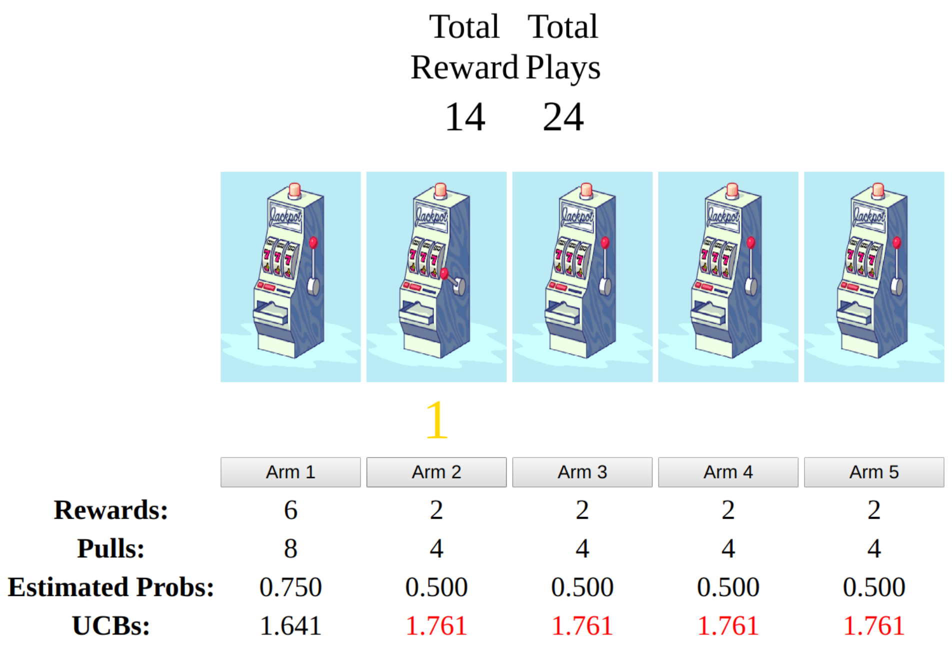
2.1.1 Finite arms and stochastic rewards

In all this thesis, we focus on the model with finite arms, that is $A(t) \in \{1, \dots, K\}$ for a fixed and known number of arms $K \in \mathbb{N}, K \geq 2$. We focus on the **stochastic** MAB, in which the reward stream are drawn from some (unknown) distributions. We also restrict to *stochastic* rewards, that is we associate a real-valued distribution to each arm, denoted ν_k for arm $k \in [K]$. Rewards are stationary, meaning that $(Y_{k,t})_{t\in\mathbb{N}^*}$ is independent and identically distributed (*i.i.d.*), and $Y_{k,t} \sim \nu_k$ for any $t \geq 1$. We only consider binary or real-valued rewards, *i.e.*, $Y_{k,t} \in \mathbb{R}$.

The most common objective for the player is to maximize its expected rewards $\mathbb{E}[\sum_{t=1}^{T} r(t)] = \sum_{t=1}^{T} \mathbb{E}[Y_{A(t),t}]$. The expectation $\mathbb{E}[\bullet]$ is taken on the rewards $(Y_{k,t})_{k,t}$, as well as on the external random variables $(U_s)_s$ (*i.e.*, on the algorithm's decisions). Other commonly studied objectives include best-arm identification (BAI) [ABM10].

An interactive demo to discover the MAB problem (for the novice). If you are discovering here the concept of bandits, I would like to recommend you to go online and play a few times with an interactive demonstration. On this demo, you will be facing a MAB problem with $K = 5$ arms, and you have $T = 100$ decisions to make. The demo is hosted on my website, at `perso.crans.org/besson/phd/MAB_interactive_demo/`, and it is illustrated below.

The webpage looks like Figure 2.2 below. The arms follow Bernoulli distributions, *i.e.*, $\nu_k = \mathcal{B}(\mu_k)$, of unknown means $\mu_k \in [0,1]$, and your goal in this interactive demonstration is to obtain the highest possible cumulated reward in $T = 100$ steps, *i.e.*, to maximize $\sum_{t=1}^{100} r(t)$. Your decisions are made sequentially: at time t, you pick one of the arms, $A(t) \in \{1,2,3,4,5\}$, then the demo shows the random reward obtained from this (virtual) casino machine (in yellow), *i.e.*, the binary value $r(t) \in \{0,1\}$, that is sampled *i.i.d.* from ν_k. The UI of the demo



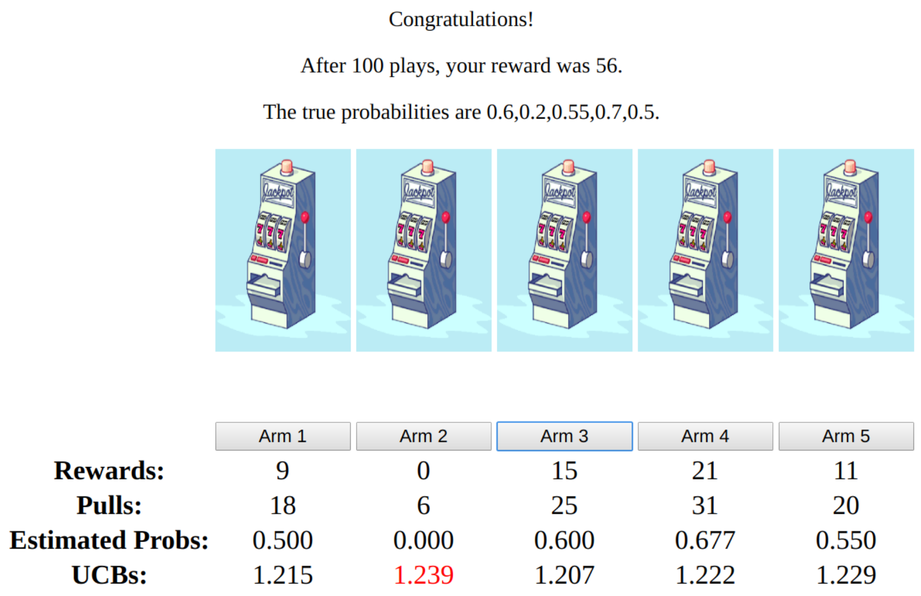
Total Reward	Total Plays
14	24

	Arm 1	Arm 2	Arm 3	Arm 4	Arm 5
Rewards:	6	2	2	2	2
Pulls:	8	4	4	4	4
Estimated Probs:	0.750	0.500	0.500	0.500	0.500
UCBs:	1.641	1.761	1.761	1.761	1.761

Figure 2.2 – Screenshot of the demonstration available online on my website, for a current step of $t = 24$.

also shows the current value of t ("total plays") and $\sum_{s=1}^{t} r(s)$ the "total reward". For each arm, we show the sum of rewards obtained from that arm, *i.e.*, $X_k(t) \doteq \sum_{s=1}^{t} r(s)\mathbb{1}(A(s) = k)$, in the "Rewards" line, and the number of pulls of that arm, *i.e.*, $N_k(t) \doteq \sum_{s=1}^{t} \mathbb{1}(A(s) = k)$ in the "Pulls" line. The demo also shows the estimated probability of each arm, that is $\widehat{\mu_k}(t) \doteq X_k(t)/N_k(t)$ (when $N_k(t) > 0$), in the "Estimated Probs" line.

In the first Figure 2.2, the current state of the game is shown at time $t = 24$. At this step, the player has collected a sum of rewards of 14, by observing $X(t) = [6, 2, 2, 2, 2]$ rewards of value 1 in the $K = 5$ different arms. Arms were sampled $N(t) = [8, 4, 4, 4, 4]$ times, meaning that the value 0 was seen respectively $[2, 2, 2, 2, 2]$ times, and currently arm 1 appears to be the best one. The true means of the arms are $\boldsymbol{\mu} = [0.6, 0.2, 0.55, 0.7, 0.5]$, and (much) more samples are needed before the player can accurately identify arm 4 as the best arm. In the second Figure 2.3 below, we display the result of an example of run, when the player was following the UCB$_1$ algorithm from [ACBF02] (we present it below in Section 2.4.2). After $T = 100$ steps, the player obtained a cumulated reward of 56, by playing mostly arms 4, 3, 5, 1, 0 (in decreasing order of number of plays). The empirical means $\widehat{\mu_k}(T)$ correctly identify the best arm (arm 4), but do not correctly rank the arms as arms 1 and 3 obtained means of $\widehat{\mu_1}(T) = 0.5 < \widehat{\mu_3}(T) = 0.6$ but the true means are $\mu_3 = 0.55 < \mu_1 = 0.6$. Other examples of such results, for different algorithms, and $T = 100$ or $T = 10000$, are given in Section 2.4.5.



Congratulations!

After 100 plays, your reward was 56.

The true probabilities are 0.6,0.2,0.55,0.7,0.5.

	Arm 1	Arm 2	Arm 3	Arm 4	Arm 5
Rewards:	9	0	15	21	11
Pulls:	18	6	25	31	20
Estimated Probs:	0.500	0.000	0.600	0.677	0.550
UCBs:	1.215	1.239	1.207	1.222	1.229

Figure 2.3 – Screenshot of the demonstration, at the end of the game after $T = 100$ steps, where the player suffers from an empirical regret of $R_T^{\mathrm{UCB}_1} = \mu^* T - 56 = 0.7T - 56 = 14$ after following the UCB_1 algorithm. You can try it on `perso.crans.org/besson/phd/MAB_interactive_demo/`

Decisions can (and should) depend on the past observations. A *bandit algorithm* $\mathcal{A}$ is also referred to as a strategy or a policy. The algorithm $\mathcal{A}$ selects an arm $A(t)$ at time t, possibly by using the past observations and the past external randomness. As shown below, being oblivious to the past yields very poor performance (cf. the pure exploration policy in Section 2.4.1), so efficient policies indeed depend on the successive feedbacks. More formally, an algorithm can be defined as a sequence of *measurable* functions $(\mathcal{A}_t)_{t\geq 1}$, where $\mathcal{A}_t$ maps the past observations $O_t \doteq (U_0, Y_{A(1),1}, U_1, \dots, Y_{A(t-1),t-1}, U_{t-1})$ to an arm $\mathcal{A}_t(O_t) \doteq A(t) \in [K]$ (we remind that we denote $r(s) = Y_{A(s),s}$ the s-th reward). The initial information is reduced to $O_1 = (U_0)$, and the first decision is $A(1) = \mathcal{A}_1(O_1)$. Usually, most algorithms starts by selecting $A(1) = 1, \dots, A(K) = K$ (or a permutation of the K arms) in the K first steps $t = 1, \dots, K$ (*e.g.*, Algorithm 2.2). An algorithm is said to be *deterministic* if it does not depend on the external randomness $U_0, U_1, \dots$, but in this thesis **we only use non-deterministic algorithms** (in particular, index policies need U_t to break ties in the $\arg\max$, see Algorithm 2.2 below).

2.1.2 Common assumptions on the reward distributions

In the example above, we consider Bernoulli distributions, but other real-valued distributions have been studied in the literature. From now on and until the last chapter of this thesis, **we only consider stochastic rewards**. The piece-wise stationary model is studied in Chapter 7. **We also focus only on real-valued rewards**, meaning that $Y_{k,t} \in \mathbb{R}$ for all arm k and time t.

An important hypothesis is whether rewards are bounded or not, and whether the player knows if they are bounded or not before starting the bandit game. Moreover, if rewards are known to be bounded, let say in an interval $[a, b]$, another important hypothesis is whether the player knows the values of a and b or not. Intuitively, the bandit game is easier if the player knows the support of the distributions, and we restrict to this case in all the thesis, *i.e.*, *a, b are always supposed to be known*. Most of the algorithms proposed in the literature follow this hypothesis as well. The mostly used infinitely supported distributions are Gaussian, exponential and Poisson, while in the literature, the Bernoulli distribution is the most common case of finitely supported distributions. Continuous-valued distributions with finite support also included truncated versions of infinitely supported distributions, in particular *truncated* Gaussian are used for numerical experiments in lots of research articles.

The normalization trick. If the player knows that reward are bounded in an interval $[a, b]$, and if she knows a and b (for $a < b$), then with no loss of generality we can restrict to the interval $[0, 1]$, as if $r \in [a, b]$, the player can instead consider the normalized reward $r' = \frac{r-a}{b-a}$ that lies in $[0, 1]$. Note that this "normalization trick" is implemented for any policy in our library SMPyBandits, with the `lower` and `amplitude` optional arguments, respectively representing a the lower-bound on rewards and $b - a$ the length of the interval of possible values of rewards.

One-dimensional exponential family. In the literature, parametric assumptions on the rewards are sometimes considered, typically the assumption that rewards belong to a real-valued distribution lying in an exponential families. One-dimensional exponential families include Bernoulli and Poisson distributions, as well as Gaussian distributions with a fixed variance. Our main interest is Bernoulli distributions, but we prefer to present the more general notations of exponential families in one dimension.

Given a measure λ, that is usually the natural Lebesgue measure on $\mathbb{R}$, an exponential family of probability distributions is defined as the distributions whose density (relative to λ) can be written as $\mathbb{P}_\eta(x|\lambda) = h(x) \exp(\eta x - A(\eta))$, for a parameter vector η (the canonical parameter), and a function h. The cumulant function $A(\eta)$ is entirely determined by η and h, as $A(\eta) = \ln\left(\int h(x) \exp(\eta x) \lambda(\mathrm{d}x)\right)$. The natural parameter space is the set of values of η such that this integral $A(\eta)$ is finite, and usually the literature focusses on regular and minimal exponential families (when the natural parameter space is an non-empty open set in $\mathbb{R}$).

We mention two important results on exponential families:

1. *A distribution in such family is entirely characterized by its parameter η.* And the mapping $\eta \mapsto \mathbb{E}_\eta[X]$ is one-to-one, if $\mathbb{E}_\eta$ is the expectation under the probabilistic model $\mathbb{P}_\eta$. That is why one-dimensional distributions are entirely characterized by their mean, $\mu = \mathbb{E}_\eta[X]$.

2. A second important result is a simplified form for the *Kullback-Leibler divergence* [KL51], for two distributions lying in the same exponential family. The KL divergence is also called the *relative entropy*, and it is a measure of how one probability distribution is different from a second, reference probability distribution.

Definition 2.1. *The* Kullback-Leibler divergence *between distributions with densities d_1 and d_2, wrt to λ, is defined as*

$$\mathrm{KL}\left(d_1, d_2\right) \doteq \int d_1 \ln\left(\frac{d_2}{d_1}\right) \lambda(\mathrm{d}x) = \mathbb{E}_{d_1}\left[\ln\left(\frac{d_2}{d_1}\right)\right] \in \mathbb{R} \cup \{\pm\infty\}. \tag{2.1}$$

Following this Definition 2.1 for two distributions $d_1 = \mathbb{P}(x|\eta_1)$ and $d_2 = \mathbb{P}(x|\eta_2)$, we can use a shorter notation and write $\mathrm{KL}(\eta_1, \eta_2) \doteq \mathrm{KL}(\mathbb{P}(x|\eta_1), \mathbb{P}(x|\eta_2))$, which can be simplified to use only the two parameters η_1 and η_2 of d_1 and d_2, and $\mu_1 = \mathbb{E}_{\eta_1}[X]$ the mean of distribution d_1, $\mathrm{KL}\left(\eta_1, \eta_2\right) = \mathbb{E}_{d_1}[(\eta_1 - \eta_2)X] - A(\eta_1) + A(\eta_2) = (\eta_1 - \eta_2)\mu_1 - A(\eta_1) + A(\eta_2)$. Without diving more in the details of exponential families, it is interesting to illustrate this definition and the notations with two important examples:

- **Bernoulli distributions** can be seen as an exponential family with $h(x) = 1$, and for a Bernoulli distribution of mean $\mu \in [0, 1]$, denoted $B(\mu)$, the parameter is $\eta = \mu/(1-\mu)$, giving $A(\eta) = \ln(1 + \mathrm{e}^{\eta})$ (with the limit behavior $\eta = +\infty$ if $\mu = 0$).

  The KL divergence between $B(x)$ and $B(y)$, of parameters η, η' is given by $\mathrm{kl}(x, y) \doteq \mathrm{KL}_B(\eta, \eta') = x\ln(x/y) + (1-x)\ln((1-x)/(1-y))$, and it is also called the *relative binary entropy*. It satisfies $\mathrm{kl}(x, y) \geq d_{1/4}(x, y)$ (Pinsker's inequality).

- **Gaussian distributions of a known variance** are a one-dimensional exponential family, and in this thesis we do not consider Gaussian with an unknown variance. For a variance of σ^2, the family uses $h(x) = 1/\sqrt{2\pi\sigma^2}\exp(-x^2/(2\sigma^2))$, and for a Gaussian distribution with mean $\mu \in \mathbb{R}$, denoted $\mathcal{N}(\mu, \sigma^2)$, the parameter is $\eta = \mu/\sigma^2$, giving $A(\eta) = \mu^2/(2\sigma^2)$.

  The KL divergence between $\mathcal{N}(x, \sigma^2)$ and $\mathcal{N}(x, \sigma^2)$, of parameters η, η' is given by $d_{\sigma^2}(x, y) \doteq \mathrm{KL}_{\mathcal{N},\sigma^2}(\eta, \eta') = (x-y)^2/(2\sigma^2)$ (see Chapter 8 of [Jor10]).

Hypotheses in this thesis. In the rest of this thesis, **we only consider bounded rewards in the mathematical developments**, and we mostly focus on Bernoulli distributions, because they are usually the most relevant choice for the considered applications. In Chapter 5, we only study models with binary rewards. Then, when we study multi-players bandits in Chapter 6, we explain that restricting to the Bernoulli case is interesting and not restrictive, as it is actually the hardest case (since continuous distributions has a null mass on $Y_{k,t} = 0$, they yield a much simpler problem as the sensing/no sensing distinction no longer makes sense), but this work can be extended to any one-dimensional exponential families. Finally, in Chapter 7 we analyze our proposed algorithm for bounded distributions, without restricting to Bernoulli distributions, but we use the fact that bounded distributions on $[0, 1]$ are sub-Bernoulli, and we use the Bernoulli Kullback-Leibler divergence kl in our analysis.

Sub-Gaussian and sub-Bernoulli distributions. A lot of research works considers rewards distributions that are not Gaussian but sub-Gaussian, meaning distributions whose moment generating function is dominated by that of a Gaussian distribution with the same mean (and a known variance). For instance, bounded distributions on $[0, 1]$ are known to be $1/4$ sub-Gaussian, and this fact is used for instance in [Mai19]. In Chapter 7, we instead consider sub-Bernoulli distributions, formally introduced in Definition 7.1. Such hypothesis is also proposed for other families of distributions, for instance the recent article [KT19] analyses their Follow-the-Perturbed-Leader algorithm for perturbations following any sub-Weibull distribution (sub-Weibull distributions generalize both sub-Gaussian and sub-Exponential distributions).

About Markov models. Finally, we note that Markov bandit models have also been considered for Cognitive Radio applications [LZ08]. They are not studied in this thesis, even though we did implemented them in SMPyBandits, They were introduced in the 1980s, by Whittle in [Whi88] and Anantharam and others in [AVW87b]. A Markov MAB model maps an arm to a Markov chain [Nor98], instead of a distribution, and thus they are no longer stochastic nor stationary. Such Markov models come in two flavors: rested or restless. For K arms, each Markov chain has a finite number of states s, each corresponding to a (constant) reward that the player obtains if she selects this arm while its Markov chain is in state s. Rested Markov models means that only the state of the selected arm's Markov chain can change, following its Markov transition matrix. Restless models remove this hypothesis, making them harder to track and solve. Such models were less studied than stationary or adversarial models, but some interesting works applied Markov models or CR applications in the last 10 years For instance, [MGMM$^+$15] proposed a CR model mixing MAB and Hidden Markov Models (HMM), solved by a mixed policy called UCB-HMM. A curious reader about Markov chains could start to read Chapter 3 of [LS19] (Section 3.2) and refer to [Nor98].

2.2 Applications of stochastic MAB

The blooming success of the research on multi-armed bandits is easily explained by the different spectrum of applications of MAB models to real-world discrete-time decision making problems. This research field has been very active since the years 2010s, but it started as early as 1933 with [Tho33], and was active since the 1980s and the seminal works by Lai and Robbins

[LR85] and by Anantharam and others [AVW87a]. MAB have been applied successfully to various decision making problems, like the following:

Clinical trials have been the first historical application of MAB models, where an arm represents a treatment, and the distribution associated with such treatment can be a Bernoulli distribution: a reward of 0 means the drug did not heal the disease, and a reward of 1 indicates a success. The mean of an arm, in this application, represents the mean success rate of a treatment. Following the "best arm identification" model, the goal of a doctor in a clinical trial is to identify the best treatment, *i.e.*, the arm with highest mean, in a number of trials as short as possible. And following our model of interest, maximizing the rewards corresponds to maximize the number of patients being successfully treated.

MAB can also be applied to a broader setting of **online content recommandation**, with more than two options. The seminal work of [LCLS10] studies the application of contextual bandit to news article recommendation, as it is used in practice on platforms such as Microsoft's Bing news website, or in applications like Netflix. In such models, the arms correspond to items to recommend (*e.g.*, articles or movies), and the contexts contain features about each user of the system. An interesting work is [LRC+16], who studies slowly-varying non-stationary models applied to recommender systems.

Using bandit algorithms for **improved machine learning** models or algorithms has been an active research domain for the last ten years or so. As presented below in Chapter 4, a certain "leader" bandit algorithm can be used to select on the run the best bandit algorithm from a pool of "followers" algorithms. Other possible use cases include hyper-parameter optimization, or features selection. Hyper-parameters include real-valued parameters, like a step size multiplier γ in a gradient descent, the width ρ of a Radial Based Function (RBF) kernel, the margin C in a Support Vector Machines (SVM), etc. Discrete-valued parameters are also common, like a choice in a fixed set of kernel functions or the depth of neural networks, and higher dimensional or more complex hyper-parameters can for instance be the entire architecture of a neural network.

Applications for Cognitive Radio. As highlighted in Chapter 1, the focus of this work is on cognitive radio and IoT networks, where arms can represent wireless orthogonal channels, but more generally any resource characterizing the communication between a wireless device and a gateway (*e.g.*, spreading factor for LoRa [KAF+18], power allocation for NOMA etc). In the model of Opportunistic Spectrum Access (OSA) with sensing [JEMP09, JEMP10], the samples $Y_{k,t}$ represents the feedback obtained by the CR-equipped device after sensing the channel k at time t. A reward of $r(t) = 1$ indicates that no Primary User was sensed, while a reward of $r(t) = 0$ indicates that the channel k is busy at time t and no uplink message can be sent. We study a similar model of using MAB for Cognitive Radio but without the OSA structre of Primary and Secondary users, *i.e.*, without sensing information but only a collision indicator, in Chapters 5 and 6.

The different applications discussed in this Section are still active research directions, and a curious reader can find other interesting applications of MAB models and algorithms in [BR19], such as game tree search or network routing in Section 1.2 of [LS19].

2.3 Measuring the performance of a MAB algorithm

As explained above, the main objective of a player facing a bandit game is to maximize its (expected) cumulated reward. In particular, an efficient algorithm should see its mean reward converge to the maximum reward. In the bandit literature, the performance of an algorithm is often measured in terms of regret, a performance measure that we now introduce.

2.3.1 Measuring performance with the (mean) regret

Let us first introduce some notations. We consider a stochastic and stationary MAB problem, with K arms of distributions $\nu_1, \dots, \nu_K = (\nu_k)_k$, that generates *i.i.d.* samples $Y_{k,t} \sim \nu_k$, for any time t. We focus on distributions fully characterized by their means, that can be Bernoulli or any one-dimensional exponential family. We denote $\mu_k \doteq \mathbb{E}[Y_{k,t}] \in \mathbb{R}$ the mean of the distribution of arm k (it will be referred to as the *mean of arm k*). To define the regret, we first need to distinguish between *optimal* and *sub-optimal* arms.

Definition 2.2. *Consider a bandit problem of K arms with distributions of means $\boldsymbol{\mu} = \mu_1, \dots, \mu_k$. Denote $\mu^* \doteq \max_k \mu_k$ the largest mean. The best arm can be non unique, and any arm k having $\mu_k = \mu^*$ is said to be* optimal, *while arms satisfying $\mu_k < \mu^*$ are called* sub-optimal.

If the goal of the player is to maximize $\mathbb{E}\left[\sum_{t=1}^{T} r(t)\right]$, the optimal strategy for this bandit problem is to always pull one of the optimal arms (it can be not unique), but it is unrealistic as the player does not know the true means, nor the optimal arms (as long as they are not all optimal, *i.e.*, in non trivial problems). Comparing the difference between the performance of a fixed baseline and that of the player is a common approach in machine learning research, and here we can compare with the oracle strategy that always obtains an (expected) reward of μ^*. For a fixed horizon T, if k^* denotes the index of any optimal arm, let us introduce the (mean) regret $R_T^{\mathcal{A}}$ of an algorithm $\mathcal{A}$ as $R_T^{\mathcal{A}} \doteq \mathbb{E}\left[\sum_{t=1}^{T}(Y_{k^*,t} - r(t))\right]$. As the rewards from arm k^* are *i.i.d.* (as for all arms), and by linearity of the expectation, we can rewrite this expression to obtain the following definition of the regret.

Definition 2.3 (Regret). *For an algorithm $\mathcal{A}$, a bandit problem of K arms characterized by their means $\mu_1, \dots, \mu_K$, with $\mu^* \doteq \max_k \mu_k$, then its* (mean) regret *at horizon T is $R_T^{\mathcal{A}}$ defined as*

$$R_T^{\mathcal{A}} = \mathbb{E}\left[\sum_{t=1}^{T} (Y_{k^*,t} - r(t))\right] = \mathbb{E}\left[\sum_{t=1}^{T} \underbrace{Y_{k^*,t}}_{\text{i.i.d. } \textit{from } \nu_{k^*}}\right] - \mathbb{E}\left[\sum_{t=1}^{T} r(t)\right] \tag{2.2}$$

$$= T \times \mathbb{E}\left[Y_{k^*,1}\right] - \sum_{t=1}^{T} \mathbb{E}\left[r(t)\right] = T\mu^* - \sum_{t=1}^{T} \mathbb{E}\left[r(t)\right]. \tag{2.3}$$

One first need to observe that $R_T^{\mathcal{A}} \geq 0$ for any T and $\mathcal{A}$, and that $R_T^{\mathcal{A}} \leq \mu^* T$, so $R_T^{\mathcal{A}} \leq T$ if the rewards lie in $[0, 1]$. Thus any algorithm has $R_T^{\mathcal{A}} = \mathcal{O}(T)$, which justifies why we are interested in efficient algorithms that achieve at least a sub-linear regret, *i.e.*, $R_T^{\mathcal{A}} = o(T)$.

A useful decomposition of the regret. Recall that $N_k(t) \doteq \sum_{s=1}^{t} \mathbb{1}(A(s) = k)$ denotes the number of times arm k was selected between times 1 and t, and that the samples $Y_{k,t}$ are all *i.i.d.* of mean μ_k. The *gap* between any arm $k \in [K]$ and an optimal arm is defined as $\Delta_k \doteq \mu^* - \mu_k$ (an arm k is thus sub-optimal if and only if $\Delta_k > 0$), and thus we can write the following decomposition on the regret. *Its proof is simple and it is not a contribution, but we include it for readers that are unfamiliar with conditioning arguments.*

Proposition 2.4 (Regret decomposition). *The (mean) regret $R_T^{\mathcal{A}}$ can be decomposed as a sum of the number of selections of sub-optimal arms k, weighted by their gaps:*

$$R_T^{\mathcal{A}} = \sum_{k=1}^{K} \Delta_k \, \mathbb{E}[N_k(T)] = \sum_{\substack{k=1,\dots,K \\ \Delta_k > 0}} \Delta_k \, \mathbb{E}[N_k(T)]. \tag{2.4}$$

Proof. We use the chain rule of expectation and a conditioning on O_t, because the expectation is taken on the randomness of the (*i.i.d.*) samples $(Y_{k,t})_t$ and on the decisions of the player $(A(t))_t$, which are measurable wrt to the past observations $O_t = (U_0, Y_{A(1),1}, U_1, \dots, Y_{A(t-1),t-1}, U_{t-1})$. Thus we can rewrite the expected cumulated reward as follows:

$$\mathbb{E}\left[\sum_{t=1}^{T} r(t)\right] = \mathbb{E}\left[\sum_{t=1}^{T}\sum_{k=1}^{K} Y_{k,t}\mathbb{1}(A(t) = k)\right] = \sum_{k=1}^{K}\sum_{t=1}^{T} \mathbb{E}\left[Y_{k,t}\mathbb{1}(A(t) = k)\right]$$

$$= \sum_{k=1}^{K}\sum_{t=1}^{T} \mathbb{E}\Big[\mathbb{E}\left[Y_{k,t} | O_t\right] \mathbb{1}(A(t) = k)\Big] = \sum_{k=1}^{K}\sum_{t=1}^{T} \mathbb{E}\Big[\mathbb{E}\left[Y_{k,1}\right] \mathbb{1}(A(t) = k)\Big]$$

$$= \sum_{k=1}^{K} \underbrace{\mathbb{E}\left[Y_{k,1}\right]}_{\mu_k} \underbrace{\sum_{t=1}^{T} \mathbb{E}\left[\mathbb{1}(A(t) = k)\right]}_{=N_k(T)} = \sum_{k=1}^{K} \mu_k \mathbb{E}\left[N_k(T)\right].$$

Thus $R_T^{\mathcal{A}} = T\mu^* - \sum_{t=1}^{T} \mathbb{E}\left[r(t)\right] = T\mu^* - \sum_{k=1}^{K} \mu_k \mathbb{E}[N_k(T)] = \sum_{k=1}^{K} (\mu^* - \mu_k)\mathbb{E}[N_k(T)]$, as $\sum_{k=1}^{K} \mathbb{E}[N_k(T)] = T$. The sum is then simplified to only count sub-optimal arms. □

Consequences. Such decomposition of the regret can be useful for at least two reasons:

On the one hand, from a numerical simulation point-of-view, when we run a finite number of repetitions of the same stochastic experiment, if we want to compute and visualize the regret, we can either use the definition with the rewards, or the decomposition and simply sum the gaps Δ_k with the number of sub-optimal draws. Both quantities are indeed equal *in expectation*, but with only a finite number of trajectories and observations, the first estimate is more noisy than the second one, since the randomness on the rewards is (partially) removed in the decomposition (2.4) (thanks to the conditioning on O_t done in the proof). In our library SMPyBandits, we implement both estimators, and all values of regret used in this thesis are based on the one using the decomposition of Proposition 2.4, because it gives faster convergence, and also "smoother" plots.

On the other hand, this decomposition is necessary as theoretical analyses of the regret of (single-player) MAB algorithms are usually based on controlling the suboptimal draws $N_k(T)$ and not directly the regret $R_T^{\mathcal{A}}$. Even if we prove that controlling $N_k(T)$ is no longer sufficient to obtain low regret for the multi-players case, we extend this decomposition in Chapter 3 (in Lemma 6.5), and it is the first quantity we prove to be bounded by $\mathcal{O}(\ln(T))$ when we prove the regret upper-bound of our proposal MCTopM-kl-UCB.

2.3.2 Regret lower bounds

We include here two well-known results about what bandit algorithms *cannot* do. First, a *problem-dependent lower-bound* states that any algorithm suffers a regret at least $\Omega(\ln T)$ on any problem (from [LR85]). Then, a *worst-case result* states that for any algorithm, there exists a certain problem instance such that the algorithm can perform as badly as $\Omega(\sqrt{KT})$ (a result from [ACBFS02]), In certain families of problems, *e.g.*, K Bernoulli distributed arms, the difficulty of a problem is characterized by a constant that depends only on the arms means. This measure of difficulty of a problem is hidden in the Ω notation in these lower-bounds.

In all this section, we restrict to stationary stochastic problems with $K \geq 2$ arms. We do not give proofs of the following theorems, as they can be found in the historical papers, and simpler proofs are given in recent references, such as [BCB12], [LS19], or Chapter 2 in [Sli19] for instance. Let $\mathcal{I}$ denote the set of all problem instances, with $K \geq 2$ arms. We assume the rewards lie in $[0, 1]$, to present the lower-bounds, we prefer to focus on Bernoulli distributions, for simplicity. To specify the dependency on an instance $I \in \mathcal{I}$, we denote the regret of $\mathcal{A}$ on instance I and horizon T by $R_T^{\mathcal{A}}(I)$. We also index the Landau notations $o(\dots)$ and $\mathcal{O}(\dots)$ by I, for instance $R_T^{\mathcal{A}}(I) = o_I(\ln(T))$ means $R_T^{\mathcal{A}}(I) = o(c_I \ln(T))$ where the "constant" c_I can depend on the problem instance I (*e.g.*, on its means and on K) but *not* on the time horizon T.

Problem-dependent lower-bound in $\Omega(\ln T)$. The following two theorems were proven in [LR85]. These lower-bounds are of highest interest to design efficient algorithms, and a significant part of the research literature on MAB algorithms has focused on finding algorithms that matches the Lai and Robbins' lower-bound asymptotically. This means that an upper-bound is proven on the regret of algorithm $\mathcal{A}$, that asymptotically matches the lower-bound, with the same constant in the big-$\mathcal{O}$ notation (in which case we say that the algorithm is asymptotically *optimal*), or with a larger constant (the algorithm is said to be *order-optimal* in such case).

Theorem 2.5. No algorithm $\mathcal{A}$ can achieve *a (mean) regret* $R_T^{\mathcal{A}}(I) = o_I(\ln(T))$ *for all Bernoulli problem instances* $I \in \mathcal{I}$. *[Sli19, Theorem 2.12]*

We consider uniformly efficient[1] algorithms, to rule out algorithms achieving low regret on some problem instances while achieving linear regret on other instances. In particular, it is necessary to rule out algorithms that always pick the same arm, as on some problem instances such fixed-arm algorithms can achieve zero regret.

Definition 2.6. *An algorithm $\mathcal{A}$ is* uniformly efficient *if its (mean) regret satisfies* $R_T^{\mathcal{A}}(I) = o_I(T^\alpha)$, *for any value* $\alpha > 0$ *and any problem instance* $I \in \mathcal{I}$.

This family is non-empty as it contains for instance the UCB$_1$ algorithm, since its regret is proven to be logarithmic on any Bernoulli problem instance (in Theorem 2.12 below). Now we can state the second logarithmic lower-bound, for algorithms in this family.

Theorem 2.7. *If $\mathcal{A}$ is uniformly efficient, then for any arbitrary Bernoulli problem instance I, its regret is asymptotically lower-bounded by $C_I \ln(T)$, or in other words,* *[Sli19, Theorem 2.13],*

$$\liminf_{T \to \infty} \frac{R_T^{\mathcal{A}}(I)}{\ln(T)} \geq C_I.$$

[1]This notion is then extended to "*strongly uniformly efficient* algorithms", in the multi-players case with Definition 6.7, where we also include a notion of (expected) fairness.

We state below Theorem 2.12 from [Sli19], to specify a possible value for the constant C_I.

Theorem 2.8. *If $\mathcal{A}$ is uniformly efficient, and I an arbitrary Bernoulli problem instance.*

- *The bound from Theorem 2.7 holds with $C_I = \sum\limits_{k:\Delta_k>0} \frac{\Delta_k}{\mathrm{kl}(\mu_k,\mu^*)}$.*
- *For any $\varepsilon > 0$, the bound also holds at finite time with $C'_I = C_I - \varepsilon$, meaning that there exists a constant C_I depending only on I, and a time T_0 such that $\forall T \geq T_0, \quad R^{\mathcal{A}}_T(I) \geq C'_I \ln(T)$.*

Moreover, it is interesting to note that the second lower-bound of Theorem 2.8 can be directly used to design efficient algorithms[2], as thanks to the regret decomposition given in Proposition 2.4 above, the expression of C_I essentially says that any efficient algorithm should sample each sub-optimal arm k about $\ln(T)/\mathrm{kl}(\mu_k, \mu^*)$ times in the total of T time steps.

Many algorithms have been proven to achieve logarithmic regret in the stochastic case, and in particular it is the case of the algorithms used in this thesis, UCB_1 from [ACBF02], Thompson sampling from [Tho33] and analyzed in [AG12, KKM12], and kl-UCB from [GC11, CGM+13]. Such bounds are valid in different settings, and in particular UCB_1 is order-optimal for bounded rewards or one-dimensional exponential families, while Thompson sampling is optimal for bounded rewards, and kl-UCB have been proven to be optimal for both cases. For more details, we refer to Chapter 16 of [LS19].

Worst-case lower-bound in $\Omega(\sqrt{T})$. For a fixed horizon T, it is interesting to note that one can find instances I that are so "hard" that a logarithmic regret (lower or upper) bound that uses a constant C_I no longer brings any information. Indeed, we can naively bound the regret by $R^{\mathcal{A}}_T \leq (\max_k \Delta_k)T$, and thus if $(\max_k \Delta_k)$ can be taken so small that $C_I \ln(T) \gg (\max_k \Delta_k)T$, a regret upper bound like $R^{\mathcal{A}}_T \leq C_I \ln(T)$ is useless.

For this reason, another family of regret bounds is relevant, that are not problem-dependent but worst-case, or also called minimax [AB09, AB10] or problem-independent. We give an example of such result below. The following theorem is from [ACBFS02], and [Sli19, Theorem 2.1]. For more details, we refer to Chapter 15 of [LS19].

Theorem 2.9. *Fix the number of arms K, and an algorithm $\mathcal{A}$. Then for any horizon T, there exists a Bernoulli problem instance I_T on which the algorithm suffers at least a regret $R^{\mathcal{A}}_T(I_T) \geq \Omega(\sqrt{KT})$.*

[2] Tracking this quantity, by using empirical estimates of the means, is used for instance in the OSSB algorithm proposed in [CMP17], which is proven to attain the lower-bound for a wider range of problems (for problems called structured stochastic bandits).

Similarly to what is considered for the first lower-bound, a natural question is to know if there is an algorithm that achieve a regret upper-bound of the form $R_T^{\mathcal{A}}(I) \leq \mathcal{O}\left(\sqrt{KT}\right)$ for any instance, independently on the problem difficulty. It is the case for the Exp3 algorithm from [ACBF02], for example. Since a few years, some algorithms were shown to achieve both a problem-dependent logarithmic and a minimax upper-bounds, like MOSS in [AB09] or recently $\mathrm{kl}\text{-}\mathrm{UCB}^{++}$ in [MG17], and such results are usually referred to as "best of both worlds".

2.3.3 Other measures of performances

The theoretical results obtained in the rest of this thesis are all expressed in terms of the mean regret $R_T^{\mathcal{A}}$ (in Chapters 6 and 7), but we quickly mention other measures of performances that we consider in our works and in this manuscript.

First, when doing numerical experiments about bandits, if one studies the regret as an empirical mean based on a "large" number of random repetitions (*e.g.*, $N = 1000$ repetitions), it is important to not only show the mean value but also the variance of the values taken by the regret on each repetition, to verify that all algorithms perform consistently. Indeed, by only visualizing the mean of 1000 values, it is possible that we miss some "bad runs": if 1 run out of the 1000 gives linear regret (*i.e.*, $R_T^{\mathcal{A}} \propto T$) and the 999 other give logarithmic regret, then the mean will appear logarithmic. This is the case of the $\mathrm{Selfish}$ algorithm that is defined and explored in Chapter 6. By visualizing the entire *distribution* (as an histogram), or the variance of the values of $R_T^{\mathcal{A}}$, and if the number N is reasonably large, we can verify that the regret appears logarithmic for all runs.

The **switching cost** $\mathrm{SC}^{\mathcal{A}}(T)$ counts how many times the player's decision has changed from one round to the next one, *i.e.*, $\mathrm{SC}^{\mathcal{A}}(T) \doteq \sum_{t=1}^{T-1} \mathbb{1}(A(t) \neq A(t+1))$. It has recently gained interest in the literature, for instance the authors of [TRY17] proposed an algorithm that adaptively tries to balance the trade-off between minimizing the regret and minimizing the switching cost. Indeed, in single-player models it is easy to show that achieving logarithmic regret directly implies a logarithmic upper-bound on the switching cost, and conversely the lower-bound from [LR85] also gives an asymptotic logarithmic lower-bound. But even if $\mathrm{SC}^{\mathcal{A}}(T) = \Theta(\ln T)$ for an efficient algorithm, it can be interesting to numerically evaluate this quantity, as a large value might indicate an algorithm that is alternating too much between the optimal arm and other arms. Because it is relevant for cognitive radio applications, as a hardware reconfiguration costs energy, and so changing channel from two consecutive time steps has a non trivial energy cost, as highlighted in [DMNM16]. And thus we find more interesting to study $\mathrm{SC}^{\mathcal{A}_1,\dots,\mathcal{A}_M}(T)$, the sum of the switching costs of the M players in a multi-players bandit game (if player j uses algorithm $\mathcal{A}_j$ for $j = 1, \dots, M$), as studied in

Chapter 6. Our proposed algorithm, MCTopM, achieves order-optimal logarithmic regret as well as a logarithmic number of switches.

Finally, another interesting measure of performance is the **fairness**. It was not studied much for single-player problems, and usually it means that each arm must be explored at least a given fraction of the total horizon T (*i.e.*, "arm-wise fairness"). It was studied in a very recent article [PGNN19], where a different notion of regret is introduced to account for the fairness constraint. The authors show that different algorithms, based on UCB_1 or Thompson sampling, can achieve logarithmic regret while respecting the fairness constraints. In this thesis, we only consider fairness in the multi-players bandit models, in Chapter 6, where fairness refers to a different notion (*i.e.*, "cooperative fairness"). We refer to Section 6.3.2 for more details.

2.4 Review of stochastic MAB algorithms

This Section reviews the most important families of stochastic MAB algorithms, from naive and simple strategies, to strategies based on the optimism or the Bayesian principles, to recent "best-of-both-worlds" strategies.

Implementation. We describe our library SMPyBandits in more details in Chapter 3. All the algorithms described in this chapter are implemented in SMPyBandits, in the `Policies` module, alongside with many more algorithms (there are about 65 for single-player stochastic problems). A complete list of the implemented policies can be found on the following web page on the documentation, `SMPyBandits.GitHub.io/docs/Policies.html`.

2.4.1 Naive or simple strategies

Pure exploitation or pure exploration. Let us first describe two naive strategies, that both fail dramatically. We recall the notations introduced above in Section 2.1.1, the sums of rewards are $X_k(t) \doteq \sum_{s=1}^{t} r(s)\mathbb{1}(A(s)=k)$, and the numbers of samples are $N_k(t) \doteq \sum_{s=1}^{t} \mathbb{1}(A(s)=k)$. The estimated means, or empirical averages, are $\widehat{\mu_k}(t) \doteq X_k(t)/N_k(t)$ (when $N_k(t) > 0$).

– The uniform strategy always plays the K arms uniformly at random, $\forall t \in [T], A(t) \sim \mathcal{U}([K])$, where $\mathcal{U}(S)$ denotes the uniform distribution on set S, and $\mathcal{U}([K])$ the uniform distribution on $[K] = \{1, \dots, K\}$ (*i.e.*, $\forall t, \forall k \in [K], \mathbb{P}(A(t)=k) = \frac{1}{K}$). The player *only explores* without using the collected information, and this strategy fails dramatically for any non trivial problem (*i.e.*, linear regret). Indeed it obtains a linear (mean) regret $R_T = \frac{1}{K}\sum_{k=1}^{K} \Delta_k T$ which gives $R_T \propto T$ for any problem with at least one sub-optimal arms (*i.e.*, all non trivial problems, rulling out the case where $\mu_1 = \mu_2 = \dots = \mu_K$).

– The "*Follow-the-Leader*" strategy consists in first playing once each arm, then always playing $A(t) \in \arg\max \widehat{\mu_k}(t)$. The player *only exploits* the collected information, and this strategy can fail dramatically, *i.e.*, obtain linear regret in some problems. Indeed consider $K = 2$ Bernoulli arms of means $\mu_1 = 1/2$ and $\mu_2 = \varepsilon$ where $\varepsilon < 1/2$, then with probability $\varepsilon/2$ the player observes a reward of 0 for arm 1 then 1 for arm 2 on the first rounds, and so she will play arm 2 for the $T - 2$ remaining rounds, giving a linear (mean) regret $R_T \geq \frac{\varepsilon}{2}\left(\frac{1}{2} - \varepsilon\right)(T-1)$.

Simple efficient strategies: ε-greedy and Explore-then-Exploit. As illustrated by the two previous examples, an efficient strategy needs to solve the trade-off between exploration and exploitation. The two following solutions both consist in splitting the T time steps into T_0 steps of exploration and $T - T_0$ steps of exploitation. They are detailed in Algorithm 2.1.

– The ε-greedy strategy consists in alternating exploration and exploitation at a certain ratio [ACBF02]. Fix $0 < \varepsilon < 1$, then at each round, with probability ε the player selects an arm uniformly at random (exploration) and with probability $1 - \varepsilon$ the arm with highest empirical mean is selected (exploitation). On the one hand, if ε is constant, then the (mean) regret still grows linearly, as it is lower bounded by $(\varepsilon \frac{1}{K} \sum_{k=1}^{K} \Delta_k)T$. In average, $T_0 = \varepsilon T$ steps are spent on exploration. On the other hand, if a lower-bound d on the positive gaps is known beforehand, we can consider a sequence $(\varepsilon_t)_{t\in\mathbb{N}^*}$ decreasing with time t, for instance $\varepsilon_t = \varepsilon_0/t$ with $\varepsilon_0 = 6K/d^2$, for a constant $0 < d < \min_{k:\Delta_k>0} \Delta_k$. Then it was shown in [ACBF02] that the regret of ε-greedy is of the order of $K\ln(T)/d + o(T)$, which leads to an order-optimal regret of $\mathcal{O}(K\ln(T))$. But this is not satisfactory, as $\min_k \Delta_k$ needs to be known in advance, which is usually not the case for real applications. Here again, an average of $T_0 \leq \varepsilon_0 \ln(T)$ steps are spent on exploration.

– The "Explore-then-Exploit" strategy, as presented for instance in [BCB12], first explores uniformly the K arms for T_0 time steps, then only exploits the arm identified as the best arm after these first T_0 steps (*i.e.*, one of the arms with highest empirical means, after T_0/K samples of each arm). Usually we restrict to T_0 being a multiple of K.

Here again, if a lower-bound $d > 0$ on the positive gaps is known beforehand, one can find a tuning of T_0 that gives a logarithmic regret, as stated in Theorem 2.10 below. It proves that the "explore-then-exploit" strategy can also obtain an order-optimal regret, $R_T \leq K(4/d^2)(1 + \ln(d^2T/4)) = \mathcal{O}(K\ln(T))$, if it is tuned with a fixed time T_0 using prior knowledge on the problem (*i.e.*, d) and the horizon T. We note that this strategy can also achieve a much larger regret bound, $R_T = \mathcal{O}\Big(T^{2/3}(K\ln(T))^{1/3}\Big)$, without prior knowledge on the problem, as shown in Section 1 of [Sli19].

Theorem 2.10. *For any instance I with K arms with Bernoulli distributions in $[0, 1]$, of means $\mu_1, \dots, \mu_k$, and any horizon $T > K$, and if $d \leq \Delta = \min_k \Delta_k$ is known, let*

```
1  for $t = 1, 2, \dots, T$ do
2      if $\varepsilon$-greedy then
3          Sample a value uniformly in $[0,1]$: $U_t \sim \mathcal{U}([0,1])$
4          if $U_t < \varepsilon$ (i.e., with probability $\varepsilon$) then
5              Play uniformly at random $A(t) \sim \mathcal{U}(\{1,\dots,K\})$          // Explore
6          else
7              Play uniformly among the arms of maximal empirical mean:
                $A(t) \sim \mathcal{U}(\arg\max_{1\leq k\leq K} \widehat{\mu_k}(t))$          // or Exploit
8      if Explore-then-Exploit then
9          if $t \leq T_0$ then
10             Play sequentially $A(t) \in 1 + (t \mod K)$          // Explore
11         else
12             if $t = T_0$ then
13                 Pick one arm $A(T_0) = k$ uniformly among the arms of maximal
                    empirical mean: $A(t) \sim \mathcal{U}(\arg\max_{1\leq k\leq K} \widehat{\mu_k}(T_0))$     // then Exploit
14             Play the same arm $A(t) = A(T_0)$
15     Observe a reward $r(t)$, and update $X_k(t)$, $N_k(t)$ and $\widehat{\mu_k}(t)$
16 end
```

Algorithm 2.1: Simple efficient strategies: ε-greedy and Explore-then-Exploit.

$T_0 = ((2K)/d^2)\ln(d^2T/(2K))$. Then the Explore-then-Exploit algorithm with parameter T_0 verifies the following finite-time regret bound

$$R_T^{\text{Explore-then-Exploit}} \leq \frac{2K}{d^2} \ln\left(\frac{d^2T}{2K}\right) = \mathcal{O}\left(\frac{K\ln(T)}{\Delta^2}\right). \tag{2.5}$$

Proof. If p denotes the probability that the chosen arm is not optimal, the algorithm suffers from a linear regret for the first T_0 rounds, then with probability p it suffers from a linear regret for the remaining $T - T_0$ rounds, and with probability $1-p$ it suffers no regret. Thus we have $R_T \leq (\max_{k:\Delta_k>0} \Delta_k)T_0 + p(T-T_0)$. We then prove that $p \leq 2K\exp(-T_0d^2/4)$.

We use Hoeffding's inequality from [Hoe63], reminded below in Lemma 2.11. First for the case of two arms: if $\mu_1 = \mu_2 + \Delta$, then $p = \mathbb{P}(\widehat{\mu_1} < \widehat{\mu_2}) \leq \mathbb{P}(\widehat{\mu_1} < \mu_1 - \Delta/2) + \mathbb{P}(\widehat{\mu_2} > \mu_2 + \Delta/2)$, and both terms can be bounded by using Hoeffding's inequality with a (non-random) number of samples $n \leq T_0/K$, to obtain $p \leq 2\exp(-T_0\Delta^2/(2K))$. Then for $K \geq 2$ arms, a simple union bound on the sub-optimal arms gives $p \leq 2(K-1)\exp(-T_0d^2/(2K))$, if d is a lower-bound on the positive gaps Δ_k.

So this bound on p gives $R_T \leq KT_0 + 2K\exp(-T_0 d^2/(2K))T$ (because $\max_{k:\Delta_k>0}\Delta_k \leq 1$ and $T - T_0 \leq T$). Optimizing the right hand-side on T_0 gives $T_0 = ((2K)/d^2)\ln(d^2T/(2K))$. This gives the announced bound. □

Lemma 2.11. *Let* $Z_1,\dots,Z_n$ *be* n i.i.d. *samples from a Bernoulli distribution of parameter* μ *(where* n *is fixed), of empirical mean* $\widehat{Z_n} \doteq \frac{1}{n}\sum_{i=1}^{n} Z_i$, Hoeffding's inequality *gives both*

$$\begin{cases} \forall x < \mu, & \mathbb{P}(\widehat{Z_n} < x) \leq \exp(-2n(x-\mu)^2), \\ \forall x > \mu, & \mathbb{P}(\widehat{Z_n} > x) \leq \exp(-2n(x-\mu)^2). \end{cases} \tag{2.6}$$

An extension of this strategy is to consider not a fixed time T_0 but a random time τ at which exploration stops. This time τ must be a *stopping time*[3], in the sense that it is a measurable random variable, dependent of the past observations. The strategy is then referred to as "explore-then-commit" (ETC), and the idea is to use a statistical test at every time step t, and stop as soon as enough samples were collected to effectively identify the best arm with a certain confidence level δ. Choosing $\delta \propto 1/T$ and using a lower-bound d on the gaps typically lead to an order-optimal algorithm as shown in [GKL16].

For both cases, the strategy obtains sub-optimal regret if it is tuned independently of the problem at hand, but can be tuned to be efficient (*i.e.*, with logarithmic regret) if a lower-bound on the gaps Δ_k is known. As such, this weakness makes the presented strategies unapplicable on an unknown problem, and so they are less interesting from a practical point-of-view.

2.4.2 The optimism principle and index policies: UCB_1, kl-UCB etc

A large family of algorithms are index based, as they compute an index $U_k(t)$ on each arm k at time t, and they play the arm that maximizes their index, *i.e.*, $A(t) = \arg\max_k U_k(t)$. If more than one indexes maximize $(U_k(t))_k$, the arm is chosen from the set, usually in a uniformly random manner: $A(t) \sim \mathcal{U}(\arg\max_k U_k(t))$ (using the external randomness, as explained above). The Algorithm 2.2 below details a generic index policy, that includes well known and efficient algorithms such as UCB_1, kl-UCB and many others.

The UCB_1 index policy: using Hoeffding's inequality to build confidence intervals. Let us consider another approach for adaptive exploration, known as "optimism under uncertainty": assume each arm is as good as it can possibly be given the observations so far, and choose the best arm based on these optimistic estimate. This intuition leads to the UCB_1 algorithm, initially introduced in [ACBF02], for bounded rewards. For a parameter α, the

[3]See Chapter 3 of [LS19], and we use this notion again in Section 7.5.

```
1 for $t = 1, 2, \dots, T$ do
2     if $t \leq K$ then
3         Play arm $A(t) = t$;
4     else
5         For each arm, compute the index $U_k^{\mathcal{A}}(t)$;
6         Play arm $A(t)$ uniformly among the arms of maximal index:
          $A(t) \sim \mathcal{U}(\arg\max_{1 \leq k \leq K} U_k^{\mathcal{A}}(t))$;
7     Observe a reward $r(t)$ and update $X_k(t)$, $N_k(t)$ and $\widehat{\mu_k}(t)$;
8 end
```

Algorithm 2.2: A generic index policy $\mathcal{A}$, using indexes $U_k^{\mathcal{A}}(t)$ (*e.g.*, UCB_1, kl-UCB etc).

UCB indexes are computed as follows, as a sum of the *average reward* $\widehat{\mu_k}(t) \doteq \frac{X_k(t)}{N_k(t)}$ and a *confidence radius* $\xi_k(t) \doteq \sqrt{\alpha \frac{\ln(t)}{N_k(t)}}$.

$$U_k^{\mathrm{UCB}_1}(t) = \mathrm{UCB}_k(t) \doteq \underbrace{\frac{X_k(t)}{N_k(t)}}_{\text{Exploitation } \widehat{\mu_k}(t)} + \underbrace{\sqrt{\alpha \frac{\ln(t)}{N_k(t)}}}_{\text{Exploration } \xi_k(t)} . \tag{2.7}$$

This selection rule $A(t) \sim \mathcal{U}(\arg\max_{1 \leq k \leq K} U_k^{\mathcal{A}}(t))$ makes sense for the following reasons: an arm k is chosen at step t because it has a large $\mathrm{UCB}_k(t)$, which can happen for two reasons. $i)$ Because the average reward $\widehat{\mu_k}(t)$ is large, in which case this arm is likely to have a high mean reward μ_k, $ii)$ and/or because the confidence radius $\xi_k(t)$ is large, in which case this arm has not been explored much. Either reason makes this arm worth choosing. One can also observe that $\xi_k(t) \to \infty$ if $t \to \infty$ while $N_k(t) \ll \ln(t)$, which can be used to prove that the UCB_1 algorithm tries all arms at least $\Omega(\ln(T))$ times (in average). Furthermore, these two terms $\widehat{\mu_k}(t)$ and $\xi_k(t)$ in the UCB defined in (2.7) respectively represent *exploitation* and *exploration*, and summing them up is a natural way of dealing with the exploitation/exploration trade off. The constant parameter α in $\xi_k(t)$ also controls this trade-off, and the theoretical analysis suggests to restrict to $\alpha \geq 1/2$, and to choose $\alpha = 1/2$ for optimal uniform performance (*i.e.*, on all problems, [ACBF02]).

We present the regret bound, which was first stated in [ACBF02], and to explain how Hoeffding's inequality is used we prove it below. Note that for the rest of the manuscript, we use the kl-UCB algorithm (and more subtle concentration inequality) whenever we analyze the regret of a newly introduced algorithm (*i.e.*, in Chapters 6 and 7), but we think the simpler proof of UCB_1 is enlightening for the reader. This theorem shows that UCB_1 achieves an order-optimal problem-dependent regret, bounded by $\mathcal{O}(K \ln(T)/\Delta^2)$ (if $\Delta \doteq \min_{k:\Delta_k>0} \Delta$), and the result also has the advantage of being non asymptotic.

Theorem 2.12 (Regret bound for UCB_1)**.** *For any instance I with K arms with bounded distributions in $[0,1]$, of means $\mu_1,\dots,\mu_k$, and any horizon $T > K$, the UCB_1 algorithm with its parameter*[4] *$\alpha = 2$ verifies the following finite-time regret bound*

$$R_T^{\mathrm{UCB}_1}(I) \leq \left(\sum_{k:\Delta_k>0} \frac{8}{\Delta_k} \right) \ln(T) + \sum_{k:\Delta_k>0} \Delta_k \left(1 + \frac{\pi^2}{3} \right). \tag{2.8}$$

Proof. We focus on bounded distributions in $[0,1]$, for which Hoeffding's inequality gives the inequality (2.6), as stated above in Lemma 2.11. Thanks to the regret decomposition from Proposition 2.4, we can focus on bounding $\mathbb{E}[N_k(T)]$ for any sub-optimal arm k.

The first trick consists in writing the following, which is true for any $n \geq 1$:

$$N_k(T) = 1 + \sum_{t=K+1}^{T} \mathbb{1}(A(t) = k) \leq 1 + n + \sum_{t=n}^{T} \mathbb{1}(A(t) = k, N_k(t) \geq n). \tag{2.9}$$

Remember that $\mathrm{UCB}_k(t) = \widehat{\mu_k}(t) + \xi_k(t)$ for any arm k. According to the Algorithm 2.2 (see line 6), a suboptimal arm k can be chosen only if $\mathrm{UCB}_k(t) \geq \mathrm{UCB}_{k^*}(t)$. Both $\widehat{\mu_k}(t)$ and $\xi_k(t)$ use $n = N_k(t)$ samples of arm k, but as (2.9) above started to isolate $N_k(t)$, we add the freedom of considering different number of samples (using n), so we introduce the notation $\widehat{\mu}_{k,n}(t)$ and $\xi_{k,n}(t)$ where $N_k(t)$ is replaced with n if $n \leq N_k(t)$, *i.e.*, $\widehat{\mu}_{k,n(t)} = \frac{1}{n}\sum_{s=1}^{t-1} r(s)\mathbb{1}(A(s) = k, N_k(s) \leq n)$ and $\xi_{k,n}(t) = \sqrt{2\ln(t)/n}$. Thus, we can continue from (2.9) and write

$$\begin{aligned}
N_k(T) &\leq 1 + n + \sum_{t=n}^{T} \mathbb{1}(A(t) = k, N_k(t) \geq n) \\
&\leq 1 + n + \sum_{t=n}^{T} \mathbb{1}(\widehat{\mu}_{k^*,N_{k^*}(t)}(t) + \xi_{k^*,N_{k^*}(t)}(t) < \widehat{\mu}_{k,N_k(t)}(t) + \xi_{k,N_k(t)}(t), N_k(t) \geq n) \\
&= 1 + n + \sum_{t=n}^{T} \mathbb{1}\left(\min_{0<n_{k^*}\leq t} \widehat{\mu}_{k^*,n_{k^*}}(t) + \xi_{k^*,n_{k^*}}(t) < \max_{n<n_k\leq t} \widehat{\mu}_{k,n_k}(t) + \xi_{k,n_k}(t) \right)
\end{aligned} \tag{2.10}$$

So two union bounds on n_{k^*} for the $\min$ and on n_k for the $\max$ give two sums

$$N_k(T) \leq 1 + n + \sum_{t=n}^{T} \sum_{n_{k^*}=1}^{t} \sum_{n_k=n}^{t} \mathbb{1}\left(\widehat{\mu}_{k^*,n_{k^*}}(t) + \xi_{k^*,n_{k^*}}(t) < \widehat{\mu}_{k,n_k}(t) + \xi_{k,n_k}(t) \right) \tag{2.11}$$

[4] For simplicity we present the theorem and its prove in the restricted case of $\alpha = 2$. More details on this proof can be found for instance in Section 1.3 of [Sli19], while for instance Chapter 2 of [BCB12] Theorem 2.1) and Chapter 7 of [LS19] (Theorem 7.2) both give more general proofs, in particular for any value of $\alpha \geq \frac{1}{2}$.

Now, at time t, this event $\widehat{\mu}_{k^*,n_{k^*}}(t) + \xi_{k^*,n_{k^*}}(t) < \widehat{\mu}_{k,n_k}(t) + \xi_{k,n_k}(t)$ implies at least one of the following three events:

$$\begin{cases} I_1 & \doteq & (\widehat{\mu}_{k^*,n_{k^*}}(t) \leq \mu_{k^*} - \xi_{k^*,n_{k^*}}(t)), \\ I_2 & \doteq & (\widehat{\mu}_{k,n_k}(t) \geq \mu_k + \xi_{k,n_k}(t)), \\ I_3 & \doteq & (\mu_{k^*} < \mu_k + 2\,\xi_{k,n_k}(t)). \end{cases}$$

- For the first two events I_1 and I_2, we carefully fixed the number of samples so they are no longer random (they are respectively to n_{k^*} and n_k), we can use Hoeffding's inequality to obtain $\mathbb{P}(I_j) \leq 1/t^4$ (for $j = 1, 2$).
- For the last event I_3, observe that $\mu_{k^*} < \mu_k + 2\xi_{k,n_k}(t)$ means $\Delta_k - 2\sqrt{2\ln(t)/n_k} < 0$, where we remind the notation of the gap $\Delta_k \doteq \mu_{k^*} - \mu_k$. Thus if $n_k \geq \left\lceil \frac{8\ln(T)}{\Delta_k^2} \right\rceil \geq \left\lceil \frac{8\ln(t)}{\Delta_k^2} \right\rceil$, then this inequality actually cannot happen: $\mathbb{P}(\Delta_k - 2\sqrt{2\ln(t)/n_k} < 0) = 0$, so as soon as $n_k \geq \frac{8\ln(T)}{\Delta_k^2}$, $\mathbb{P}(I_3) = 0$.

Taking the expectation of $N_k(T)$ from (2.10), and setting $n = n_T \doteq \lceil \frac{8\ln(T)}{\Delta_k^2} \rceil$ thus gives

$$\begin{aligned} \mathbb{E}[N_k(T)] &\leq 1 + \frac{8\ln(T)}{\Delta_k^2} + \sum_{t=n_T}^{T} \sum_{n_{k^*}=1}^{t} \sum_{n_k=n_T}^{t} \mathbb{P}\left(\widehat{\mu}_{k^*,n_{k^*}}(t) + \xi_{k^*,n_{k^*}}(t) < \widehat{\mu}_{k,n_k}(t) + \xi_{k,n_k}(t)\right) \\ &\leq 1 + \frac{8\ln(T)}{\Delta_k^2} + \sum_{t=n_T}^{T} \sum_{n_{k^*}=1}^{t} \sum_{n_k=n_T}^{t} [\underbrace{\mathbb{P}(I_1)}_{\leq 1/t^4} + \underbrace{\mathbb{P}(I_2)}_{\leq 1/t^4} + \underbrace{\mathbb{P}(I_3)}_{=0}] \\ &\leq 1 + \frac{8\ln(T)}{\Delta_k^2} + \sum_{t=n_T}^{T} \sum_{n_{k^*}=1}^{t} \sum_{n_k=n_T}^{t} 2\frac{1}{t^4} \end{aligned}$$

We actually have that $\sum_{n_{k^*}=1}^{t} \sum_{n_k=n_T}^{t} 2\frac{1}{t^4} \leq 2 \sum_{n_{k^*}=1}^{t} \frac{1}{t^3} \leq 2\frac{1}{t^2}$, whose sum for $t = 1$ to ∞ is $2\frac{\pi^2}{6}$.

$$\leq \frac{8\ln(T)}{\Delta_k^2} + 1 + 2\sum_{t=1}^{T} \frac{1}{t^2} \leq \frac{8\ln(T)}{\Delta_k^2} + 1 + 2\sum_{t=1}^{\infty} \frac{1}{t^2} \leq \frac{8\ln(T)}{\Delta_k^2} + 1 + \frac{\pi^2}{3}. \tag{2.12}$$

To sum up, we obtain the following finite-time bound on $N_k(T)$ for any sub-optimal arm k

$$\mathbb{E}[N_k(T)] \leq \frac{8\ln(T)}{\Delta_k^2} + \left(1 + \frac{\pi^2}{3}\right).$$

By using the regret decomposition from Proposition 2.4, this gives

$$R_T^{\mathrm{UCB}_1} = \sum_{k:\Delta_k>0} \Delta_k \mathbb{E}[N_k(T)]$$

$$\leq \sum_{k:\Delta_k>0} \frac{8}{\Delta_k} \ln(T) + \sum_{k:\Delta_k>0} \Delta_k \left(1 + \frac{\pi^2}{3}\right) = \sum_{k:\Delta_k>0} \frac{8}{\Delta_k} \ln(T) + o(\ln(T)) \,.$$

□

We see in this Theorem 2.12 that UCB_1 has the advantage of not relying on any prior knowledge of the problem difficulty, of being anytime, and computationally simple, for both memory and time aspects, and it achieves an order-optimal problem-dependent regret upper bound in $\mathcal{O}((\sum_{k:\Delta_k>0} \frac{1}{\Delta_k}) \ln(T))$. UCB_1 requires a constant storage capacity for each arm (storing $X_k(t)$ and $N_k(t)$ require two integers or a `float` and an integer, *i.e.*, $\mathcal{O}(1)$), giving a storage capacity of $\mathcal{O}(K)$, independently of the horizon T. It also needs to perform some mathematical operations, but a square-root and a logarithm are cheap, thus it costs a $\mathcal{O}(1)$ constant time for each arm and time step, thus a time $\mathcal{O}(KT)$ for the T time steps. We use UCB_1 as the base building block for other policies in Chapter 5, for these reasons. Moreover, the UCB_1 algorithm has the advantage of being easy to explain and understand, and its decisions can be interpreted clearly: by reading the values of $\widehat{\mu_k}(t)$ and $\mathrm{UCB}_k(t)$, anyone can see why an arm was chosen at anytime. This is why it is used in our demonstration presented in Section 5.3.

The kl-UCB **index policy.** The kl-UCB algorithm follows the same spirit as the UCB_1 algorithm presented above, it is an index policy (see Algorithm 2.2), which also builds upper confidence bounds on the unknown mean of each arm. The KL-UCB algorithm was first proposed in [GC11] for one-dimensional exponential families, and then the kl-UCB algorithm was introduced in [CGM+13] for bounded rewards. In all this thesis, we either restrict to exponential families or to bounded rewards in $[0, 1]$ and use the fact that they are sub-Bernoulli, thus we prefer to only present kl-UCB.

Instead of using a confidence radius $\xi_k(t)$ based on Hoeffding's inequality, the kl-UCB algorithm rather uses the Chernoff concentration inequality for the considered exponential family The indexes are computed as follows, for an exploration function $f(t)$ that is typically chosen as $f(t) \doteq \ln(t) + c\ln(\ln(t))$ for the theoretical analyses (with $c \geq 3$), and $f(t) \doteq \ln(t)$ in practice.

$$U_k^{\text{kl-UCB}}(t) \doteq \sup_{q\in[0,1]} \left\{ q : \mathrm{kl}\left(\frac{X_k(t)}{N_k(t)}, q\right) \leq \frac{f(t}{N_k(t)} \right\}. \tag{2.13}$$

For Bernoulli distributions, kl-UCB builds tighter confidence intervals than UCB_1, in the sense that $U_k^{\text{kl-UCB}}(t) < U_k^{\text{UCB}}(t)$ but with the same probability coverage. This is justified by Pinkser's inequality, which gives $\mathrm{kl}(x, y) \geq 2(x-y)^2$. The KL-UCB and kl-UCB algorithms achieve finite-time regret logarithmic upper-bounds on their regret, and have been proven to be asymptotically optimal, respectively for one-dimensional exponential families and for bounded rewards distributions [GC11, CGM+13].

A note about the implementation of kl-UCB **algorithms.** Implementing this kind of policy requires to implement two things. The first piece is usually easy: one has to implement a way to compute $\mathrm{kl}(x, y)$ the Kullback-Leibler divergence of the considered one-dimensional family, for instance kl for Bernoulli distributions has a closed form and is easy to compute. The second piece is less obvious: one has to solve the optimisation problem defining the $\sup$ in (2.13). Any iterative algorithm for numerical optimization of one-dimensional functions can be used (*e.g.*, see [BV04] for a survey), but as the $\mathrm{kl}(x, \bullet)$ function is usually convex (for any x), two practical and mathematically correct solutions are Newton's method or a bisection search. The search space for q can be restricted to $[X_k(t)/N_k(t), 1]$ instead of $[0, 1]$, and for simplicity and efficiency, in this case a naive bisection search works well. For instance, $\varepsilon = 10^{-8}$ is enough for numerical simulations, and typically the bisection search terminates in between 5 to 20 steps.

If we perform numerical simulations with kl-UCB algorithms, we are interested in having efficient implementations, and ideally kl-UCB should not be significantly slower than UCB. The bottleneck of the performance is the computations of the KL functions and the optimization problem, and we note that our library SMPyBandits implements both parts in both a naive Python implementation (file `kullback.py`) and in a Python module written in a C file[5], in an optimized way (file `kullback_py3.c`). For the supported families of distributions (including Bernoulli, Gaussian, exponential and Poisson), we wrote both a naive and an optimized version of $\mathrm{KL}(x, y)$ and the $\sup_q \mathrm{KL}(x, q) \leq z$ optimization problem, using Newton's method.

Variants on UCB_1**.** Many variants of the UCB_1 algorithms have been studied in the multi-armed bandits literature. For instance, UCB-H replaces the $\ln(t)$ by a $\ln(T)$ (as well as kl-UCB-H), and it actually corresponds to the first algorithm analyzed in [ACBF02]. Using an estimator for the variance of the arms' samples gives UCB-V, which was for instance proven to be efficient against Gaussian distributions with unknown variance. Other variants include UCB^{+}, $\mathrm{UCB}^{\dagger}$ from [Lat18], UCBoost, and many more. All these variants are usually comparable to the original UCB_1 algorithm, in terms of time and memory complexity, and they achieve improved regret upper bounds in the same setting or some extensions of the initial setting.

[5] Note that we also wrote an intermediate implementation that uses Cython [BBS+19], to have a file with a syntax closer to Python, while gaining from almost the same performance speed-up as with the optimized C version, see file `kullback_cython.pyx`. These implementations of $\mathrm{KL}(x, y)$ and kl-UCB indexes are also published as an independent Python package, hosted on `GitHub.com/Naereen/Kullback-Leibler-divergences-and-kl-UCB-indexes` and installable from Pypi. A Julia version of the same package is also available at `GitHub.com/Naereen/KullbackLeibler.jl`.

2.4.3 Bayesian policies

Thompson sampling is the oldest known MAB algorithms, dating back from the first paper by Thompson in 1933 [Tho33]. It became popular since its asymptotical and finite-time analyses in [AG12, KKM12]. Thompson sampling uses a Bayesian point of view: instead of building confidence intervals on the means of the arms like UCB_1 does, it starts with a prior $\pi_k(0)$ on the K distributions (*e.g.*, a flat uniform prior), and then after every observation $(k, r(t))$ it updates the posterior distribution of the arm k with a new data $r(t)$, from $\pi_k(t)$ to $\pi_k(t+1)$. Thompson sampling makes its decision at time t by first sampling a random sample $s_k \sim \pi_k(t)$, and then playing according to the "optimism under uncertainty" philosophy if these samples represent the arms (expected) quality, *i.e.*, $A(t) \in \arg\max_k s_k$.

For problems with Bernoulli distributions on the K arms, the best choice of prior is a Beta distribution, that starts as an initially uniform prior (*i.e.*, $\forall k, \pi_k(0) \doteq \mathrm{Beta}(1,1)$), and maintain a distribution $\pi_k(t) \doteq \mathrm{Beta}(X_k(t)+1, N_k(t)-X_k(t)+1)$, as $X_k(t)$ counts the number of successes observed on arm k, and $N_k(t) - X_k(t)$ counts the number of failures. We detail the Thompson sampling algorithm in this case, below in Algorithm 2.3, as it corresponds to the case used in Chapter 5 in Section 5.2.4. In this setting as well as for Gaussian distributions, Thompson sampling was proven to achieve asymptotical optimal problem-dependent bounds [KKM12, AG12].

```
1 for $t = 1, 2, \dots, T$ do
2     For each arm, sample $s_k(t) \sim \pi_k(t-1)$ from the posterior of arm $k$,
        $\pi_k(t-1) = \mathrm{Beta}(X_k(t-1)+1, N_k(t-1) - X_k(t-1) + 1)$;
3     Play arm $A(t)$ uniformly among the arms of maximal random sample:
        $A(t) \sim \mathcal{U}(\arg\max_{1 \leq k \leq K} U_k(t))$;
4     Observe a reward $r(t)$ and update $X_k(t)$ and $N_k(t)$;
5 end
```

Algorithm 2.3: Thompson Sampling for Beta prior and posteriors, on Bernoulli rewards.

Bayes-UCB from [KCG12] also uses the Bayesian point of view, updated like for Thompson sampling, but used in a different way. Instead of sampling from the posteriors and playing the arms with maximum sample, Bayes-UCB computes the $(1 - 1/(t(\ln(t))^5)$ quantile of each arm, at time t, and plays the arm with the largest quantile. It was also proven to achieve asymptotical optimal problem-dependent bounds [KCG12]. Bayes-UCB is comparable to Thompson sampling in terms of memory complexity, but it can be slower, as computing a quantile is generally more costly than just sampling from a distribution.

For curiosity, we mention a third Bayesian algorithm, AdBandits introduced in [TdSCC13], mainly because it has not gained any popularity despite its good empirical performance.

AdBandits follows the same assumptions as Thompson sampling and Bayes-UCB, and for instance for Bernoulli problems it uses a Beta posterior on each arm. It uses a random mixture between the two algorithms, and with a certain probability it uses the Thompson sampling decision rule, otherwise it uses the Bayes-UCB decision rule. It was proposed in this article [TdSCC13] and illustrated on empirical simulations, but no theoretical analysis was given. We found that in practice it can achieve very good performance, as illustrated in Section 3.3.

Finally, we would like to mention that a recent work [KT19] presented an approach that closes the gap between the two aforementioned point-of-views, frequentist algorithms such as UCB and Bayesian algorithms such as Thompson sampling. The authors of [KT19] showed that their Randomized Confidence Bound (RCB) algorithm can achieve the same regret upper bounds as UCB_1 or Thompson sampling, with uniformly distributed perturbations. Their RCB algorithm ressembles UCB_1, and builds the same confidence interval $[\mathrm{LCB}_k(t), \mathrm{UCB}_k(t)]$ on arm k at time t (the lower confidence bound is $\mathrm{LCB}_k(t) \doteq \widehat{\mu_k}(t) - \xi_k(t)$), but instead of maximizing the UCB, a random sample is taken on each interval $s_k \sim \mathcal{U}([\mathrm{LCB}_k(t), \mathrm{UCB}_k(t)])$, and the played arm is the one maximizing this sample, *i.e.*, $A(t) \in \arg\max_k s_k$.

2.4.4 Other policies

We can briefly mention other families of policies.

BESA (Best Empirical Sampled Average) was introduced in [BMM14], in order to fix a simple observation: if arm 1 has 100 samples and arm 2 has 50 samples, it should make more sense to only use 50 samples of arm 1 to compute and compare their empirical means. Based on this idea, the BESA algorithm for $K = 2$ arms is quite simple: at each time step, if arms 1 and 2 have respectively n_1 and n_2 samples, and if $n_1 > n_2$ (for instance), first it sub-samples n_2 observations from n_1, then it computes the mean of these observations, and play the arm maximizing this mean (now that there is the same number of observations for both arms, it makes more sense to compare these quantities). It was analyzed partially, for the two-armed case it was proven to be asymptotically optimal for one-dimensional exponential families, and the authors illustrated that it can be very efficient empirically. Analyzing it in the generic case was found to be difficult, and it is still an open problem.

Policies for adversarial bandits. Another important family of bandits algorithms are the algorithms proposed for the adversarial setting [ACBFS02]. The first of them is the Exp3 policy, for "*Exp*onential weights for *Exp*loitation and *Exp*loration", which is an example of the well-known *multiplicative weights update algorithm*[6], a family of algorithms developed since the 1950s, and applied to a wide range of different domains. The Exp3 policy maintains weights

[6] For more details see this blog post by Jeremy Kun, `JeremyKun.com/2017/02/27/the-reasonable-effectiveness-of-the-multiplicative-weights-update-algorithm`, which comes with an interactive online demonstration to help understanding the behavior of the algorithm, hosted at `j2kun.GitHub.io/mwua/` (accessed the 24th of May 2019). More details can be found in the survey [AHK12].

$w_k(t)$ on the K arms, and at time t it samples an arm from the distribution which is a mixture of $\boldsymbol{w}(t) = [w_1(t), \dots, w_K(t)]$ and the uniform distribution, *i.e.*, $\mathbb{P}(A(t) = k) \doteq (1-\gamma) w_k(t) + \gamma \frac{1}{K}$. The weights are initially uniform, *i.e.*, $w_k(0) = 1/K$, and then at each time after observing a reward $Y_{k,t}$ from arm k, Exp3 updates the weight of the chosen arm multiplicatively, using $w_k(t+1) = w_k(t) \times \exp(\widehat{r_k}(t)/(\gamma N_k(t)))$. The weights are then renormalized, to have a sum $\sum_k w_k(t+1) = 1$. This notation $\widehat{r_k}(t)$ means an unbiased estimator of the reward $r(t) = Y_{k,t}$, that is, $\widehat{r_k}(t) = r(t)/\mathbb{P}(A(t) = k)$. Like for the ε-greedy policy, the parameter γ can be constant, or can be a non-increasing sequence $(\gamma_t)_{t\in\mathbb{N}^*}$. It was proven in [ACBFS02] that Exp3 with constant parameter γ achieves $R_T = \mathcal{O}\left(\sqrt{KT\ln(T)}\right)$ problem-independent regret, while using a decreasing sequence, such as $\gamma_t = \sqrt{\ln(K)/K \ln(t)/t}$ gives an order-optimal problem-independent regret upper-bound of $\mathcal{O}\left(\sqrt{KT}\right)$. A short proof of this results and more details on Exp3 are given in [BCB12]. Many other policies have been proposed for adversarial bandits, like Follow-the-Perturbed-Leader or Follow-the-Regularized-Leader for linear adversarial bandits, and we refer to Parts III and VI of [LS19] for more details.

Hybrid policies. As explained before, in the last few years, the MAB research community has been interested in finding algorithms which achieves both a problem-dependent logarithmic and a minimax upper-bounds. We present here some solutions that are index-based (see Algorithm 2.2). The first example is MOSS from [AB09], or kl-UCB^{++} in [MG17]. If we denote $g(t,T,K) = \ln^+(y(t)(1+\ln^+(y(t))^2))$, and $y(t) = \frac{T}{Kt}$, with $\ln^+(x) = \max(0, \ln(x))$, then they use the following indexes:

$$U_k^{\text{MOSS}}(t) \doteq \frac{X_k(t)}{N_k(t)} + \max\left(0, \sqrt{\frac{\ln\left(\frac{t}{KN_k(t)}\right)}{N_k(t)}}\right).$$

$$U_k^{\text{kl-UCB}^{++}}(t) \doteq \sup_{q\in[0,1]} \left\{ q : \mathrm{kl}\left(\frac{X_k(t)}{N_k(t)}, q\right) \leq \frac{g(N_k(t), T, K)}{N_k(t)} \right\}.$$

Both algorithms are index policies, but they are not anytime. For any K, T, and problem instances I in a certain family $\mathcal{I}$ (*e.g.*, with Bernoulli distributions), they enjoy regret bounds like the following, for two constants $c_{\mathcal{A}}$ and $c'_{\mathcal{A}}$ that depend on the algorithm, and a constant C_I that depend on the problem instance only: $R_T^{\mathcal{A}} \leq \min(c_{\mathcal{A}}\sqrt{KT}, c'_{\mathcal{A}} C_I \ln(T))$.

More recently, the kl-UCB-switch from [GHMS18] is the first algorithm to be proven to enjoy simultaneously a distribution-free regret bound of optimal order $\sqrt{KT}$, and a distribution-dependent regret bound of optimal order as well, that is, matching the $C_I \ln(T)$ lower bound by [LR85]. It is an index policy that uses modified versions of the indexes of kl-UCB$^+$ for arms that are not sampled much (see [Hon19] for more details), and of MOSS$^+$ for the other arms. If the two indexes are defined by $U_k^{\text{MOSS}+}(t) \doteq \frac{X_k(t)}{N_k(t)} + \max\left(0, \sqrt{\ln\left(T/(KN_k(t))\right)/N_k(t)}\right)$ and $U_k^{\text{KL-UCB}+}(t) \doteq \sup_{q\in[0,1]} \left\{q : \mathrm{kl}\left(\frac{X_k(t)}{N_k(t)}, q\right) \leq \ln(T/(KN_k(t)))/N_k(t)\right\}$, and if $f(T,K) = \lfloor (T/K)^\gamma \rfloor$

for $\gamma = 1/5$, then kl-UCB-switch uses the index defined as follows:

$$U_k^{\text{kl-UCB-switch}}(t) \doteq \begin{cases} U_k^{\text{KL-UCB+}}(t) & \text{if } N_k(t) \leq f(T,K), \\ U_k^{\text{MOSS+}}(t) & \text{if } N_k(t) > f(T,K). \end{cases}.$$

Finally, it is worth mentioning the Exp3^{++} algorithm, as a recent variant of Exp3, proposed by [SL17], that uses an adaptive tuning of its parameters. It is anytime, and it was also proven to obtain good "best-of-both-world" performance.

2.4.5 Comparison of the main algorithms on the example Bernoulli problem from the online interactive demonstration

We finish this Section 2.4 by illustrating the performance of different algorithms, on the small MAB problem used in the online demonstration shown above (see Figures 2.2 and 2.3). We consider $K = 5$ arms following Bernoulli distributions, of means $\mu_1 = 0.6, \mu_2 = 0.2, \mu_3 = 0.55, \mu_4 = 0.7, \mu_5 = 0.5$ (unknown to the algorithms), and first an horizon $T = 100$ (like in the online demo) then a large value of $T = 10000$. The best arm is μ_4 and the smallest gap is $\Delta = 0.1$, thus a lower-bound on the gaps is taken as $d = \Delta$.

The different algorithms being compared are the following, in the order they were presented above: first we include the pure-exploration and pure-exploitation algorithms, then the ε-greedy (with $\varepsilon_t = \varepsilon_0/t$ and $\varepsilon_0 = 6K/d^2 = 3000$) and Explore-then-Commit (with $T_0 = K4/d^2 \ln(d^2T/4) = 2000\ln(T/400)$) algorithms from Section 2.4.1, which both uses a tuning based on a prior knowledge of Δ (that gives large values of ε_0 and T_0 yielding linear regret for small horizon). We then include efficient algorithms, that do not need prior knowledge of the T (they are anytime) nor Δ (their are robust and efficient on unknown problems): UCB$_1$ (with $\alpha = 1$), kl-UCB (with Bernoulli $\mathrm{kl} = \mathrm{KL}_B$), and Thompson sampling (with Beta posteriors and flat uniform priors).

We first give in Table 2.1 the cumulated rewards as well as the estimated regrets of the different algorithms, *for one simulation*. First, the rewards are computed as $\frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T} r_{A(t)}^{(j)}(t)$, where $r_k^{(j)}(t)$ means the reward stream of the different runs $j = 1, \dots, N$. Then the estimated regret is *not* computed as $\frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T} r_{k^*}^{(j)}(t) - r_{A(t)}^{(j)}(t)$, as this first formula could give a negative regret and is more noisy, but as $\mu^* T - \frac{1}{N}\sum_{j=1}^{N}\sum_{k=1}^{K}\mu_k N_k^{(j)}(T)$, as this later formula gives a more robust estimator, as noted the proof of Proposition 2.4 (in Page 36 above). It shows that the performance can fluctuate a lot, as on both cases the pure exploitation strategy was very lucky and obtains close-to-optimal performance, even though we proved that it suffers from a linear mean regret.

Then in Table 2.2 we show the *results of the average on $N = 1000$ independent runs*. We also include in Figures 2.4, 2.5 below the mean cumulated rewards and mean regrets as functions

Algorithms	rewards	regret for $T = 100$	rewards	regret for $T = 10000$
Pure exploration	46	20.9	5171	1886.4
Pure exploitation	**65**	**1.15**	**6997**	**0.95**
ε-greedy	50	20.9	6657	382.2
Explore-then-Commit	49	15	5766	1199.8
UCB_1	63	10.3	6920	78.4
kl-UCB	56	12.4	6922	70.2
Thompson sampling	51	12.2	6949	44.2

Table 2.1 – Cumulated rewards and regret, for horizons $T = 100$ and $T = 10000$, *for only one run of the simulation*, for different algorithms.

of the current time step t, for $T = 10000$ (and $N = 1000$). The reward of the best arm is also plotted in Figure 2.4, and it serves as an *upper-bound* to illustrate convergence of the average rewards (*i.e.*, $\frac{1}{t}(r(1) + \cdots + r(t))$), which should converge to μ^* when $t \to \infty$ for efficient algorithms. Conversely, we also add the Lai & Robbins' logarithmic *lower-bound* on regret ($C_I \ln(t)$ from Theorem 2.8) on the regret plot in Figure 2.5. It should not be surprising that the black line of the lower-bound is *above* the values of the regret as the lower-bound is only asymptotic. We can clearly identify the behavior of the different strategies: the three efficient strategies indeed achieve a sub-linear regret, that looks logarithmic as supported by the theory, while the naive strategies achieve linear regret. The two simple strategies also achieve sub-linear regret but are less efficient: both ε-greedy and Explore-then-Commit starts as pure exploration, respectively until ε_0/t becomes smaller than 1 and until $t > T_0$, then they usually identify correctly the best arm and start to catch up with the best policies.

Algorithms	rewards	regret for $T = 100$	rewards	regret for $T = 10000$
Pure exploration	51	19	5099	1900
Pure exploitation	62	8.3	6349	653
ε-greedy	51	18.7	6460	542
Explore-then-Commit	57	12.9	5795	1207
UCB_1	**60**	**9.8**	6921	79
kl-UCB	61	9.3	6927	72
Thompson sampling	**60**	**9.6**	**6951**	**49**

Table 2.2 – Cumulated rewards and regret, for horizons $T = 100$ and $T = 10000$, *averaged over $N = 1000$ independent simulations* ($j = 1, \dots, N$), for different algorithms.

Summary of this review of MAB algorithms. We gave an overview of the most well known families of algorithms for multi-armed bandits, focussing on algorithms designed and analyzed for the single-player stationary and stochastic MAB model. Except the naive and simple strategies given as introductory examples, most of the policies presented above are order-optimal for Bernoulli distributed problems or bounded rewards (or one-dimensional exponential families), and some of them are known to be optimal in different settings. To clearly understand the different in their empirical behaviors, we illustrated the performance of two naive and two simple strategies against three efficient ones, on the example of Bernoulli-

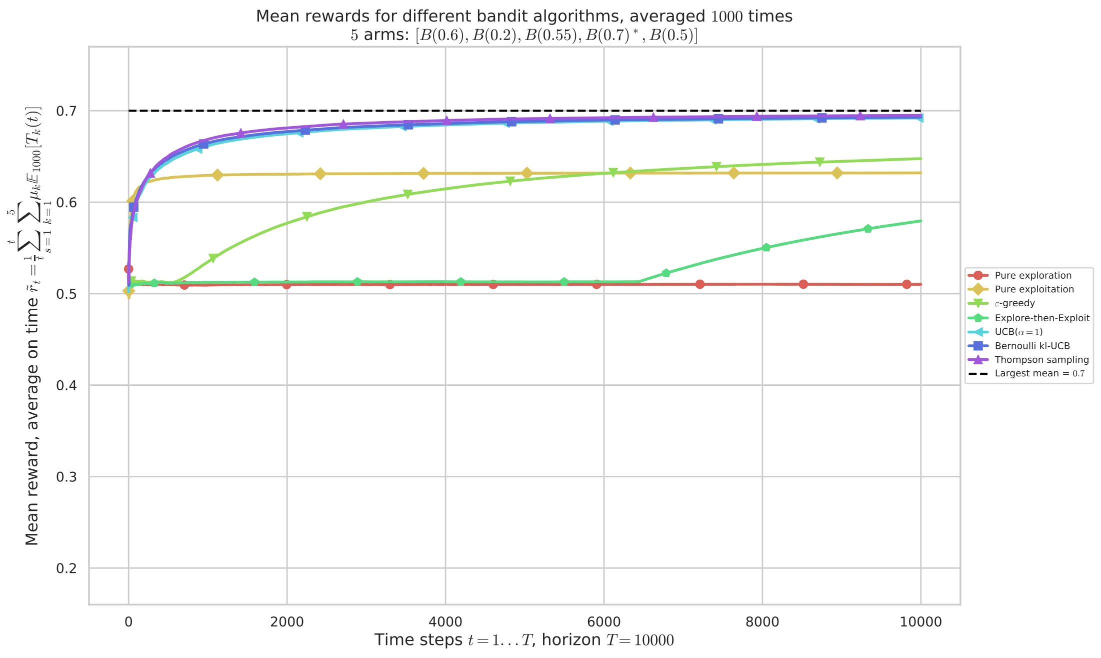


Figure 2.4 – Average of the cumulated rewards, as function of t, for $T = 10000$ and $N = 1000$.

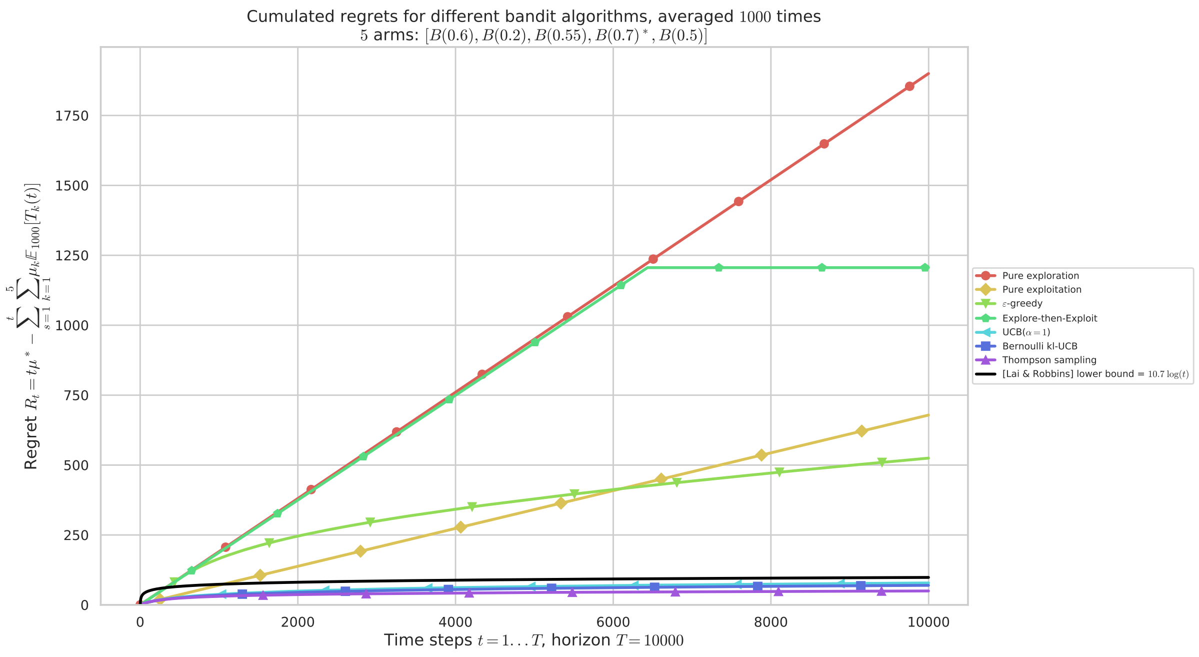


Figure 2.5 – Mean regret R_t as function of t for $T = 10000$ and $N = 1000$.

distributed problem used for the online demonstration from Section 2.1 (see Figures 2.2 and 2.3). This also serves as a first example of numerical simulation performed with our library SMPyBandits, before the large-scale experiments of Section 3.3 below. In the rest of this thesis, we mainly use the UCB_1 or Thompson sampling algorithms when simplicity and realistic deployment is favored (*i.e.*, Chapter 5), and the kl-UCB algorithm when we give mathematical developments (*i.e.*, Chapters 6 and 7).

2.5 Conclusion

In this chapter, we presented the multi-armed bandit model, focussing on a finite number of arms, and real-valued rewards. Our main focus is on one-dimensional exponential families of distributions, and on stochastic and stationary problems. By showcasing an interactive demonstration made available online, we presented the notations used in all this thesis. We also showed a small numerical simulation, comparing some algorithms on a small Bernoulli-distributed problem, using our Python open-source library, SMPyBandits, that is presented in details in the next Chapter 3.

We presented the simplest MAB model studied in this chapter, by focussing on one agent playing a stationary game. Both hypotheses are removed or weaken in the rest of this manuscript, first by considering players facing a stationary problem in a decentralized way in Chapter 6, and then by considering a single player facing a non-stationary or a piece-wise stationary problem in Chapter 7. For both directions, we present in the two final chapters natural extensions of the base model, and we detail our contributions that obtained state-of-the-art results for the two problems of stationary multi-players and piece-wise stationary bandits.

We take another approach in Chapter 5, where the MAB model is generalized to study decentralized learning of a large set of independent players, all having different activation times. This extension is significantly harder than the two previously evoked ones, and we were unfortunately unable to obtain any strong theoretical results under these loose hypotheses. This model is however more generic and as such it was found suitable for applications to Internet of Things (IoT) networks, where arms model orthogonal wireless channels, players model communicating devices (*i.e.*, IoT end-devices) and rewards model successes or failures of a wireless packet sent by a device.

Possible future works. We focused in this thesis on finite-arms and one-dimensional bandit problems, and thus two possible directions of future works could be to extend our works to MAB models with either multidimensional rewards, like contextual bandits, or infinite arms, like Lipschitz bandits.

Chapter 3

SMPyBandits: a state-of-the-art Python library to simulate MAB problems

SMPyBandits is a Python package I have developed since the beginning of my PhD. It is designed to allow easy and fast numerical simulations on single-player and multi-players Multi-Armed Bandits (MAB) algorithms. It is the most exhaustive open-source implementation of state-of-the-art algorithms and different kinds of MAB models. This chapter details the organization of the library, what it implements in terms of arm distributions, models, algorithms and visualizations. Then we use it to perform a numerical comparisons of the main state-of-the-art single-player MAB algorithms, and a study to compare time and memory costs of the main algorithms.

Contents

3.1 Introduction

SMPyBandits is a Python package I have developed since the beginning of my PhD. It is designed to allow easy and fast numerical simulations on single-player and multi-players Multi-Armed Bandits (MAB) algorithms. This library is by far the most exhaustive open-source implementation of state-of-the-art algorithms and different kinds of MAB models. It aims at being exhaustive, simple to use and maintain, and has a clean and well documented codebase, and it uses the popular open-source Python language. It allows fast prototyping of simulations, with an easy configuration system and command-line options to customize experiments. Experiment results are saved in an optimized binary format (HDF5) as well as high quality plots of many useful visualizations. More than two years of active development have shown how easy it can be to add new algorithms, new arm distributions, and new bandit models (*e.g.*, Markov or non-stationary). It is hosted on GitHub, and uses the state-of-the-art development technologies, by using two online services of continuous integration to run automated tests and to build its online documentation.

SMPyBandits stands for *S*ingle- and *M*ulti-*P*layer *Bandits* in P*y*thon. The library does not aim at being blazing fast or optimized in terms of memory usage, as it comes with a pure Python implementation [Fou17], and uses only standard open-source Python packages (*i.e.*, no hard to install dependencies). Some critical parts are also available as a `C` Python extension, and the just-in-time Numba compiler [I^{+}17] is used whenever it is possible, so we can note that we optimized the time efficiency of what could be (easily) optimized. However if simulation speed really matters, one should rather refer to less exhaustive but faster implementations, like for example [Lat16a] in `C++` or [Raj17] in Julia. Note that both are not maintained anymore, and contain just a few algorithms for the simple stationary MAB model.

In this Chapter 3, we start by presenting in Section 3.2 the organization of the library. We use the library to compare the most famous and most efficient single-player MAB algorithms in Section 3.3, then we discuss in Section 3.4 about the time and memory costs of MAB algorithms. The take away messages are two-fold: from a practical point-of-view, it is usually advised to use the simplest algorithm and we advise to use UCB (as we do in the next Chapter 5), and that for theoretical works where optimality of the MAB algorithm matters, we advise to use kl-UCB (as we do in the next Chapters 6 and 7). Finally, in Appendix 3.6 we include some small files showing how to use the library, and we conclude with some details regarding the use of parallel computing to speed-up simulations.

References. The code for SMPyBandits is hosted at `GitHub.com/SMPyBandits/SMPyBandits`, its documentation is at `SMPyBandits.GitHub.io`, and all the library is freely publicly released under the MIT open-source License. This chapter is based on our article [Bes18].

3.2 Presentation of the library

We start by explaining how SMPyBandits simulates MAB problems, by detailing its components: MAB problems (single- or multi-players), and reward distributions. We then detail how it implements MAB algorithms (*e.g.*, UCB), and what kind of information is displayed, saved and plotted after a batch of simulations of bandit problems. The main goal of the library is to be easily able to simulate MAB algorithms (*e.g.*, three algorithms like UCB, kl-UCB and Thompson sampling) on one or more bandit models (single- or multi-players) defined by the number of arms K and the distributions ν_k, for some time from $t = 1$ to the horizon T. After the simulation, the library then displays statistical summary of the (mean) rewards accumulated by each algorithm, as well as regret and other visualizations. The same problem is simulated for N independent repetitions (*e.g.*, $N = 100$) in order to show mean results with a low variance.

3.2.1 Single- and multi-players MAB problems

For the classical single-player stochastic MAB model, as defined in Chapter 2, a stochastic MAB problem is defined by $K > 1$ distributions ν_k (also called arms), used to generate the *i.i.d.* samples $Y_{k,t} \sim \nu_k, \forall t$. An agent chooses arm $A(t) \in [K]$ at time t and observes the reward $r(t) = Y_{A(t),t}$ without knowing the other (hidden) rewards. Her goal is to maximize $\sum_{t=1}^{T} r(t)$ by sequentially exploring the K arms, and she essentially has to find and exploit the best one as fast as possible.

Simulation loop for single-player MAB. Any simulation library targeting single-player bandit problems must implement at least three components: reward distributions, MAB algorithm, and a simulation loop that essentially looks like this. First, initialize the MAB problem and one or more algorithms, Then, for $t = 1$ to $t = T$ (known before hand), repeat the following block (for each algorithm). Ask algorithm $\mathcal{A}$ her chosen arm $A(t)$, then sample a (random) reward $r(t)$ (*i.i.d.*) from distribution $\nu_{A(t)}$, and finally feeds the observation $(A(t), r(t))$ to the algorithm, At the end, compute the cumulated reward, the regret, plot visualizations etc. Note that the second step is repeated a large number of times (*e.g.*, $N = 100$), in order to study *mean* regret and not only the regret in one single trajectory. If one wants to compare algorithms on a same problem, it is possible to sample all the rewards $(Y_{k,t})_{k\in[K],t\in[T]}$ before-hand, and store them so that for each repetition of the simulation, the randomness from the environment (*i.e.*, the rewards) has the same impact on all the algorithms.

Simulation loop for multi-players MAB. For Cognitive Radio dealing with OSA problems and other applications, a well-studied extension is to consider $M \geq 2$ players, interacting on the same K arms. Whenever two or more players select the same arm at the same time (*e.g.*, for OSA, the same frequency channel), they all suffer from a radio collision. Different collision models have been proposed, and the simplest one consists in giving a 0 reward to each colliding players. Without any centralized supervision or coordination between players, they must learn to access the M best resources (*i.e.*, arms with highest means) without collisions. We refer to Chapter 6 which introduces the multi-players bandit model.

SMPyBandits implements all collision models found in the literature (in the module `Environment.CollisionModels`), as well as all the algorithms known by the authors (in the module `PoliciesMultiPlayers`). It includes `rhoRand` from [AMTA11], `MEGA` from [AM15], `MusicalChair` from [RSS16], and our state-of-the-art algorithms `RandTopM` and `MCTopM` from [BK18a]. For comparison, realistic (*e.g.*, UCB for multiple play) or full-knowledge centralized algorithms are also implemented.

Any simulation library targeting multi-players bandit problems must implement at least another component: a simulation loop that essentially looks like this. First, initialize the MAB problem with M players, and one or more cohorts of M players (one player is one algorithm, usually M times the same one), Then, for $t = 1$ to $t = T$ (known before hand), repeat the following block (for each algorithm). Ask each player $\mathcal{A}^{(j)}$ her chosen arm $A^{(j)}(t)$, then query the collision model[1] to know which player will get a zero reward (for a collision) and which player will get a random reward from the environment. The sample (random) feedback $Y_{k,t}$ (*i.i.d.*) from distributions ν_k, and compute the rewards $r^{(j)}(t)$ from the random feedback and the collision model. Finally feeds the observation $(A^{(j)}(t), Y_{A^{(j)}(t)}(t), r^{(j)}(t))$ to player j, for all the M players, At the end, compute the accumulated reward, the regret, plot visualizations etc.

3.2.2 Reward distributions

We focus on one dimensional distributions, implemented in the `Arms` module. The library supports discrete distributions: *Bernoulli* (`Bernoulli`), *binomial* (`Binomial`), *Poisson* (`Poisson`), and a generic *discrete* distribution (`DiscreteArm`), as well as continuous distributions, which can be truncated to an interval $[a, b]$ or have unbounded support ($\mathbb{R}$): *exponential* (`Exponential`), *gamma* (`Gamma`), *Gaussian* (`Gaussian`) and *uniform* (`Uniform`). The default is to use the same distribution for the K arms, as this is the setting studied in the literature, but it also possible to mix them. For instance the following code in Code 3.1 creates three arms, following a Bernoulli distribution with respective means $0.1, 0.5, 0.9$, and a `MAB` object which encapsulates the problem. We give another example for truncated Gaussian distributions, and a visualization of a histogram of 10000 rewards obtained from the arms, below in Figure 3.1.

For more details, an interested reader can refer to the following Jupyter notebook [K+16]: `SMPyBandits.GitHub.io/notebooks/Easily_creating_MAB_problems.html`.

[1] The default collision model is the most widely studied in the literature, where a player encounters a collision (*i.e.*, received a zero reward $Y^{(j)}(t) = 0$) if she is not the only one to have chosen an arm $k = A^{(j)}(t)$, otherwise $Y^{(j)}(t) = r_{A^{(j)}(t)}(t)$ if she is the only one playing this arm.

```python
from SMPyBandits.Arms import *
arm1 = Bernoulli(0.1)
arm2 = Bernoulli(0.5)
arm3 = Bernoulli(0.9)

means = [0.1, 0.5, 0.9]
from SMPyBandits.Environment import MAB
bernoulli_problem = MAB([arm1, arm2, arm3])
# but also
bernoulli_problem = MAB({'arm_type': Bernoulli, 'params': means})

gaussian_problem = MAB({'arm_type': Gaussian, 'params': means})
gaussian_problem.plotHistogram()  # display the histogram shown below
```

Code Example 3.1 – Example of Python code to create Bernoulli and Gaussian arms, a MAB problem with $K = 3$ arms, a plot a histogram of rewards, with SMPyBandits

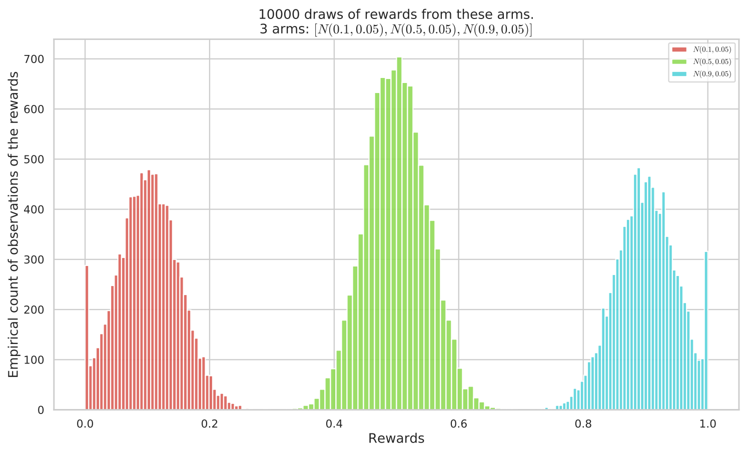


Figure 3.1 – Histogram of 10000 *i.i.d.* rewards obtained from three arms with a Gaussian distribution truncated to $[0, 1]$, of respective means 0.1, 0.5 and 0.9.

We have not yet added support for higher dimensional distributions of rewards, *e.g.*, linear bandit, but it would be an interesting extension of SMPyBandits. However, note that our library does support finite-state real-valued Markov MAB models, where arms correspond to Markov chains [Nor98], in the two *rested*/*restless* variants, as introduced by [AVW87b].

3.2.3 Multi-Armed Bandits algorithms

SMPyBandits is a complete open-source implementation of single-player (classical) bandit algorithms, containing over 65 algorithms (in the module `Policies`). It uses a well-designed hierarchical structure and class inheritance scheme (as detailed on the various UML diagrams shown on the `uml_diagrams` folder) to minimize redundancy in the codebase. For instance, most existing algorithms are index policies (see Algorithm 2.2), and new ones are easily written by inheriting from the `IndexPolicy` class (`Policies.IndexPolicy`). They compute an index $I_k(t) \in \mathbb{R}$ for each arm k at time t, and play $A(t) = \arg\max_k I_k(t)$. For instance the code specific to UCB_1 is as short as this (and fully documented):

```python
from numpy import sqrt, log
from .IndexPolicy import IndexPolicy

class UCB(IndexPolicy):
  """ The UCB policy for bounded bandits.
  Reference: [Lai & Robbins, 1985]. """

  def computeIndex(self, arm):
    r""" Compute the current index, at time t and
    after :math:`N_k(t)` pulls of arm k:

    .. math::

        I_k(t) = \frac{X_k(t)}{N_k(t)}
        + \sqrt{\frac{2 \log(t)}{N_k(t)}}.
    """
    if self.pulls[arm] < 1:   # forced exploration
      return float('+inf')    # in the first steps
    else:                     # or compute UCB index
      estimated_mean = (self.rewards[arm] / self.pulls[arm])
      exploration_bias = sqrt((2 * log(self.t)) / self.pulls[arm])
      return estimated_mean + exploration_bias
```

Code Example 3.2 – Code defining the UCB algorithm, as a simple example of an Index Policy.

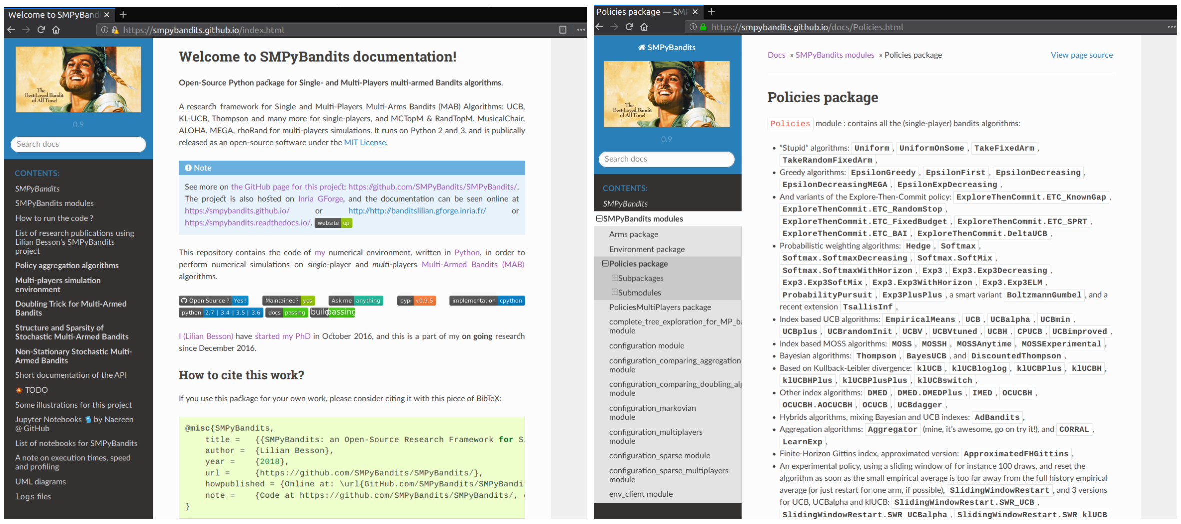
3.2.4 Summary of the features

With this numerical framework, simulations can run on a single CPU or a multi-core machine using joblib [Var17] (see Appendix 3.6.6), and summary plots are automatically saved as high-quality PNG, PDF and EPS, using matplotlib [Hun07] and seaborn [W+17]. Raw data from each simulation is also saved in an HDF5 file, using h5py [C+18], an efficient and compressed binary format, to allow easy post-mortem manipulation of any simulation results. Making new simulations is very easy, one only needs to write a configuration script (`configuration.py`), without needing a complete knowledge of the internal code architecture.

3.2.5 Documentation

A complete documentation, for each algorithm and the entire codebase, is available online at SMPyBandits.GitHub.io. It uses the Sphinx software [B+18], and the content is directly written in the Python files as docstrings, so users have access to the documentation from within their IDE or the Python console. The most interesting component of the library is the many MAB algorithms being implemented: for each of them, the documentation gives a reference to a research paper (*e.g.*, [KCG12] for `BayesUCB`), as well as a bird-eye view of its behavior. For most algorithms, especially for index policies, the internal variables of the implementation are carefully linked to the notations of each paper, and formulas explaining the way the algorithm selects its arm are usually given. Whenever it is needed, we also included warnings or information about the empirical performance of the more costly algorithms.

The documentation also contains extensive examples of all intermediate numerical functions, that are also used as tests (using `doctest.testmod` from the standard library). For example, Figures 3.2 and 3.3 below show four screenshots from the documentation. We show the main page, the list of algorithms in the `Policies` module, an example of the documentation of one algorithm (for the kl-UCB-Switch algorithm from [GHMS18]), and a page detailing the API and organization of the library as well as instructions to follow if someone wants to implement a new model or a new algorithm.



Welcome to SMPyBandi
https://smpybandits.github.io/index.html
The Best-Loved Bandit of All Time!
0.9
Search docs
CONTENTS:
SMPyBandits
SMPyBandits modules
How to run the code ?
List of research publications using Lilian Besson's SMPyBandits project
Policy aggregation algorithms
Multi-players simulation environment
Doubling Trick for Multi-Armed Bandits
Structure and Sparsity of Stochastic Multi-Armed Bandits
Non-Stationary Stochastic Multi-Armed Bandits
Short documentation of the API
TODO
Some illustrations for this project
Jupyter Notebooks by Naereen @ GitHub
List of notebooks for SMPyBandits
A note on execution times, speed and profiling
UML diagrams
logs files
Open Source ? Yes!
Maintained? yes
Ask me anything
pypi v0.9.5
implementation cpython
python 2.7 | 3.4 | 3.5 | 3.6
docs passing
build passing
website up
Policies package — SMP
https://smpybandits.github.io/docs/Policies.html
SMPyBandits
Docs » SMPyBandits modules » Policies package
View page source
Arms package
Environment package
Policies package
Subpackages
Submodules
PoliciesMultiPlayers package
complete_tree_exploration_for_MP_b module
configuration module
configuration_comparing_aggregation module
configuration_comparing_doubling_al module
configuration_markovian module
configuration_multiplayers module
configuration_sparse module
configuration_sparse_multiplayers module
env_client module

Welcome to SMPyBandits documentation!

Open-Source Python package for Single- and Multi-Players multi-armed Bandits algorithms.

A research framework for Single and Multi-Players Multi-Arms Bandits (MAB) Algorithms: UCB, KL-UCB, Thompson and many more for single-players, and MCTopM & RandTopM, MusicalChair, ALOHA, MEGA, rhoRand for multi-players simulations. It runs on Python 2 and 3, and is publically released as an open-source software under the MIT License.

Note

See more on the GitHub page for this project: https://github.com/SMPyBandits/SMPyBandits/. The project is also hosted on Inria GForge, and the documentation can be seen online at https://smpybandits.github.io/ or http://http://banditslilian.gforge.inria.fr/ or https://smpybandits.readthedocs.io/.

This repository contains the code of my numerical environment, written in Python, in order to perform numerical simulations on *single*-player and *multi*-players Multi-Armed Bandits (MAB) algorithms.

I (Lilian Besson) have started my PhD in October 2016, and this is a part of my **on going** research since December 2016.

How to cite this work?

If you use this package for your own work, please consider citing it with this piece of BibTeX:

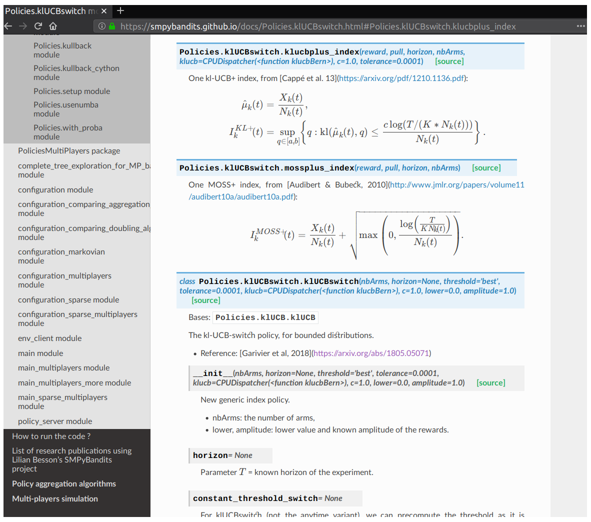
```
@misc{SMPyBandits,
    title =   {{SMPyBandits: an Open-Source Research Framework for S
    author =  {Lilian Besson},
    year =    {2018},
    url =     {https://github.com/SMPyBandits/SMPyBandits/},
    howpublished = {Online at: \url{GitHub.com/SMPyBandits/SMPyBandi
    note =    {Code at https://github.com/SMPyBandits/SMPyBandits/,
}
```

Policies package

`Policies` module : contains all the (single-player) bandits algorithms:

- "Stupid" algorithms: `Uniform`, `UniformOnSome`, `TakeFixedArm`, `TakeRandomFixedArm`,
- Greedy algorithms: `EpsilonGreedy`, `EpsilonFirst`, `EpsilonDecreasing`, `EpsilonDecreasingMEGA`, `EpsilonExpDecreasing`,
- And variants of the Explore-Then-Commit policy: `ExploreThenCommit.ETC_KnownGap`, `ExploreThenCommit.ETC_RandomStop`, `ExploreThenCommit.ETC_FixedBudget`, `ExploreThenCommit.ETC_SPRT`, `ExploreThenCommit.ETC_BAI`, `ExploreThenCommit.DeltaUCB`,
- Probabilistic weighting algorithms: `Hedge`, `Softmax`, `Softmax.SoftmaxDecreasing`, `Softmax.SoftMix`, `Softmax.SoftmaxWithHorizon`, `Exp3`, `Exp3.Exp3Decreasing`, `Exp3.Exp3SoftMix`, `Exp3.Exp3WithHorizon`, `Exp3.Exp3ELM`, `ProbabilityPursuit`, `Exp3PlusPlus`, a smart variant `BoltzmannGumbel`, and a recent extension `TsallisInf`,
- Index based UCB algorithms: `EmpiricalMeans`, `UCB`, `UCBalpha`, `UCBmin`, `UCBplus`, `UCBrandomInit`, `UCBV`, `UCBVtuned`, `UCBH`, `CPUCB`, `UCBimproved`,
- Index based MOSS algorithms: `MOSS`, `MOSSH`, `MOSSAnytime`, `MOSSExperimental`,
- Bayesian algorithms: `Thompson`, `BayesUCB`, and `DiscountedThompson`,
- Based on Kullback-Leibler divergence: `klUCB`, `klUCBloglog`, `klUCBPlus`, `klUCBH`, `klUCBHPlus`, `klUCBPlusPlus`, `klUCBswitch`,
- Other index algorithms: `DMED`, `DMED.DMEDPlus`, `IMED`, `OCUCBH`, `OCUCBH.AOCUCBH`, `OCUCB`, `UCBdagger`,
- Hybrids algorithms, mixing Bayesian and UCB indexes: `AdBandits`,
- Aggregation algorithms: `Aggregator` (mine, it's awesome, go on try it!), and `CORRAL`, `LearnExp`,
- Finite-Horizon Gittins index, approximated version: `ApproximatedFHGittins`,
- An experimental policy, using a sliding window of for instance 100 draws, and reset the algorithm as soon as the small empirical average is too far away from the full history empirical average (or just restart for one arm, if possible), `SlidingWindowRestart`, and 3 versions for UCB, UCBalpha and klUCB: `SlidingWindowRestart.SWR_UCB`, `SlidingWindowRestart.SWR_UCBalpha`, `SlidingWindowRestart.SWR_klUCB`

Figure 3.2 – Screenshots from two pages of the documentation: the homepage (SMPyBandits.GitHub.io), and a list of all the algorithms in the `Policies` module.


Policies.klUCBswitch m
https://smpybandits.github.io/docs/Policies.klUCBswitch.html#Policies.klUCBswitch.klucbplus_index
Policies.kullback module
Policies.kullback_cython module
Policies.setup module
Policies.usenumba module
Policies.with_proba module
PoliciesMultiPlayers package
complete_tree_exploration_for_MP_b module
configuration module
configuration_comparing_aggregation module
configuration_comparing_doubling_al module
configuration_markovian module
configuration_multiplayers module
configuration_sparse module
configuration_sparse_multiplayers module
env_client module
main module
main_multiplayers module
main_multiplayers_more module
main_sparse_multiplayers module
policy_server module
How to run the code ?
List of research publications using Lilian Besson's SMPyBandits project
Policy aggregation algorithms
Multi-players simulation

`Policies.klUCBswitch.klucbplus_index`(*reward, pull, horizon, nbArms, klucb=CPUDispatcher(<function klucbBern>), c=1.0, tolerance=0.0001*) [source]

One kl-UCB+ index, from [Cappé et al. 13](https://arxiv.org/pdf/1210.1136.pdf):

$$\hat{\mu}_k(t) = \frac{X_k(t)}{N_k(t)},$$
$$I_k^{KL+}(t) = \sup_{q\in[a,b]} \left\{ q : \mathrm{kl}(\hat{\mu}_k(t), q) \leq \frac{c \log(T/(K * N_k(t)))}{N_k(t)} \right\}.$$

`Policies.klUCBswitch.mossplus_index`(*reward, pull, horizon, nbArms*) [source]

One MOSS+ index, from [Audibert & Bubeck, 2010](http://www.jmlr.org/papers/volume11/audibert10a/audibert10a.pdf):

$$I_k^{MOSS+}(t) = \frac{X_k(t)}{N_k(t)} + \sqrt{\max\left(0, \frac{\log\left(\frac{T}{K N_k(t)}\right)}{N_k(t)}\right)}.$$

class **`Policies.klUCBswitch.klUCBswitch`**(*nbArms, horizon=None, threshold='best', tolerance=0.0001, klucb=CPUDispatcher(<function klucbBern>), c=1.0, lower=0.0, amplitude=1.0*) [source]

Bases: `Policies.klUCB.klUCB`

The kl-UCB-switch policy, for bounded distributions.

- Reference: [Garivier et al, 2018](https://arxiv.org/abs/1805.05071)

`__init__`(*nbArms, horizon=None, threshold='best', tolerance=0.0001, klucb=CPUDispatcher(<function klucbBern>), c=1.0, lower=0.0, amplitude=1.0*) [source]

New generic index policy.

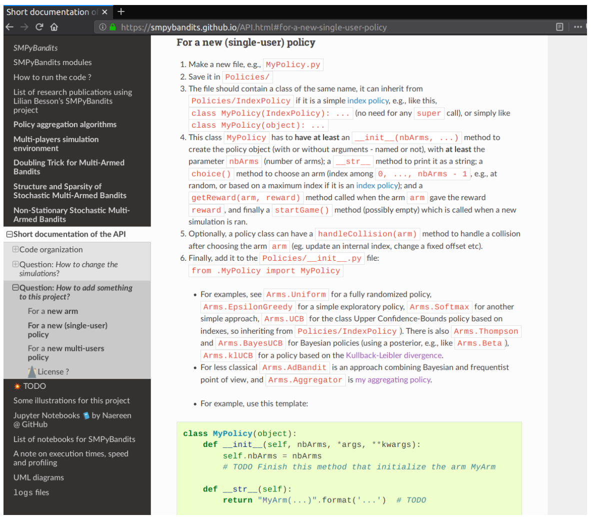
- nbArms: the number of arms,
- lower, amplitude: lower value and known amplitude of the rewards.

`horizon`= *None*

Parameter T = known horizon of the experiment.

`constant_threshold_switch`= *None*

For klUCBswitch (not the anytime variant), we can precompute the threshold as it is


Short documentation o
https://smpybandits.github.io/API.html#for-a-new-single-user-policy
SMPyBandits
SMPyBandits modules
How to run the code ?
List of research publications using Lilian Besson's SMPyBandits project
Policy aggregation algorithms
Multi-players simulation environment
Doubling Trick for Multi-Armed Bandits
Structure and Sparsity of Stochastic Multi-Armed Bandits
Non-Stationary Stochastic Multi-Armed Bandits
Short documentation of the API
Code organization
Question: How to change the simulations?
Question: How to add something to this project?
For a new arm
For a new (single-user) policy
For a new multi-users policy
License ?
TODO
Some illustrations for this project
Jupyter Notebooks by Naereen @ GitHub
List of notebooks for SMPyBandits
A note on execution times, speed and profiling
UML diagrams
logs files

For a new (single-user) policy

1. Make a new file, e.g., `MyPolicy.py`
2. Save it in `Policies/`
3. The file should contain a class of the same name, it can inherit from `Policies/IndexPolicy` if it is a simple index policy, e.g., like this, `class MyPolicy(IndexPolicy): ...` (no need for any `super` call), or simply like `class MyPolicy(object): ...`
4. This class `MyPolicy` has to **have at least** an `__init__(nbArms, ...)` method to create the policy object (with or without arguments - named or not), with **at least** the parameter `nbArms` (number of arms); a `__str__` method to print it as a string; a `choice()` method to choose an arm (index among `0, ..., nbArms - 1`, e.g., at random, or based on a maximum index if it is an index policy); and a `getReward(arm, reward)` method called when the arm `arm` gave the reward `reward`, and finally a `startGame()` method (possibly empty) which is called when a new simulation is ran.
5. Optionally, a policy class can have a `handleCollision(arm)` method to handle a collision after choosing the arm `arm` (eg. update an internal index, change a fixed offset etc).
6. Finally, add it to the `Policies/__init__.py` file: `from .MyPolicy import MyPolicy`

- For examples, see `Arms.Uniform` for a fully randomized policy, `Arms.EpsilonGreedy` for a simple exploratory policy, `Arms.Softmax` for another simple approach, `Arms.UCB` for the class Upper Confidence-Bounds policy based on indexes, so inheriting from `Policies/IndexPolicy`). There is also `Arms.Thompson` and `Arms.BayesUCB` for Bayesian policies (using a posterior, e.g., like `Arms.Beta`), `Arms.klUCB` for a policy based on the Kullback-Leibler divergence.
- For less classical `Arms.AdBandit` is an approach combining Bayesian and frequentist point of view, and `Arms.Aggregator` is my aggregating policy.

- For example, use this template:

```python
class MyPolicy(object):
    def __init__(self, nbArms, *args, **kwargs):
        self.nbArms = nbArms
        # TODO Finish this method that initialize the arm MyArm

    def __str__(self):
        return "MyArm(...)".format('...')  # TODO
```

Figure 3.3 – Screenshots from two other pages of the documentation: the `klUCBSwitch` policy, and details of the code organization, to know how to add a new policy.

3.2.6 How to run experiments?

We show how to install SMPyBandits, and an example of how to run a simple experiment. See this page `SMPyBandits.GitHub.io/How_to_run_the_code.html` for more details. SMPyBandits is also available on Pypi, see `pypi.org/project/SMPyBandits`, you can install it directly with `sudo pip install SMPyBandits`, or from a `virtualenv` [BP+16]. This bash code shows how to clone the code, and install the requirements for Python 3 (once):

```bash
# 1. get the code in the folder you want
$ git clone https://GitHub.com/SMPyBandits/SMPyBandits.git
$ cd SMPyBandits.git
# 2. install the requirements
$ pip install -r requirements.txt
```

Code Example 3.3 – Example of Bash code to download and install dependencies of SMPyBandits.

Simulations are easily executed, *e.g.*, Code 3.4 shows how to start $N = 1000$ repetitions of a simple non-Bayesian Bernoulli-distributed problem, for $K = 9$ arms, an horizon of $T = 10000$ and on 4 CPU. It takes about 20 min, on a 4-core 64-bit GNU/Linux laptop. Environment variables (`N=1000` etc) in the command line are not required, but they are convenient.

3.2.7 Example of a simulation and illustration

A small script `configuration.py` is used to import the arm classes (`Arms` module), the policy classes (`Policies` module) and define the problems and the experiments. Choosing

```bash
# 3. run a single-player simulation
$ BAYES=False ARM_TYPE=Bernoulli N=1000 T=10000 K=9 N_JOBS=4 \
  MEANS=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9] \
  python3 main.py configuration.py
```

Code Example 3.4 – Example of Bash code to run a simple experiment with SMPyBandits.

the algorithms to include is done by changing the `configuration["policies"]` list in the `configuration.py` file. For instance, one can compare the standard anytime kl-UCB algorithm (class `klUCB` in `Policies` module) against the non-anytime variant kl-UCB^{++} algorithm (`klUCBPlusPlus`), and also UCB with $\alpha = 1$ (`UCB`) and Thompson sampling (`Thompson`) with a Beta posterior (`Posterior.Beta`)).

```python
configuration["policies"] = [
  {"archtype": klUCB, "params": {"klucb": klucbBern}},
  {"archtype": klUCBPlusPlus,
   "params": {"horizon": T, "klucb": klucbBern}},
  {"archtype": UCBalpha, "params": {"alpha": 1}},
  {"archtype": Thompson, "params": {"posterior": Beta}}
]
```

Code Example 3.5 – Example of Python code to configure the list of algorithms tested on a problem.

Running the simulation as shown above will save figures in a sub-folder, as well as save data (pulls, rewards, regret and other data) in a HDF5 file[2] [C$^+$18]. Figure 3.4 above shows the average regret for these 4 algorithms. The Figure 3.5 below shows the histogram of regret obtained at the end of the experiment (*i.e.*, R_T) for the same example.

3.2.8 Dependencies

This library is written in Python [Fou17], for versions 2.7+ or 3.4+, using `matplotlib` [Hun07] for 2D plotting, `numpy` [vdWCV11] for data storing, random number generations and operations on arrays, `scipy` [JOP$^+$01] for statistical and special functions, and `seaborn` [W$^+$17] for clean plotting and colorblind-aware colormaps.

Optional dependencies include `joblib` [Var17] for parallel simulations, `numba` [I$^+$17] for automatic speed-up on small functions using a Just-in-Time compiler, `sphinx` [B$^+$18] for generating the documentation, `virtualenv` [BP$^+$16] for launching simulations in isolated environments, and `jupyter` [K$^+$16] used with `ipython` [PG07] to experiment with the code.

[2] For example, this simulation produced this HDF5 file
`GitHub.com/SMPyBandits/SMPyBandits/blob/master/plots/paper/example.hdf5`

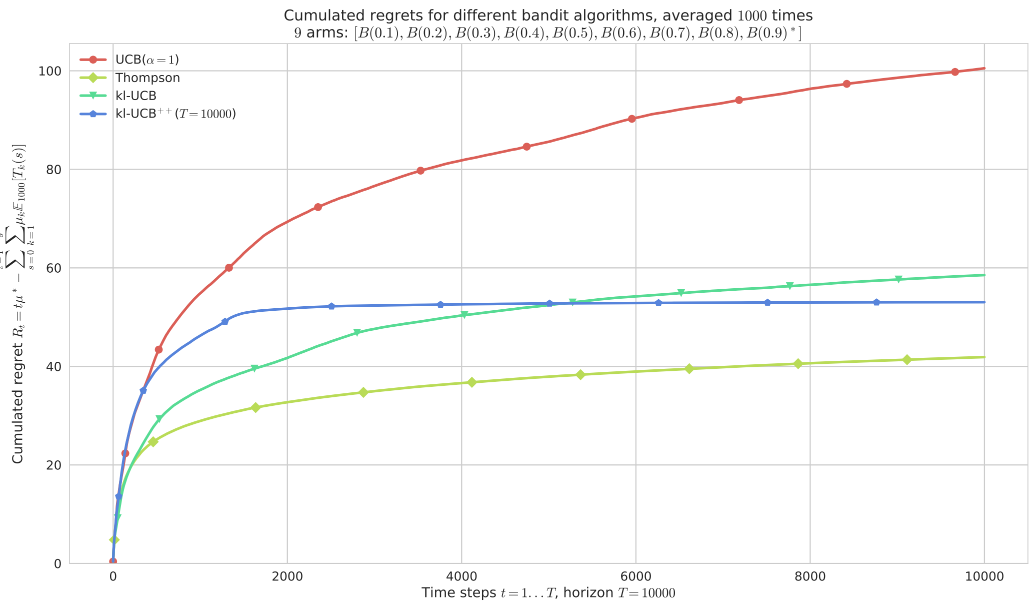


Figure 3.4 – Example of a single-player simulation showing the average regret of four algorithms. They all perform very well: each algorithm is known to be order-optimal (*i.e.*, its regret is proven to match the lower-bound up-to a constant), and each but UCB is known to be asymptotically optimal (*i.e.*, with the constant matching the lower-bound).

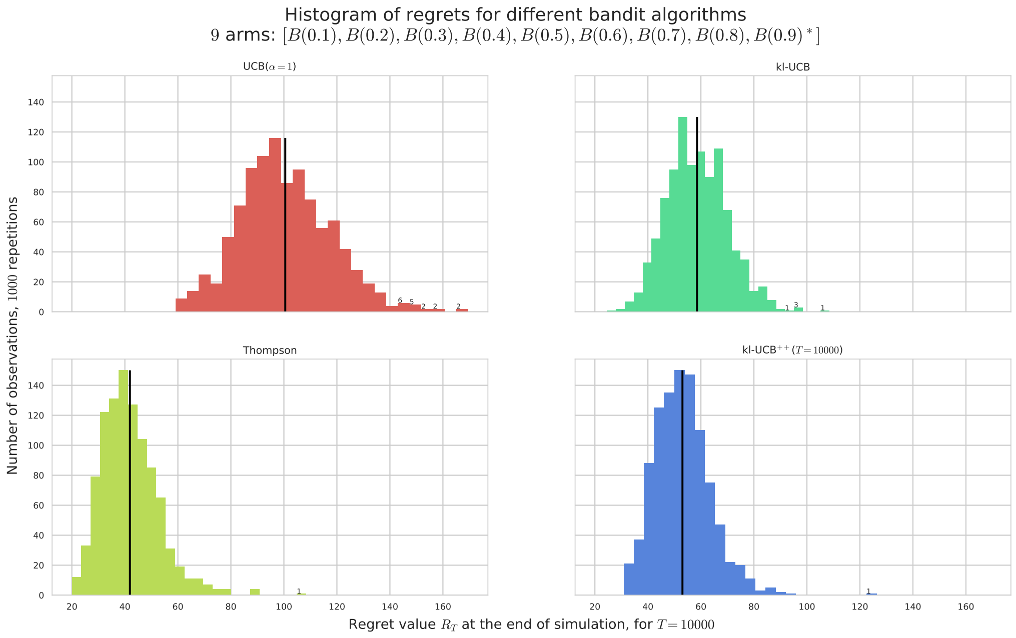


Figure 3.5 – Histogram of regret for the same experiment as of Figure 3.4. For instance, Thomson sampling is very efficient in average (in yellow), and UCB shows a larger variance (in red).

All the quoted libraries are free and open-source and can be installed in one command using the `pip` (pip.PyPa.io) or `conda` (conda.anaconda.org) package manager. When SMPyBandits is installed using `pip install SMPyBandits`, the dependencies are of course automatically installed if not already present.

3.2.9 Continuous integration with Read the Docs and Travis CI

Since 2017 and 2018, SMPyBandits is using two online continuous integration (CI) service, to automatically build and host the documentation, and to automatically test the library on some numerical simulations. These CI services are free for open-source works, and are both triggered whenever a modification of the codebase is sent to the hosting platform (*i.e.*, GitHub).

Read the Docs. Since 2017, we have been using the free web service provided by Read the Docs (ReadTheDocs.org). Read the Docs allows to automatically build the documentation after every commit. This allows to regularly check that the library is well formatted and can be imported correctly, as well as keeping the online documentation up-to-date. Moreover, they offer to host the documentation online, at SMPyBandits.ReadTheDocs.io. For more details, see ReadTheDocs.org/projects/SMPyBandits. Since its first use, the service did about 180 builds, and about 10% of them were useful to detect newly introduced issues or bugs in the code. The build is configured using the `.readthedocs.yml` file in SMPyBandits main folder, and it builds the documentation in about two minutes. I have also been building the documentation manually on a weekly basis, to also host it on SMPyBandits.GitHub.io thanks to GitHub pages. In the last two years, the documentation saw about 8500 unique visits, while the GitHub project have been visited about 3500 unique visits.

Travis CI. Since 2018, we use the free web service provided by Travis CI (Travis-CI.org). Travis CI allows to automatically run short numerical simulations after every commit. This allows to check that each modification on any part of the codebase does not break anything, as well as giving an up-to-date example of log files that shows online the results of different examples of experiments. The tests covers all the main models and almost all the algorithms implemented in SMPyBandits, and it has been proven very useful to quickly find and fix new bugs. For more details, see Travis-CI.org/SMPyBandits/SMPyBandits. Since its first use, the service did about 160 builds, and about 30% of them were useful to detect newly introduced issues or bugs in the code. The build is configured using the `.travis.yml` file in SMPyBandits main folder, and it runs numerical experiments with a short horizon (*e.g.*, $T = 100$ or $T = 1000$) and a small number of repetitions (*i.e.*, $N = 4$), but covering all the different models implemented by SMPyBandits (and even including some that are not covered in this chapter, like the Markov model, or the sparse multi-players model). Builds typically run for 15 minutes, and Travis CI have been proven to be very useful in our development process.

3.2.10 List of research works using SMPyBandits

SMPyBandits has been used for the following research articles since 2017, and it is used for the previous and the next chapters of this thesis (except in Chapter 5).

- In [BKM18] and in Chapter 2 above, we used SMPyBandits to illustrate and compare different aggregation algorithms[3]. We designed a variant of the Exp4 algorithm for online aggregation of experts [BCB12], called `Aggregator`.
- In [BK18a] and in Chapter 6 below, we used SMPyBandits for all the simulations for multi-players bandit algorithms[4]. We designed the two `RandTopM` and `MCTopM` algorithms and proved than they obtain logarithmic regret in the usual setting, and outperform significantly the previous state-of-the-art solutions (*i.e.*, `rhoRand`, `MEGA` and `MusicalChair`).
- In [BK18b] (quickly presented in Appendix A), we used SMPyBandits to illustrate and compare different "doubling trick" schemes[5].
- In [BK19b] and [BK19a], and in Chapter 7 below, we used SMPyBandits for piece-wise stationary MAB models (only for the single-player case). We illustrate and compare different algorithms designed for this family of non-stationary problems[6]. We designed the `GLRklUCB` policy, and implemented most of the state-of-the-art passive or active adaptive policies, both adversarial- or stochastic-based, designed to tackle the piece-wise stationary MAB problems. We also implemented a large benchmark of different problems.

3.3 Experimental comparisons of state-of-the-art algorithms

In this section, we use the SMPyBandits library to compare experimentally various state-of-the-art (single-player) algorithms on some MAB problems. We first detail the list of different algorithms that we compare. We then give the results of the numerical experiments, in terms of mean regret at the end of the experiment of different horizons T. Additional results in terms of real measurements of time and memory are given and discussed in the next Section 3.4.

[3] See the page `SMPyBandits.GitHub.io/Aggregation.html` on the documentation, which gives detailed instructions to reproduce the results presented in this research paper, details about the models and the notations, and bibliographic references.
[4] Similarly, see the page `SMPyBandits.GitHub.io/MultiPlayers.html` on the documentation.
[5] Similarly, see the page `SMPyBandits.GitHub.io/DoublingTrick.html` on the documentation.
[6] Similarly, see the page `SMPyBandits.GitHub.io/NonStationary.html` on the documentation.

3.3.1 Experimental setup: algorithms and problems

We consider the 9 following algorithms, and 7 more are described in Appendix 3.6.4. For each of them, we give a bibliographic reference, that corresponds to a recent article studying it and not the first one which introduced it. We give the chosen tuning of its parameters, for parametric algorithms. Algorithms that are "not anytime" use the exact value of the horizon T, and when increasing values of T are studied for the same problem, they use the correct successive values. For reproducibility, we also give in "`typewriter font`" the name of the corresponding class in the `Policies` module in SMPyBandits (*e.g.*, UCB$_1$ is `Policies.UCBalpha`).

1. ε-greedy [BCB12] (`EpsilonDecreasing`), using $\varepsilon_t = \varepsilon_0 / t$, and $\varepsilon_0 = 0.1$ (chosen arbitrarily). It has a very small cost in terms of time and memory, but it usually achieves linear regret.

2. Explore-then-Commit [GKL16] (`ETC_KnownGap`), that knows the horizon, and a lower-bound on the gap between arms (chosen arbitrarily as $\delta = 0.01$, valid for all problems). Similarly to ε-greedy, it has a very small footprint in terms of time and memory, but most of the times it only obtains large (linear) regret.

3. Exp3^{++} from [SL17] (`Exp3PlusPlus`), using $\alpha = 3$ and $\beta = 256$ as advised. It is a recent anytime variant of the Exp3 algorithm, which was proven to obtain good "best of both world" performances.

4. UCB$_1$ from [ACBF02] (`UCBalpha`), shown in Algorithm 2.2 above, using $\alpha = 1$. It achieves order-optimal problem-dependent bounds with a $\mathcal{O}(K \ln(T)/\Delta^2)$ regret.

5. kl-UCB from [CGM$^+$13] (`klUCB`), also shown in Algorithm 2.2 above, using the Bernoulli KL divergence and the corresponding kl-UCB indexes (coded as `kullback.klucbBern`). It is computationally more costly that UCB, as at each time step t and for each arm k, a convex optimization problem must be solved approximately, in order to compute the K indexes. In practice, a few steps of a simple bisection search method are enough to obtain a good numerical precision, and empirically we found that even in the worst cases kl-UCB is no more than one order of magnitude slower than UCB.

6. Thompson sampling from [KKM12] (`Thompson`), using a Beta prior and posterior, initially uniform (*i.e.*, $\pi_k(0) = \mathrm{Beta}(1,1)$). It is efficient in terms of storage, and even if K random samples must be sampled from the arms posteriors at each time step t, Thompson sampling is not much slower than UCB, if the random number generator used for the simulations is efficient (it is the case for SMPyBandits, as we use `numpy.random` module which relies on highly optimized C or Cython code).

7. Bayes-UCB from [KCG12] (`BayesUCB`), using a Beta prior and posterior, initially uniform, *i.e.*, $\pi_k(0) = \mathrm{Beta}(1, 1)$. It is comparable to Thompson sampling in terms of memory complexity, but slower as computing a quantile is more costly than sampling from a distribution. But in particular for Bernoulli distributed arms and for Beta posteriors and priors, Bayes-UCB is only about twice as slow as Thompson sampling.

8. AdBandits from [TdSCC13] (`AdBandits`), using $\alpha = 1$. In practice, it is usually (slightly) more efficient than both Thompson sampling and Bayes-UCB in terms of regret, it is comparable in terms of computation times, but costs more memory.

9. BESA (Best Empirical Sampled Average) from [BMM14], which is implemented with a binary tournament, written in a naive but efficient way (*i.e.*, not using recursive functions, by using the `BESA` class with the `"non_recursive"=True` option). But the extension to $K > 2$ arms is still costly in terms of both time and memory, as illustrated below, and blows-up exponentially when K increases.

We focused here on the most well known algorithms, discussed in Section 2.4, and two others that are efficient but less known (AdBandits and BESA). To highlight that SMPyBandits implements many other MAB algorithms proposed in recent works, we detail in Appendix 3.6.4 7 other algorithms. They are more recent (some from 2016, 3 from 2018 and 2 from early 2019) and are usually more complex, or are based on a different point-of-view. The numerical results presented below include these extra algorithms, to justify empirically that despite their respective qualities, and despite being very efficient, it is reasonable to focus on UCB and kl-UCB in the rest of this thesis, as they stay comparable with the state-of-the-art algorithms but are simpler to implement (*e.g.*, when comparing UCB to `ApproximatedFHGittins` or MOSS-Anytime) to manipulate theoretically (*e.g.*, when comparing kl-UCB to BESA, PHE or RCB).

Randomly sampled problems. For brevity, we prefer to focus on a single kind of distributions (Bernoulli), but similar results were observed for other distributions, in particular for (possibly truncated) Gaussian distributions.

For a fixed number of arms $K \geq 2$, instead of simulating a large number of times the same problem, we generate a new random problem for each independent run. We consider a randomly generated vector of means, each being sampled uniformly at random in $[0, 1]$, with a minimum gap of $\Delta^{\min}$, $\boldsymbol{\mu} \sim \mathcal{E}(\Delta^{\min}, \mu_{\min}, \mu_{\max})$, as well as minimum and maximum values of $\mu_{\min}$ and $\mu_{\max}$. This space $\mathcal{E}_{\Delta^{\min}}$ is defined as $\{\boldsymbol{\mu} \in [\mu_{\min}, \mu_{\max}]^K : \min_{k \neq k'} |\mu_k - \mu_{k'}| \geq \Delta^{\min}\}$. We use $\mu_{\min} = \Delta^{\min}$ and $\mu_{\max} = 1 - \Delta^{\min}$, and the value used for $\Delta^{\min}$ is 0.1 for $K \leq 3$ (for "easy" problems), and $\frac{1}{3K}$ for $K \geq 5$ (for "harder" problems).

Other parameters of the experiments. On the one hand, we fix $K = 8$ and we consider increasing values for the horizon, from $T = 1000$ to $T = 50000$. On the other hand, we fix

$T = 5000$ and we consider as well an increasing number of arms, from $K = 2$ to $K = 32$. For all experiments, we run $N = 1000$ independent simulations. We show in Code 3.10 in the Appendix below more details about how to run these experiments.

3.3.2 Experiments results

We give in the figures below the results of the experiments.

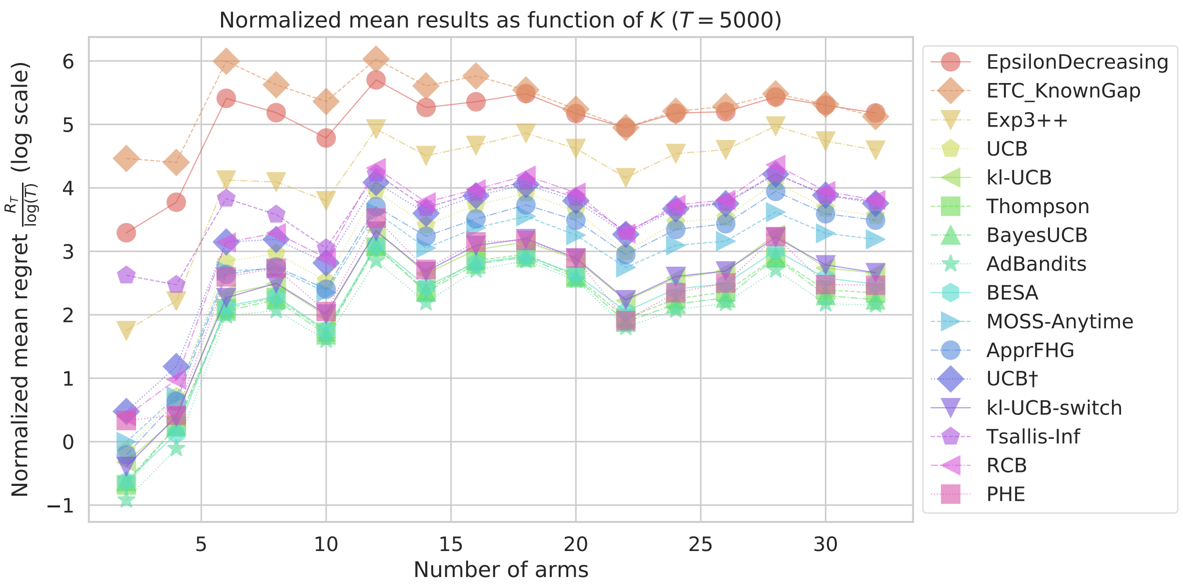


Figure 3.6 – Regret vs different values of K ($R_T / \ln(T)$), for $T = 5000$ and for 16 algorithms. The y-axis is in log-scale. All algorithms appear to have a regret slowly growing with respect to K, as predicted by the regret bounds which are linear with respect to the number of arms, as predicted. Bayesian algorithms appear to be the most efficient, and kl-UCB as well as UCB are also shown to be efficient. Both the ε-greedy and Explore-then-Commit algorithms performed poorly, actually they achieve linear regret.

Summary of the experiments. In the two Figures 3.6 and 3.7 included in this section, we are able to check empirically that the regret of all the efficient algorithm indeed scales as predicted by the theory, that is linearly in the number of arms, and logarithmically in the horizon, *i.e.*, $R_T = \mathcal{O}(K \ln T)$. The main take-away message is the following: in all the rest of this thesis, we focus on two algorithms depending if simplicity or efficiency is favored:

- UCB is used in the more applied Chapter 5, when simplicity is favored,
- kl-UCB is used, in the two more theoretical Chapters 6 and 7, when efficiency is favored, and when we analyze mathematically the performance of an algorithm that is running on top of a classical stationary MAB policy, like our two contributions, MCTopM and GLR.

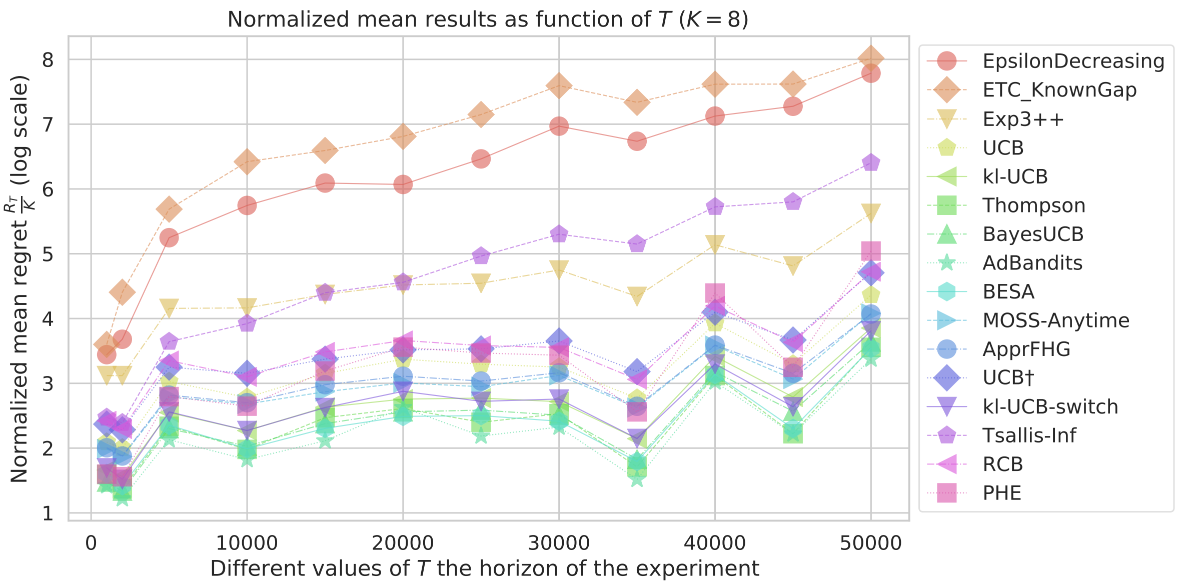


Figure 3.7 – Regret vs different values of T (R_T/K), for $K = 32$ and for 16 algorithms. All efficient algorithms appear to have a logarithmic regret with respect to the horizon, as predicted. The respective ranking of all the algorithms also appears to remain preserved for different values of T, which is also backed-up by theoretical results: if two algorithms $\mathcal{A}$ and $\mathcal{A}'$ has a regret close to their regret bounds, of the form $R_T \leq \mathcal{O}(K \ln(T))$, and the bound for $\mathcal{A}$ use a smaller constant than the bound for $\mathcal{A}'$, $\mathcal{A}$ should obtain a smaller regret than $\mathcal{A}'$ no matter the horizon. Bayesian algorithms appear to be the most efficient, and kl-UCB as well as UCB are also shown to be efficient. Finally, we also observe once more than both the ε-greedy and Explore-then-Commit algorithms performed poorly, achieving linear regret.

3.4 Comparing real measurements of time and memory costs

In this section, we report additional experiment results from the simulations described in the previous Section 3.3. Instead of studying on the efficiency of the algorithms (*i.e.*, their regret), we report results of real measurements in terms of computational time as well as memory storage. While the results reported in the previous section should not depend on the implementation of the different algorithms, the results in this section concern real measurements of both time and memory consumptions of the simulation software used for these simulations. Hence, the reported results highly depend on many factors, including how the code is written, and where and when it is run. We take precautions to ensure the fairness of the comparison between the different algorithms, as detailed in Appendix 3.6.5.

Computational time. In Figure 3.8, we can see that Bayesian algorithms appear to be the most efficient (*i.e.*, in the bottom left corner), and kl-UCB is very close to their best empirical performances. UCB, algorithms that were proven to perform similarly to UCB (*i.e.*, PHE and RCB), and other index policies inspired by UCB (ApprFHG, UCB†) enjoy very similar performances: a larger regret than Bayesian algorithms and kl-UCB, but a shorter running time. This highlights a trade-off between optimality in terms of regret and computational efficiency.

Moreover, in the two Figures 3.9 and 3.10, we are able to check empirically that the computational time of all the efficient algorithm indeed scales as predicted by the theory, that is linearly in the number of arms in the horizon, *i.e.*, $\mathcal{T}_T = \mathcal{O}(KT)$.

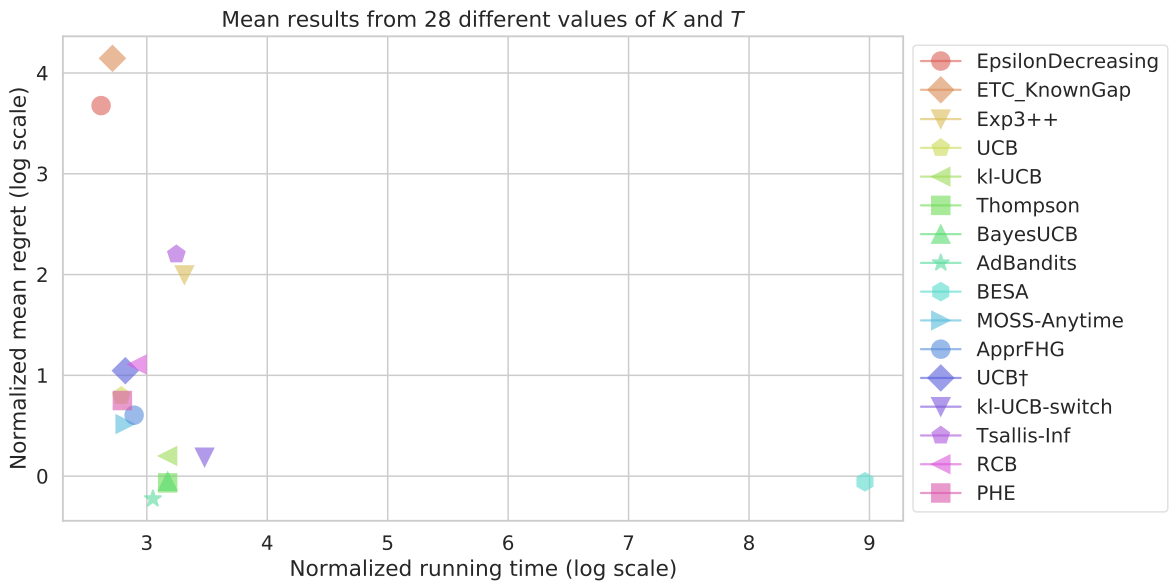


Figure 3.8 – Normalized mean regret vs normalized running time (in micro-seconds), aggregating the results from different values of K and T, for 16 algorithms. Both the x-axis and y-axis are in log-scale. UCB and kl-UCB are among the best algorithms, while AdBandits, Thompson sampling and Bayes-UCB slightly outperform them in terms of regret, and have similar running times.

Memory cost. Like for the computational cost discussed above, in Figure 3.11, we can see that Bayesian algorithms like Thompson sampling again appear to be the most efficient (*i.e.*, in the bottom left corner), and kl-UCB is close to their best empirical performances. UCB and other index policies inspired by UCB (ApprFHG, UCB†) enjoy very similar performances: a larger regret than Bayesian algorithms and kl-UCB, but a similar memory cost. This time, there is no clear trade-off between optimality in terms of regret and memory cost, and based on simply this Figure 3.11, one could advise to use Thompson sampling rather than kl-UCB.

Moreover, in the two Figures 3.12 and 3.13, we are able to check empirically that the memory cost of all the efficient algorithm indeed scales as predicted by the theory, that is linearly in the number of arms but independently of the horizon, *i.e.*, $\mathcal{M}_T = \mathcal{O}(K)$.

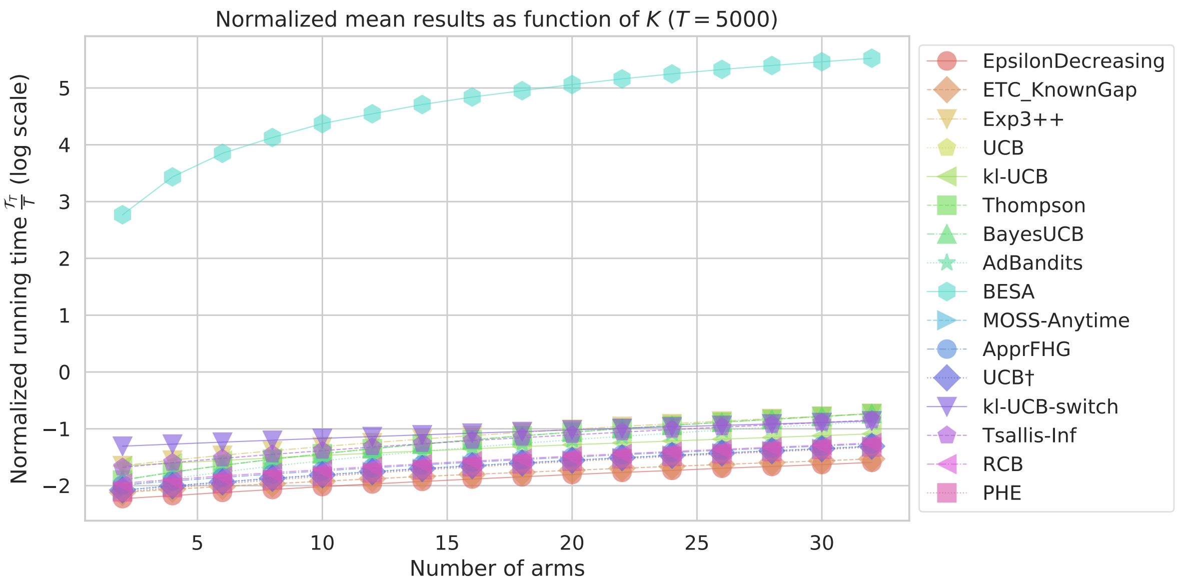


Figure 3.9 – Normalized running time vs different values of K ($\mathcal{T}_T/T$), for $T = 5000$ and for 16 algorithms. y-axis is in log-scale. All algorithms except BESA has a linear normalized running time, meaning that for K arms they use a computation time proportional to K, as predicted: $\mathcal{T}_T = \mathcal{O}(KT)$.

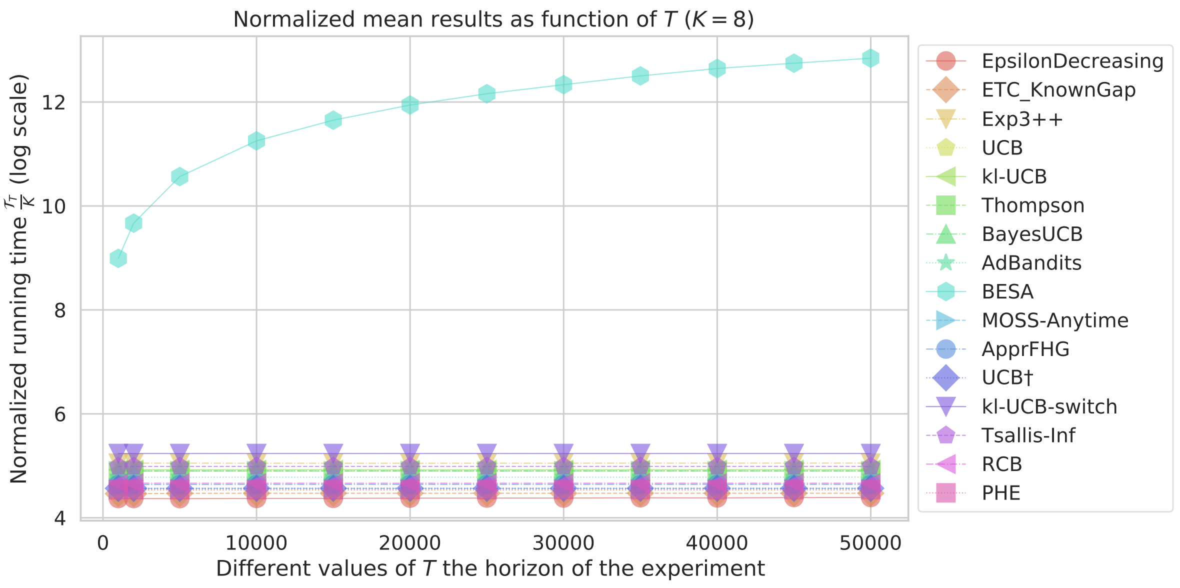


Figure 3.10 – Normalized running time vs different values of T ($\mathcal{T}_T/T$), for $K = 32$ and for 16 algorithms. y-axis is in log-scale. All algorithms except BESA has a constant normalized running time, meaning that for T rounds they use a computation time proportional to T, as predicted: $\mathcal{T}_T = \mathcal{O}(KT)$.

Summary of the experiments. The main take-away message is the following: the simplest (but most efficient) algorithms have a memory cost proportional to the number of arms but

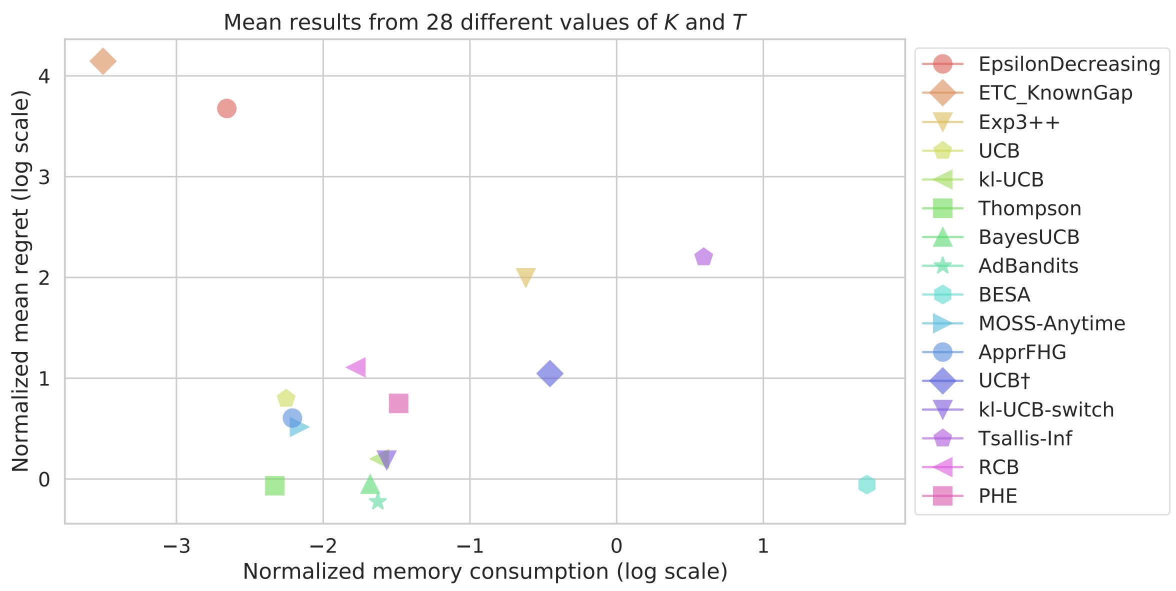


Figure 3.11 – Normalized mean regret vs normalized memory costs (in bytes), aggregating the results from different values of K and T, for 16 algorithms. Both the x-axis and y-axis are in log-scale. Thompson sampling appears as the best algorithm in this visualization, while UCB has the advantage of being memory efficient, and kl-UCB obtains similar performances.

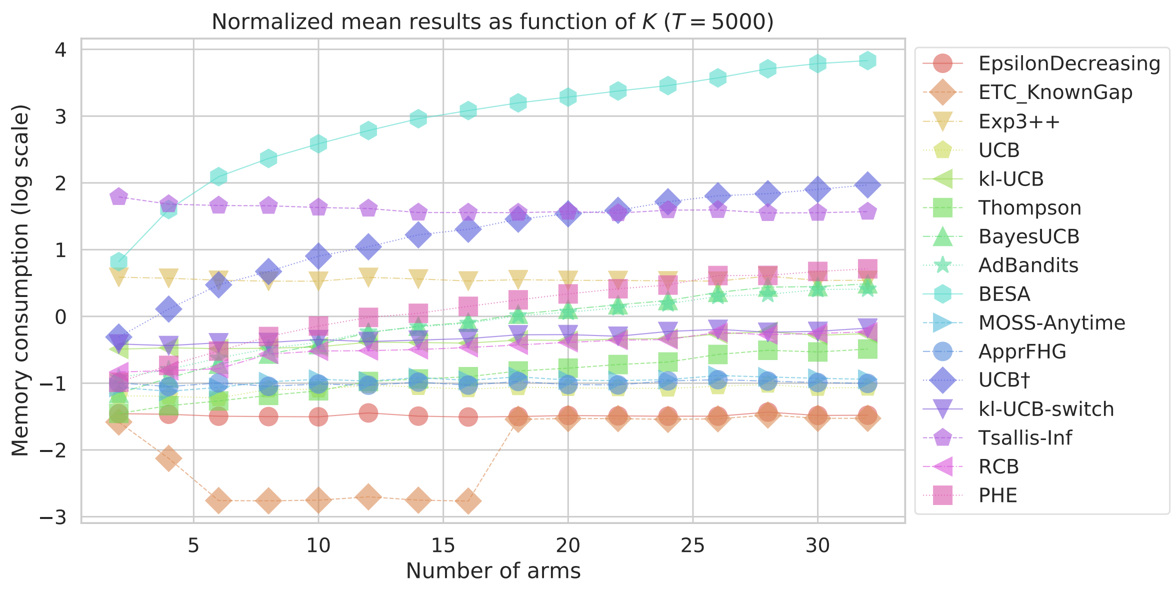


Figure 3.12 – Normalized memory cost vs different values of K ($\mathcal{M}_T/K$), for $T = 5000$ and for 16 algorithms. y-axis is in log-scale. All algorithms (except BESA) has an almost constant memory cost, meaning that for K arms they use a storage proportional to K, as predicted: $\mathcal{M}_T = \mathcal{O}(K)$.

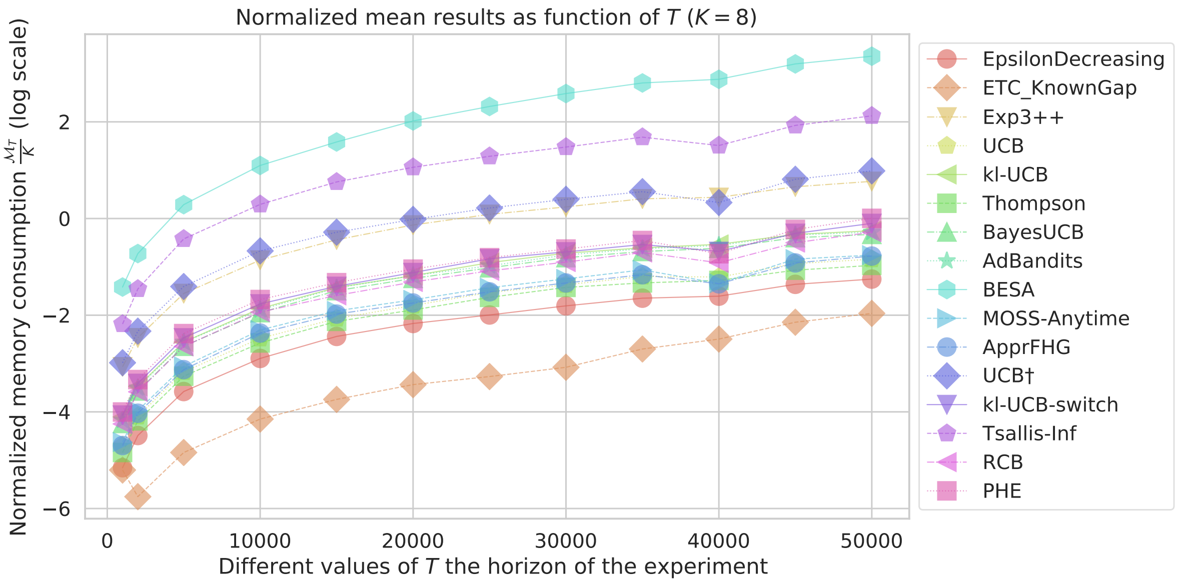


Figure 3.13 – Normalized memory cost vs different values of T ($\mathcal{M}_T/K$), for $K = 32$ and for 16 algorithms. y-axis is in log-scale. All algorithms have a growing normalized memory cost, meaning that for T rounds they use a computation time almost independent of T, as predicted: $\mathcal{M}_T = \mathcal{O}(K)$.

independent of the horizons, *i.e.*, bounded by $\mathcal{O}(K)$, and a time complexity at each time step $t \in [T]$ bounded by $\mathcal{O}(K)$, independent of t.

We also shown a trade-off between optimality in terms of regret, and efficiency in terms of time complexity or memory cost. Our position regarding this trade-off is motivated by Occam's razor principle. On the first hand, if the algorithm should be implemented on a cognitive radio device that has limited hardware capacity (for instance), one can reasonably aim at the simplest yet order-optimal algorithms, and we advise to use UCB and algorithms running on top of this simple index policy, as we do in Chapter 5. On the other hand, if one is more interested in the mathematical developments and wants to prove the tightest possible regret upper bounds, aiming at more efficient but more complex algorithms is interesting, and we chose the kl-UCB algorithm in Chapters 6 and 7.

3.5 Conclusion

In this chapter, we presented our Python library SMPyBandits, by detailing three main points: (i) its purpose and qualities, (ii) its organization, and (iii) an overview of its usage:

(i) Its purpose is to easily implement numerical simulations of stochastic or piece-wise stochastic problems of single- or multi-players multi-armed bandits. SMPyBandits is distributed on GitHub and Pypi freely, under an open-source license, and it is extensively documented (at `SMPyBandits.GitHub.io`). Our library allows any researcher to easily run numerical simulations of different kinds of multi-armed bandit problems, requiring only a small knowledge of Python thanks to its documentation, its well-designed API, and many examples of simulation scripts and configuration files included.

(ii) We detailed how SMPyBandits is implementing arms, problems, algorithms, and use these components to implement a simulation loop, with various visualizations being performed after the simulation. As far as now, SMPyBandits is restricted to the finite-arm cases, but it supports a wide range of arms distributions. Different kinds of models are implemented, from stationary single-player to piece-wise stationary multi-players with different collision models. One of the main qualities of the library is that it is quite exhaustive, as all the main families of algorithms covering these different models have been implemented, even very recent algorithms from the literature, as we tried our best in following the active research since December 2016 to June 2019. More than 65 algorithms or variants of algorithms are implemented for the single-player case, 5 for the expert aggregation problem, about 15 for the multi-players case, and about 20 for the piece-wise stationary problem. All the codebase is fully documented, and the library is using continuous integration to run automated tests on the code after every modification. When comparing algorithms on a problem, the main performance measure is the regret, but the library also computes, stores and visualizes other measures, such as best-arm selection rate or mean cumulated reward, as well as real time and memory costs.

(iii) Finally, we presented how to use SMPyBandits, if one wants to run some pre-designed simulations or design new simulations. Running a simulation is very easy, and different examples showed that the main parameters such as the time horizon T or the number of repetitions can be configured directly from the command line when running the Python script, or by modifying the code. Designing a new simulation requires to have a basic knowledge of Python, but not to dive into the implementation of the library. We give below in Appendix 3.6 a small but complete example of the two files that has to be written for a new simulation: a `main` file which essentially defines the desired visualizations after the simulation, and a `configuration` file which defines the problems to simulate and their parameters, as well as the algorithms that will be compared.

We want to conclude by highlighting that a significant amount of time during my PhD was devoted to the development of SMPyBandits, and as such the library, its documentation and this chapter are considered an important contribution of this thesis. Our library is used for the numerical simulations in all this thesis, except Chapter 5. We also used it in other publications not included in this thesis, like our work on doubling tricks [BK18b] (see Appendix A).

Future works. An interesting future work left on our library SMPyBandits could be to implement variants of the single-player stochastic models, as well as variants for the multi-players or the non-stationary cases. For more details, see the issue tickets at `GitHub.com/SMPyBandits/SMPyBandits/issues/`. For instance it would be interesting to extend the library for problems with non-finite arms, *e.g.*, linear or contextual bandit (ticket 117), or to add support for the "dynamic case" of multi-players bandits to allow arrivals or departures of players (ticket 124). At the end of Chapter 6, we present many extensions of the multi-players bandit model, and even if some have already been implemented, a major future work is to implement the most interesting ones (tickets 120, 124, 185).

Since 2017, I also received by emails or by GitHub issues some requests to implement new features or models. Most of them were implemented quickly, but some would require more time and are left as possible future works. Two example of requests include a model where the number of arm can change at some or all time step (an example of such model is called "sleeping bandit", issue 123), to implement the model of rotting bandits (issue 156), or to add a web-based user interface for an even easier usage of the library requiring no command line or knowledge of Python (issue 133).

Finally, the most exciting thing that could happen to our SMPyBandits library would be to see it gaining popularity! Its documentation has seen already about 20000 visits, the project on GitHub had 88 stars in June 2019, and based on the features requested and the emails received about it, we counted between 15 to 25 researchers in other labs we use or used SMPyBandits. While this is a good start, we believe that the library is mature and interesting enough to hope to see it being used by more people. We have submitted a summary paper presenting SMPyBandits [Bes18] to the MLOSS track of the Journal of Machine Learning Research (JMLR), and we hope to be accepted, as it would give more visibility to our work.

As a personal note, I would like to continue working on SMPyBandits, and implement the most important requested features, as well as maintain it, and possibly continue to add recent algorithms by following actively the research in this community. I would also love to be able in the next to teach a graduate or under-graduate course on multi-armed bandits, and use my library as a support to illustrate the course, as well as practical sessions.

3.6 Appendix

We start by giving a small but complete example of use of SMPyBandits, then we give additional experimental results completing those presented in Section 3.3 and 3.4 above. We conclude by a discussion on parallel multi-core CPU computations to speed-up SMPyBandits.

3.6.1 Minimalist example of use of SMPyBandits

Below is presented in Code 3.6 a minimalist example of use of the library, to define a toy bandit problem with $K = 2$ Bernoulli distributed arms (of means $\mu = 0.1$ and $\mu = 0.9$), one UCB algorithm, and play one experiment of horizon $T = 1000$. Only the mean reward is printed at the end of the simulation. With Python 3.6 this example shows that the algorithm got a mean reward very close to the optimal arm, $\hat{\mu} = 0.886 \simeq 0.9 = \mu^* = \mu_2$.

```
The UCB algorithm obtains here a mean reward = 0.886
```

```python
""" Example of use of SMPyBandits.
See https://SMPyBandits.GitHub.io/API.html for more details!"""
import numpy as np
np.random.seed(0)  # for reproducibility
from SMPyBandits.Arms import Bernoulli
arms = [Bernoulli(0.1), Bernoulli(0.9)]
from SMPyBandits.Environment import MAB
my_MAB_problem = MAB(arms)
nbArms = my_MAB_problem.nbArms  # 2 arms !
from SMPyBandits.Policies import UCB
my_UCB_algo = UCB(nbArms)
my_UCB_algo.startGame()  # reset internal memory

horizon = 1000
for t in range(horizon):  # simulation loop
    chosen_arm = my_UCB_algo.choice()
    observed_reward = my_MAB_problem.draw(chosen_arm)
    my_UCB_algo.getReward(chosen_arm, observed_reward)

cumulated_reward = sum(my_UCB_algo.rewards)  # random!
number_of_plays  = sum(my_UCB_algo.pulls)    # horizon = 1000
mean_reward = cumulated_reward / number_of_plays
print("The UCB algorithm obtains here a mean reward =", mean_reward)
```

Code Example 3.6 – Example of use of SMPyBandits.

3.6.2 A minimalist `main.py` file

We present below in Code 3.7 a full example of a minimalist `main.py` file, which is used to load the configuration, run the experiment and then print and plot various visualizations for the simulation.

```python
""" An example of a simple 'main' script.
Main scripts load the config, run the simulations, and plot them/
For the single player case.
"""
from example_of_configuration_singleplayer import configuration
from SMPyBandits.Environment import Evaluator

configuration['showplot'] = True
evaluation = Evaluator(configuration)

# Start the evaluation and then print final ranking
# and do and show plot, for each environment
for envId, env in enumerate(evaluation.envs):
    # Evaluate just that env
    evaluation.startOneEnv(envId, env)

# Compare them
for envId, env in enumerate(evaluation.envs):
    evaluation.plotHistoryOfMeans(envId)

    print("\nGiving all the vector of final regrets ...")
    evaluation.printLastRegrets(envId)
    print("\nGiving the final ranking ...")
    evaluation.printFinalRanking(envId)

    print("\n\n- Plotting the last regrets...")
    evaluation.plotLastRegrets(envId, boxplot=True)

    print("\nGiving the mean and std running times ...")
    evaluation.printRunningTimes(envId)
    evaluation.plotRunningTimes(envId)

    print("\nGiving the mean and std running times ...")
    evaluation.printMemoryConsumption(envId)
    evaluation.plotMemoryConsumption(envId)

    print("\n\n- Plotting the mean reward...")
    evaluation.plotRegrets(envId, meanReward=True)

    print("\n\n- Plotting the regret...")
    evaluation.plotRegrets(envId)

    print("\n- Plotting the probability of picking the best arm...")
    evaluation.plotBestArmPulls(envId)

    print("\n- Plotting the histograms of regrets...")
    evaluation.plotLastRegrets(envId, sharex=True, sharey=True)

# Done
print("Done for simulations example_of_main_singleplayer ...")
```

Code Example 3.7 – Example of `main.py` file.

3.6.3 A minimalist `configuration.py` file

We present below in Code 3.9 a full example of a minimalist `configuration.py` file, which is used to define the experiment. It defines the number of arms K and their distribution, the horizon T, the policies to test etc. It uses environment variables, so a user can configure some parameters of the simulation when launching them from a terminal:

```bash
$ T=50000 N=100 K=10 ARM_TYPE=Gaussian python main.py
```

Code Example 3.8 – Example of Bash code to run an experiment

```python
""" An example of a configuration file to launch some the simulations
For the single-player case.
"""
from os import getenv  # to get command line variables

# Import arms
from SMPyBandits.Arms import *
# Import contained classes
from SMPyBandits.Environment import MAB
# Import single-player algorithms
from SMPyBandits.Policies import *

# HORIZON : number of time steps of the experiments.
# Warning Should be >= 10000 to be interesting "asymptotically".
```

```
HORIZON = int(getenv('T', 10000))

# REPETITIONS : number of repetitions of the experiments.
# Warning: Should be >= 10 to be statistically trustworthy.
REPETITIONS = int(getenv('N', 100))

# Number of jobs to use for the parallel computations.
# -1 means all the CPU cores, 1 means no parallelization.
N_JOBS = int(getenv('N_JOBS', -1))

# Number of arms for non-hard-coded problems
NB_ARMS = int(getenv('K', 3))

# Lower value of means
LOWER = float(getenv('LOWER', 0))
# Amplitude value of means
AMPLITUDE = float(getenv('AMPLITUDE', 1))

# Type of arms for non-hard-coded problems
ARM_TYPE = mapping_ARM_TYPE[str(getenv('ARM_TYPE', "Bernoulli"))]

# Means of arms for non-hard-coded problems
MEANS = uniformMeans(nbArms=NB_ARMS, delta=0.05,
    lower=LOWER, amplitude=AMPLITUDE, isSorted=True
)

# This dictionary configures the experiments
configuration = {
    # --- Duration of the experiment
    "horizon": HORIZON,
    # --- Number of repetition of the experiment (to have an average)
    "repetitions": REPETITIONS,
    # --- Parameters for the use of joblib.Parallel
    "n_jobs": N_JOBS,    # = nb of CPU cores
    # --- Should we plot the lower-bounds or not?
    "plot_lowerbounds": True,  # Default
    # --- Arms: a list of configuration of environments
    "environment": [
        {   # Use vector from command line
            "arm_type": ARM_TYPE,
            "params": MEANS
        },
    ],
    # --- Policies: a list of configuration of algorithms
    "policies": [
        # --- Stupid algorithm
        {
            "archtype": Uniform,   # The stupidest policy, fully uniform
            "params": {}
        }, {  # --- Fixed arm algorithm
            "archtype": TakeFixedArm,   # Take worse arm!
            "params": { "armIndex": 0 },
        }, {
            "archtype": TakeFixedArm,   # Take second worse arm!
            "params": { "armIndex": 1 },
        }, {
            "archtype": TakeFixedArm,   # Take third worse arm!
            "params": { "armIndex": min(2, nbArms - 1) },
        }, {  # --- UCB algorithm
            "archtype": UCBalpha,   # UCB with alpha=1 parameter
            "params": { "alpha": 1.0 }
        }, {  # --- Thompson algorithm
            "archtype": Thompson,
            "params": { "posterior": Beta }
        }, {  # --- KL UCB algorithm
            "archtype": klUCB,
            "params": { "klucb": klBern }
        },
    ]
}  # DONE

print("Loaded experiments configuration:")
print("configuration =", configuration)  # DEBUG
```

Code Example 3.9 – Example of `configuration.py` file.

3.6.4 Additional details about numerical experiments in Section 3.3

This Appendix section starts by describing 7 more algorithms that are also compared with the 9 presented above in Section 3.3.

10. MOSS-Anytime from [DP16], using $\alpha = 1$ (`MOSSAnytime`). It is a recent variant of the MOSS algorithm [BS12], using an adaptive tuning of its parameter. It is anytime, and obtains good "best of both world" performances, meaning that it achieves $\mathcal{O}(K \ln(T)/\Delta^2)$ regret for problem-dependent bounds, and $\mathcal{O}(\sqrt{KT})$ for problem-independent bounds. Empirically, it performs usually similarly to UCB, and is slightly more costly in terms of both time and memory.

11. Approximated Finite-Horizon Gittins from [Lat16b] (`ApproximatedFHGittins`), using $\alpha = 1$. It mimics the UCB algorithm but uses a more complicated function to compute the upper confidence bounds. It achieves order-optimal problem-dependent bounds with a $\mathcal{O}(K \ln(T)/\Delta^2)$ regret. Empirically, it performs usually similarly to UCB and MOSS-Anytime, and is slightly more costly in terms of both time and memory.

12. UCB† from the same author [Lat18] (`UCBdagger`), using $\alpha = 1$. It mimics the UCB algorithm but uses a much more complicated function to compute the upper confidence bounds, and uses more storage. It also achieves order-optimal problem-dependent bounds with a $\mathcal{O}(K \ln(T)/\Delta^2)$ regret. Empirically, it performs usually similarly to UCB and MOSS-Anytime, and is slightly more costly in computation time, but it is much more costly in terms of memory.

13. Anytime variant of KL-UCB-Switch from [GHMS18] (`klUCBswitchAnytime`), is a very recent variant of the KL-UCB algorithm [CGM+13]. It mimics the KL-UCB algorithm but uses a much more complicated function to compute the upper confidence bounds, and uses more storage. It is anytime, and was prove to obtain good "best of both world" performances, meaning that it achieves an asymptotically optimal $\mathcal{O}(\ln(T))$ regret for problem-dependent bounds, and $\mathcal{O}(\sqrt{KT})$ for problem-independent bounds. Empirically, it usually outperforms (slightly) kl-UCB in terms of regret, and costs about the same memory, but it is slower.

14. Tsallis-Inf from [ZS18] (`TsallisInf`), using $\alpha = 1/2$ (it seems to be the most efficient of the algorithms described in the article). It is based on the online mirror descent algorithm with a Tsallis entropy regularizer, and it is anytime. It was also proven to obtain good "best of both world" performances. Empirically, we were unable to find any problem were it performs as well as UCB (or any other efficient algorithm), and we are confident that our implementation is correct[7]. Tsallis-Inf is also slower than UCB (about as slow as KL-UCB-Switch), and costs an-order-of-magnitude more memory.

15. RCB (Randomized Confidence Bound) from [KT19] (`RCB`), using $\alpha = 1$ and perturbations uniformly sampled in $[0, 1]$. We prefer to use the simplest of the algorithms described in the article, and other algorithms in the same article used other families of distributions for the perturbations. It was analyzed and found to be order-optimal for problem-dependent bounds, and empirically we found that it is usually performs slightly worse than UCB, in terms of regret, time and memory. It is included because it remains comparably efficient with the other state-of-the-art algorithms, and because the key message from

[7]Despite a serious amount of time spent on this implementation, maybe it has some bug which explains the poor empirical performance obtained by Tsallis-Inf in our simulations. Exploring in more details this kind of algorithms is very interesting, as algorithms based on online mirror descent make a rich connexion between multi-armed bandit and online convex optimization [H+16], and a short-term future work in SMPyBandits will be to explore different values of α for Tsallis entropy regularization, or other families of regularization.

[KT19] is very interesting: "the optimism embedded in UCB can be replaced by simple randomization".

16. PHE (Perturbed-History Exploration) from [KSGB19] (`PHE`), using a perturbation scale of 0.5 as advised. The algorithm adds $\mathcal{O}(t)$ *i.i.d.* pseudo-rewards to its history in round t, and then pulls the arm with the highest estimated value in its perturbed history. It was also analyzed and found to be comparable with UCB. Like RCB, PHE was found to be empirically slightly worse than other state-of-the-art algorithms. Interestingly, its time and memory consumption stay constant with respect to the step number t and the horizon T, because generating and summing t pseudo-rewards from a Bernoulli distribution can actually be done in $\mathcal{O}(1)$ time and space, by using efficient sampling methods for sampling from a Binomial distribution.

Bash command to run these experiments. We give below in Code 3.10 a Bash command to run the experiments presented in Sections 3.3 and 3.6.4 above. It uses `for` loops and environment variables from Bash, and not Python, mainly for concision, but we could also have done the same by writing a Python script that calls the `main.py` script for these values of T and K.

```bash
$ for T in $(seq 5000 5000 50000); do \
    DEBUG=False NOPLOTS=False SAVEALL=True N_JOBS=-1 N=1000 BAYES=True \
    T=$T   K=8  python3 main.py configuration.py \
    && cp logs/main_py3_log.txt logs/main_py3_log__N1000_BAYES_T{T}_K8.txt \
  done
$ for K in $(seq 2 2 32); do \
    DEBUG=False NOPLOTS=False SAVEALL=True N_JOBS=-1 N=1000 BAYES=True \
    T=5000 K=$K python3 main.py configuration.py \
    && cp logs/main_py3_log.txt logs/main_py3_log__N1000_BAYES_T5000_K{K}.txt \
  done
```

Code Example 3.10 – Bash code to run the large-scale experiments presented in Sections 3.3 and 3.6.4, for $K = 8$ and $T \in \{5000, \dots, 50000\}$ and $T = 5000$ and $K \in \{2, \dots, 32\}$.

3.6.5 Methodology details for measurements of time and memory

The results reported in Section 3.4 highly depend on many factors, including how the code is written, and where and when it is run. We detail both below:

Quality and optimization of the code: The same level of optimization is used in all the codebase of SMPyBandits, and most of the time, the code is pure and naive Python and was not optimized. Mathematical functions (*e.g.*, $\sqrt{\bullet}$, $\ln$) and random numbers used are using `numpy` and `numpy.random` modules [vdWCV11], which are based on C and Cython code [BBS+19] and can be considered highly efficient. Computations of Kullback-Leibler and indexes for kl-UCB algorithms are based on a compiled CPython extension written in C [Fou17] and can also be considered highly efficient. As we can safely affirm that the code of the different algorithms has the same level of optimization, the comparison is fair, and we do not favor any algorithm in their implementation in SMPyBandits.

About the experimental environment: the exact measurements used for the figures displayed below highly depend on the machine used for the simulation, the version of the language and its libraries (see above in Section 3.2.8 for details about SMPyBandits), as well as the number of CPU cores being used (here, we used 12 cores in a 12-core desktop) etc. As long as the different algorithms and simulations are performed on the same environment, the comparison we make from them are fair and do make sense.

About the measurement protocol. To be precise, we used the two following approaches to measure the real time and memory costs of the different algorithms.

For time, we used the `time.time()` function from Python's standard library, that gives the current time in micro-seconds. In the `Evaluator` object of SMPyBandits, before launching the "for" loop on $t \in [T]$ for one algorithm, the system time is stored, and when the loop is finished, the time used for this loop is the current system time minus the starting time. This measurement gets averaged on the $N = 100$ (independent) repetitions, and consistently measure the time efficiency of the algorithms.

For memory, we use two different approaches whether the code runs on a UNIX or a Windows system. On UNIX, the `resource` module allows to measure the memory of the current process, and by counting the difference in memory used by the `Evaluator` object before and after the "for" loop, we can track the memory used by the algorithm to store all its interior variables.

In both cases, the "for" loop takes some time and stores many variables, but the only difference in terms of both time or memory between two algorithms is explained by the difference in the implementation of the algorithms.

Comparing measurements. But what is important in these simulation results is not the exact values, but rather to compare the costs of the different algorithms, and thus only the relative difference in terms of time and memory costs are important. Relative difference for costs do not depend much on the code and the machine used for the simulation. The computation time is *normalized*, that is it is divided by the horizon, to count the (average) time used by an algorithm at time t from $t = 1$ to $t = T$, while the memory cost is *not normalized*. Thus, we can check that efficient algorithms have a running time independent on the horizon T, instead of just observing a linear dependency, and we can check that efficient algorithms have a memory cost independent on the horizon. All the considered algorithms in this Chapter are efficient in this aspect, but in Chapter 7 we study algorithms that essentially have to store, and run computations, on an increasing number of observations (*e.g.*, CUSUM-UCB), so their computation cost at time t increases (linearly) with t, and thus they are more costly. We can also check that efficient algorithms have a running time and a memory cost linear in the number of arms K, while more complex algorithms like BESA suffer from an exponential blow-up of their complexity when K increases.

Normalizing data. The two figures below regret normalized data, in the following way. For each algorithm, we ran simulations to obtain its (empirical) regret, computation time and space requirement, for different problems with different values of K and T. Simply considering the average of such measurements makes no sense, as for instance two values of the regret for $T = 1000$ or $T = 50000$ do not have the same order of magnitude. We know that efficient algorithms are expected to follow these patterns:

- The final regret R_T should scale as $\mathcal{O}(K \ln(T))$,
- The total computation time $\mathcal{T}_T$ should scale as $\mathcal{O}(KT)$ (*i.e.*, a computation time of $\mathcal{O}(K)$ for each time step t),
- The total memory cost $\mathcal{M}_T$ should scale as $\mathcal{O}(K)$, independent of the horizon.

Thus, on the one hand, when we show aggregated results from all the different values of K and T, we normalized the data by dividing the regrets by $K \ln(T)$, the times by KT and the memory by K. On the other hand, when we plot a quantity (R_T, $\mathcal{T}_T$, $\mathcal{M}_T$) as a function of T (resp. of K) we only normalize by K (resp. by T).

3.6.6 Using multi-cores parallelism to speed-up simulations in SMPyBandits

Nowadays, parallelism is everywhere in the computational world, and any serious framework for numerical simulations must try to gain performance from parallelism. In this last section, we quickly review the chosen approach for SMPyBandits (multi-core on one machine).

For all the numerical simulations presented in this thesis, the setting is the same: we consider a small set of p different problems, of time horizon T that we want to simulate for N independent runs (*e.g.*, $p = 6$, $T = 10000$ and $N = 1000$ like in Chapter 7). Because of the fundamentally sequential nature of bandit games, each repetition of the simulation must be sequential regarding the time steps $t = 1, \dots, T$, and so no parallelism can be done to speed up this axis. But parallelism can help greatly for the two other axes: if we have a way to run in parallel 4 processes, and we have $p = 4$ problems to simulate, then running a process for each problem directly brings a speed-up factor of 4. Similarly, if we want to run 1000 repetitions of the same (random) problem, and we can run 4 processes in parallel, then running $1000/4 = 25$ repetitions on each process also bring a speed-up factor close to 4.

`Joblib` for multi-core simulations. The first approach is to use multiple cores of the same machines, and because it is both the simplest and the less financially as well as ecologically costly, this is the approach implemented in SMPyBandits. The machines I had access to during my thesis, either my own laptop or a workstation hosted the SCEE team in CentraleSupélec campus, were equipped with $i5$ or $i7$ Intel CPU with 4 or 12 cores. As explained in the page `SMPyBandits.GitHub.io/How_to_run_the_code.html`, we implemented in SMPyBandits an easy way to run any simulations on n cores of a machine, using the `Joblib` [Var17] library.

It is implemented in a completely transparent way, and if someone uses the command-line variable to configure experiments, switching from using one core to using all the cores of the machine is as simple as changing `N_JOBS=1` to `N_JOBS=-1`, like in the Code Example 3.11 below. Similarly, changing from within the Python code is simple, by changing the `configuration["n_jobs"]` value.

```
$ BAYES=False ARM_TYPE=Bernoulli N=1000 T=10000 K=9 N_JOBS=1 \
  python3 main.py configuration.py  # on only one core

$ BAYES=False ARM_TYPE=Bernoulli N=1000 T=10000 K=9 N_JOBS=-1 \
  python3 main.py configuration.py  # parallel on all the cores

$ BAYES=False ARM_TYPE=Bernoulli N=1000 T=10000 K=9 N_JOBS=100 \
  python3 main.py configuration.py  # parallel with too many jobs
```

Code Example 3.11 – Running a simulation using either one CPU core with `N_JOBS=1`, or all of them with `N_JOBS=-1`, or more jobs than CPU cores with `N_JOBS=20` (if there are 4 cores). The later option is not advised, and `N_JOBS=-1` gives the optimal speed-up factor.

Effect of the number of jobs. As long as the number of jobs (`N_JOBS` here) is less then or equal to the number of physical cores in the CPU of the computer, the final speed-up in terms of total computation runtime is almost optimal. But jobs are implemented as threads, so the speed-up cannot be more than the number of cores, and using for instance 20 jobs on 4-cores for the 20 repetitions is sub-optimal, as the CPU will essentially spend all its time (and memory) managing the different jobs, and not actually doing the simulations. Using the above example, we illustrate the effect of using multi-jobs and multi-cores on the time efficiency of simulations using SMPyBandits. We consider three values of `N_JOBS`, 1 to use only one core and one job, 4 to use all the 4 cores of my $i5$ Intel CPU, and 100 jobs. We give in Table 3.1 an example of running time of an experiment with $T = 10000$, and different number of repetitions and number of jobs. These three experiments correspond to the three lines of Code Example 3.11 shown above. The results of Table 3.1 clearly illustrate that using more jobs than the number of CPU is sub-optimal, and that as soon as the number of repetitions is large enough, using one job by available CPU core (*i.e.*, here 4 jobs) gives a significant speed-up time. Due to the cost of orchestrating the different jobs, and memory exchanges at the end of each repetition, the parallel version is *not* 4 times faster, but empirically we always found it to be 2 to 3.5 times faster. Similar conclusions can be drawn from Figure 3.14, which reports the results of the same experiment on a machine with $n_{\text{cores}} = 12$ CPU cores. More details are included in the documentation page `SMPyBandits.github.io/About_parallel_computations.html`

Repetitions	`N_JOBS`$= 1$	`N_JOBS`$= 4$ $(= n_{\text{cores}})$	`N_JOBS`$=20$ $(> n_{\text{cores}})$
$N = 1$ repetition	15 s	26 s (0.57%)	43 s (0.34%)
$N = 10$ repetitions	87 s	51 s (1.70%)	76 s (1.14%)
$N = 100$ repetitions	749 s	272 s (2.75%)	308 s (2.43%)
$N = 500$ repetitions	2944 s	1530 s (1.92%)	1846 s (1.59%)

Table 3.1 – Effect on the running time of using `N_JOBS` jobs in parallel, on a machine with $n_{\text{cores}} = 4$ CPU cores, for a simulation with 9 different algorithms, for $K = 9$ arms, a time horizon of $T = 10000$. The percentages are the relative gains in comparison with the non-parallel simulation.

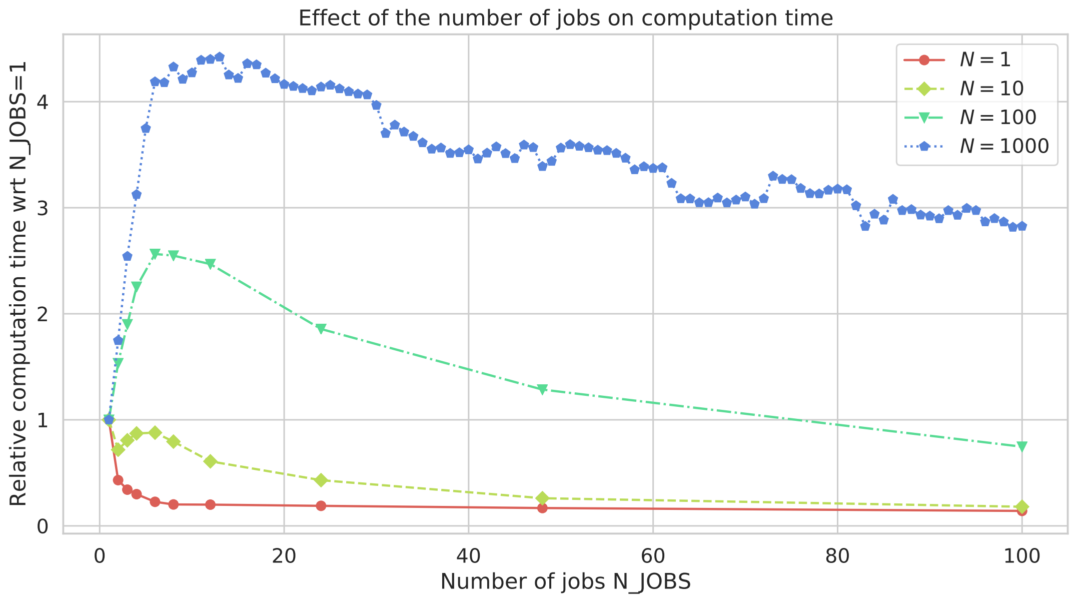


Figure 3.14 – Relative gain on the running time in comparison to using one job, as a function of the number of jobs `N_JOBS`, for a $n_{\text{cores}} = 12$-core machine running the simulation of Code Example 3.11.

Chapter 4

Expert aggregation for online MAB algorithms selection

In this chapter, we discuss the question of online algorithm selection. We tackle the question of how to select a particular bandit algorithm when a practitioner is facing a particular (unknown) bandit problem. Instead of always choosing a fixed algorithm, or running costly benchmarks before real-world deployment of the chosen algorithm, we advise to select a few candidate algorithms, where at least one is expected to be very efficient for the given problem, and use online algorithm selection to automatically and dynamically decide the best candidate. We proposed an extension of the Exp4 algorithm for this problem, Aggregator, and illustrate its performance on some bandit problems.

Contents

4.1 Different approaches on algorithm selection

For any real-world applications of MAB algorithms, several solutions have been explored based on various models, and for any model, typically there are many algorithms available, as we have seen above. We gave in Section 2.4 an overview of the main families of algorithms designed to tackle stochastic MAB problems, and we mainly focused on the UCB_1, kl-UCB and Thompson sampling algorithms. But a rich research literature has produced many different algorithms tackling the MAB problem, as suggested for instance by the length of the reference book [LS19]. Thus, when a practitioner is facing a problem where MAB algorithms could be used, it is not an easy task to decide which algorithm to use. It is hard to predict which solution could be the best for real-world conditions at every instant, and even by assuming a stationary environment, when one is facing a certain problem but has limited information about it, it is hard to know beforehand which algorithm can be the best solution.

In this chapter, we first present two naive approaches for selecting an algorithm when facing a new problem, and then we detail the online approach that uses a "leader" MAB algorithm running on top of a pool of "followers" algorithms, and we present our contribution that is a new "leader" algorithm based on $\mathrm{Exp4}$.

Publication. This chapter is based on our article [BKM18].

Outline. This chapter is organized as follows. First, we give more motivations in the rest of this section, then we explain how to combine such stationary MAB algorithms and aggregate them. We present our proposed algorithm, called $\mathrm{Aggregator}$, in Section 4.2, Finally, we present numerical experiments in Section 4.3, on Bernoulli and non-Bernoulli MAB problems, comparing the regret of several algorithms against different aggregation algorithms. Theoretical guarantees are shortly discussed, before we conclude in Section 4.4.

4.1.1 Motivation for online algorithm selection

Many different learning algorithms have been proposed by the machine learning community, and most of them depend on several parameters, for instance α for UCB_1, the prior for Thompson sampling or BayesUCB, the kl function for kl-UCB etc. Every time a new MAB algorithm $\mathcal{A}$ is introduced, it is compared and benchmarked on some bandit instances, parameterized by $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)$, usually by focusing on its expected regret $R_T^{\mathcal{A}}$. For a known and specific instance, simulations help to select the best algorithm in a pool of algorithms, but when one wants to tackle an *unknown* real-world problem, one expects to be efficient against *any* problem, of any size and complexity in a certain family: ideally one would like to use an algorithm that can be applied identically against any problem of such family.

Naive approaches: On the one hand, a practitioner can decide to pick one algorithm, maybe because it seems efficient on other problems, or maybe because it is simple enough to be used in its application. It might be unrealistic to implement complicated algorithms on limited hardware such as embedded chips in a very low-cost IoT end-device, and for instance a practitioner could chose to only consider the UCB_1 algorithm (or other low-cost algorithms). On the other hand, if prior knowledge on the application at hand is available, one could implement some benchmarks, and compare a set of algorithms on different problems. If a leader appears clearly, it is then possible to choose it for the application.

Illustrative example: For instance, if you know that the considered problem can either have K arms with very close means, or one optimal arm far away from the other, two versions of UCB_1 will perform quite differently in the two problems: using a large α, *i.e.*, favoring exploration, will give low regret in the first case, while using a low α, *i.e.*, favoring exploitation, will give low regret in the second case. One approach can be to use an intermediate value, as $\alpha = 1/2$ suggested by theory, but another approach could be to consider an aggregated vote of different versions of UCB_1, each running with a different value of α (*e.g.*, in a logarithmic grid), and let another learning algorithm decide which value of α is the best *for the problem at hand*.

4.1.2 Online algorithm selection with expert aggregation

Another possibility is to consider an online approach, which is interested in the case where the computation power or memory storage of the application is not a limitation factor, but where one cannot run benchmarks before deploying the application. We consider a fixed set of algorithms, and we use another learning algorithm on top of this set, to learn *on the fly* which one should be trusted more, and eventually, used on its own.

The aggregation approach is especially interesting if we know that the problem the application will face is one of a few known kinds of problems. In such cases, if there are N different sorts of problems, and if the practitioner has prior knowledge on it, one can use the naive approach to select an algorithm $\mathcal{A}_i$ which should perform well on problem i, for $i \in [N]$, and use the aggregation of $\mathcal{A}_1, \dots, \mathcal{A}_N$ when facing the unknown problem.

As we said, a third possible approach is to select *on the run* the best algorithm for a specific situation, for which the so called *expert aggregation algorithms* framework can be useful. To the best of our knowledge, aggregation algorithms, such as $\mathrm{Exp4}$ which dates back from 2002 [ACBF02], have never been used in practice for stochastic MAB problems. For instance, if we look at the applications of MAB for Opportunistic Spectrum Access, like it was proposed in [JEMP09, JEMP10, JMP12], online selection of MAB algorithms was never discussed, and usually the considered algorithm is simply UCB_1, without discussion regarding this choice.

4.1.3 Aggregating bandit algorithms

We assume to have $N \geq 2$ MAB algorithms, $\mathcal{A}_1, \dots, \mathcal{A}_N$, and let $\mathcal{A}_{\text{aggr}}$ be an aggregation algorithm, which runs the N algorithms in parallel (with the same slotted time), and use them to choose its arms based on a voting from their N decisions. $\mathcal{A}_{\text{aggr}}$ depends on a pool of algorithms and a set of parameters. We would like that $\mathcal{A}_{\text{aggr}}$ performs almost as well as the best of the $\mathcal{A}_a$, with a good choice of its parameters, independently of the MAB problem. Ideally $\mathcal{A}_{\text{aggr}}$ should perform similarly to the best of the $\mathcal{A}_a$. To simplify the presentation, we restrict in Algorithm 4.1 to bandit algorithms that give deterministic recommendations: one arm is chosen with probability 1 and the others with probability 0. However, both $\mathrm{Exp4}$ and $\mathrm{Aggregator}$ can be adapted to aggregate randomized bandit algorithms, *i.e.*, algorithms that output a probability distribution $q(t) = (q_k(t))_{k\in[K]}$ over the K arms at each time step, and draw the next selected arm according to this distribution. For instance, Thompson sampling uses its posterior distribution, or UCB can use a distribution with a mass of $1/n$ if there are $n \geq 2$ arms of maximal index, in case of tie in the $\arg\max$.

The aggregation algorithm maintains a probability distribution π^t on the N algorithms $\mathcal{A}_a$ (*i.e.*, $\pi^t \in \Delta_N$ the simplex of dimension N), starting from a uniform distribution: π^t_a is the probability of trusting the decision made by algorithm $\mathcal{A}_a$ at time t. $\mathcal{A}_{\text{aggr}}$ then simply performs a weighted vote on its algorithms: it first decides whom to trust by sampling $a \in [N]$ from π^t, then follows $\mathcal{A}_a$'s decision: $a \sim \pi^t, A_{\text{aggr}}(t) = A(t) = A_a(t)$. We prove below that it is equivalent to first let all the algorithms decide their arm, *i.e.*, $A_a(t)$, and then to compute $\forall k \in [K], p^t_k \doteq \sum_{a=1}^N \pi^t_a \times \mathbb{1}(\{A_a(t) = k\})$, the weighted probability that one of the N aggregated algorithms chose arm k, and finally to sample $A(t) \sim \pi^t$.

Proposition 4.1. *The two following sampling schemes for the aggregated algorithm $\mathcal{A}_{\text{aggr}}[\mathcal{A}_1, \dots, \mathcal{A}_N]$ are equivalent,* i.e., *they give the same distribution on the action chosen by $\mathcal{A}_{\text{aggr}}$, $A_{\text{aggr}}(t) = A(t)$.*

1. *Sample $a(t) \sim \pi^t$ first, then trusts $\mathcal{A}_{a(t)}$'s decision: $A_{\text{aggr}}(t) = A(t) = A_{a(t)}(t)$,*
2. *Let $A_a(t)$ be the arm chosen by algorithm $\mathcal{A}_a$ (sampled from q^t_a), for each $a \in [N]$, and compute $\forall k \in [K], p^t_k \doteq \sum\limits_{a=1}^{N} \pi^t_a \times \mathbb{1}(A_a(t) = k)$. Then, play $A_{\text{aggr}}(t) = A(t) \sim p^t$.*

Proof. For any fixed time step $t \in [T]$, we denote respectively $P(t)$ and $P'(t)$ the distribution of action of the aggregated algorithm $\mathcal{A}_{\text{aggr}}$ at time t, respectively under the first and second sampling scheme. Let $q^t_a \in \Delta_K$ be the distribution of the chosen action by algorithm $\mathcal{A}_a$ (for $a \in [N]$), that depends on the past observations (as well as some external randomness, see Section 2.1). The first sampling scheme gives $P_k(t) = \mathbb{P}(A_{\text{aggr}}(t) = k) = \mathbb{P}(\cup_{a=1}^N A_{\text{aggr}}(t) = A_a(t) = k, a(t) = a) = \sum_{a=1}^N \mathbb{P}(A_a(t) = k)\mathbb{P}(a(t) = a) = \sum_{a=1}^N q_{a,k}(t)\pi^t_a$, by independence of $a(t) \sim \pi^t$ and $A_i(t) \sim q_i(t)$ (for all i). The second sampling scheme gives $P'_k(t) = \mathbb{P}(A_{\text{aggr}}(t) = k) = \mathbb{E}[p^t_k]$ by definition, and $\mathbb{E}[p^t_k] = \sum_{a=1}^{N} \pi^t_a \mathbb{E}[\mathbb{1}(A_a(t) = k)] = \sum_{a=1}^{N} \pi^t_a \mathbb{P}(A_a(t) = k) = \sum_{a=1}^{N} q_{a,k}(t)\pi^t_a$. Thus we showed that $P_k(t) = P'_k(t)$, so the two sampling schemes are equivalent, as claimed. □

The main questions are then to know what observations (*i.e.*, arms and rewards) should be given as feedback to which algorithms, and how to update the trusts at each step, and our proposal Aggregator differs from Exp4 on these very points. The considered aggregation algorithms are special cases of the well-known *multiplicative weights update algorithm*.

4.2 Our contribution: the Aggregator algorithm

Our proposed Aggregator is detailed in Algorithm 4.1. At every time step, after having observed a reward $r(t) = Y_{A(t),t}$ for its chosen action $A(t)$, the algorithm updates the trust probabilities from π^t to π^{t+1} by a multiplicative exponential factor (using the learning rate and the *unbiased* reward). Only the algorithms $\mathcal{A}_a$ who advised the last decision get their trust updated, in order to trust more the "reliable" algorithms.

Unbiased estimate of the rewards. The reward estimate is unbiased in the following sense. If one had access to the samples $Y_{k,t}$ for all arms k, the reward incurred by algorithm a at time t would be $r^a(t) = Y_{A_a(t),t}$. But we are not in the full information setting (see [CBL06]), but in the bandit setting, so only one of the samples $Y_{k,t}$ can be observed, $r(t) = Y_{A(t),t}$. This quantity can only be observed for those algorithms a for which $A^t_a = A(t)$. However, by dividing by the probability of observing this recommendation, one obtains an unbiased estimate of $r^a(t)$. More precisely, if we define the estimate by

$$\widehat{r^a}(t) \doteq \frac{Y_{A_a(t),t}}{p^t_{A_a(t)}} \mathbb{1}(A_a(t) = A(t)) \tag{4.1}$$

then it satisfies $\mathbb{E}_{O_t}[\widehat{r^a}(t)|O_t] = Y_{A_a(t),t}$, for all a, where the expectation is taken conditionally to the history of observations up to round t, O_t. Observe that $\widehat{r^a}(t) = 0$ for all algorithms a such that $A_a(t) \neq A(t)$.

An important feature of Aggregator is the feedback provided to each underlying bandit algorithm, upon the observation of arm $A(t)$. Rather than updating only the trusted algorithms (that is the algorithms which would have drawn arm $A(t)$) with the observed reward $r(t)$, we found that updating each algorithm with the (original) reward observed for arm $A(t)$ improves the performance drastically. As expected, the more feedback they get, the faster the underlying algorithms learn, and the better the aggregation algorithm is. This intuition is backed up by theory explained in [MM11].

```
1  Input: $N$ bandit algorithms, $\mathcal{A}_1, \dots, \mathcal{A}_N$, with $N \geq 2$
2  Input: A sequence of learning rates, $(\eta_t)_{t \geq 1}$, e.g., $\eta_t = \ln(N)/(tK)$
3  Data: Initial uniform distribution, $\pi^0 = \mathcal{U}([N])$
4  Result: $\mathcal{A}_{\mathrm{aggr}} = \mathrm{Aggregator}\,[\mathcal{A}_1, \dots, \mathcal{A}_N]$
5  for $t = 1, \dots, T$ do                                    // At every time step
6      for $a = 1, \dots, N$ do                                // Can be parallel
7          $\mathcal{A}_a$ updates its internal state (e.g., UCB indexes);
8          It chooses $A_a(t) \in [K]$.
9      end
10     Let $p^t_k \doteq \sum_{a=1}^{N} \pi^t_a \times \mathbb{1}(A_a(t) = k), \;\forall k \in [K]$;
11     Then $\mathcal{A}_{\mathrm{aggr}}$ chooses arm $A(t) \sim p^t$;
12     Give original reward $(A(t), r(t))$ to each $\mathcal{A}_a$ (maybe not on its chosen arm);
13     Compute an unbiased estimate of the reward, $\widehat{r}(t) = r(t)/p^t_{A(t)}$;
14     for $a = 1, \dots, N$ do
15         if $\mathcal{A}_a$ was trusted, i.e., $A_a(t) = A(t)$ then
16             $\pi^{t+1}_a = \exp(\eta_t \widehat{r}(t)) \times \pi^t_a$ ;        // Trusted more
17         else
18             $\pi^{t+1}_a = \pi^t_a$ ;                          // Do not update the trust now
19     end
20     Renormalize the new weights $\pi^{t+1}$: $\pi^{t+1} \doteq \pi^{t+1} / \sum_{a=1}^{N} \pi^{t+1}_a$.
21 end
```

Algorithm 4.1: Our algorithm Aggregator aggregating $\mathcal{A}_1, \dots, \mathcal{A}_N$.

Regarding the update of π^t, one can note that the trust probabilities are not all updated before the normalization step, and an alternative would be to increase π_a if $A_a(t) = A(t)$ and to *decrease it otherwise*. It would not be so different, as there is a final renormalization step, and empirically this variation has little impact on the performance of Aggregator.

Comparison with Exp4: The Exp4 algorithm dates from [ACBFS02], and for instance it is also well explained in Section 4.2 of [BCB12]. It is similar to Aggregator, presented in Algorithm 4.1, but differs in the two following points. The first difference is that $a \sim \pi^t$ is sampled first and the arm chosen by $\mathcal{A}_a$ is trusted, whereas Aggregator needs to listen to the N decisions to perform the updates on π^t. Then, Exp4 gives back an observation (*i.e.*, a pair arm, reward) only to the last trusted algorithm whereas Aggregator gives it to all algorithms. The second difference is that after having computed the reward estimate $\widehat{r^a}(t)$, Exp4 updates the estimated cumulative reward for each algorithm, $\widetilde{R}_a(t) = \sum_{s=1}^{t} \widehat{r^a}(t) \times \mathbb{1}(A^s_a = A^s_{\mathrm{aggr}})$. Instead of updating π^t multiplicatively as we do for our proposal, Exp4 recomputes it, proportionally to $\exp(\eta_t \widetilde{R}_a(t))$. Note that there is no difference for the case of constant learning rates, and it does not differ much for decreasing learning rates as well.

We also note that Exp4 uses *losses* instead of rewards, with $\ell(t) = 1 - r(t)$, but this is only a change of vocabulary.

How to choose the learning rates (η_t): The sequence of non-negative learning rates $(\eta_t)_{t\geq 1}$ used by Exp4 can be arbitrary. It can be constant but should be non-increasing [BCB12, Theorem 4.2]. If the horizon T is known (and fixed), the best choice is given by $\eta_t \doteq \eta = 2\ln(N)/(TK)$. However, for real-world communication problems, it can be considered unrealistic to assume a fixed and known time horizon. For more discussion on this hypothesis, we refer to Section 1 of our article [BK18b], that we shortly present in Appendix A. Thus we prefer the alternative horizon-free choice of learning rates, $\eta_t \doteq \ln(N)/(tK)$ suggested by [BCB12]. It is non-increasing, and is obtained by minimizing the upper-bound on the regret derived in [BCB12, pp48]. We compared both approaches empirically, and the second one usually performs better. We also stick to this sequence $(\eta_t)_{t\geq 1}$ of decreasing learning rates for Aggregator.

4.3 Experiments on simulated MAB problems

We focus on *i.i.d.* MAB problems, with $K = 9$ arms[1]. For Bernoulli problem, the first one uses $\boldsymbol{\mu} = [0.1, \dots, 0.9]$, and is considered as a "simple" problem. The second one is divided in three groups: 2 very bad arms ($\mu = 0.01, 0.02$), 5 average arms ($\mu = 0.3$ to 0.6) and 3 very good arms ($\mu = 0.78, 0.8, 0.82$), and it is considered as a "harder" problem. The horizon is set to $T = 20000$ (but its value is unknown to all algorithms), and simulations are repeated 1000 times, to estimate the expected regret. This empirical estimation of the expected regret R_T is plotted below[2], as a function of T, comparing some algorithms $\mathcal{A}_1, \dots, \mathcal{A}_N$ (for $N = 6$), and their aggregation with Aggregator (displayed in orange bold), using the parameter-free learning rate sequence, $\eta_t \doteq \ln N/(tK)$.

Note that for each of the $N = 1000$ simulations, we choose to generate all the rewards beforehand, *i.e.*, one full matrix $(r_k(t))_{1\leq k\leq K, 1\leq t\leq T}$ (for every repetition), in order to compare the algorithms on the same realizations[3] of the MAB problem.

We compare our Aggregator algorithm, as well as other aggregation algorithms, Exp4 from [BCB12], CORRAL from [ALNS17] and LearnExp from [AHK17], all with their default parameters. The aggregated algorithms consist in: a naive uniform exploration (to have at

[1] Similar behaviors are observed for other values, *e.g.*, trying up-to $K = 100$ gave the same results.

[2] The Lai & Robbins' logarithmic lower-bound [LR85] is also plotted. It corresponds to the second case in Theorem 2.7. It is crucial to note that the lower-bound is only asymptotic, and as such one should not be surprised to see regret curves smaller than the lower-bound (*e.g.*, for the easier Bernoulli problem in Figure 4.1).

[3] Similar plots and similar results are obtained if this choice is not made, but it makes more sense to compare them against the same randomization. Note that this choice is the default in SMPyBandits, but the other possibility is implemented as well, by changing `cache_rewards` to `true` or `false` in the `configuration` dictionary. See Section 3.6.3 for explanations on this `configuration.py` file, or the online documentation.

least one algorithm with bad performances, *i.e.*, linear regret, that is *not* included in the plots to reduce clutter), UCB with $\alpha = 1/2$, three kl-UCB algorithms (resp. with Bernoulli, Gaussian and exponential kl functions), and BayesUCB and Thompson sampling with uniform prior.

Figures 4.1 and 4.4 are in semilog-y scale, this helps to see that the best algorithms can be an *order of magnitude* more efficient than the worst, and the $\mathrm{Aggregator}$ performs similarly to the best ones, when the other aggregation algorithms are usually amongst the worst. Figure 4.5 is in semilog-x scale to show that the regret of efficient algorithms are indeed logarithmic.

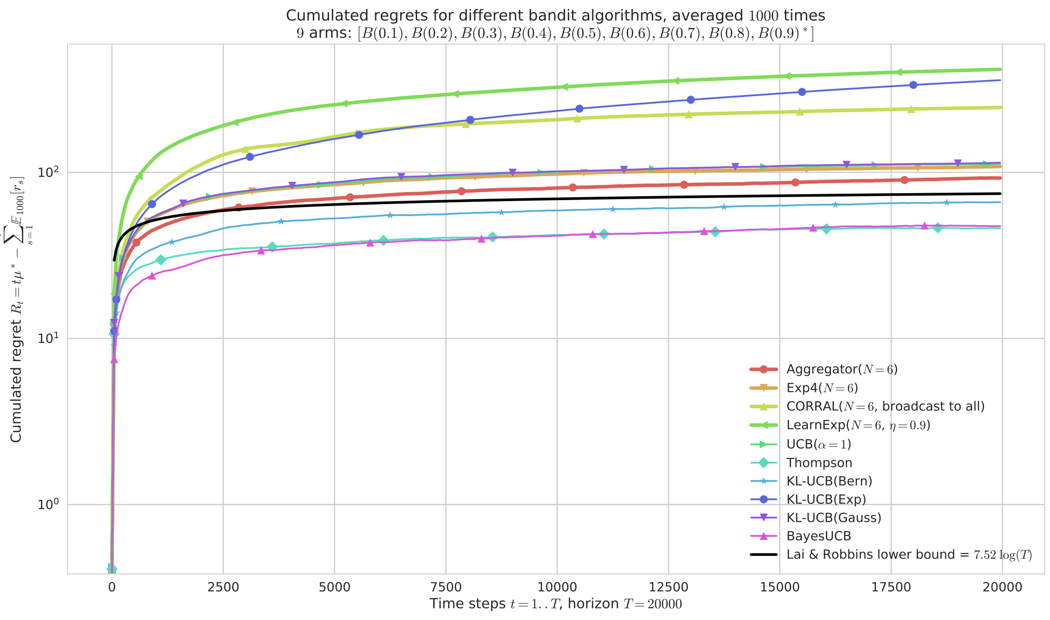


Figure 4.1 – On a "simple" Bernoulli problem (semilog-y scale).

For Bernoulli problems (Figures 4.1 and 4.2), UCB with $\alpha = 1/2$, Thompson sampling, BayesUCB and kl-UCB$^+$ (with the binary kl function) all perform similarly, and $\mathrm{Aggregator}$ is found to be as efficient as all of them. For Gaussian and exponential arms, rewards are truncated into $[0, 1]$, and the variance of Gaussian distributions is fixed to $\sigma^2 = 0.05$ for all arms, and can be known to the algorithms (the kl function is adapted to this one-dimensional exponential family). Figure 4.3 uses only Gaussian arms, with a large gap between their means and a relatively small variance, giving an "easy" problem. And Figure 4.4 shows a considerably harder "mixed" problem, when the distributions are no longer in the same one-dimensional exponential family and so the Lai & Robbins' lower-bound no longer holds.

For each of the 4 problems considered, the $\mathrm{Aggregator}$ algorithm with default option (*i.e.*, broadcast rewards to all the aggregated algorithms) is the best of all the aggregation algorithms, and its regret is very close to the best of the aggregated algorithms. Especially in difficult problems with mixed or unknown distributions, $\mathrm{Aggregator}$ showed to be more efficient

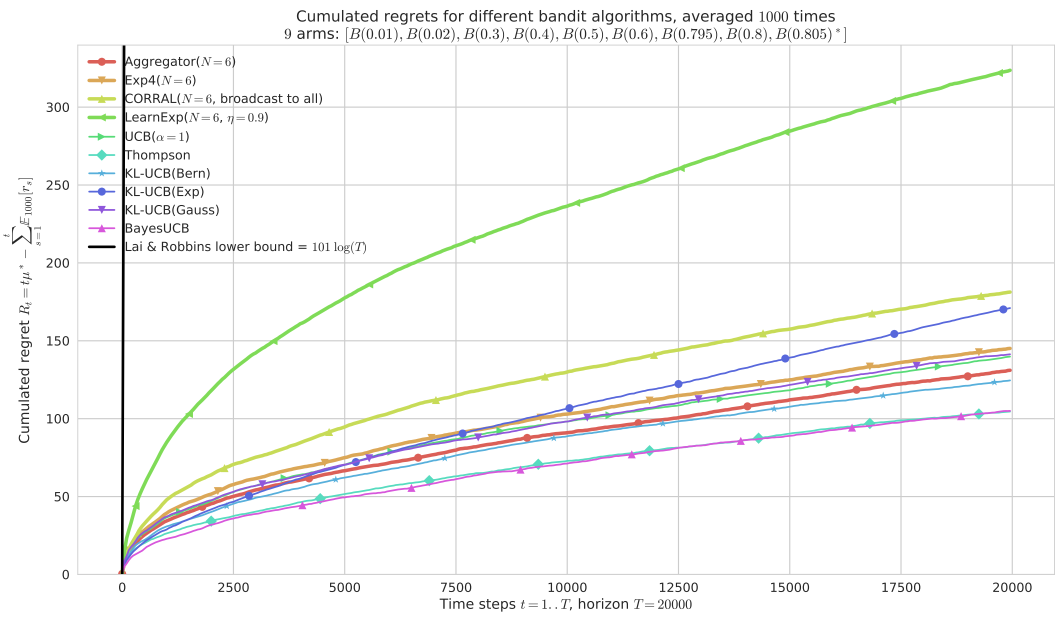


Figure 4.2 – On a "harder" Bernoulli problem, they all have similar performances, except LearnExp, and our proposal Aggregator outperforms its competitors.

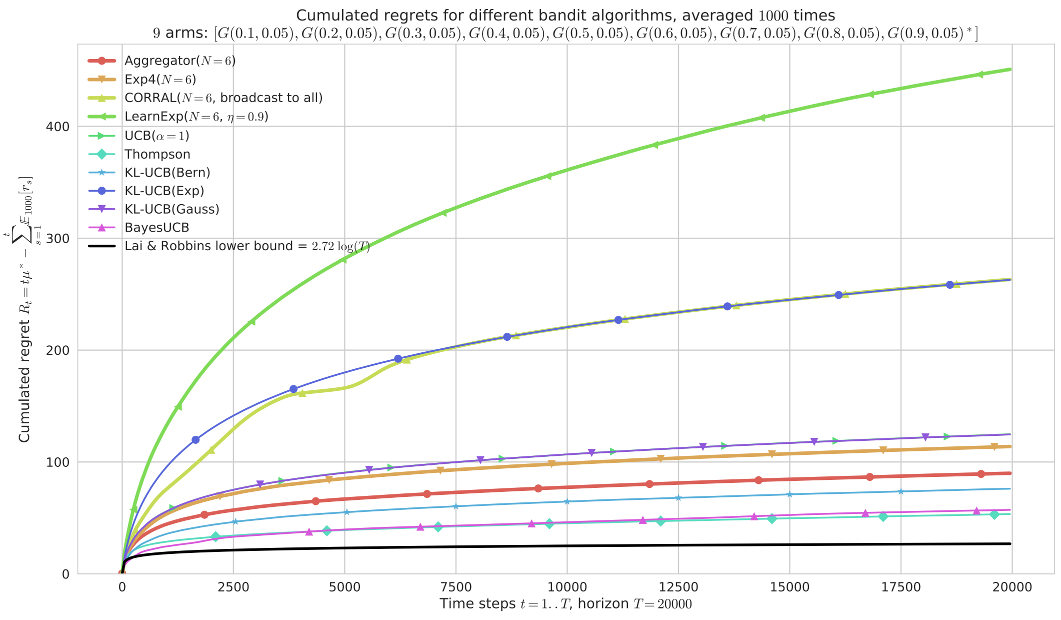


Figure 4.3 – On an "easy" Gaussian problem, only Aggregator shows reasonable performances, thanks to BayesUCB and Thompson sampling.

that Exp4 and orders of magnitude better than the other reference aggregation algorithms LearnExp and CORRAL (see Figures 4.4 and 4.5).

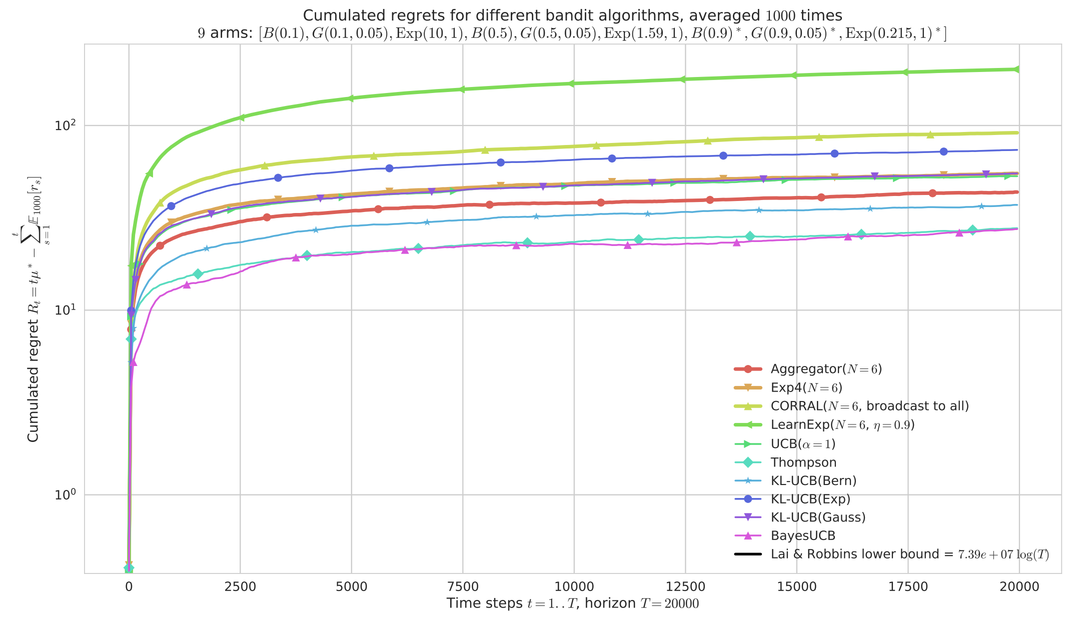


Figure 4.4 – On a harder problem, mixing Bernoulli, Gaussian, Exponential arms, with 3 arms of each type with the *same mean*.

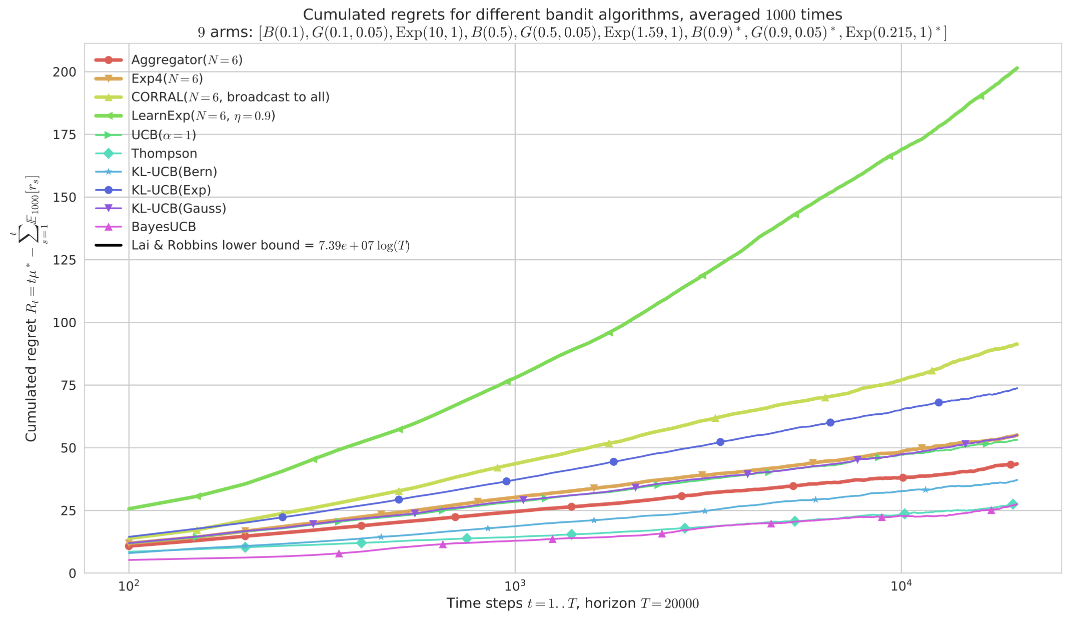


Figure 4.5 – The semilog-x scale clearly shows the logarithmic growth of the regret for the best algorithms and our proposal Aggregator, even in a hard "mixed" problem (*cf.* Figure 4.4).

4.3.1 No theoretical guarantees.

The $\mathrm{Aggregator}$ does not have satisfying theoretical guarantees in terms of regret R_T, unlike many bandit algorithms. Another notion, the *adversarial regret*, denoted by $\overline{R_T}$, measures the difference in terms of rewards, between the aggregation algorithm $\mathcal{A}_{\text{aggr}}$ and the best aggregated algorithm $\mathcal{A}_a$. This is in contrast with the (classical) regret, which measure the difference with the best fixed-arm strategy (Definition 2.3). Thus, even if the aggregated algorithms have logarithmic (classical) regret, having an adversarial regret scaling as $\sqrt{T}$ does not permit to obtain a logarithmic (classical) regret for the aggregation algorithm. Under some additional hypotheses, [BCB12, Theorem 4.2] proves that $\mathrm{Exp4}$ have a bounded adversarial regret, $\overline{R_T} \leq 2\sqrt{TN\ln(K)}$, with the good choice of the learning rate sequence $(\eta_t)_{t\geq 1}$.

Our proposed algorithm follows quite closely the architecture of $\mathrm{Exp4}$, and a similar bound for $\mathrm{Aggregator}$ is expected to hold. This would be a first theoretical guarantee, but not satisfactory as we saw above that simple algorithms (like UCB) have regrets scaling as $\ln(T)$ [ACBF02, BCB12], not $\sqrt{T}$. Regret bounds in several different settings are proven for the CORRAL algorithm [ALNS17], but no logarithmic upper-bound can be obtained from their technique, even in the simplest setting of stochastic bandits. However, $\mathrm{Aggregator}$ always seems to have a (finite-horizon) logarithmic regret in all the experiments we performed, for both Bernoulli and non-Bernoulli problems (*e.g.*, Gaussian, exponential and Poisson distributions).

4.4 Conclusion

In this chapter, we presented the first contribution of this manuscript, that corresponds to our article [BKM18]. We presented the use of aggregation algorithms, especially for the real-world setting of unknown problem instances, when tuning parameters before-hand is no longer possible and an adaptive algorithm is preferable. Our proposed Aggregator was presented in details, and we also highlighted its differences with Exp4.

We presented experiments on simple MAB problems, coming from previous work on bandits for OSA [JEMP10], and the simulations results showed that Aggregator is able to identify on the fly the best algorithm to trust for a specific problem, as expected. Experiments on problems mixing different families of distributions were also presented, with similar conclusions in favor of Aggregator.

Exp4 has theoretical guarantees in terms of adversarial regret, and even if the same result could hold for Aggregator, results in terms of classical regret are yet to be proven. Empirically, Aggregator showed to always have a logarithmic regret R_T if it aggregates algorithms with logarithmic regrets (like UCB, kl-UCB, Thompson sampling, BayesUCB etc). It usually succeeds to be close to the best of the aggregated algorithms, both in term of regret and best arm pull frequency. As expected, the Aggregator is never able to outperform any of the aggregated algorithms, but this was an over-optimistic goal.

What matters the most is that, empirically, Aggregator is able to quickly discover *on the fly* the best algorithms to trust, and then performs almost as well as if it was following it from the beginning. Our Aggregator algorithm could probably be rewritten as an Online Mirror Descent (for more details, see for instance [H$^+$16, ZS18]), as it is the case for both Exp4 and CORRAL. But this direction does not appear very useful, as in the case of CORRAL the analysis cannot bring a logarithmic regret bound, even by aggregating asymptotically optimal algorithms.

Reproducility of the experiments. Experiments presented in this Section are based on our library SMPyBandits, and the page `SMPyBandits.GitHub.io/Aggregation.html` gives instructions to reproduce them.

Part II

Multi-Armed Bandit Models for Internet of Things Networks

Chapter 5

Improving Spectrum Usage of IoT Networks with Selfish MAB Learning

In this chapter, we propose two models of Internet of Things (IoT) networks, composed of many independent IoT end-devices, that can use low-cost RL (Reinforcement Learning) algorithms to learn to improve their spectrum access. Decentralized RL for IoT lets the devices use acknowledgements sent by their Base Station as a reward, instead of sensing feedback like for OSA. We consider many independent "dynamic" devices, communicating with a small probability at every instant. Simulations show that dynamic devices can improve their spectrum efficiency, by using MAB algorithms like UCB. We also developed a proof-of-concept using USRP platforms, for a real-world validation of this approach. The first model is interesting for its simplicity, and because considering no retransmission can improve the battery life of IoT devices, at the cost of a lower Quality of Service (QoS).

However, in case of failed transmission of a message, a dynamic device can also retransmit it up-to a fixed number of retransmission. We compare different heuristics, based on UCB, and numerical simulations confirm that the non-naive heuristics significantly improve the network efficiency. The second model is more interesting if the main target is an improvement of the QoS, rather than a longer battery life.

Contents

5.1 Introduction and motivations for MAB learning for IoT Networks

After the first chapters that presented the model of MAB, we go back to the initial problems studied in this thesis, and thus we focus on Internet of Things (IoT) networks. As explained in the introduction in Chapter 1, unlicensed bands are more and more used and considered for mobile and LAN communication standards (WiFi, LTE-U), and for Internet of Things (IoT) standards for short-range (ZigBee, Z-Wave, Bluetooth) and long-range (LoRaWAN, SIGFOX, Ingenu, Weightless) communications [CVZZ16]. This heavy use of unlicensed bands, in particular with the expected exponential growth of the number of IoT devices, will cause performance drop, due to radio collisions that could even compromise IoT promises.

Efficient Medium Access (MAC) policies allow devices to avoid interfering traffic and can significantly reduce the spectrum contention problem in unlicensed bands. As end-devices battery life is a key constraint of IoT networks, and as IoT networks are decentralized, because the devices initiate transmissions, this leads to IoT protocols using as low signaling overhead as possible and simple ALOHA-based mechanisms. In this chapter, we analyze the performance of Multi-Armed Bandits (MAB) algorithms, that could be used in combination with a time-frequency slotted ALOHA-based protocol. We highlight that even without changing anything on the level of IoT standards, our proposal is just an add-on capability that can be used on a unit-per-unit basis. We consider the Upper-Confidence Bound (UCB) [ACBF02], and the Thompson-Sampling (TS) algorithms [Tho33, AG12, KKM12], for the first model. For the demonstration as well as for the second model, without loss of generality, we preferred to focus on heuristics based on the simplest algorithm (*i.e.*, UCB), to give a clear presentation of the different ideas explored to solve the problem of learning in order to retransmit efficiently.

We present in Section 5.2 how the MAB algorithms can be used in a unlicensed but frequency- and time-slotted IoT network. Several devices are using bandit algorithms, and the assumptions made by the stochastic bandit algorithms are not satisfied: as several agents learn simultaneously and their activation processes are random, their behavior is not stationary. As far as we know, we provide the first practical study to confirm robustness of the use of stochastic bandit algorithms for decision making in IoT networks with a large number of intelligent devices in the network, which makes the environment "highly not stationary". This specific context makes it very hard to give mathematical proofs of convergence and efficiency for bandit algorithms (that is why we relax the hypothesis and only consider up-to $M \leq K$ players in Chapter 6). We then validate the model with a hardware implementation on real radio signals, detailed in Section 5.3. We conclude this chapter by presenting in Section 5.4 an extension of this model to take into account another aspect of the ALOHA protocol, that is the possibility for dynamic devices to retransmit their packet if the *Ack* was not received.

Publications. This chapter is mainly based on our articles [BBM+17, BBM18, BBM19, BBMVM19, MB19].

5.2 Selfish learning for many dynamic devices with low activation probabilities in an IoT network

As explained before in Chapter 1, the future IoT networks will require to support more and more communicating devices. In this section, we show that intelligent devices in unlicensed bands can use MAB learning algorithms to improve spectral resource exploitation. We evaluate the performance of two classical MAB learning algorithms, UCB and Thomson Sampling, to handle the decentralized decision-making of Spectrum Access, applied to IoT networks. We mean by *decentralized* that learning is performed on the device side, and is spread on the totality of the devices in a network. We also evaluate the learning performance when the number of intelligent end-devices grows. We show that using learning algorithms firstly extends devices battery autonomy (by reducing the packet-loss-ratio or equivalently, by improving the successful-communication rate), and secondly does help to fit more devices in such networks, even when all end-devices are intelligent and are dynamically switching from channel to channel. In the studied settings, stochastic MAB learning has near optimal performance even in non-stationary settings with a majority of intelligent devices.

The aim of this section is to assess the potential gain of learning algorithms in IoT scenarios, even when the number of intelligent devices in the network increases, and the network usage is more and more fluctuatating. To do that, we suppose an IoT network made of two types of devices: *static devices* that use only one channel (fixed in time), and *dynamic devices* that can choose the channel for each of their transmissions. Static devices form an interfering traffic, which could be generated by devices using other standards as well. Note that instead of assuming that each static device uses a fixed channel, we could also assume a looser hypothesis: if each static device uses a fixed sub-set of the K channels, and a purely uniform random access in its set of considered channels, then *in average* the observed occupancy of the K channels can be modeled as if it were occupied by (more) static devices using only one channel. So our first hypothesis is actually not constraining. We first evaluate the probability of collision if dynamic devices randomly select channels (that is, a naive approach), and if a centralized controller would optimally distribute all of them in the channels at the beginning of the scenario. This second approach is ideal, but not realistic the most common situation of decentralized co-located IoT networks, and it is just used here as a reference. Then, these three reference scenarios allow to evaluate the performance of bandit algorithms, such as UCB and TS, in a decentralized network, in terms of successful communication rate, as it reflects the network efficiency. We show that these algorithms have near-optimal performance, even when the proportion of end-devices increases and the interfering traffic from other devices becomes less and less stochastic, more and more fluctuating and unpredictable.

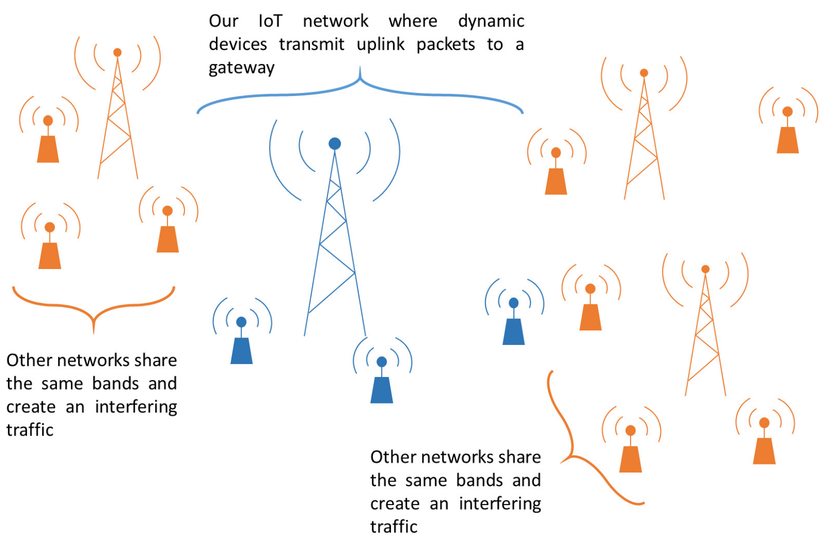

Our IoT network where dynamic devices transmit uplink packets to a gateway
Other networks share the same bands and create an interfering traffic
Other networks share the same bands and create an interfering traffic

Figure 5.1 – In our system model, some dynamic devices (in the IoT network in blue) transmit packets to a gateway and suffer from the interference generated by neighboring networks (in orange left/right).

5.2.1 System model and notations

We present our system model, which consists of one gateway and many IoT devices, using a frequency- and time- slotted protocol.

One gateway and many devices. We consider the system model presented in Figure 5.1, where a set of devices all sends uplink packets to a (unique) network gateway. Messages can be sent in a fixed number $K \geq 2$ of wireless channels, that are assumed to be orthogonal, that is, a message in channel k will not cause collision to a message in another channel $j \neq k$. The communication between IoT devices and this gateway is done through a simple pure acknowledged ALOHA-based protocol where devices transmit uplink packets of fixed duration whenever they want. We mean here by ALOHA-based that *devices do not use sensing*, and *we do not consider retransmissions of packets* (we relax this hypothesis in Section 5.4).

The devices can transmit their packets in one of the K channels. In the case where the gateway –of the corresponding IoT network– receives an uplink message in one channel, it transmits an acknowledgement to the end-device in the same channel, after a fixed delay. This is realistic in IoT networks such as networks based on LoRaWAN. In a more general setting, if there were more than one gateway in the same IoT network being considered, the network would reply to the device by letting only one gateway sends back an acknowledgement. The natural choice is for the network to send this acknowledgement by the gateway which received the uplink message. For simplicity but without loss of generality, we consider a network with only one gateway, but still it is important to observe the distinction between the gateway, which is simply a RF node, and the IoT network in charge of receiving, handling, and replying to the incoming uplink messages from the IoT devices being paired in the network.

These communications operate in unlicensed ISM bands and, consequently, they can suffer from interference generated by uncoordinated co-localized and/or neighboring networks (two are shown in orange in Figure 5.1, in left or right). This interfering traffic is uncontrolled, and is most likely unevenly distributed over the K different channels, as each IoT protocol or IoT networks can choose to use a different subset of channels. In order to simulate networks designed for the IoT, we consider a protocol **with no sensing** and no repetition of uplink messages. The gateway is in charge of sending back an acknowledgement, after some fixed-time delay, to any device paired with this gateway in this network (*i.e.*, in blue in Figure 5.1) who succeeded in sending an uplink packet. By considering a small number of orthogonal wireless channels, and a unique PHY layer configuration (*i.e.*, modulation, waveform, etc), and in case of a non-uniform traffic in the different channels, the device can improve their usage of the network if they are able to *learn* on the fly the best channels to use, that is, the most vacant one. Indeed, in this model, the *quality* of the channels (*i.e.*, the ressources) are identified by their vacancy, or one minus their occupancy rates.

We note that MAB learning has also started to be applied to also optimize the PHY layer parameters, see for instance [KAF$^+$18] following our article [BBM$^+$17], but in this thesis we chose to restrict to the spectrum access problem and thus we only MAB models where the arms correspond to orthogonal frequency bands.

Slotted protocol. For simplicity, and for consistency with the rest of this thesis, we only consider a discrete protocol (*i.e.*, slotted). As illustrated in Figure 5.2, we suppose a slotted protocol, in both time and frequency. All devices share a synchronized time, and know in advance K, the finite number of available RF channels. In each time slot, the devices try to send packets to the unique gateway, which listens continuously to all channels, following an ALOHA-based communication, with no sensing. Each time slot is divided in two parts: first for uplink communications in which data packets are sent by end-devices to the gateway. In one channel, if only one packet is sent in this part of the slot, the gateway can decode it and sends an acknowledgement to the device in the second part (on the same channel). If two or more devices send an uplink packet in the same slot, the uplink packets collide, and the acknowledgement (*Ack*) is not transmitted. In this case, we say that there is a *collision* in this time slot. In other words, if the gateway cannot successfully decode the incoming (uplink) messages because they were corrupted by collisions, it does not send any *Ack* back. This way, no collision can occur on the downlink messages, easing the analysis of collisions.

Static vs dynamic devices. We make the following assumptions on the network. We assume that there are two types of end-devices in the network:

- *Static* end-devices are identical, and each of them uses only *one* channel to communicate with the gateway, with no loss of generality. Their choice is assumed to be fixed in time

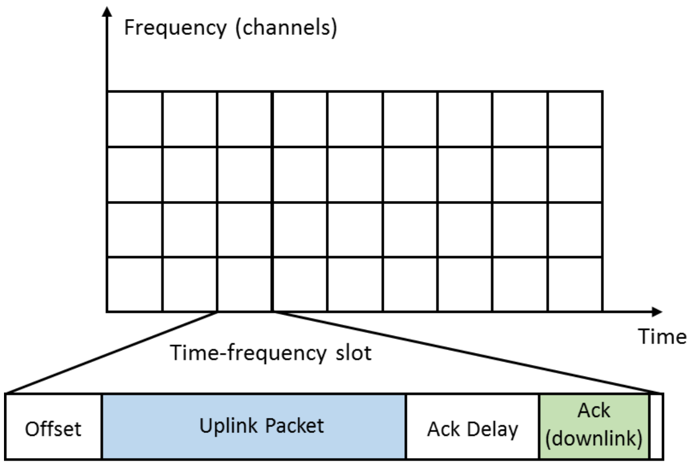


Figure 5.2 – The considered time-frequency slotted protocol. Each frame is composed by a fixed duration uplink slot in which the end-devices transmit their (uplink) packets. If a packet is well received, the gateway replies by transmitting an *Ack*, after the ack delay.

(*i.e.*, stationary). The traffic generated by these devices is considered as an interfering traffic for other devices.

- *Dynamic* (or *smart*) end-devices can use all the available channels, by quickly changing their communication channel at any time slot, following a Machine Learning policy. For that end, they use their history of past communication successes or failures they experienced in each channel, to learn about channel availability. We also assume that the dynamic end-devices can run simple embedded decision making algorithms, and have limited but reasonable computing as well as storage capacities. Note that of course we assume a limited storage capacity, so no device stores the full history of communication successes and failures, but they only store average rates (which can be stored using one float number for each of the K channels). Our goal is to propose a learning algorithm than can be implemented by each dynamic device, in a decentralized and independent manner, in order to improve their successful communication rate automatically.

We further assume that there are $K \geq 2$ channels, $D \geq 0$ dynamic end-devices, and $S \geq 0$ static devices with $0 \leq S_k \leq S$ static devices in channel $k \in [K]$ (so $S = \sum_{k=1}^{K} S_k$).

Random emission patterns: We suppose that all devices follow the same emission pattern, being fixed in time, and we choose to model it as a simple Bernoulli process: all devices have the same probability to send a packet in any (discrete) temporal slot, and we denote $p \in (0, 1)$ this probability[1]. The parameter p essentially controls the frequency of communication for each device, once the time scale is fixed, and $1/p$ is proportional to the *duty cycle*. For instance,

[1] In the experiments below, p is about 10^{-3}, because in a crowded network p should be smaller than $K/(S + D)$ for all devices to communicate successfully (in average).

for IoT devices sending messages with a duration of 1 second and possibly at every second, then on a daily basis we have $p = 1/(12 \times 60 \times 60) \simeq 1.5 \times 10^{-5}$.

Some assumptions on the network occupation: We focus on *dense networks*, in which the number of devices $S+D$ is very large compared to K (for instance, about 1000 to 100000, while K is about 4 to 256). As this problem is only interesting if devices are able to communicate reasonably efficiently with the gateway, we assume that devices only communicate occasionally, *i.e.*, with a low *duty cycle*, as it is always considered for IoT. Note that even unlicensed bands have such limitations, as for instance this is a strict requirement for any device using the 868 MHz ISM band in Europe (and its regulation also enforces a maximum transmission power, but we do not consider power transmission in this work). We prefer this choice rather than non-crowded networks, *i.e.*, where $S + D \leq K$, as the former makes more sense for realistic IoT networks.

On the opposite, imagine if there were only $K = 4$ channels being occupied by $D+S = 100$ devices, each communicating with a high rate of $p = 1/10$, with a non-zero occupancy in each channel (*i.e.*, $\forall k, S_k > 0$). Then, almost all time slots will lead to collisions in all channels, for a uniformly random access scheme, and thus the network efficiency (*i.e.*, successful communication rate) will be so close to zero that using learning algorithms cannot improve much. Such scenario is not our target of interest, and thus we prefer to only consider feasible scenarios where p is smaller than $K/(S + D)$, in order to *have an average of active devices not larger*[2] *than the number of channels*.

Link with the MAB model: All devices follow a Bernoulli emission process. Consider the network from the point of view of one dynamic device: every time a dynamic device has to communicate with the gateway, it has to choose one channel (at each transmission $t \geq 1, t \in \mathbb{N}$), denoted as $A(t) = k \in [K]$. Then, the dynamic device waits in this channel $A(t)$ for an acknowledgement sent by the gateway, during a certain period (*e.g.*, 5 seconds for the example considered above). Before sending another message (*i.e.*, at time $t + 1$), the dynamic device knows if it received or not this *Ack* message. For this reason, selecting channel (or arm) k at time t yields a (random) feedback, called a (binary) *reward*, $r(t) \doteq Y_{k,t} \in \{0, 1\}$, being 0 if no *Ack* was received before the next message, or 1 if *Ack* was successfully received. The goal of the dynamic device is to minimize its packet loss ratio, or equivalently, to maximize its number of successful transmissions, or its cumulative reward, $\sum_{t=1}^{T} r(t)$, which is the usual objective in MAB problems (see Section 2.1).

This problem is a special case of the stochastic MAB with Bernoulli distributions. Contrarily to many previous work done in the CR field and for Opportunistic Spectrum Access [JEMP10, JMP12], the reward $r(t)$ does *not* come from a sensing phase before sending the t-th message, as it would do for any "listen-before-talk" model. In our model, rewards rather come from receiving or not an acknowledgement from the gateway, between the t-th and $t+1$-th messages.

[2] In Chapter 6 we simply consider the case of $p = 1$ and $M = D \leq K$ dynamic devices, referred to as *players*.

A reward of one indicates that the acknowledgement was received on time, and a reward of zero indicates the opposite.

The problem parameters are $S_1, \dots, S_K$ which represent the (stationary) occupancy of the channels by the static devices, and they are unknown to each dynamic device, so to maximize their cumulated rewards, they must learn the distributions of the channels, in order to be able to progressively improve their respective successful communication rate. The goal is thus to propose a simple sequential algorithm to be applied identically and independently by each dynamic device, in a fully distributed setting (each device runs its own algorithm, from its observations), in order to minimize collisions and maximize the fraction of successful transmissions of all the dynamic devices. This requires to tackle the so-called *exploration-exploitation dilemma*: a device has to try all channels a sufficient number of times to get a robust estimate of their qualities, while not selecting the worst channels too many times. Before presenting the MAB algorithms used in the experiments, we present some simple baseline (reference) policies.

5.2.2 Three reference policies

We present three different policies that can be used to assess the efficiency of the learning algorithms presented later on. The first one is naive but can be used in practice, while the two others are very efficient but require full knowledge on the system (*i.e.*, an oracle) and are thus unpractical. They are however useful for our numerical simulations, and are used as references, to show that our MAB-based approaches quickly learn to perform almost optimally. Their short names are used in the legend on Figures 5.3, 5.5), and are given in "quotes" in the corresponding paragraphs.

1st - Naive policy: Random Channel Selection ("Random")

We derive here the probability of having a successful transmission, for a dynamic device, in the case where all the dynamic devices make a purely random channel selection (*i.e.*, uniform on $i \in [K]$). This reflects a naive policy that could be implemented by all the dynamic devices, and it provides a reference scenario to compare against. Note that even nowadays this is still the solution implemented for real IoT devices deployed in new LoRaWAN networks.

In this case, for one dynamic device, a successful transmission happens if it is the only device to choose channel k, at that time slot. the S_k static devices in each channel k are assumed to be independent, and static and dynamic devices are assumed to *not* transmit at each time t with a fixed probability $1 - p$, so probability of successful transmission is computed as follows

$$\mathbb{P}(\text{success}|\text{sent}) = \sum_{k=1}^{K} \underbrace{\mathbb{P}(\text{success}|\text{sent in channel } k)}_{\text{No one else sent in channel } k} \underbrace{\mathbb{P}(\text{sent in channel } k)}_{=1/K, \text{ by uniform choice}} \tag{5.1}$$

All dynamic devices follow the same policy in this case, so the probability of transmitting at that time in channel k for any dynamic device is p/K, and there are $D-1$ other dynamic devices. As they are independent, the probability that no other dynamic device sent in i is $q = \mathbb{P}(\bigcap_{k=1}^{D-1} \text{device } k \text{ did not sent in } i) = \prod_{k=1}^{D-1} \mathbb{P}(\text{device } k \text{ did not sent in } i)$. And $\mathbb{P}(\text{device } k \text{ sent in } i) = p \times 1/K$, by uniform choice on channels and the Bernoulli emission hypothesis. So $q = \prod_{k=1}^{D-1}(1-p/K) = (1-p/K)^{D-1}$. Thus we can conclude,

$$\begin{aligned}\mathbb{P}(\text{success}|\text{sent}) &= \sum_{k=1}^{K} \underbrace{(1-p/K)^{D-1}}_{\text{No other dynamic device}} \times \underbrace{(1-p)^{S_k}}_{\text{No static device}} \times \frac{1}{K} \\ &= \frac{1}{K}\left(1-\frac{p}{K}\right)^{D-1} \sum_{k=1}^{K} (1-p)^{S_k}.\end{aligned} \tag{5.2}$$

This expression (5.2) is constant (in time), and easy to compute numerically, but comparing the successful transmission rate of any policy against this naive policy is important, as any efficient learning algorithm should outperform it (maybe after a long enough initial learning period).

2nd - (Unachievable) Optimal oracle policy (“Optimal”)

We investigate here the optimal policy that can be achieved if the dynamic devices have a perfect knowledge of the repartition of static devices $(S_k)_k$, and a fully centralized decision making[3] is possible. We want to find the stationary repartition of devices into channels that maximizes the probability of having a successful transmission.

If the oracle chooses a fixed configuration of dynamic devices, it means that for each dynamic device the oracle affects it to a unique channel for all time steps. Then there is a number D_k of devices affected to channel k being fixed in time (*i.e.*, stationary), and thus this probability is computed as before:

$$\begin{aligned}\mathbb{P}(\text{success}|\text{sent}) &= \sum_{k=1}^{K} \mathbb{P}(\text{success}|\text{sent in channel } k)\, \mathbb{P}(\text{sent in channel } k) \\ &= \sum_{k=1}^{K} \underbrace{(1-p)^{D_k-1}}_{D_k-1 \text{ others}} \times \underbrace{(1-p)^{S_k}}_{\text{No static device}} \times \underbrace{D_k/D}_{\text{Sent in channel } k}.\end{aligned} \tag{5.3}$$

[3] This optimal policy needs an *oracle* seeing the entire system, and affecting all the dynamic devices, once and for all, for a fixed stationary scenario, in order to avoid any signaling overhead. This is not possible in IoT contexts as several completely independent networks may operate in a single place.

Consequently, an optimal allocation vector $(D_1, \dots, D_K)$ is a solution of the following real-valued constraint optimization problem:

$$\underset{D_1,\dots,D_K}{\arg\max} \sum_{k=1}^{K} D_k (1-p)^{S_k+D_k-1}, \tag{5.4a}$$

$$\text{such that} \sum_{k=1}^{K} D_k = D, \tag{5.4b}$$

$$D_k \geq 0 \qquad \forall k \in [\![1; K]\!]. \tag{5.4c}$$

Proposition 5.1. *The* Lagrange multipliers *method [BV04] can be used to solve the constraint real-valued maximization problem introduced in equation (5.4). It gives a closed form expression for the (unique) optimal solution $D_k^*(\lambda)$, depending on the system parameters, and the unknown Lagrange multiplier $\lambda \in \mathbb{R}$.*

$$D_k^*(\lambda) = \left(\frac{1}{\ln(1-p)} \left[\mathcal{W}\left(\frac{\lambda \mathrm{e}}{(1-p)^{S_k-1}} \right) - 1 \right] \right)^+. \tag{5.5}$$

Proof.

- In a realistic scenario, we can assume that $D_k \leq \frac{-2}{\ln(1-p)} \approx \frac{2}{p}, \quad \forall k \in [\![1; K]\!]$. For such values for D_k, the objective function $f : (D_1, \dots, D_K) \mapsto \sum_{k=1}^{K} D_k (1-p)^{S_k+D_k-1}$ is concave as the sum of concave functions [4].
- The Lagrange multipliers method can be applied to the optimization problem (5.4a), with a concave objective function f, linear equality constraints (5.4b) and linear inequality constraints (5.4c). The strong duality condition is satisfied in this case [BV04], so finding the saddle points will be enough to find the maximizers.

More details are given in Section 5.6.1 in the Appendix of this Chapter. □

Where in the equation (5.5), $(a)^+ \doteq \max(a, 0)$, and $\mathcal{W}$ denotes the $\mathcal{W}$-Lambert function which is the reciprocal bijection of $x \mapsto xe^x$ on $\mathbb{R}^+ = [0, +\infty)$ (which can be computed numerically in an efficient manner, [CGH⁺96]). Moreover, condition (5.4b) implies that the Lagrange multiplier λ is the solution of the constraint $\sum_{k=1}^{K} D_k^*(\lambda) = D$. This single constraint can be solved numerically, with simple one-dimensional root finding algorithms. Solving the optimization problem provides the optimal real number value for D_k^*, which has to be rounded[5] to find the optimal number of devices for channel k: $\widehat{D_k} = \lfloor D_k^* \rfloor$ for $1 \leq k < K$, and $\widehat{D_K} = D - \sum_{k=1}^{K-1} \widehat{D_k}$.

[4] It is worth noting that f is neither concave nor quasi-concave on $[0, \infty)^K$ [Lue68, Yaa77].

[5] Any rounding choice will give about the same repartition, up-to a difference of only one device by channel, and so we chose to round from below for the first channels.

3rd - A greedy approach of the oracle strategy ("Good sub-optimal")

We propose a *sequential* approximation of the optimal policy: the third solution is a sub-optimal naive policy, simple to set up, but also unpractical as it also needs an oracle. End-devices are iteratively inserted in the channels with the lowest load (*i.e.*, the index k minimizing $S_k + D_k(\tau)$ at global time step τ). Once the number of devices in each channel is computed, the probability of sending successfully a message is also given by equation (5.3). This is the policy that would be used by dynamic devices if they were inserted one after the other, and if they had a perfect knowledge of the channel loads.

5.2.3 Sequential policies based on bandit algorithms

While the stochastic MAB model has been used to describe some aspects of Cognitive Radio systems, it is in principle not suitable for our IoT model, due to the non-stationarity of the channels occupancy, caused partly by the learning policy used by dynamic devices, but mainly by their random activation processes. In our model, every dynamic device implements its own learning algorithm, *independently*. For one device, the time t refers to the number of time it accessed the network (following its Bernoulli transmission process, *i.e.*, its duty cycle), *not* the total number of time slots from the beginning, as rewards are only obtained after a transmission, and IoT devices only transmit sporadically, due to low transmission duty cycles.

Using a bandit algorithm for IoT devices: Our IoT application is challenging in that there are *multiple* players (the dynamic devices) interacting with the *same* arms (the channels), without any centralized communication (they do not even know the total number of dynamic devices). We propose algorithms in which each dynamic device ignores all the other one and implements a learning algorithm to play a bandit game. In each time slot, if it has to communicate (which happens with probability p), then it chooses a channel and it receives a reward 1 if the transmission is successful, 0 otherwise. Each device aims at maximizing the sum of the rewards collected during its communication instants, which shall indeed maximize the fraction of successful transmissions. Besides the modified time scale (rewards are no longer collected at every time step), this looks like a usual bandit problem. However, it cannot be modeled as a stochastic MAB, as the rewards are (unfortunately) *not i.i.d.*: they not only depend on the (stationary, *i.i.d.*) behavior of the static devices, but also on the behavior of other dynamic devices, that is not stationary (because of learning and random activation of each device). Despite this, we show in the next subsection that running a stochastic bandit algorithm for each device based on its own rewards is surprisingly successful.

Considered algorithms. The two algorithms we consider are UCB ("UCB" in the figures) and Thompson sampling (TS) ("Thompson-sampling"), and they are presented in Section 2.4. The UCB algorithm uses $\alpha = 1/2$ in our experiments. The TS algorithm is known to be empirically efficient, and for these reasons it has been used successfully in various applications, including problems from Cognitive Radio [TCLM16, MTC+16], and also in previous work on decentralized IoT-like networks [DMP16].

***Adversarial* bandit algorithms?** Instead of using MAB algorithms assuming a stochastic hypothesis on the system, we could try to use MAB algorithms designed to tackle a more general problem, that makes no hypothesis on the interfering traffic. The *adversarial MAB* algorithms is a broader family, of which a well-known and efficient example is the Exp3 algorithm [ACBF02, BCB12]. Empirically, the Exp3 algorithm turned out to perform significantly worse than both UCB and TS in the same experiments, therefore we did not report results here.

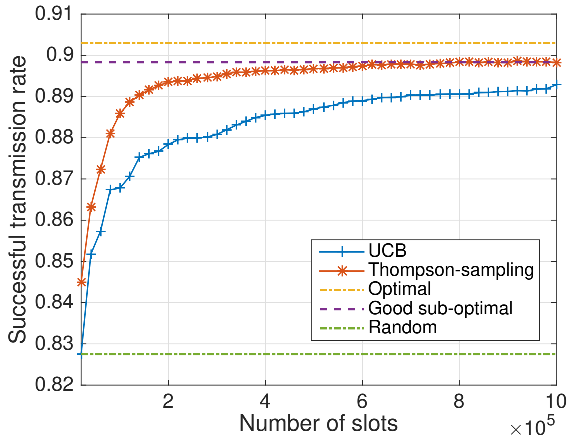
5.2.4 Numerical results

We suppose a network with $S + D = 2000$ end-devices, and one IoT gateway. Each device sends packets following a Bernoulli process, of probability $p = 10^{-3}$ (*e.g.*, this is realistic: one packet sent about every 20 minutes, for time slots of 1s). The RF band is divided in $K = 10$ channels. Each static device only uses one channel, and their uneven repartition in the 10 channels is chosen as $(S_1, \cdots, S_K) = S \times (0.3,\ 0.2,\ 0.1,\ 0.1,\ 0.05,\ 0.05,\ 0.02,\ 0.08,\ 0.01,\ 0.09)$, to keep the same proportions when S decreases. The dynamic devices have access to all the channels, and use learning algorithms. We simulate the network during 10^6 discrete time slots, during which each device transmits on average 1000 packets (*i.e.*, the learning time is about 1000 steps, for each algorithm). We tried similar experiments with other values for K and this repartition vector, and results were similar for non-homogeneous repartitions.

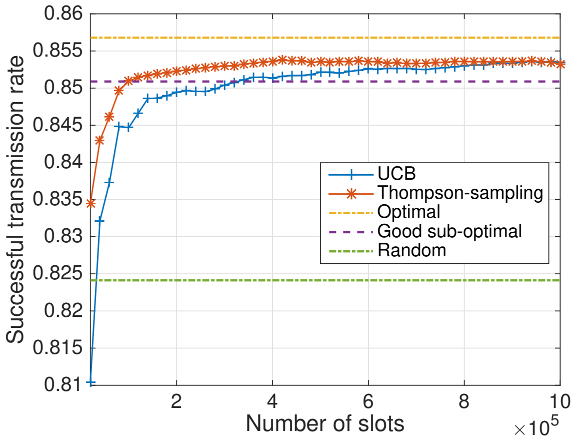
Clearly, the problem is less interesting for a homogeneous repartition, as all channels appear the same for dynamic devices, and so even with D small in comparison to S, the system behaves like in Figure 5.3d, where the performance of the five approaches are very close.

First results. Figure 5.3 presents the successful transmission rate as a function of time. The two MAB algorithms, UCB and Thompson Sampling (TS), are compared against the naive random policy (which are outperformed easily by the MAB algorithms), and the two (optimal and greedy) oracle policies (which outperform slightly the MAB algorithms). The results are displayed when 10%, 30%, 50% and 100% of the traffic is generated by dynamic devices.

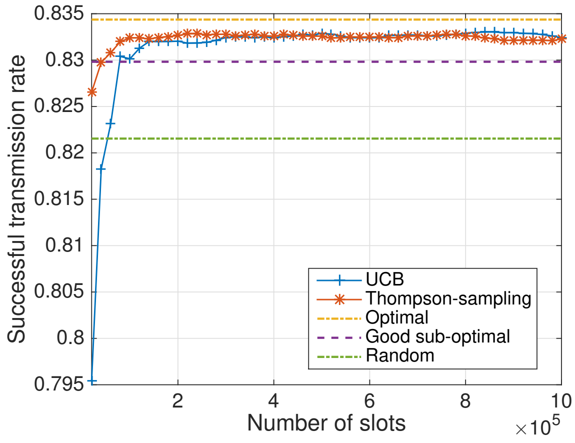
We can see in Figure 5.3 that the TS algorithm (in red) outperforms the UCB algorithm (in blue), when the number of end-devices is below 50%. When the number of end-devices is higher, both algorithms have almost the same performance, and perform well after a small number of transmissions (*i.e.*, they show quick convergence). Moreover, we can see in Figures 5.3a, 5.3b, and 5.3c that both have better success rate than the random policy and the probability of successful transmission is between the optimal oracle and suboptimal oracle policies. For instance, for 10% of dynamic devices, after about 1000 transmissions, using UCB



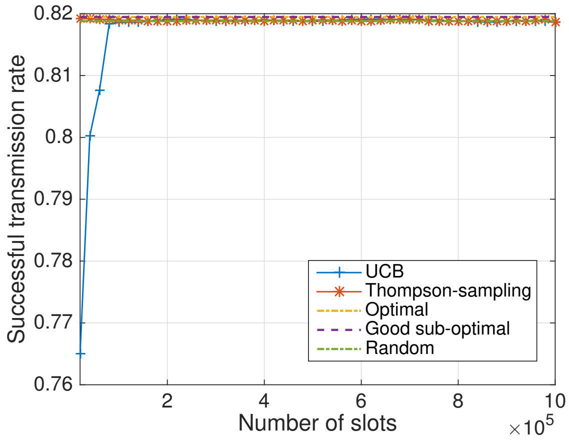
(a) 10% of dynamic devices



(b) 30% of dynamic devices



(c) 50% of dynamic devices



(d) 100% of dynamic devices

Figure 5.3 – Performance of two MAB algorithms (UCB and Thompson Sampling), compared to extreme reference policies without learning or oracle knowledge, when the proportion of dynamic end-devices in the network increases, from 10% to 100%.

over the naive uniform policy improved the successful transmission rate from 83% to 88%, and using Thompson Sampling improved it to 89%. Increasing the number of end-devices decreases the gap between the optimal and random policies: as expected intuitively, the more dynamic devices, the less useful are learning algorithms, and basically for networks with only dynamic devices, the random policy is as efficient as the optimal one, as seen in Figures 5.3d and on the right end side of Figure 5.4.

Successful transmission rate as a function of the number of dynamic devices. To better assess the evolution of the optimal policy compared to the random one, we have displayed on Figure 5.4 the evolution of the gain, in terms of successful transmission rate, provided by the optimal oracle and the two learning policies, after 10^6 time slots, *i.e.*, about 1000 transmissions for each IoT device. We can see that when the proportion of end-devices is low (*e.g.*, 1% of devices are dynamic), the optimal policy provides an improvement of 16% compared to

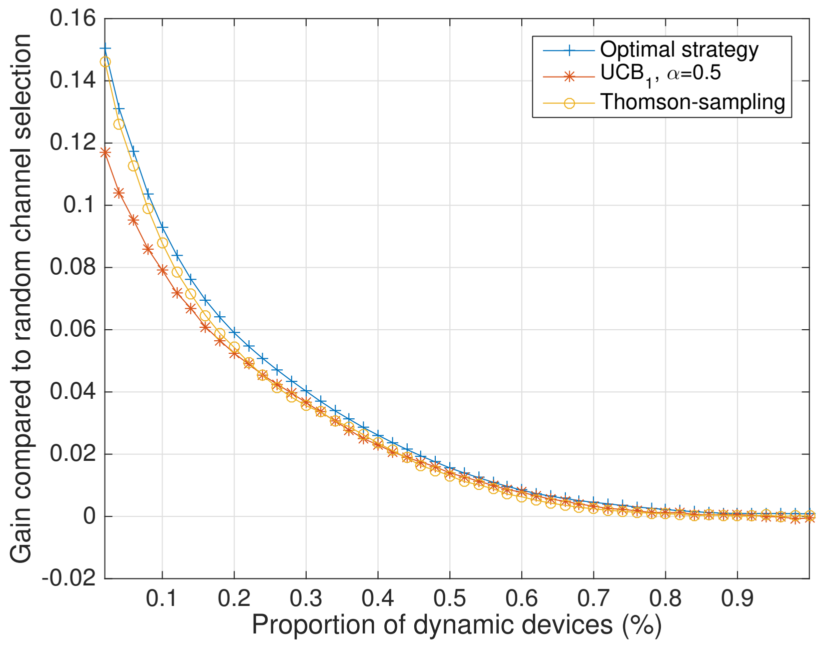
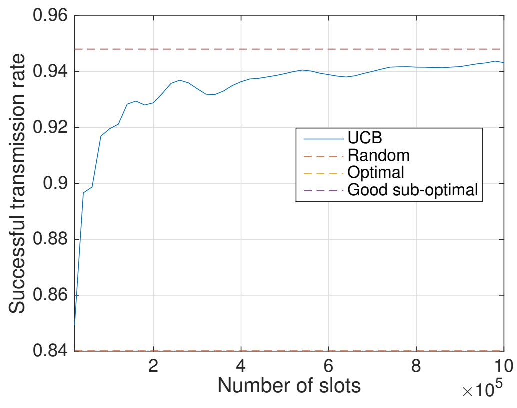


Figure 5.4 – Learning with UCB and Thomson Sampling, with many dynamic devices. The learning gain, for each device, decreases with the proportion of dynamic devices in the network. Note that the values (16% etc) depend on the repartition of static devices into the K channels (*i.e.*, $S_1, \dots, S_K$) but the general profile of this plot does *not* depend much on these parameters.

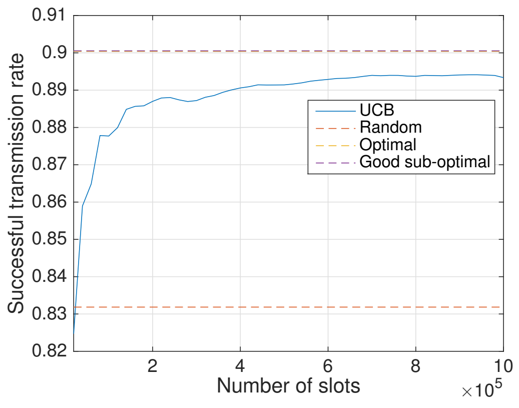
random channel selection. The TS algorithm always provides near-optimal performance, but the UCB algorithm has a lowest rate of convergence and performs consequently worse after 1000 transmissions, for instance it only provides a gain of 12% for the same proportion of dynamic devices (1%), for the considered values of $S_1, \dots, S_K$.

Figure 5.4 also shows that learning keeps near-optimal performance even when the proportion of devices becomes large. Note that when this proportion increases, the assumptions of a stochastic MAB model are clearly violated, and there is no mathematical justification for the efficiency of TS and UCB algorithms. Hence, it is surprising to have near optimal performance with stochastic MAB algorithms applied to partly or fully dynamic scenarios.

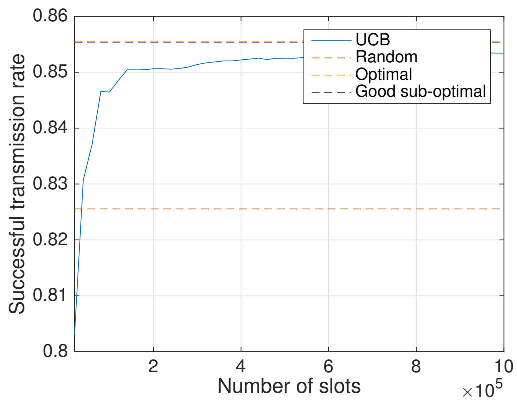
A safety check. We include another simulation in Figure 5.5, with a uniform repartition of static devices (*i.e.*, $\forall k, S_k = S/K$), to check that learning approach (here, only UCB) also gives interesting gain of performance, and achieve a close-to optimal successful transmission rate. Except for the limit case of 100% of dynamic devices, which corresponds to Figures 5.3d and 5.5d, where the uniform access performs as well as the optimal oracle solution, the MAB-based approach almost instantly outperforms the baseline.



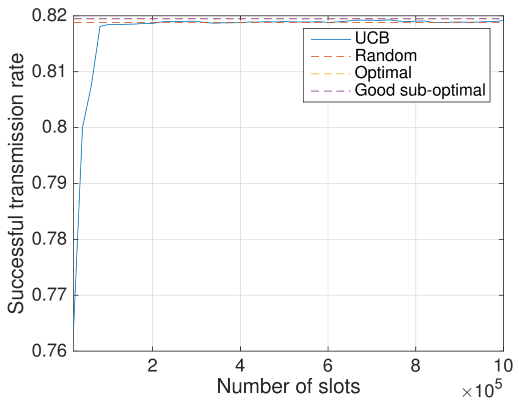
(a) 10% of intelligent devices



(b) 30% of intelligent devices



(c) 50% of intelligent devices



(d) 100% of intelligent devices

Figure 5.5 – Performance of the UCB bandit algorithm for the special case of uniform repartition of the static devices, when the proportion of intelligent devices in the network increases, from 10% to 100%.

Reproducibility. The simulation code used for the experiments in Section 5.2.4 is for MATLAB or GNU Octave, and it was written in collaboration with Rémi Bonnefoi, in May 2017. Instructions to reproduce our experiments are given, and the code is open-sourced under the MIT License, at `Bitbucket.org/scee_ietr/rl_slotted_iot_networks`.

5.3 Test-bed implementation of our model for real-world validation

In this section, we present a demonstration showcased at the International Conference on Telecommunication (ICT) in June 2018 [BBM18, BBM19], implementing a proof-of-concept (PoC) of the model introduced in Section 5.2. As far as we know, this is the first demonstration of running learning algorithms on the end-device side for Internet of Things networks, in real radio conditions, as highlighted it in our paper [MB19].

This PoC implements an IoT network the following way: one gateway, one or several intelligent (*i.e.*, learning) devices, embedding the proposed solution, and a traffic generator that emulates radio interference from many other devices. Intelligent devices communicate with the gateway with a wireless ALOHA-based protocol with acknowledgements, which does not require any specific overhead for the learning. Similarly to the previous section, network access is modelled as a discrete sequential decision making problem, and using the framework and algorithms from MAB learning, we show that intelligent devices can improve their access rate to the network, by using low complexity and decentralized algorithms, such as UCB and Thompson Sampling. This solution could be added in a straightforward and costless manner in most IoT networks, such as LoRaWAN networks, just by adding this feature at the higher level of the MAC layer, in all or only some of the devices, without any modification on the network side, and no signalling overhead for the devices.

Related works. This work is new for the IoT context, but previous works have similarly implemented proof-of-concepts to show that Reinforcement Learning (RL) algorithms can be used within real-world wireless communications. Starting from 2010, the works of W. Jouini and C. Moy [JEMP09, JEMP10, JMP12] were among the first ones to propose to use RL for Cognitive Radio, especially MAB and the UCB algorithm, and proof-of-concepts were developed with C. Robert from 2013 [RMZ14, Moy14]. Between 2015 and 2017, C. Moy, N. Modi and S. Darak (of our team SCEE) continued to work on this direction [DNMP16, DMP16, DMNM16, KDY$^+$16]. Since 2017, S. Darak and his team at IIIT Delhi (India) have been very active in the research on CR using MAB, and some of their recent works are illustrated with real-world demo using USRP and the MATLAB/Simulink system [KYDH18, SKHD18, H. 18].

5.3.1 Context of this demonstration

We describe the way we implemented a demo where we evaluate MAB algorithms, used in combination with a pure ALOHA-based protocol, such as the ones employed in LPWAN. This demonstration is the first implementation which aims at assessing the potential gain of MAB learning algorithms in IoT scenarios. We use a TestBed designed in 2017 by the SCEE team at the Rennes campus of CentraleSupélec [Bod17, Appendix 3], containing different USRP boards [Ett], controlled by a single laptop running the GNU Radio software [GNUb], and where the intelligence of each device corresponds to a learning algorithm, implemented as a GNU Radio block [GNUa] and written in Python or C++.

In our demo, we consider a simple wireless network, that reproduces the model of Section 5.2, consisting of one gateway (*i.e.*, radio access point), and a certain interfering background traffic, assumed to be stationary (*i.i.d.*), which is generated by end-devices communicating in other networks. Some dynamic intelligent devices (end-user or autonomous devices) try to communicate with the gateway, with a low-overhead protocol. This communication can be done in different channels which are also shared by devices using other networks. Once the gateway receives a packet transmitted by a dynamic device in one channel (*i.e.*, if no collision occurred), it transmits back to it an acknowledgement in the same channel, after a fixed-time delay, as it is done in the LoRaWAN standard. This *Ack* allows the device to learn about the channel quality (*i.e.*, mean availability) and thus, to use learning algorithms for the purpose of best channel selection.

We can generate scenarios with different parameters (number of channels, interfering traffic load on each channel, etc) in order to evaluate the performance of learning in various settings. Moreover, we compare the performance of learning strategies with that of the random uniform access to channels, which is the current state-of-the-art of commercial LPWAN solutions [RKS17]. This allows to check that in case of uniform traffic, when there is nothing to learn, the intelligent devices at least do not reduce their successful communication rate in comparison to the naive devices. This also shows that in case of non-uniform stationary traffic, MAB learning algorithms indeed help to increase the global efficiency of the network by improving the success rate of the intelligent devices. The benefits are twice and of primary importance for IoT networks: the proposed approach can mitigate RF collisions, and enhance intelligent device battery lifetime if they do retransmissions.

5.3.2 Physical model and user interface of our GNU Radio implementation

In this section, we present our implementation of MAB algorithms in the model of IoT networks presented in Section 5.2.1. We first describe the simplified physical layer of this demo, then we present our GNU Radio implementation.

Physical layer and protocol. We implement a simple PHY/MAC layers solution, in order to demonstrate the possibility of improvement of the performance of IoT communications in unlicensed bands. We could have used any physical layer and any ALOHA-based protocol. We choose to implement our own physical layer and protocol, for both clarity and conciseness, and because developing a complete IoT network protocol stack is no more my research work and would have fall outside of the scope of this thesis.

Regarding the physical layer, we consider a QPSK constellation (*Quadrature Phase-Shift Keying*). Moreover, we use simplified packets composed of two parts. The first part is the *preamble* which is used for the purpose of synchronization (phase correction). Then, we have the *index* of the user, which is a sequence of QPSK symbols. For example, this index can be a simple QPSK symbol ($\pm 1 \pm 1j$), or a sequence of QPSK symols if the PoC has to consider more users. With ℓ QPSK symbols, we can indeed fit at most $4^\ell/2 = 2^{2\ell-1}$ devices, and not 4^ℓ due to the use of a conjugate index to send back acknowledgements from the gateway. Once the gateway receives an uplink packet, it detects this index and transmits an acknowledgement which has the same frame structure, but where the index is the conjugate of the index of the uplink packet ($z \mapsto \overline{z}$, *e.g.*, $1 + j \mapsto 1 - j$). Thanks to this index, we can have several devices communicating with the same gateway. In turn, the end-device that receives the acknowledgement demodulates it, and checks if the index is the conjugate of its own index. In this case, the *Ack* was sent for him, and it knows that its packet has been received and decoded correctly by the gateway.

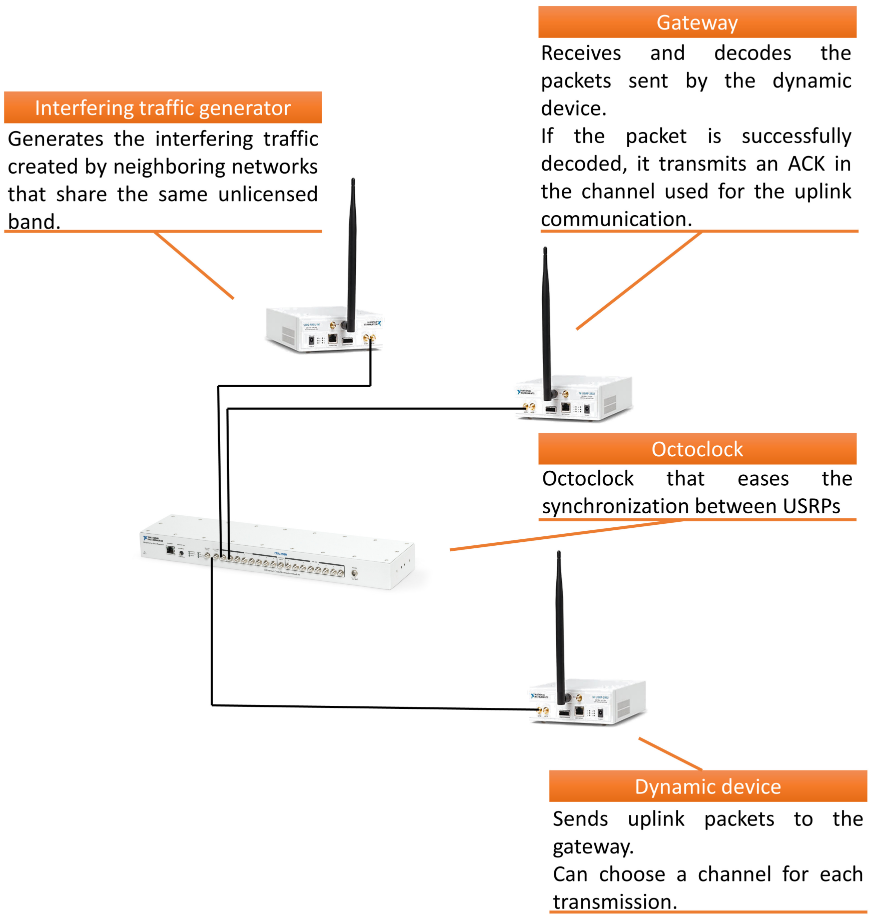

Gateway
Receives and decodes the packets sent by the dynamic device.
If the packet is successfully decoded, it transmits an ACK in the channel used for the uplink communication.
Interfering traffic generator
Generates the interfering traffic created by neighboring networks that share the same unlicensed band.
Octoclock
Octoclock that eases the synchronization between USRPs
Dynamic device
Sends uplink packets to the gateway.
Can choose a channel for each transmission.

Figure 5.6 – Schematic of our implementation that presents the role of each USRP platform.

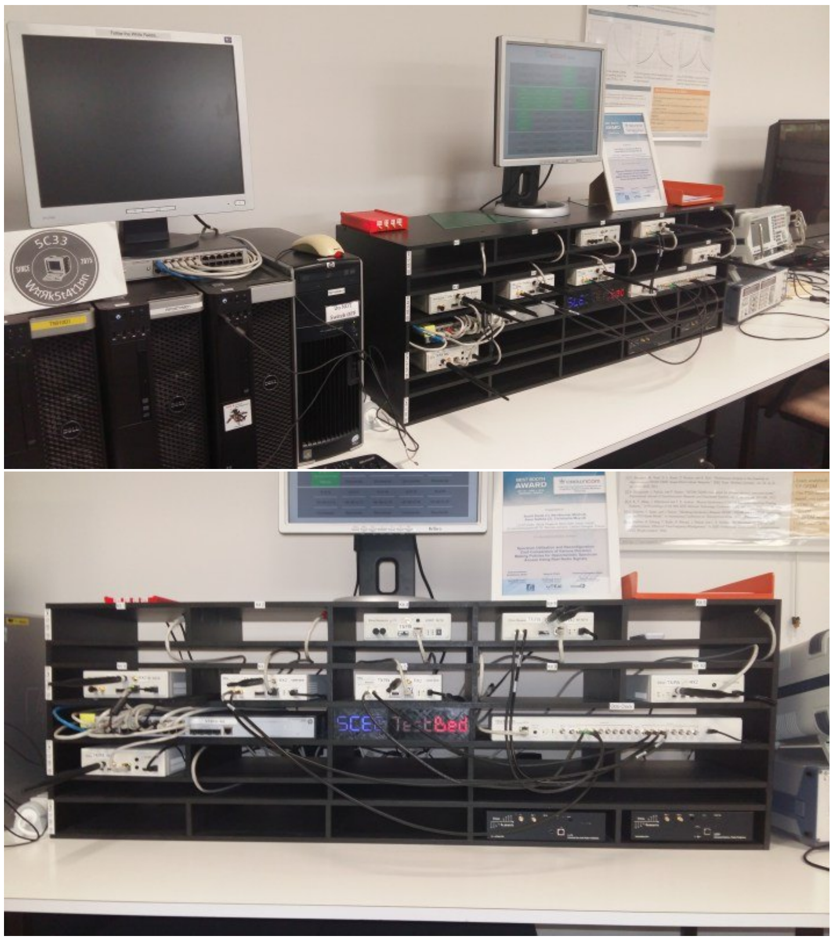


Figure 5.7 – Two pictures showing the SCEE testbed [Bod17, Appendix 3], taken in 2018.

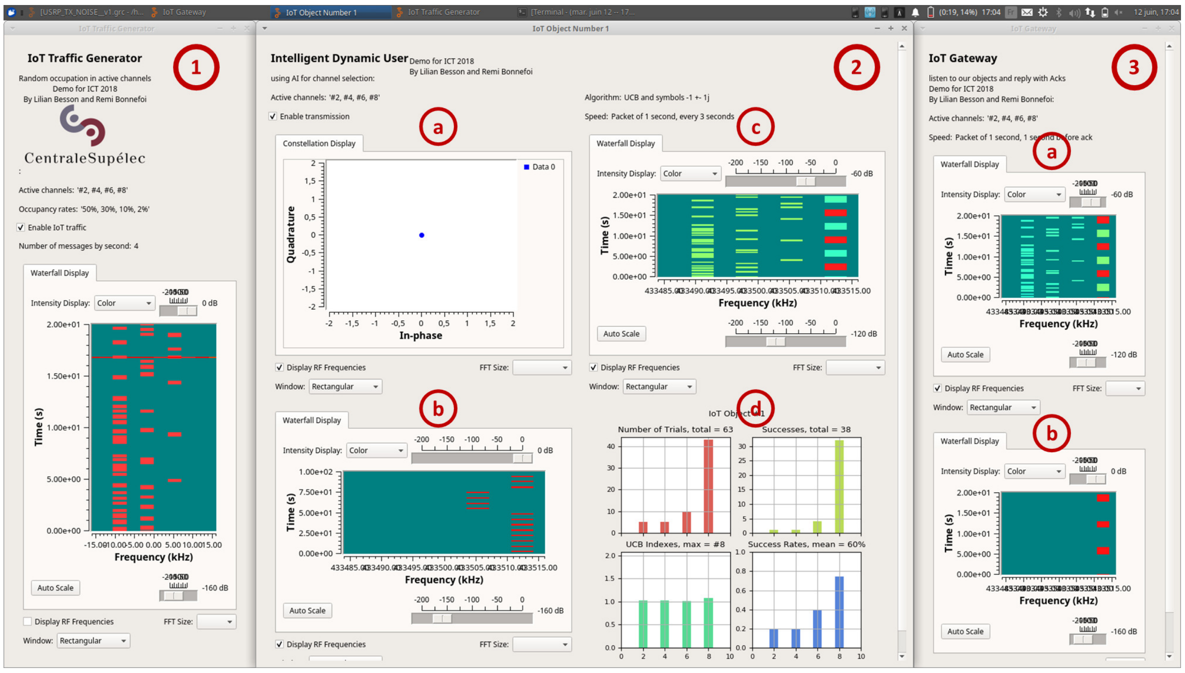
Equipment. We use USRP N210 boards [Ett], from Ettus Research (National Instrument), with version 4 of their FPGA system and version 5.1 of the RBX system. As illustrated in Figure 5.6, our implementation is composed of at least 3 USRP: the gateway, a traffic generator which emulates the interfering traffic (made by surrounding static devices), and at least one dynamic device. Each dynamic device has its own USRP and its own learning algorithm.

The boards have their own power supply, and are all connected to a local Ethernet switch, itself connected to a single laptop, running GNU/Linux and Ubuntu. The pictures in Figure 5.7 show the testbed used for these experiments. To ease the synchronization in both time and frequency between the boards representing the dynamic devices and the gateway, we use an Octoclock [Oct], also a product of Ettus Research, and coaxial cables connecting every platform to the Octoclock for time (PPS) and frequency synchronization, but this is not mandatory.

Details about our implementation. We used the GNU Radio Companion software (GRC, version 3.7 in 2017), and a laptop runs a GRC design to configure and control each USRP platform. As such, one laptop can run in parallel the control program of any number of boards[6]. The GNU Radio software provides the framework and tools to build and run software radio or just general signal-processing applications. GNU Radio applications are flow-graphs: a series of signal processing blocks connected together to describe a data flow. For maximum efficiency, we wrote all of our blocks in C++. These flow-graphs can be written in either C++ or the Python programming language. The GNU Radio infrastructure is written entirely in C++, and many of the user tools are written in Python. GNU Radio Companion is a graphical user interface (UI) used to develop GNU Radio applications: GRC is effectively a Python code-generation tool. When a flow-graph is compiled in GRC, a Python code is produced, which can be executed to connect to the USRP, create the desired GUI windows and widgets, and create and connect the blocks in the flow-graph.

Illustrations of the flow-graph for the three components of the presented demonstration are included in Appendix 5.6.2, in Figures 5.15, 5.16 and 5.17.

User Interface. We have designed a user interface in order to visualize the results obtained with our experimental demonstration. This user interface is shown in Figure 5.8. We can see that it is made of three parts, one for each USRP, as highlighted in circled red numbers.


IoT Traffic Generator
Random occupation in active channels
Demo for ICT 2018
By Lilian Besson and Remi Bonnefoi
Active channels: '#2, #4, #6, #8'
Occupancy rates: '50%, 30%, 10%, 2%'
✓ Enable IoT traffic
Number of messages by second: 4

Intelligent Dynamic User using AI for channel selection:
Demo for ICT 2018
By Lilian Besson and Remi Bonnefoi
Active channels: '#2, #4, #6, #8'
✓ Enable transmission
Algorithm: UCB and symbols -1 +- 1j
Speed: Packet of 1 second, every 3 seconds

IoT Gateway
listen to our objects and reply with Acks
Demo for ICT 2018
By Lilian Besson and Remi Bonnefoi:
Active channels: '#2, #4, #6, #8'
Speed: Packet of 1 second, 1 second before ack

Figure 5.8 – User interface of our demonstration.

[6] Even if in practice, maximum efficiency is kept as long as there is not more than one GRC design by CPU core.

(1) The first part is the interface of the IoT traffic generator, where we see the traffic generated by this USRP, presented in a waterfall view in the time vs frequency domain. Messages of the random traffic, generated by surrounding static devices, are shorter in time by purpose (they could be coming from other IoT standard), in order to distinguish them from an intelligent device traffic on the "waterfall" visualizations of the traffic.

(2) The second part is the interface of the intelligent device which is made of four parts. At the top left, we observe the constellation of the transmitted packet *(a)*. At the bottom left, we have a time/frequency view of the lasts packets transmitted by the device *(b)*. We can see, in this view that the device transmitted its last 9 packets in channels $\#3$ and $\#4$. Then, at the top right of this interface *(c)*, we can see the traffic observed by this device, where we have the interfering traffic (green), the uplink packets transmitted by this device (red) and the acknowledgements sent by the gateway (blue). Colors in the "waterfall" represent the RF power level received at the device antenna. Hot colors are for closer elements, as for instance the device Tx antenna (reception) is close to the device Rx antenna (transmission), and consequently for the device waterfall these signals are colored in red (see graph *(c)* in part (2)). Finally, at the bottom right *(d)*, we have four histograms showing the performance indicators of the chosen MAB algorithm (number of transmissions, number of successful transmissions, UCB indexes and success rates, in each channel).

(3) The last part is the interface of the gateway, where we can see the traffic observed by the gateway *(a)* and the channels in which the last acknowledgements have been sent *(b)*. The observed traffic is coherent with *(c)* of part (2), as the elements are very close the one from the others on the testbed. Colors may change, as they depend on the exact distance between the different transmitters and receivers.

5.3.3 Experimental results

We compare in this PoC the two algorithms described in Section 2.4 (UCB and Thompson sampling) against a uniform access algorithm, that uniformly selects its channel at random. Note that we could run more algorithms, but with no real added-valued in terms of validation of the proposed learning-based approach, which is our goal. For one dynamic device, three algorithms are compared by their mean successful communication rates, on a horizon of $T = 2000$ communication slots, and were using three algorithms: uniform random access (in cyan), Thompson Sampling ("TS", in green) and UCB (in red).

In Figure 5.9 below, we show the results averaged on 10 repetitions using the same conditions. Each experiment has been done on a duration of about half a day, due to the IoT sporadic transmission mode that we want to respect (like in our model of Section 5.2). However, we make devices generate one message every 5 seconds, in order to artificially speed up the process and with no loss of generality (as we are using real hardware). Learning can be

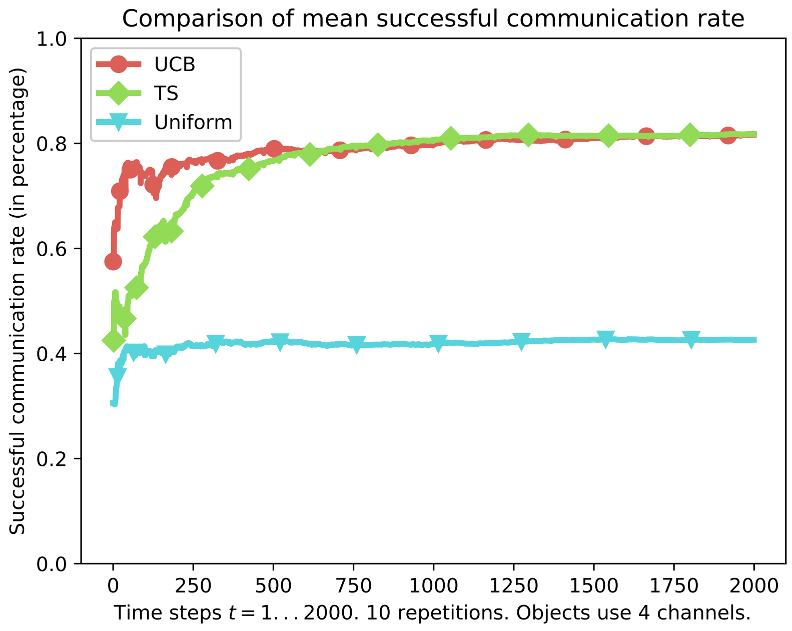


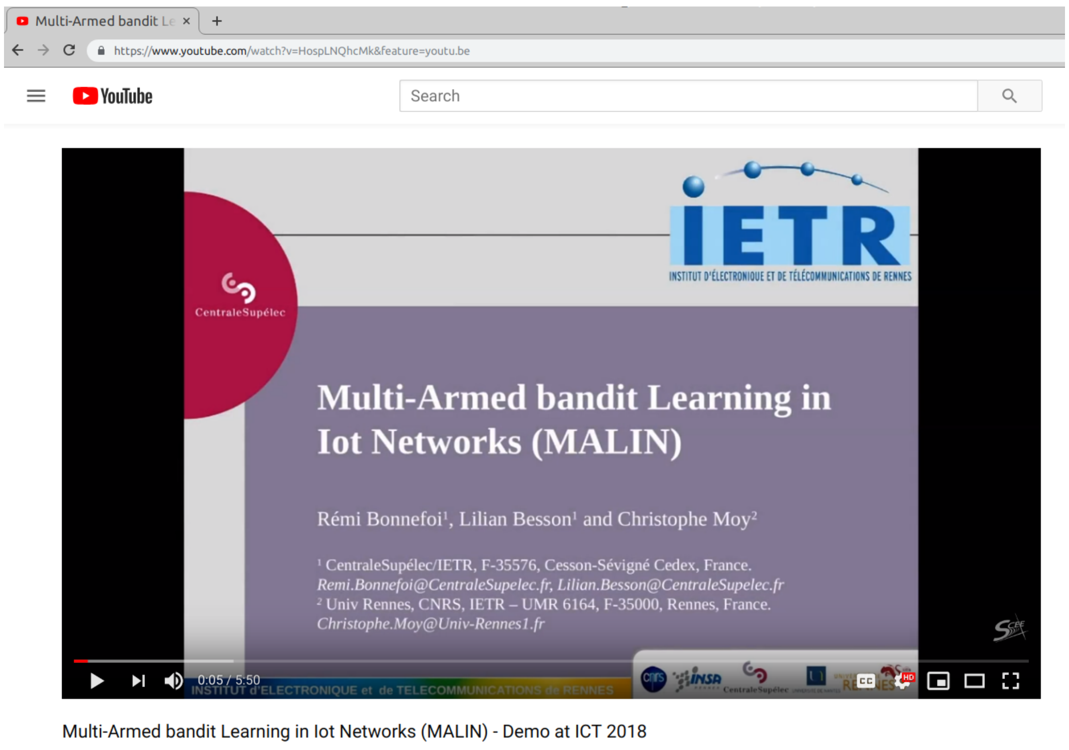
Figure 5.9 – Less than 400 communication slots (*i.e.*, less than 100 trials in each channel) are sufficient for the two learning devices to reach a successful communication rate close to 80%, which is **twice as much** as the non-learning (uniform) device, which stays around 40% of success. Similar gains of performance were obtained in many different scenarios.

useful only when there is a large enough difference between "good" and "bad" channels, Each device was learning to access 4 different non-overlapping channels, that we chose to have occupancy rates of $(\mu_k)_k = [15\%, 10\%, 2\%, 1\%]$. Note that a maximum occupancy rate of 15% could seem not so high, but indeed it is, because for a pure ALOHA access mode, a naive dynamic device only enjoys about 40% success rate, under such occupancy of the channels (see the "uniform" plot in Figure 5.9). The occupancy rate of a channel, which denotes the mean occupancy, is implemented using the traffic generator, to emulate the presence of S_i static devices with emission probability p (that is, $\mu_i = S_i \times p$ here). When facing the same stationary background traffic, we see that the learning devices are both very quickly more efficient than the naive uniform device. We obtain an improvement in terms of successful communication rate from 40% to about 60% in only 100 communications (about 16 min), and up-to 80% in only 400 communications. In stationary environments, both the TS and UCB algorithms are very efficient and converge quickly, resulting in a very strong decrease in collisions and failed communication slots. UCB is faster to learn but eventually TS gives a (slightly) better average performance.

Similar results are obtained for overlapping channels, when dynamic devices are learning in the presence of multiple devices, all using the same learning algorithm. However, our experimental testbed can not run hundreds of intelligent devices. Empirical results confirm the simulations presented in Section 5.2 (see Figure 5.4). Such results are very encouraging, and illustrate well the various strong possibilities of MAB learning applied to IoT networks.

Availability of data and materials. The source code of our demonstration is fully available online, open-sourced under GPLv3 license, at `bitbucket.org/scee_ietr/malin-multi-arm-bandit-learning-for-iot-networks-with-grc/`. It contains both the GNU Radio Companion flowcharts and blocks, with ready-to-use `Makefiles` to easily compile, install and launch the demonstration. The demonstration only requires a laptop and open-source free softwares, as the laptop should run a GNU/Linux distribution (like Ubuntu or Debian), in addition to USRP platforms from Ettus Research.

Video. As depicted in Figure 5.10 below, we realized a 6-minute **video** to sum-up our demonstration and advertise our work online, and it is available on the YouTube hosting platform, at `youtu.be/HospLNQhcMk`. The video shows examples of 3 dynamic devices learning simultaneously, confirming the results of Figure 5.9 for overlapping channels. It also shows the connections between the USRP boards, the Octoclock, the master laptop etc, completing the presentation of the SCEE testbed already shown in Figure 5.7.


Multi-Armed bandit Le
https://www.youtube.com/watch?v=HospLNQhcMk&feature=youtu.be
YouTube
Search
IETR
INSTITUT D'ÉLECTRONIQUE ET DE TÉLÉCOMMUNICATIONS DE RENNES
CentraleSupélec
0:05 / 5:50

Multi-Armed bandit Learning in Iot Networks (MALIN)

Rémi Bonnefoi[1], Lilian Besson[1] and Christophe Moy[2]

[1] CentraleSupélec/IETR, F-35576, Cesson-Sévigné Cedex, France.
Remi.Bonnefoi@CentraleSupelec.fr, Lilian.Besson@CentraleSupelec.fr
[2] Univ Rennes, CNRS, IETR – UMR 6164, F-35000, Rennes, France.
Christophe.Moy@Univ-Rennes1.fr

Multi-Armed bandit Learning in Iot Networks (MALIN) - Demo at ICT 2018

Figure 5.10 – Screenshot of the **video** of our demonstration, `youtu.be/HospLNQhcMk`.

5.4 Extending the model to account for retransmissions

In this section, we extend the previous model to take into account the possibility for retransmissions of a message after a collision. This is of major importance as most protocols for real-world IoT networks can use retransmissions, it is for instance the case of the LoRaWAN standard. As before, we propose and evaluate different learning strategies based on MAB algorithms. However, the price to be paid is a shorter battery lifetime for IoT devices. So this approach would be used only for devices with strong delivery constraints (*e.g.*, for health-care applications), or that can refuel their energy over time (*e.g.*, for robotic applications).

The presented strategies allow IoT devices to improve their access to the network and their autonomy, while taking into account the impact of encountered radio collisions. For that end, several heuristics employing UCB algorithms are examined, to explore the information provided by the number of retransmissions. In this section, our results show that approaches based on UCB obtain a significant improvement in terms of successful transmission probabilities, compared to a naive approach which does not learn. Furthermore, it also reveals that a pure UCB channel access is as efficient as more sophisticated learning strategies.

We presented in Section 5.2 the impact of non-stationarity on the network performance using MAB algorithms is studied. Low-cost algorithms following two well-known approaches, such as the Upper-Confidence Bound (UCB), and the Thompson Sampling (TS) algorithms have reported encouraging results. When considering the application of MAB algorithms for slotted wireless protocols in a decentralized manner, other recent directions of research include theoretical analysis, like what we present in the next Chapter 6, or realistic empirical PoC like in [Moy14, RMZ14, DMNM16, DNMP16, KDY+16, KDY+17] or the previous Section 5.3, and finally applications to multi-hoping networks [MTC+16, TCLM16] or other kinds of networks [WCN+19, WBMB+19]. None of the mentioned works discuss in detail the impact of retransmissions on the performance of MAB learning algorithms as we do now.

The aim of this section is to assess the performance of MAB algorithms for channel selection in LPWA networks operating in unlicensed bands, while taking into account the impact of retransmissions on the network performance. For this reason, several decision making strategies are applied after a first retransmission (*i.e.*, when a collision occurs). The proposed approach employs contextual information provided by the number of retransmissions, and is again implemented independently by each device, so that no coordination among them is needed. Moreover, our UCB-based heuristics show low complexity making them suitable for being embedded in LPWA devices.

The contributions of this section are summarized as follows. Firstly, we provide a close form approximation of the radio collision probability after a first retransmission. By doing this, we highlight the need to develop a learning approach for channel selection upon collision. Secondly, different heuristics are proposed to cope with retransmissions. Lastly, we conduct simulations in order to compare the performance of the proposed heuristics with a naive uniform random approach, and a UCB strategy (*i.e.*, without any learning for the retransmissions, that is, the same channel is used for retransmission).

5.4.1 Presentation of the model with retransmissions

LPWA network. Like before in this chapter, we consider an LPWA network composed of a gateway and a large number of end-devices that regularly send short data packets, where K channels ($K > 1$) are available for the transmission of their packets. We assume that this network is constituted by two types of devices: On the one hand, we have *static* devices that operate in one channel[7] in order to communicate with the gateway. On the other hand, there are IoT devices, that possess the additional advantage of being able to select any of the K available channels to perform their transmissions.

Like in the previous model presented in Section 5.2, regardless the type of devices, each of them follows a slotted ALOHA protocol [Rob75], and has a probability $p > 0$ to transmit a packet in a time slot. We make the hypothesis that the transmission is successful if the channel is available, otherwise it fails upon radio collision. The novelty compare to the previous model is that in case of RF (uplink or downlink) collision that prevent the device to receive the *Ack* from the gateway, these devices will attempt to retransmit their packet up-to $\mathrm{MaxBackOff}$ times[8], with $\mathrm{MaxBackOff} \in \mathbb{N}^*$. It is important to note that, every retransmission is carried out after a random back-off time, uniformly distributed in $\{0, \dots, m-1\}$, where $m \in \mathbb{N}^*$ is the length of the back-off interval (note the difference between m and $\mathrm{MaxBackOff}$).

Model of IoT devices. The aforementioned contention process can be described by a Markov chain model [Nor98] similar to the one presented in [YFE12], as it is depicted in Figure 5.11. When a device has a packet to transmit, it goes from an idle state to a transmission state, while considering retransmissions due to different collision probabilities, $\{p_c, p_{c1}, \dots, p_{\mathrm{MaxBackOff}-2}\}$, at each $\mathrm{MaxBackOff}$ back-off stage. At each time slot, a transition from an idle state to a transmission state (denoted as `Trans.`) occurs if a packet transmission is required, while waiting states (denoted as `Wait`), correspond to a m back-off interval.

A device aims to select a channel with the highest probability of successful transmissions, for which it uses a reinforcement learning approach, again formulated as a MAB problem. Contrarily to what may appear at first sight, *the goal is not to minimize the number of retransmissions*, but to maximize the probability of successful transmission, considering both the first transmission of a message and the retransmissions of the same message. Indeed, the objective of each device is to *maximize its battery life by minimizing its **total** number of transmissions*. We

[7] Note that, for unlicensed bands, this definition also encompasses any device following a different standard or trying to establish communication with gateways of other networks.

[8] We denote it $\mathrm{MaxBackOff}$ instead of M like we did in our paper [BBMVM19], as M is used in next Chapter 6.

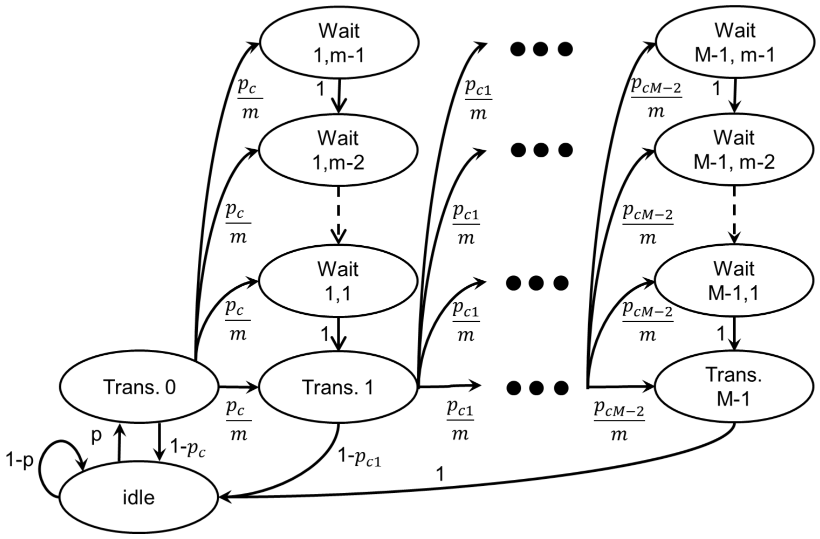


Figure 5.11 – The Markov model of the behavior of all devices paired to the considered IoT network using the ALOHA protocol.

address the problem of channel selection taking into account the described Markov model for the retransmissions of end-devices. It motivates our present work for which we consider the number of retransmissions, carried out by each device.

5.4.2 Motivations for the proposed approach

We consider IoT devices with a constraint on their QoS, imposing the successful delivery of their messages. When such an IoT device experiments a collision, it goes in a back-off state to retransmit the same packet on the same or another channel. If all devices remain in the same channel for retransmissions, it is a well-known result that it could result in a sequence of successive collisions with the same packets' devices that previously collided. Thus, it seems interesting to consider in the decision making policy the possibility for a device to retransmit in a different channel. One of our motivations to develop new MAB algorithms for our problem is this option of using a different communication channel between the first transmission and the next retransmissions.

By considering this possibility, the device will have to learn more, thus, we expect the learning time to be longer, but it could be possible that the final performance gain increases too, in terms of network performance. We present in Section 5.4.5 an analysis to check this performance gain, for various heuristics based on the UCB algorithm. Here after, we start by presenting a mathematical derivation that backups this idea. To do so, we study the collision probabilities considering the Markov process depicted in Figure 5.11, and foresee the impact of using bandit strategies, as well as setting guidelines for the design of heuristic approaches.

Probability of collision at a second transmission slot. It is well known [Abr70, Rob75] that having a collision during an access time can be overcome by a retransmission procedure (this can take several retransmission attempts). Our goal here is to obtain a mathematical approximation of the collision probability at the second transmission slot p_{c1}, as a function of the first collision probability p_c. We make two approximations $\mathcal{H}_1$ and $\mathcal{H}_2$ defined as (they are hypotheses on which the rest of this section is built),

- $\mathcal{H}_1$: The probability p_{c1}, is composed by the sum of two probabilities: i) the probability of colliding consecutively twice, *i.e.*, the devices that collide at a given time slot and collide again when retransmitting their packets, and ii) the probability of collision among devices that did not collide in the same previous collision. Moreover, we suppose that the number of devices involved in a collision is small in comparison to the total number of devices. This is very realistic as a very small proportion of devices transmit at the same period, due to their low duty cycle.

- $\mathcal{H}_2$: The total number of back-off stages at time t is constant, and it is assumed to be large enough to consider that no device will ever be in the last failure state (this case is the one on the right side in Figure 5.11), after $\mathrm{MaxBackOff}$ successive failed retransmissions (otherwise, its battery life can be threatened if it does not have re-fueling capabilities).

Considering one device and one channel, we denote x_t^i the probability that it is transmitting a packet for the $(i+1)$-th time in a given time slot t (with $i \in \{0, \dots, \mathrm{MaxBackOff} - 1\}$), and we denote $x_t = \sum_{i=0}^{\mathrm{MaxBackOff}-1} x_t^i$ the probability that it transmits a packet (*i.e.*, just the sum on i of x_t^i). We consider $N > 0$ active devices following the same policy.

We assume to be in the steady state [Nor98], in our Markov chain model depicted in Figure 5.11, and thus the probabilities no longer depend on the slot number t (*i.e.*, $\forall t, x_t = x$). Therefore, the probability that this device has a collision at the first transmission is p_c, and has the following expression

$$p_c = 1 - (1-x)^{N-1} \iff x = 1 - (1-p_c)^{\frac{1}{N-1}} . \tag{5.6}$$

Moreover, from (5.6) we define the probability $p_{cp}(n)$ that involves the collision of n packets sent by each IoT device (for any $1 \leq n \leq N-1$), during the first transmission slot, and is defined by $p_{cp}(n) = \binom{N-1}{n} x^n (1-x)^{N-1-n}$. As explained above, if an IoT device experiences a collision at the first transmission, it proceeds for the retransmission of its packet after a random back-off interval. We denote p_{ca} the probability to have a collision with a packet involved in the previous collision. Under the $\mathcal{H}_1$ assumption, the number of packets involved in the same previous collision remains very small in comparison to the total number of devices that may transmit during this time. In other words, this collision probability does not depend on previous retransmissions and is equal to p_c. So, the probability that the same device's packet experiences again a collision at the second time slot is

$$p_{c1} = p_{ca} + (1 - p_{ca})\, p_c. \tag{5.7}$$

If the device has a collision at the first attempt, we consider $p_{bp}(n)$ the probability that it has a collision with *exactly* n packets (for any $1 \leq n \leq N-1$), and that *at least one* of the n devices involved in this first collision chooses the same back-off interval,

$$p_{bp}(n) = \binom{N-1}{n} x^n (1-x)^{N-1-n} \left[1 - \left(1 - \frac{1}{m}\right)^n\right]. \tag{5.8}$$

Besides, p_{ca} is the conditional probability of collision with a packet sent by a device involved in the previous collision given that the packet experienced collision at its first transmission. Hence, under hypothesis $\mathcal{H}_2$, we can use Bayes theorem and the law of total probability to relate p_{ca} with $p_{bp}(n)$, and the different probabilities that a device experienced a collision during the first slot and has the same back-off interval for its retransmission is, $p_{ca} = \frac{1}{p_c}\sum_{n=1}^{N-1} p_{bp}(n)$. Therefore, the expression of p_{ca} is

$$\begin{aligned}&\frac{1}{p_c}\sum_{n=1}^{N-1}\binom{N-1}{n} x^n (1-x)^{N-1-n}\left[1-\left(1-\frac{1}{m}\right)^n\right]\\ &= 1 - \frac{1}{p_c}\sum_{n=1}^{N-1}\binom{N-1}{n} x^n (1-x)^{N-1-n}\left(1-\frac{1}{m}\right)^n.\end{aligned} \tag{5.9}$$

Once again under $\mathcal{H}_1$, assuming that the number of devices involved in the first collision is small compared to $N-1$, the first $N_0 \ll N-1$ terms of the sum in (5.9) are predominant. We derive $p_{ca} \simeq 1 - \frac{1}{p_c}\sum_{n=1}^{N_0}\binom{N-1}{n}x^n (1-x)^{N-1-n}\left(1-\frac{1}{m}\right)^n$. Moreover, for these terms, n is small compared to $N-1$, and so $N-1-n$ can be approximated to $N-1$. Thus it gives,

$$p_{ca} \simeq 1 - \frac{(1-x)^{N-1}}{p_c}\sum_{n=1}^{N_0}\binom{N-1}{n} x^n \left(1-\frac{1}{m}\right)^n. \tag{5.10}$$

Assuming $\mathcal{H}_1$ amounts to consider that $x \ll 1$. As a consequence, the sum in equation (5.10) can be supplemented by negligible terms,

$$p_{ca} \simeq 1 - \frac{(1-x)^{N-1}}{p_c}\sum_{n=1}^{N-1}\binom{N-1}{n} x^n \left(1-\frac{1}{m}\right)^n. \tag{5.11}$$

The binomial theorem expands the sum in (5.11), so we can rewrite the expression of p_{ca}

$$p_{ca} \simeq 1 - \left(\frac{1}{p_c} - 1\right)\left[1 + \left(1 - (1-p_c)^{\frac{1}{N-1}}\right)\left(1 - \frac{1}{m}\right)\right]^{N-1}. \tag{5.12}$$

Finally, our approximation of p_{c1} can be obtained by inserting (5.12) in (5.7).

$$\begin{aligned} p_{c1} &= p_{ca} + (1-p_{ca})\, p_c = (1-p_c)p_{ca} + p_c \\ &\simeq (1-p_c)\,(1 - \left(\frac{1}{p_c} - 1\right) \left[1 + \left(1 - (1-p_c)^{\frac{1}{N-1}}\right)\left(1 - \frac{1}{m}\right)\right]^{N-1} + p_c. \end{aligned} \tag{5.13}$$

Behavior analysis of p_c and p_{c1} In order to assess the proposed approximation, we suppose a unique channel where all the devices follow the same contention Markov process. We simulate an ALOHA protocol with a maximum number of retransmissions $\mathrm{MaxBackOff} = 10$, a maximum back-off interval $m = 10$, and a transmission probability $p = 10^{-3}$. In Figure 5.12, we show the collision probabilities for different number of devices N (from $N = 50$ up-to $N = 400$), for both p_c and p_{c1}. From this simulations, we can verify that our approximation is very precise for lower values $p_{c1} \leq 30\%$ (*i.e.*, red and orange curves are quite close). Moreover, a significant gap between p_{c1} and p_c, of up-to 10%, can be observed, which suggests us to resort to MAB algorithms for the channel selection for both the first transmission and next retransmissions.

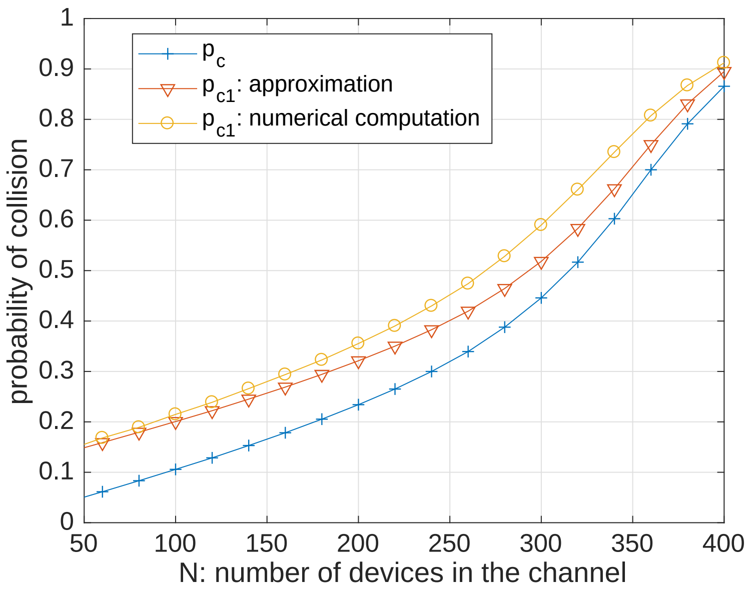


Figure 5.12 – Proposed approximation for the probability of collision at the second transmission. It is more precise for smaller values of N.

Learning is useful for non-congested networks. It is worth to highlight that, if we write (5.7) as $p_{c1} = p_c + p_{ca}(1-p_c)$, then it is obvious that p_{c1} is always larger than p_c (as $p_{ca}(1-p_c) > 0$). But for large values of p_c, $p_{ca}(1-p_c) \simeq 0$ so the gap gets small, and for small values of p_c the gap is significant. Moreover, we can verify (*e.g.*, numerically or by differentiating) that the gap decreases when p_c increases (for fixed N and m). This backups mathematically the observation we made from Figure 5.12: the smaller the p_c, the larger is the gap between p_c and p_{c1}. We interpret this fact in two different situations:

- On the one hand, in a congested network, when devices suffer from a large probability of collision on their first transmission (*i.e.*, p_c is not so small), then $p_{c1} \simeq p_c$ and so devices cannot really hope to reduce their collision probabilities even if the use a different channel for retransmission.
- On the other hand, if p_c is small enough, *i.e.*, in a network not yet too congested, then our derivation shows that $p_{c1} > p_c$, meaning that the possible gain of retransmitting in a different channel that the one used for the first transmission can be large, in terms of collision probability (*e.g.*, up-to 10% in this experimental setting). In other words, when learning can be useful (small p_c), learning to retransmit in a different channel can have a large impact on the global collision rate, thus justifying our approach.

5.4.3 The first heuristic: UCB unaware of retransmissions

This first heuristic we propose is unaware of retransmission: the same channel is used for retransmissions. The UCB algorithm is implemented independently by each device, we denote it “first-stage” UCB, and we present it in Algorithm 5.1. It is the same algorithm as the Algorithm 2.2 in Section 2.4.2 (using the same UCB indexes as we defined them in equation (2.7)), but we write it again to make clear the difference with first transmission and retransmission of messages. Note that a device using this first approach is only able to select a channel for the first transmission (using UCB, line 3-5), and then it uses the same channel for all the corresponding retransmissions of a packet (if retransmissions happen, line 7-8).

```
1 for $t = 1, \dots, T$ do
2     if First packet transmission then
3         Compute $\forall k, U_k(t) = \widehat{\mu_k}(t) + \sqrt{\alpha \ln(t)/N_k(t)}$;
4         Transmit in channel $A(t) \sim \mathcal{U}(\arg\max_k U_k(t))$;
5         Reward $r(t) = 1$, if Ack is received, else $0$;
6     else                                   // Retransmit in same channel
7         j $\leftarrow$ last channel selected by first-stage UCB;
8         Transmit in channel $A(t) = j$;
9 end
```

Algorithm 5.1: First-stage UCB and retransmission in same channel (“Only UCB”).

More formally, for one device, let $N_k(t)$ be the number of times the channel k (for $k \in [K]$) was selected up-to time $t-1$, for $t \geq 1$ for any $t \in \mathbb{N}$, $N_k(t) = \sum_{\tau=1}^{t-1} \mathbb{1}(A(\tau) = k)$. The empirical mean estimator $\widehat{\mu_k}(t)$ of channel k is defined as $\widehat{\mu_k}(t) = \frac{1}{N_k(t)} \sum_{\tau=1}^{t-1} r(\tau)\mathbb{1}(A(\tau) = k)$, where $r(t) = Y_{k,t}$ is the reward obtained after transmission in channel k at time t. The upper confidence bound in each channel k is defined as $U_k(t) = \widehat{\mu_k}(t) + \sqrt{\alpha \ln(t)/N_k(t)}$. Finally, the transmission channel at time step t is $A(t) \sim \mathcal{U}(\arg\max_k U_k(t))$.

Small warning about notations: local (device) time vs global (gateway) time: In all the algorithms and notations presented in this Section, the time steps t do not refer to the *global* time steps (as seen from the gateway), but rather the current number of uplink transmissions (or retransmissions) that was carried out by the device. Each device wakes up whenever it has to send some data, following its random emission process (here, we propose a Bernoulli process of small probability, *e.g.*, $p = 10^{-3}$), and when it woke up, it sends its packet, and will send again for at most $\mathrm{MaxBackOff}$ times (*e.g.*, $\mathrm{MaxBackOff} = 10$) until it receives an *Ack*. In other words, the time steps t in the following algorithms denote the number of time the device sent an uplink packet, and these uplink (re)transmissions can happen at different (real-world) times, more or less spread out in time, it does not matter to the learning part of this model and work.

5.4.4 Heuristics to (try to) learn how to retransmit efficiently

A device that implements the UCB algorithm is led to focus its transmissions and retransmissions in the channel which is currently identified as the best. As explained above in Section 5.4.2, focusing in one channel could increase the collision probability in retransmissions. We describe here the proposed heuristics for the channel selection in a retransmission. It is carried out taking into account that a device can incorporate a different channel selection strategy while being in a back-off state. Hence, a natural question is to evaluate whether using this additional contextual information can improve the performance of a learning policy.

For that end, all of our heuristics comprise two stages: the first stage is a UCB algorithm employed for the first attempt to transmit, and the second stage is another algorithm used for channel selections for the next retransmissions. We present below four heuristics for this second stage. Their short names (with their colors) are used in the legend on Figures 5.13, 5.14), and are given in "quotes" in the corresponding paragraphs.

Uniform random retransmission "(Random"). In this first proposal, the device uses a random channel selection, following a uniform distribution (on $[K]$). It is described below in Algorithm 5.2. More precisely, the first-stage UCB use rewards built from the acknowledgments that the device received or not for its first-stage transmission, but do not use any feedback about any of the retransmissions of any message.

UCB **for retransmission ("UCB").** Instead of applying a random channel selection for retransmission, another heuristic is to use a second UCB algorithm in the second stage. In other words, we expect that this algorithm is able to learn the best channel to retransmit a packet. It is described in Algorithm 5.3, and it is still a practical approach, since the storage

```
for $t = 1, \dots, T$ do
    if *First packet transmission* then
        Use first-stage UCB as in Algorithm 5.1;
    else                                          // Random retransmission
        Transmit in channel $A(t) \sim \mathcal{U}(1, \dots, K)$;
end
```

Algorithm 5.2: Heuristic: uniform random retransmission "(Random").

requirements and time complexity remains linear w.r.t. the number of channels K (*i.e.*, $\mathcal{O}(K)$). Note that, we use the subscript (^{r}) to denote the variables $\widehat{\mu^r}(t)$, $B_k^r(t)$ and $U_k^r(t)$, related to the UCB algorithm employed for the retransmission.

```
for $t = 1, \dots, T$ do
    if *First packet transmission* then
        Use first-stage UCB as in Algorithm 5.1;
    else                                   // Packet retransmission with UCB$^r$
        Compute $\forall k, U_k^r(t) = \widehat{\mu_k^r}(t) + \sqrt{\alpha \ln(t) / N_k^r(t)}$;
        Transmit in channel $C^r(t) \sim \mathcal{U}(\arg\max_k U_k^r(t))$;
        Reward $r^r(t) = 1$, if *Ack* is received, else $0$;
end
```

Algorithm 5.3: Heuristic: UCB for retransmission ("UCB").

K different UCBs for retransmission ("K UCB") Another heuristic is to not use the same algorithm no matter where the collision occurred, but to use K different second-stage UCB algorithms. It means that after a failed first transmission in channel j, the device relies on the j-th algorithm to decide its retransmission. The corresponding algorithm is depicted in Algorithm 5.4. Each of these algorithms are denoted using the subscript (^{j}), for $j \in [K]$.

Although, this approach increases the complexity and storage requirements (now, of order $\mathcal{O}(K^2)$). For our LPWA networks of interest, such as LoRaWAN, the cost of its implementation is still affordable, since a small number of channels is used. For instance, for $K = 4$ channels, the memory to store $K + 1 = 5$ algorithms is of the order of the requirements to store one.

Note that for other networks this heuristic could not be practical. The storage requirements and time complexity is now quadratic in K, and as such we no longer consider this heuristic to be a practical proposal in some LPWA networks, as for instance Sigfox networks consist in a large number of very narrow-band channels (*e.g.*, $K = 128$). But for LoRaWAN networks with $K = 4$, storing $K + 1 = 5$ algorithms does not cost much more than storing 2.

```
for $t = 1, \dots, T$ do                                  // At every time step
    if First packet transmission then
        Use first-stage UCB as in Algorithm 5.1;
    else                                    // Packet retransmission with UCB$^j$
        j $\leftarrow$ last channel selected by first-stage UCB;
        Compute $\forall k, U_k^j(t) = \widehat{\mu_k}^j(t) + \sqrt{\alpha \ln(t)/N_k^j(t)}$;
        Transmit in channel $A^j(t) \sim \mathcal{U}(\arg\max_k U_k^j(t))$;
        Reward $r^j_{A^j(t)}(t) = 1$ if Ack is received, else $0$;
end
```
Algorithm 5.4: Heuristic: K different UCBs for retransmission ("K UCB").

Delayed UCB **for retransmission ("Delayed** UCB**").** This last heuristic is a composite of the random retransmission (Algorithm 5.2) and the UCB retransmission (Algorithm 5.3) approaches. Instead of starting the second stage UCB directly from the first retransmission, we introduce a fixed delay $\Delta \in \mathbb{N}$, $\Delta \geq 1$, and start to rely on the second stage UCB after Δ transmissions. The selection for the first steps is handled with the random retransmission.

The idea behind this delay is to allow the first stage UCB to start learning the best channel, before starting the second stage UCB (see details in Algorithm 5.5). The number of transmissions to wait before applying the second algorithm is denoted by Δ, it has to be fixed before-hand[9]. Note that, we use the subscript $(^d)$ to denote the variables related to the delayed second-stage UCB algorithm.

```
for $t = 1, \dots, T$ do                                  // At every time step
    if First packet transmission then
        Use first-stage UCB as in Algorithm 5.1;
    else if $t \leq \Delta$ then                          // Random retransmission
        Transmit randomly in a channel $A(t) \sim \mathcal{U}(1, \dots, K)$.
    else                          // Packet retransmission with delayed UCB$^d$
        Compute $\forall k, U_k^d(t) = \widehat{\mu_k^d}(t) + \sqrt{\alpha \ln(t)/N_k^d(t)}$;
        Transmit in channel $A^d(t) \sim \mathcal{U}(\arg\max_k U_k^d(t))$;
        Reward $r^d_{A^d(t)}(t) = 1$ if Ack is received, else $0$;
end
```
Algorithm 5.5: Heuristic: delayed UCB for retransmission ("Delayed UCB").

[9] Choosing the value of Δ could be done by extensive benchmarks but such approach goes against the reinforcement learning idea: an heuristic should work against any problem, without the need to simulate the problem before-hand to find a good value of some internal parameter. As such, we only consider a delay of $\Delta = 100$ in our experiments, and we did not try to optimize it.

Other ideas that we did not explore. Instead of considering a first-stage UCB followed by a random channel retransmission, we could have considered a random first-stage channel retransmission followed by a second-stage UCB. This was not considered in our study [BBMVM19], and we preferred to not add it to the experiments presented below, for three reasons: to avoid clutter in the plots, to win some time as these experiments had to run for quite a long time, but mainly because the first-stage channel selection is most important one as we illustrate by the large difference in terms of performance between the uniform channel selection and the "Only UCB" heuristic, below in Figures 5.13 and 5.14.

Using another bandit policy? We could have chose any bandit algorithm for the "first stage" component used in this Section, and for simplicity and clarity we focused on a simple but efficient one (UCB). We believe that the same empirical results, but more importantly, the same conclusions, could be given if we used Bayes-UCB or other bandit algorithm for the base building block in the different heuristics proposed in this section. Our goal here was not to optimize on the bandit policy (as we presented in Section 3.3 some numerical simulations doing precisely this), but rather to compare the different heuristics, for a fixed bandit policy (for which we preferred to use UCB for simplicity).

5.4.5 Numerical results

We simulate our network considering N devices following the contention Markov process described in Section 5.4.1, and a LoRaWAN standard with $K = 4$ channels, as in Section 5.2. Each device is set to transmit with a fixed probability $p = 10^{-3}$, *i.e.*, a packet about every 20 minutes for time slots of 1 s. For the evaluation of the proposed heuristics, a total number of $T = 20 \times 10^4$ time slots is considered, and the results are averaged over 1000 independent random simulations.

In a first scenario, we consider a total number of $N = 1000$ IoT devices, with a non-uniform repartition of static devices given by $10\%, 30\%, 30\%, 30\%$ for the four channels. In other words, the channels are occupied[10] respectively 10%, 30%, 30%, and 30% of time, and the contention Markov process considered is given by $\mathrm{MaxBackOff} = 5$, and $m = 5$. In Figure 5.13, we show the successful transmission rate versus the number of slots, for all the proposed heuristics.

A first result is that all the heuristics clearly outperform the non-learning approach that simply uses random channel selection for both transmissions and retransmissions (*i.e.*, the "**no** UCB" curve in black), which is (still) the current state-of-the-art in IoT networks. The improvement of the heuristics over the non-learning approach is clear, and for every heuristic

[10] Note that we consider higher occupancy rates that the ones considered previously in Section 5.3, because the goal of this experimental section is to evaluate our approach that use learning to optimize the way each device retransmits its packets. We need to have a large enough probability p_c of retransmission if we want to see a difference between the different learning heuristics, which is why we consider a network more densily occupied than in Section 5.3.

that uses a kind of learning mechanism it can be observed that the successful transmission rate increases rapidly (or equivalently the PLR decreases). Moreover, all of these approaches show a fast convergence making them suitable for the targeted application. It is also worth mentioning that the employment of the same UCB algorithm for retransmissions, denoted here as "Only UCB", achieves a better performance, while a "Random" retransmission features a slight degradation. This result can be explained as follows: the loss of performance related to the separation of information for several algorithms is greater than the gain obtained by considering the first transmissions and retransmissions separately.

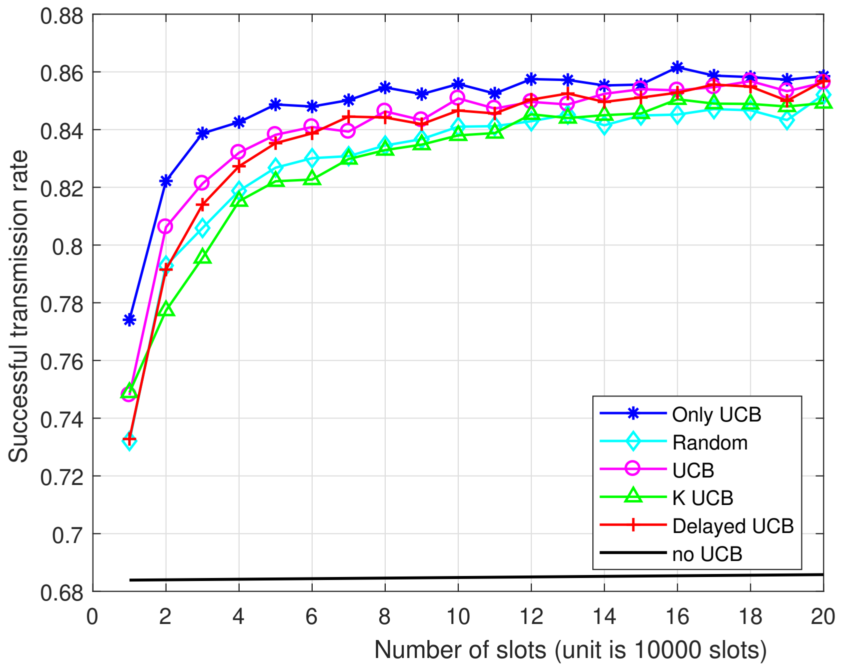


Figure 5.13 – Comparison between the exposed heuristics for the retransmission:" Only UCB", "Random", "UCB", "K UCB", and "Delayed UCB". The usage of the same learning policy for transmissions and retransmission is named "Only UCB", whereas the usage of a random channel selection, for both transmission and retransmission, is labeled as "**no** UCB". First scenario: learning helps but learning to retransmit smartly is not needed, as we observe that the random retransmission heuristic achieves similar performance than the others. We considered $N = 1000$ static IoT devices, that occupy the 4 channels in a static way leading to mean occupancy rates of $10\%, 30\%, 30\%, 30\%$.

We also consider in our experiment the case of an ALOHA protocol using $\mathrm{MaxBackOff} = 5$, and $m = 10$, a statistic distribution of the devices about $40\%, 30\%, 20\%, 10\%$ for the four channels, and $N = 2000$ IoT devices. The corresponding results are depicted in Figure 5.14. In this case the successful transmission rate is degraded compared with achieved results in Figure 5.13. We can explain this by the fact that we are considering in our network more devices, which increases the collision probability. It is important to highlight, that the "Random" retransmission heuristic shows a poor performance in comparison to the other heuristics, and it can be attributed to the fact that the number of retransmission is increased, and consequently a learning approach is able to take advantage of it. Furthermore, the "UCB", "K UCB" and "Delayed UCB" heuristics behave similarly to "Only UCB", after a similar convergence time.

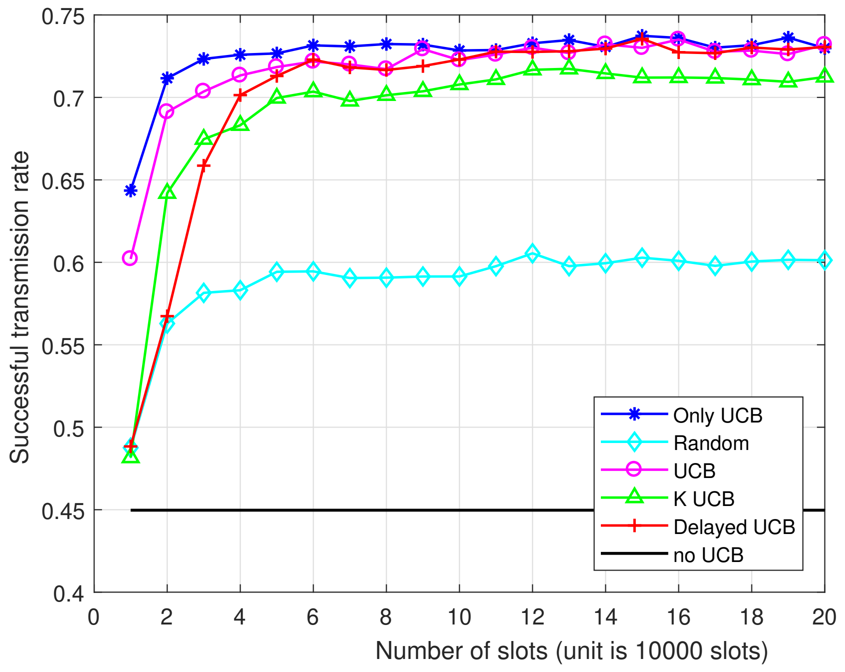


Figure 5.14 – Second scenario: learning helps a lot (a gain of 30% in terms of collision probability), and learning to retransmit smartly is needed. We observe that the random retransmission achieves poor performance compared to the others. We considered $N = 2000$ static IoT devices, that occupy the 4 channels in a static way leading to mean occupancy rates of $10\%, 30\%, 30\%, 30\%$.

The conclusions we can draw from these results are twofold. Firstly, MAB learning algorithms are very useful to reduce the collision rate in LPWA networks, a gain of up-to 30% of successful transmission rate is observed after convergence. Secondly, using learning mechanisms for retransmissions can be a simple yet efficient and interesting way to reduce collisions in IoT networks, even in networks with massive deployments of IoT as this can be checked in Figure 5.14, where the random retransmission heuristic is greatly outperformed by the any of the UCB-based approaches, that use learning for channel selection during the retransmission procedure. With 10% to 30% occupancy rates the considered example of IoT network can indeed be considered as an IoT network with a massive deployment of devices.

Reproducility of the experiments. The source code (MATLAB or Octave) used for the simulations and the figures of this section is open-sourced under the MIT License, and published at `Bitbucket.org/scee_ietr/ucb_smart_retrans`. It was written in collaboration with Rémi Bonnefoi and Julio César Mango-Vasquez, in July and August 2018.

5.5 Conclusion

To sum-up, we focused in this chapter on models of IoT networks, and we proposed to use classical stationary multi-armed bandit learning algorithms implemented in a selfish and decentralized manner by each of the dynamic devices of the IoT network. We presented two models of wireless IoT network without sensing, inspired by the ALOHA protocol, and with or without retransmission of uplink packages in case of collisions. In both cases, we try to model the existing standards, like the LoRa standard, and we demonstrated the efficiency of the proposed MAB-based approach in both numerical simulations and empirical measurements on real wireless radio signals.

A satisfying message is that this learning-based approach seems to be very efficient, as it allows the IoT devices to automatically and independently increase their successful transmission rate. It is also quite surprising that stochastic MAB algorithms can be of any use in such non-stationary applications. Sadly, analyzing our model with thousands of independent devices communicating and learning in their own (random) time scales turned out to be of extreme difficulty. We can also highlight that the two models we studied are complementary: the first model (without retransmission of packets), is interesting for its simplicity, and focused on improving the battery life of IoT devices, at the cost of a lower Quality of Service (QoS), while the second model is more interesting if the main target is an improvement of the QoS, rather than a longer battery life.

Multi-Players MAB. On the first hand, we are actually able to analyze a simpler model, if we assume to have at most $M \leq K$ devices with a transmission probability of $p = 1$. Instead of the experiment-driven direction pursued in this chapter, another possibility is to consider a *multi-players MAB* model to describe our problem. The main difference between the two models is the fact that in this chapter, devices do not transmit their messages at every time step, but following a random activation process (with a fixed transmission probability $p < 1$). If static and dynamic devices that have to transmit at a fixed time are denoted *active devices*, then their random activation patter makes the number of active devices an (unpredictable) random variable. Analyzing multi-players MAB models under this hypothesis is much harder, and is left as a future work. We study this first direction in the next Chapter 6.

Non-stationary MAB. On the other hand, while it is hard to analyze the models of this chapter due to their non-stationary behaviors, we are also able to analyze a simpler model, if we focus on a single player accessing a network which is assumed to be *piece-wise stationary* (MAB problems which are stationary on “long enough” intervals, of unknown locations and lengths). We also study this second direction in Chapter 7.

5.6 Appendix

5.6.1 Proof of Proposition 5.1

We include here the missing details of the proof of Proposition 5.1. First, we need to justify that the objective function is quasi-convex, in each of its coordinates. Then, we develop the computation of $D_i^*(\lambda)$, as a closed form expression of the system parameters (K, p), the repartition of static devices $(S_1, \dots, S_K)$ and the Lagrange multiplier λ.

Quasi-convexity.

- For $0 < \gamma < 1$, the function $g(x) \doteq x\gamma^x$ is quasi-convex on $[0, \infty)$, *i.e.*, $g(\eta x + (1-\eta)y) \leq \max(g(x), g(y))$ for any $x, y \in [0, \infty)$ and $\eta \in [0, 1]$ (definition from [Lue68]). Indeed, $g(\eta x + (1-\eta)y) = \eta\,[x(\gamma^x)^\eta]\,\gamma^{((1-\eta)y)} + (1-\eta)\,[y(\gamma^y)^{1-\eta}]\,\gamma^{\eta x}$, and $\gamma^{((1-\eta)y)} \leq 1$ and $\gamma^{\eta x} \leq 1$. But also $(\gamma^x)^\eta \leq \gamma^x$ as $\eta \leq 1$, and the same holds for $(\gamma^y)^{1-\eta} \leq \gamma^y$. So $g(\eta x + (1-\eta)y) \leq \eta(x\gamma^x) + (1-\eta)(y\gamma^y)$ which is a convex combination of $x\gamma^x$ and $y\gamma^y$, so smaller than the larger of the two values, and so $g(\eta x + (1-\eta)y) \leq \max(x\gamma^x, y\gamma^y)$.
- The function $f(D_1, \dots, D_K) \doteq \sum_{i=1}^{K} D_i(1-p)^{S_i+D_i-1}$ is quasi-convex in each of its coordinates, on $[0, \infty)^K$, as a sum of component-wise quasi-convex functions (with $\gamma = (1-p) \in (0, 1)$, thanks to the first point). $\square$

Derivation of the Lagrange multiplier solution. Let us now prove the expression for D_i^* given above in (5.5).

- If $\boldsymbol{D} \doteq (D_1, \dots, D_K)$, the Lagrangian is denoted $\mathcal{L}(\boldsymbol{D}, \lambda) \doteq f(\boldsymbol{D}) + \lambda(D - \sum_{i=1}^{K} D_i)$, and its derivative w.r.t. D_i is $\frac{\partial}{\partial D_i}\mathcal{L}(\boldsymbol{D}, \lambda) = (1-p)^{S_i+D_i-1} + \ln(1-p)D_i(1-p)^{S_i+D_i-1} - \lambda$.
- So the gradient is zero $\frac{\partial}{\partial D_i}\mathcal{L}(\boldsymbol{D}, \lambda)|_{D_i = D_i^*} = 0$ iff D_i^* satisfies $(1-p)^{D_i^*}(1 + \ln(1-p)D_i^*) = \lambda/(1-p)^{S_i-1}$. Let $x = \ln(1-p)D_i^*$ this is equivalent to $\mathrm{e}^x(1+x) = \lambda/(1-p)^{S_i-1}$ and with $y = 1+x$, we get $\mathrm{e}^y y = \lambda\mathrm{e}/(1-p)^{S_i-1}$.
- By using the $\mathcal{W}$-Lambert function $\mathcal{W}$ [CGH+96], reciprocal of $y \mapsto \mathrm{e}^y y$, we get $x = y - 1 = \mathcal{W}(\lambda\mathrm{e}/(1-p)^{S_i-1}) - 1$. So the gradient of the Lagrangian is zero iff $D_i^* = \max(0, x/\ln(1-p)) = \left[\frac{1}{\ln(1-p)}\mathcal{W}(\lambda\mathrm{e}/(1-p)^{S_i-1}) - 1\right]^+$, because D_i^* has to be non-negative. This gives the i-th coordinate of the unique saddle point of $f(\boldsymbol{D})$, and so the unique solution to the maximization problem (5.4a), thanks to [Lue68, Theorem 1]. $\square$

5.6.2 Illustration of the GNU Radio Companion Flowcharts

For the curious reader, we wanted to include here an illustration and a description of the three components of the demonstration we presented above in Section 5.3.

The code corresponding to the following components is written in `C++` and `Python` for the hand-written blocks, and using the GUI of the GNU Radio Companion software. The flow-graphs are saved as XML files, and the complete code of our demo is fully available online, open-sourced under GPLv3 license, at `bitbucket.org/scee_ietr/malin-multi-arm-bandit-learning-for-iot-networks-with-grc/`.

Random traffic generator

Figure 5.15 shows the **random traffic generator** flow-graph. `Generator` is the only hand-written block, which is configured with a list of active channels, a list of occupancy rate, and constants about the PHY layer (preamble length and data length). It uses one USRP only for emitting data ("Sink" mode).

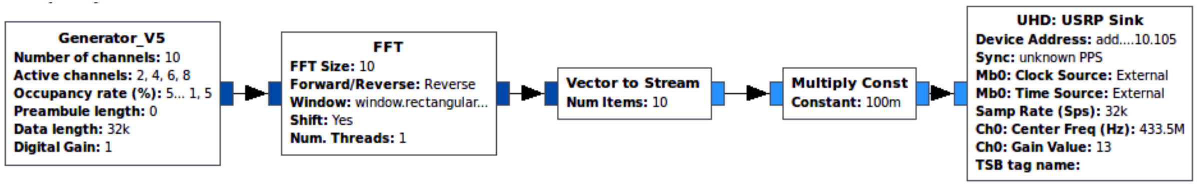


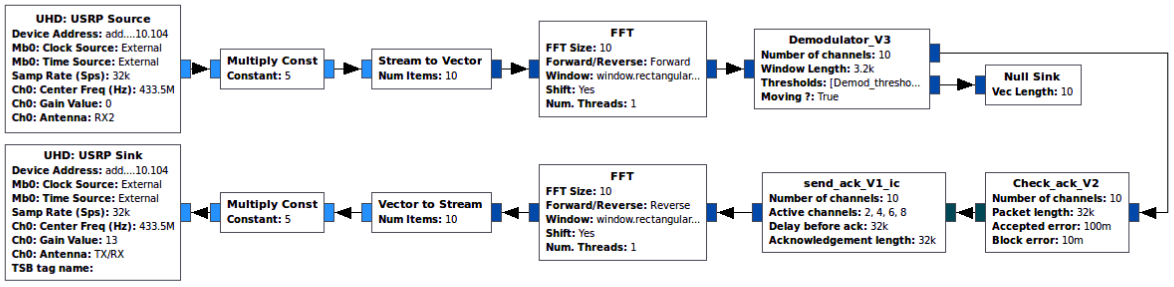
Figure 5.15 – The **random traffic generator** flow-graph.

IoT gateway

The **IoT gateway** is presented in Figure 5.16 below. We wrote the following blocks:

- 1) the `Demodulator` block, which is configured with a list of detection threshold (on the received power),
- 2) the `Check_ack` block, which is configured with a maximum block error and an accepted error rates (to decide when a message is close enough to an acknowledgement),
- and finally 3) the `send_ack` block, which is configured with the list of active channels, and knowledge about the PHY layer (delay before sending the Ack and its length).

All blocks need to know constants about the PHY layer (preamble length and data length). The IoT gateway uses one USRP for both emitting and receiving data ("Sink" and "Source" modes).


UHD: USRP Source
Device Address: add....10.104
Mb0: Clock Source: External
Mb0: Time Source: External
Samp Rate (Sps): 32k
Ch0: Center Freq (Hz): 433.5M
Ch0: Gain Value: 0
Ch0: Antenna: RX2

Multiply Const
Constant: 5

Stream to Vector
Num Items: 10

FFT
FFT Size: 10
Forward/Reverse: Forward
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

Demodulator_V3
Number of channels: 10
Window Length: 3.2k
Thresholds: [Demod_thresho...
Moving ?: True

Null Sink
Vec Length: 10

UHD: USRP Sink
Device Address: add....10.104
Mb0: Clock Source: External
Mb0: Time Source: External
Samp Rate (Sps): 32k
Ch0: Center Freq (Hz): 433.5M
Ch0: Gain Value: 13
Ch0: Antenna: TX/RX
TSB tag name:

Multiply Const
Constant: 5

Vector to Stream
Num Items: 10

FFT
FFT Size: 10
Forward/Reverse: Reverse
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

send_ack_V1_ic
Number of channels: 10
Active channels: 2, 4, 6, 8
Delay before ack: 32k
Acknowledgement length: 32k

Check_ack_V2
Number of channels: 10
Packet length: 32k
Accepted error: 100m
Block error: 10m

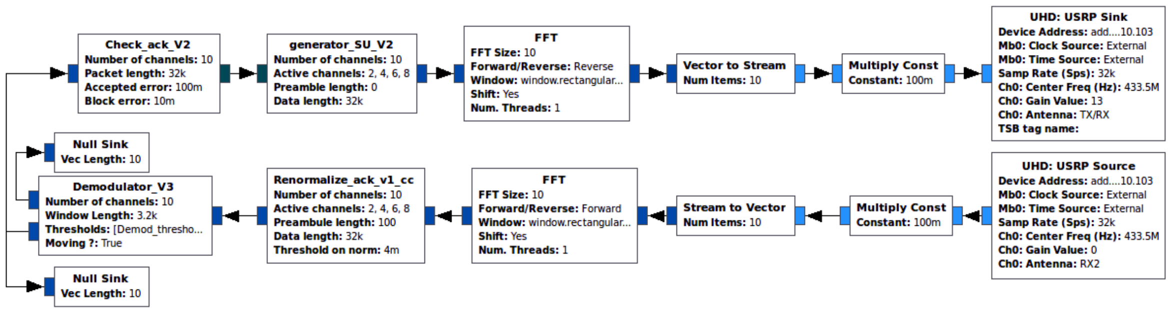
Figure 5.16 – The **IoT gateway** flow-graph.

IoT dynamic device

Figure 5.17 below shows the **IoT dynamic device** flow-graph. The hand-written blocks are the

- 1) the `Renormalized_ack` block, which is configured with the list of active channels and extra knowledge about the PHY layer (preamble length, message length, and a threshold to tune detection of the incoming Ack),
- 2) the `Demodulator` and 3) the `Check_ack` blocks, both shared with the IoT gateway,
- and finally 4) the `generator_SU` block, which embeds the UCB or Thompson Sampling algorithm, and which is configured with the list of active channels.

Most blocks need to know constants about the PHY layer (preamble length and data length). Any of the IoT dynamic devices use one USRP for both emitting and receiving data (“Sink” and “Source” modes)


Check_ack_V2
Number of channels: 10
Packet length: 32k
Accepted error: 100m
Block error: 10m

generator_SU_V2
Number of channels: 10
Active channels: 2, 4, 6, 8
Preamble length: 0
Data length: 32k

FFT
FFT Size: 10
Forward/Reverse: Reverse
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

Vector to Stream
Num Items: 10

Multiply Const
Constant: 100m

UHD: USRP Sink
Device Address: add....10.103
Mb0: Clock Source: External
Mb0: Time Source: External
Samp Rate (Sps): 32k
Ch0: Center Freq (Hz): 433.5M
Ch0: Gain Value: 13
Ch0: Antenna: TX/RX
TSB tag name:

Null Sink
Vec Length: 10

Demodulator_V3
Number of channels: 10
Window Length: 3.2k
Thresholds: [Demod_thresho...
Moving ?: True

Null Sink
Vec Length: 10

Renormalize_ack_v1_cc
Number of channels: 10
Active channels: 2, 4, 6, 8
Preambule length: 100
Data length: 32k
Threshold on norm: 4m

FFT
FFT Size: 10
Forward/Reverse: Forward
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

Stream to Vector
Num Items: 10

Multiply Const
Constant: 100m

UHD: USRP Source
Device Address: add....10.103
Mb0: Clock Source: External
Mb0: Time Source: External
Samp Rate (Sps): 32k
Ch0: Center Freq (Hz): 433.5M
Ch0: Gain Value: 0
Ch0: Antenna: RX2

Figure 5.17 – The **IoT dynamic device** flow-graph.

Chapter 6

Multi-Players Multi-Armed Bandits

In this chapter, we are interested in a more formal approach to the decentralized learning problem presented in Chapter 5. We restrict to the easier case of at most $M \leq K$ devices in a network with K channels, because it was found very hard to analyze the aforementioned IoT network model. We discuss three feedback levels that give variants of the multi-players MAB model previously studied in the literature. Using a decomposition of the centralized system regret, we start by explaining the intuition about what an efficient decentralized algorithm should do, and then we propose $\mathrm{RandTopM}$ and MCTopM, two new orthogonalization schemes. Combining them with the kl-UCB index policy gives order-optimal algorithms, that achieve state-of-the-art performance, as illustrated by numerical simulations. We conclude by reviewing different extensions of the multi-players models.

Contents

6.1 Motivations for multi-players MAB models

As we saw in Chapters 1 and 5, a crucial step for the development of Cognitive Radio is to insert *multiple* smart devices in the *same* background traffic (at least $M \geq 2$). In this Chapter 6, we are interested in a more formal approach to the decentralized learning problem presented in the previous Chapter 5. Such networks can be modelled using a decentralized multi-players multi-armed bandit problem, where arms are channels and players are dynamic IoT devices. The decentralized hypothesis means that the central base station does not given any direct order to the devices, they are all in charge of deciding the channel they each want to use at each time step. Moreover, the devices are independent and do not directly communicate with each other either. We show that dynamic end-devices are able to use limited feedback sent by the gateway to learn to find orthogonal affectations of the group of end-devices to the best radio channels, automatically and without explicit communication with each other. In other words, the end-devices are able to learn to cooperate efficiently, in a completely decentralized and autonomous manner.

We consider M identical dynamic IoT devices, communicating with a unique gateway, in K orthogonal channels and in a acknowledgment-based wireless protocol slotted in time. A perfect time and frequency synchronization is assumed, and some *i.i.d.* background traffic is assumed to be non-uniformly in the K channels. As before, if two or more devices decide to use the same channel at the same time, a *collision* arises and none of sent uplink packet can be received by the gateway. Unfortunately, it is very hard to formally analyze the IoT network models we presented in Chapter 5, mainly because of the random activation process of all the dynamic (*i.e.*, learning) devices. Even if there are many identical bandit algorithms learning independently and in a decentralized way, the difficulty mainly comes from the fact that all of them are only communicating at some (random) time steps, and at each time step the number of communicating devices is random and unpredictable.

Mainly for these two reasons, for this Chapter 6 we prefer to only consider at most $M \leq K$ devices, communicating at each time step. Note that in the model of Chapter 5, this hypothesis $M \leq K$ corresponds to choosing a probability of activation of $p = 1$, as we assumed that (in average) no more than K devices should be active at the same step. We start by reviewing previous works on multi-players MAB models, which all considered the easier case of *sensing feedback*. But in our model in this chapter, each device also uses a Multi-Armed Bandit algorithm, to maximize its number of successful communications, by using the received acknowledgement *Ack* as a (random) binary reward after each uplink message (*i.e.*, at each time step). With the presence of a central controller that can assign the devices to separate channels, this amounts to choosing at each time step *several* arms of a MAB in order to maximize the global rewards, and can thus be viewed as an application of the multiple-play bandit, introduced by [AVW87a] and recently studied by [KHN15]. They essentially proved that existing algorithms can be easily extended to the multiple-play case, with provable guarantees on their regret. Due to the communication cost implied by a central controller, a more relevant model is the *decentralized multi-players* multi-armed bandit model, introduced by [LZ10] and further studied shortly after in [AMT10, AMTA11], in which players select arms individually and collisions may occur, that yield a loss of reward. Further algorithms were proposed in similar models by [TL12] and [KNJ12] (under the assumption that each arm is a Markov chain), and by [AM15, AM16] and [RSS16] for *i.i.d.* arms. In the point of view of the wireless protocol, each time frame is separated as before in a *sensing* phase (during which the device senses for the background traffic), an *uplink* phase (during which the device sends a packet to the gateway if it sensed the chosen channel to be free), and a *downlink* phase (during which it waits for an *Ack* from the gateway). In this first model, the binary reward is 1 only if the channel was sensed to be free of background traffic **and** if *Ack* was received, and the device has access to both information. We present two algorithms, $\mathrm{RandTopM}$ and MCTopM, based on the combination of an efficient MAB index policy (we chose kl-UCB) and a smart orthogonalization procedure, based on a random hoping procedure called Musical Chair. Like in previous works, we consider the centralized system regret (multi-players regret), or simply referred to as regret. We start by showing an improved asymptotical regret lower-bound for any algorithm of a certain class, including previous solutions such as $\mathrm{RhoRand}$ and our two solutions. We then analyze our MCTopM algorithm and we show that its regret upper-bound is logarithmic and order-optimal, improving over the previous state-of-the-art. We also present extensive numerical simulations that show that our proposal outperforms all previous solutions and is much more efficient in this easier model of sensing feedback with a fixed and known number of players M accessing $K \geq M$ channels.

The goal for every player is to select most of the time one of the M best arms, without colliding too often with other players. A first difficulty relies in the well-known trade-off between *exploration* and *exploitation*: players need to explore all the arms to estimate their means, while trying to focus on the best arms to gain as much rewards as possible. The decentralized setting considers wireless protocol with no direct or explicit exchange of information between players, and assumes that the players only know K and M. To avoid collisions, players should furthermore find orthogonal configurations, *i.e.*, the M players use the M best arms without any collision, without explicitly communicating[1] with each other. Hence, in that case the trade-off is to be found between exploration, exploitation *and* low collisions.

All these above-mentioned works are motivated by the OSA problem, in which it is assumed that *sensing* occurs, that is, each smart device observes the availability of a channel (*i.e.*, a reward from the arm) *before* trying to transmit and possibly experiment a collision

[1]One must now be careful about this aspect when stating that "no explicit communications are allowed between players" in the model, as [BP18] proved that even in the no sensing case, the "communication trick" can be used to exchange information between players, by generating collisions at some pre-agreed times. We discuss this work more in details at the end of this chapter, in Section 6.7.3.

with other smart devices. However some real radio networks do not use sensing at all, *e.g.*, emerging standards currently or recently developed for *Internet of Things* (IoT) networks, such as LoRaWAN. Thus, to take into account these new applications, algorithms with additional constraints on the available feedback have to be proposed within the multiple-player MAB model. Especially, the typical approach that combines a (single-player) bandit algorithm based on the sensing information –to learn the quality of the channels while targeting the best ones– with a low-complexity decentralized collision avoidance protocol, is no longer possible. Our article [BK18a] was the first to study this other model of multi-players bandits for the “no sensing” case, for which we only proposed an heuristic, the naive Selfish strategy as it was already used in Chapter 5. Even if empirical simulations showed that Selfish-kl-UCB performs very well, it is a mistake to only consider mean regret, as we found on numerical simulations as well as formal derivation on a simple example of $K = M = 2$ that the Selfish heuristic can have a linear regret with a low probability (and so asymptotically it has linear expected regret and fails to solve “no sensing” multi-players MAB problems). We do not propose any other efficient algorithm, but our work presented in April 2018 has strongly inspired two articles shortly after [LM18, BP18]. They confirmed our finding that Selfish can have a linear regret, and they both proposed new algorithms. We include some numerical simulations to compare some of them, and we present in details the current state-of-the-art of research on multi-players MAB models without sensing.

Outline. The rest of this chapter is organized as follows. We introduce the multi-players bandit model with three feedback levels in Section 6.2. We present a useful regret decomposition in Section 6.3, and illustrate it in the Appendix 6.9. The Selfish, RandTopM and MCTopM algorithms are introduced in Section 6.4, for which we present our theoretical analysis in Section 6.5. The result of our experimental study reported in Section 6.6. Finally, we present in Section 6.7 a review of the recent literature which studies variants of the model presented in this Chapter. For some extensions, we discuss how to adapt our proposals and illustrate the empirical performances of such modification of MCTopM-kl-UCB, leaving theoretical analyses as future works.

Publication. This chapter is mainly based on our article [BK18a].

6.2 Three feedback levels for the multi-players bandit model

Similarly to what is presented in Chapter 2, we consider a K-armed Bernoulli bandit model, in which arm k is a Bernoulli distribution with mean $\mu_k \in [0, 1]$. We denote $(Y_{k,t})_{t\in\mathbb{N}}$ the *i.i.d.* (binary) *reward stream* for arm k, that satisfies $\mathbb{P}(Y_{k,t} = 1) = \mu_k$ and that is independent from the other rewards streams.

Only Bernoulli. However we mention that our lower bound and all our algorithms (and their analysis) can be easily extended to one-dimensional exponential families (just like for the kl-UCB algorithm of [CGM+13]). For simplicity, we focus on the Bernoulli case, that is also the most relevant for Cognitive Radio problems, based on channel availability.

In the multi-players MAB setting, there are $M \in [K]$ players (or agents), that have to make decisions at some pre-specified time instants. At time step $t \in \mathbb{N}, t \geq 1$, player j selects an arm $A^j(t)$, independently from the other players' selections.

Definition 6.1. *A* collision *occurs at time t if at least two players choose the same arm. We introduce the two events, for $j \in [M]$ and $k \in [K]$,*

$$C^j(t) \doteq \{\exists j' \neq j : A^{j'}(t) = A^j(t)\} \quad \textit{and} \quad C_k(t) \doteq \left\{\#\{j : A^j(t) = k\} > 1\right\}, \tag{6.1}$$

that respectively indicate that a collision occurs at time t for player j and that a collision occurs at time t on arm k.

Each player j then receives (and observes) the *binary rewards* $r^j(t) \in \{0, 1\}$,

$$r^j(t) \doteq Y_{A^j(t),t}\, \mathbb{1}(\overline{C^j(t)}). \tag{6.2}$$

In other words, she receives the reward of the selected arm if she is the only one to select this arm, and a reward zero otherwise. This provides another reason to focus on the Bernoulli model. It is the hardest model, in the sense that receiving a reward zero is *not enough* to detect collisions. For other models, the data streams $(Y_{k,s})_s$ are usually continuously distributed, with no probability mass at the value of zero (*e.g.*, Gaussian). Hence receiving $r^j(t) = 0$ directly gives $\mathbb{1}(C^j(t)) = 1$. Note that other models for rewards loss have also been proposed in the literature, for instance the reward is randomly allocated to one of the players selecting it. To stay consistent and for simplicity, with the model presented in the previous Chapter 5, we preferred to focus on full reward occlusion in this chapter.

A multi-players MAB strategy is formally defined as a tuple $\mathcal{A} = (\mathcal{A}_1, \dots, \mathcal{A}_M)$ of arm selection strategies for of each of the M players, and the goal is to propose a strategy that maximizes the total reward of the system, under some constraints. First, each player j should adopt a *sequential* strategy $\mathcal{A}_j$, that decides which arm to select at time t based on *previous observations*. Previous observations for player j at time t always include the previously chosen arms $A^j(s)$ and received rewards $r^j(s)$ for $s < t$, but may also include the *sensing information* $Y_{A^j(t),t}$ or the *collision information* $C^j(t)$. More precisely, depending on the application, one may consider the following three observation models, (I), (II) and (III).

If the arms have continuous distributions (or are such that $\mathbb{P}(Y_{k,s} = 0) = 0$), the *sensing information* $Y_{A^j(t),t}$ and *collision information* $C^j(t)$ can always be extracted from the reward information. But for Bernoulli distributions, one may consider the following three observation models, that are not equivalent:

(I) **Simultaneous sensing and collision**: player j observes $Y_{A^j(t),t}$ *and* $C^j(t)$. We note that this first model was never previously studied, but we do not focus on it because of its unrealistic aspect, and its simplicity (it is easier than the following model, for which we obtain good theoretical results).

(II) **Sensing, then collision**: player j observes $Y_{A^j(t),t}$, *then* observes the reward, and thus also $C^j(t)$ only if $Y_{A^j(t),t} = 1$. This common setup, studied for example by [AMTA11, RSS16], is relevant to model the OSA problem: the device first checks for the presence of primary users in the chosen channel, if this channel is free ($Y_{A^j(t),t} = 1$), the transmission is successful ($r^j(t) = 1$) if no collision occurs with other smart devices ($\overline{C^j(t)}$).

(III) **No sensing**: player j only observes the reward $r^j(t)$. For IoT networks, this reward can be interpreted as an acknowledgement from a Base Station, received when a communication was successful. A lack of acknowledgment may be due to a collision with a device from the background traffic ($Y_{A^j(t),t} = 0$), or to a collision with one of the others players ($C^j(t)$). However, the sensing and collision information are censored. Recently, our work presented in Chapter 5 [BBM$^+$17] presented the first (bandit-based) algorithmic solutions under this (harder) feedback model, in a slightly different setup, more suited to large scale IoT applications, but as explained above we did not present theoretical results for this third model (yet).

Under each of these three models, we define $\mathcal{F}_t^j$ to be the filtration generated by the observations gathered by player j up to time t (which contains different information under models (I), (II) and (III)). While a *centralized* algorithm may select the vector of actions for all players $(A^1(t), \dots, A^M(t))$ based on all the observations from $\bigcup_j \mathcal{F}_{t-1}^j$, under a *decentralized* algorithm the arm selected at time t by player j only depends on the past observation of this player. More formally, $A^j(t)$ is assumed to be $\mathcal{F}_{t-1}^j$-measurable.

Definition 6.2. *We denote by μ_1^* the best mean, μ_2^* the second best etc, and by M-*best *the (non-sorted) set of the indices of the M arms with largest mean* (best arms)*: if $\mu_1^* = \mu_{k_1}, \dots, \mu_M^* = \mu_{k_M}$ then $M\text{-best} = \{k_1, \dots, k_M\}$. Similarly, M-*worst *denotes the set of indices of the $K - M$ arms with smallest means* (worst arms)*, $[K] \setminus M\text{-best}$.*

Note that they are both uniquely defined if $\mu_M^* > \mu_{M+1}^*$.

Following a natural approach in the bandit literature, we evaluate the performance of a multi-players strategy using the *expected regret* (later simply referred to as regret), that measures the performance gap with respect to the best possible strategy. The regret of the strategy $\mathcal{A}$ at horizon T is the difference between the cumulated reward of an oracle strategy, assigning in this case the M players to $M\text{-best}$, and the cumulated reward of strategy $\mathcal{A}$:

Definition 6.3. *The excepted centralized multi-players regret is defined by*

$$R_T(\boldsymbol{\mu}, M, \mathcal{A}) \doteq \left(\sum_{k=1}^{M} \mu_k^*\right) T - \mathbb{E}_\mu \left[\sum_{t=1}^{T} \sum_{j=1}^{M} r^j(t)\right]. \tag{6.3}$$

With this definition, maximizing the expected sum of global reward of the system is indeed equivalent to minimizing the regret, and we investigate the best possible *regret rate* of a decentralized multi-players algorithm in the next section.

6.3 Decomposing the regret to get an intuition for the design of efficient decentralized algorithms

In this section, we provide a useful decomposition of the regret (Lemma 6.5) that permits to establish a new problem-dependent lower bound on the regret (Theorem 6.8), and also provides key insights on the derivation of regret upper bounds (Lemma 6.9).

Warning. The regret lower bound that we gave in [BK18a] is not applicable to any algorithm, as it was discovered by Boursier and Perchet, as they explain in Section 2.4 in [BP18]. The proof we gave in the Appendix of our paper [BK18a] was wrong in just one step, but the results stated in this section are now correct, thanks to the modifications proposed by Kaufmann and Mehrabian in Appendix E of [KM19]. The cost for this modification is to restrict the lower-bound to a smaller class of algorithms, which *should* contain RhoRand, RandTopM and MCTopM, but not the two algorithms proposed SIC-MMAB from [BP18] and Multiplayer Explore-then-Commit (M-ETC) from [KM19]. Proving that the proposed algorithms are in this class is left as an open question.

6.3.1 A useful regret decomposition

We introduce additional notations in the following definition.

Definition 6.4. *Let $N_k^j(T) \doteq \sum_{t=1}^{T} \mathbb{1}(A^j(t) = k)$, and denote $N_k(T) \doteq \sum_{j=1}^{M} N_k^j(T)$ the* number of selections *of arm $k \in [K]$ by any player $j \in [M]$, up to time T.*

Let $\mathcal{C}_k(T)$ *be the* number of colliding players *on arm* $k \in [K]$ *up to horizon* T*:*

$$\mathcal{C}_k(T) \doteq \sum_{t=1}^{T} \sum_{j=1}^{M} \mathbb{1}(C^j(t)) \mathbb{1}(A^j(t) = k). \tag{6.4}$$

Note that when n players choose arm k at time t, this counts as n collisions, not just one. So $\mathcal{C}_k(T)$ counts the total *number of colliding players* rather than the number of collision events. Hence there is small abuse of notation when calling it a number of collisions. By denoting $\mathcal{P}_M \doteq \left\{ \boldsymbol{\mu} \in [0,1]^K : \mu_M^* > \mu_{M+1}^* \right\}$ the set of bandit instances with a strict gap between the M best arms and the other arms, we now provide a regret decomposition for any $\boldsymbol{\mu} \in \mathcal{P}_M$.

Lemma 6.5. *For any bandit instance* $\boldsymbol{\mu} \in \mathcal{P}_M$ *such that* $\mu_M^* > \mu_{M+1}^*$*, it holds that*

$$R_T(\boldsymbol{\mu}, M, \mathcal{A}) = \underbrace{\sum_{k \in M\text{-worst}} (\mu_M^* - \mu_k) \mathbb{E}_\mu[N_k(T)]}_{\text{(a)}} \tag{6.5}$$

$$+ \underbrace{\sum_{k \in M\text{-best}} (\mu_k - \mu_M^*)(T - \mathbb{E}_\mu[N_k(T)])}_{\text{(b)}} + \underbrace{\sum_{k=1}^{K} \mu_k \mathbb{E}_\mu[\mathcal{C}_k(T)]}_{\text{(c)}}.$$

In this decomposition, term (a) counts the lost rewards due to *sub-optimal arms* selections ($k \in M$-worst), term (b) counts the number of times the *best arms* were *not* selected ($k \in M$-best), and term (c) counts the weighted number of collisions, on *all arms*.

Proof. Using the definition of regret $R_T(\boldsymbol{\mu}, M, \mathcal{A})$ from (6.3), denoted here R_T, and this collision indicator $\eta^j(t) \doteq \mathbb{1}(\overline{C^j(t)})$,

$$R_T = \left(\sum_{k=1}^{M} \mu_k^* \right) T - \mathbb{E}_\mu \left[\sum_{t=1}^{T} \sum_{j=1}^{M} Y_{A^j(t),t} \eta^j(t) \right] = \left(\sum_{k=1}^{M} \mu_k^* \right) T - \mathbb{E}_\mu \left[\sum_{t=1}^{T} \sum_{j=1}^{M} \mu_{A^j(t)} \eta^j(t) \right]$$

The last equality comes from the linearity of expectations, and the fact that $\mathbb{E}_\mu[Y_{k,t}] = \mu_k$ (for all t, from the *i.i.d.* hypothesis), and the independence from $A^j(t)$, $\eta^j(t)$ and $Y_{k,t}$ (observed *after* playing $A^j(t)$). So $\mathbb{E}_\mu[Y_{A^j(t),t} \eta^j(t)] = \sum_k \mathbb{E}_\mu[\mu_k \mathbb{1}(A^j(t), t) \eta^j(t)] = \mathbb{E}_\mu[\mu_{A^j(t)} \eta^j(t)]$. And so

$$R_T = \mathbb{E}_\mu \left[\sum_{t=1}^{T} \sum_{j \in M\text{-best}} \mu_j - \sum_{t=1}^{T} \sum_{j=1}^{M} \mu_{A^j(t)} \eta^j(t) \right]$$

$$= \left(\frac{1}{M} \sum_{j \in M\text{-best}} \mu_j \right) - \sum_{k=1}^{K} \sum_{j=1}^{M} \mu_k \mathbb{E}_\mu \left[N_k^j(T) \right] + \sum_{k=1}^{K} \mu_k \mathbb{E}_\mu \left[\mathcal{C}_k(T) \right].$$

For the first term, we have $TM = \sum\limits_{k=1}^{K} \sum\limits_{j=1}^{M} \mathbb{E}_\mu \left[N_k^j(T) \right]$, and if we denote $\overline{\mu}^* \doteq \frac{1}{M} \sum\limits_{j \in M\text{-best}} \mu_j$ the average mean of the M-best arms, then,

$$= \sum_{k=1}^{K} \sum_{j=1}^{M} (\overline{\mu}^* - \mu_k) \mathbb{E}_\mu \left[N_k^j(T) \right] + \sum_{k=1}^{K} \mu_k \mathbb{E}_\mu \left[\mathcal{C}_k(T) \right].$$

Let $\overline{\Delta_k} \doteq \overline{\mu}^* - \mu_k$ be the gap between the mean of the arm k and the M-best average mean, and if M^* denotes the index of the worst of the M-best arms (*i.e.*, $M^* = \arg\min_{k \in M\text{-best}}(\mu_k)$), then by splitting $[K]$ into three disjoint sets $M\text{-best} \cup M\text{-worst} = (M\text{-best} \setminus \{M^*\}) \cup \{M^*\} \cup M\text{-worst}$, we get

$$= \sum_{k \in M\text{-best} \setminus \{M\}} \overline{\Delta_k} \mathbb{E}_\mu \left[N_k(T) \right] + \overline{\Delta_{M^*}} \mathbb{E}_\mu \left[N_{M^*}(T) \right]$$
$$+ \sum_{k \in M\text{-worst}} \overline{\Delta_k} \mathbb{E}_\mu \left[N_k(T) \right] + \sum_{k=1}^{K} \mu_k \mathbb{E}_\mu \left[\mathcal{C}_k(T) \right].$$

But for $k = M^*$, $N_{M^*}(T) = TM^* - \sum\limits_{k \in M\text{-best} \setminus \{M\}} \mathbb{E}_\mu \left[N_k(T) \right] - \sum\limits_{k \in M\text{-worst}} \mathbb{E}_\mu \left[N_k(T) \right]$, so by recombining the terms, we obtain,

$$= \sum_{k \in M\text{-best} \setminus \{M\}} (\overline{\Delta_k} - \overline{\Delta_{M^*}}) \mathbb{E}_\mu \left[N_k(T) \right] + \overline{\Delta_{M^*}} TM^*$$
$$+ \sum_{k \in M\text{-worst}} (\overline{\Delta_k} - \overline{\Delta_{M^*}}) \mathbb{E}_\mu \left[N_k(T) \right] + \sum_{k=1}^{K} \mu_k \mathbb{E}_\mu \left[\mathcal{C}_k(T) \right].$$

The term $\overline{\Delta_k} - \overline{\Delta_{M^*}}$ simplifies to $\mu_{M^*} - \mu_k$, and so $\overline{\Delta_{M^*}} = \frac{1}{M} \sum_{k=1}^{M} \mu_k - \mu_{M^*}$ by definition of $\overline{\mu}^*$. And for $k = M^*$, $\mu_{M^*} - \mu_k = 0$, so the first sum can be written for $k = 1, \dots, M$ only, soIt

$$R_T = \sum_{k \in M\text{-best}} (\mu_{M^*} - \mu_k) \mathbb{E}_\mu \left[N_k(T) \right] + \sum_{k \in M\text{-best}} (\mu_k - \mu_{M^*}) T$$
$$+ \sum_{k \in M\text{-worst}} (\mu_{M^*} - \mu_k) \mathbb{E}_\mu \left[N_k(T) \right] + \sum_{k=1}^{K} \mu_k \mathbb{E}_\mu \left[\mathcal{C}_k(T) \right]$$

And so we obtain the decomposition with three terms (a), (b) and (c).

$$R_T = \sum_{k \in M\text{-best}} (\mu_k - \mu_{M^*}) \left(T - \mathbb{E}_\mu \left[N_k(T) \right] \right)$$
$$+ \sum_{k \in M\text{-worst}} (\mu_{M^*} - \mu_k) \mathbb{E}_\mu \left[N_k(T) \right] + \sum_{k=1}^{K} \mu_k \mathbb{E}_\mu \left[\mathcal{C}_k(T) \right].$$

Which is exactly the decomposition we wanted to prove. □

The regret decomposition in Lemma 6.5 is valid for both centralized and decentralized algorithms. For centralized algorithms, due to the absence of collisions, (c) is obviously zero, and (b) is non-negative, as $N_k(T) \leq T$. For decentralized algorithms, (c) may be significantly large, and term (b) may be negative, as many collisions on arm k may lead to $N_k(T) > T$ (which is counter intuitive with such notations). However, a careful manipulation of this decomposition shows that the regret is always lower bounded by term (a). The Figure 6.10 in Appendix illustrate two cases of $M < K$ and $M = K$ and the different impact of the three terms (a), (b) and (c) on the regret.

Lemma 6.6. *For any strategy $\mathcal{A}$ and $\boldsymbol{\mu} \in \mathcal{P}_M$, the regret is lower-bounded by (a):*

$$R_T(\boldsymbol{\mu}, M, \mathcal{A}) \geq \sum_{k \in M\text{-worst}} (\mu_M^* - \mu_k)\mathbb{E}_\mu[N_k(T)].$$

Proof. Note that term (c) is clearly lower bounded by 0 but it is not obvious for (b) as there is no reason for $N_k(T)$ to be upper bounded by T (it counts the selections of arm k by *all* the players, and for instance if player 1 is fixed on arm 1 and player 2 plays it at least once, then $N_1(T) \geq T+1$). Let $N_k^!(T) \doteq \sum_{t=1}^T \mathbb{1}(\exists! j, A^j(t) = k)$, where the notation $\exists!$ stands for “there exists a unique”. Then $N_k(T) = \sum_{t=1}^T \sum_{j=1}^M \mathbb{1}(A^j(t) = k)$ can be decomposed as

$$N_k(T) = \sum_{t=1}^T \mathbb{1}(\exists! j, A^j(t) = k) + \sum_{t=1}^T \sum_{j=1}^M c_{k,t}\mathbb{1}(A^j(t) = k) = N_k^!(T) + C_k(T).$$

We focus on the two terms $(b) + (c)$ from the decomposition of $R_T(\boldsymbol{\mu}, M, \mathcal{A})$ from Lemma 6.5,

$$\begin{aligned}
(b) + (c) &= \sum_{k \in M\text{-best}} (\mu_k - \mu_M^*)(T - \mathbb{E}_\mu[N_k^!(T)]) + \sum_{k \in M\text{-best}} \mu_M^* \mathbb{E}_\mu[C_k(T)] \\
&\quad + \sum_{k=1}^M \mu_k \mathbb{E}_\mu[C_k(T)] - \sum_{k \in M\text{-best}} \mu_k \mathbb{E}_\mu[C_k(T)] \\
&= \sum_{k \in M\text{-best}} (\mu_k - \mu_M^*)(T - \mathbb{E}_\mu[N_k^!(T)]) + \sum_{k \in M\text{-best}} \mu_M^* \mathbb{E}_\mu[C_k(T)] + \sum_{k \in M\text{-worst}} \mu_k \mathbb{E}_\mu[C_k(T)] \\
&= \sum_{k \in M\text{-best}} (\mu_k - \mu_M^*)(T - \mathbb{E}_\mu[N_k^!(T)]) + \sum_{k=1}^M \min(\mu_M^*, \mu_k)\mathbb{E}_\mu[C_k(T)].
\end{aligned}$$

And now both terms are non-negative, as $N_k^!(T) \leq T$, $\min(\mu_M^*, \mu_k) \geq 0$, and $C_k(T) \geq 0$, so $(b) + (c) \geq 0$ which proves that $R_T(\boldsymbol{\mu}, M, \mathcal{A}) = (a) + (b) + (c) \geq (a)$, as wanted. □

6.3.2 An improved asymptotic lower bound on the regret

Similarly to what we present above in Section 2.3, we use the Kullback-Leibler divergence to express our lower bound. We remind that $\mathrm{kl}(x, y) \doteq x \ln(x/y) + (1 - x) \ln((1 - x)/(1 - y))$ is the KL divergence between the Bernoulli distributions of means x and y. We first introduce the assumption under which we derive a regret lower bound, that generalizes the assumption of uniform efficiency given in Definition 2.6, a classical assumption made by [LR85] in single-player bandit models.

Definition 6.7. *An algorithm $\mathcal{A}$ is* **strongly uniformly efficient** *if for all $\boldsymbol{\mu} \in \mathcal{P}_M$,*

$$\forall a \in (0,1) R_T(\boldsymbol{\mu}, M, \mathcal{A}) \underset{T\to+\infty}{=} o(T^\alpha) \tag{6.6}$$

$$\textit{and} \forall a \in (0,1)\ \forall j \in [M], k \in M\text{-best},\ \frac{T}{M} - \mathbb{E}_\mu[N_k^j(T)] \underset{T\to+\infty}{=} o(T^\alpha). \tag{6.7}$$

Having a small regret on every problem instance, *i.e.*, being uniformly efficient, is a natural assumption, that rules out algorithms tuned to perform well on specific instances only (*e.g.*, fixed-armed algorithms). From this assumption $(R_T(\boldsymbol{\mu}, M, \mathcal{A}) = o(T^\alpha))$ and the decomposition of Lemma 6.5 one can see[2] that for every $k \in M$-best, $T - \mathbb{E}_\mu[N_k(T)]{=}o(T^\alpha)$, and so

$$\sum_{j=1}^{M} \left(\frac{T}{M} - \mathbb{E}_\mu[N_k^j(T)] \right) = o(T^\alpha). \tag{6.8}$$

The additional assumption in (6.7) further implies some notion of *fairness*, as it suggests that each of the M players spends on average the same amount of time on each of the M best arms. Note that this assumption is satisfied by any strategy that is invariant under every permutation of the players, *i.e.*, for which the distribution of the observations under $\mathcal{A}_\sigma = (\mathcal{A}_{\sigma(1)}, \dots, \mathcal{A}_{\sigma(M)})$ is independent from the choice of permutation $\sigma \in \Sigma_M$ of the set $\{1, \dots, M\}$. In that case, it holds that $\mathbb{E}_\mu[N_k^j(T)] = \mathbb{E}_\mu[N_k^{j'}(T)]$ for every arm k and $(j, j') \in [M]$, hence (6.7) and (6.8) are equivalent, and strong uniform efficiency is equivalent to standard uniform efficiency. Note that the algorithms studied in Section 6.4 are permutation invariant, and MCTopM-kl-UCB is thus an example of strongly uniformly efficient algorithm, as we prove in Section 6.5 that its regret is logarithmic on every instance $\mu \in \mathcal{P}_M$.

In the proofs of the regret bounds for RhoRand in [AMTA11], the authors briefly mention that their policy is invariant under permutation and that this yields a certain fairness guarantee, but without formalizing it more. The notion of fairness defined above can be interpreted as a "cooperative fairness", opposed to the notion of "arm fairness" that has been studied in a few

[2]With some arguments used in the proof of Lemma 6.6 to circumvent the fact that (b) may be negative.

papers on single-player stochastic MAB. For instance, [PGNN19] requires that each arm must be sampled at least a given fraction of the total time steps, *i.e.*, $\forall k, \mathbb{E}[N_k(T)] \geq r_k T$, where the constraint vector $[r_1, \dots, r_K]$ is given and known to the algorithm.

The following notations and the hypothesis on the collision information term (6.9) come from Appendix E of [KM19]. Similarly to what we did in Section 2.1, consider the observations O_t^j that she gathered after t rounds of its algorithm $\mathcal{A}_j$ for a fixed player $j \in [M]$, defined as $O_t^j \doteq \Big(U^j(1), Y_{A^j(1),1}, C^j(1), \dots, U^j(t), Y_{A^j(t),t}, C^j(t)\Big)$, where $U^j(t)$ denotes some external source of randomness useful to select $A^j(t+1)$ (*e.g.*, for an algorithm based on ranks and UCB indexes, like $\mathrm{RhoRand}$, when two arms have the same maximum index, the decision is an $\arg\max$ that is usually a uniform random selection among the arms with maximum index).

Fix a problem $\boldsymbol{\mu}$, then introduce an alternative model parametrized by $\boldsymbol{\lambda}$, with a small difference between $\boldsymbol{\mu}$. Fix a sub-optimal arm k and $\varepsilon > 0$, and $\lambda_k = \mu_M^* + \varepsilon$ and $\lambda_\ell = \mu_\ell$ for any $\ell \neq k$. We denote $\mathbb{P}_{\boldsymbol{\mu}}^{O_t^j}$ the distribution of the vector O_t under the model $\boldsymbol{\mu}$ when using algorithm $\mathcal{A}_j$. Then we introduce the *collision information term* as:

$$\mathcal{I}_{\boldsymbol{\mu},\boldsymbol{\lambda}}(\mathcal{A}_j, T) \doteq \sum_{t=1}^{T} \mathrm{KL}(\mathbb{P}_{\boldsymbol{\mu}}^{C_t | O_{t-1}}, \mathbb{P}_{\boldsymbol{\lambda}}^{C_t | O_{t-1}^j}). \tag{6.9}$$

For a *decentralized strategy* $\mathcal{A}$ that has access to the sensing information (*i.e.*, ruling out model (III)), and satisfies $\mathcal{I}_{\boldsymbol{\mu},\boldsymbol{\lambda}}(\mathcal{A}_j, T) = o(\ln(T))$, we now state a problem-dependent asymptotic lower bound on the number of sub-optimal arms selections. The additional hypothesis on $\mathcal{I}_{\boldsymbol{\mu},\boldsymbol{\lambda}}(\mathcal{A}, T)$ essentially says that "the collisions do not bring too much information on the arm means", as stated in [KM19]. The theorem stated below is proven in the Appendix of [BK18a], and it also yields an asymptotic logarithmic lower bound on the regret.

Theorem 6.8. *Under observation models* (*I*) *and* (*II*)*, consider a strongly uniformly efficient* decentralized *policy* $\mathcal{A} = (\mathcal{A}_1, \dots, \mathcal{A}_M)$ *and a problem* $\boldsymbol{\mu} \in \mathcal{P}_M$*. Furthermore, if* $\mathcal{A}_j$ *satisfies* $\mathcal{I}_{\boldsymbol{\mu},\boldsymbol{\lambda}}(\mathcal{A}_j, T) = o(\ln(T))$*, then*

$$\forall j \in [M],\ \forall k \in M\text{-worst},\quad \liminf_{T\to\infty} \frac{\mathbb{E}_\mu[N_k^j(T)]}{\ln(T)} \geq \frac{1}{\mathrm{kl}(\mu_k, \mu_M^*)}. \tag{6.10}$$

And so if $\mathcal{A}_1, \dots, \mathcal{A}_M$ *all satisfy this hypothesis, from Lemma 6.6, it follows that*

$$\liminf_{T\to+\infty} \frac{R_T(\boldsymbol{\mu}, M, \mathcal{A})}{\ln(T)} \geq M \times \left(\sum_{k \in M\text{-worst}} \frac{(\mu_M^* - \mu_k)}{\mathrm{kl}(\mu_k, \mu_M^*)} \right). \tag{6.11}$$

Observe that the regret lower bound (6.11) is tighter than the state-of-the-art lower bound in this setup given by [LZ10], that states that

$$\liminf_{T\to+\infty} \frac{R_T(\boldsymbol{\mu}, M, \mathcal{A})}{\ln(T)} \geq \sum_{k\in M\text{-worst}} \left(\sum_{j=1}^{M} \frac{(\mu_M^* - \mu_k)}{\mathrm{kl}(\mu_k, \mu_j^*)} \right), \tag{6.12}$$

as for every $k \in M$-worst and $j \in [M]$, $\mathrm{kl}(\mu_k, \mu_j^*) \geq \mathrm{kl}(\mu_k, \mu_M^*)$ (see Figure 6.9 in Appendix 6.9). It is worth mentioning that [LZ10] considered the more general assumption for $\mathcal{A}$ that there exists some numbers (a_k^j) such that $a_k^j T - \mathbb{E}_\mu[N_k^j(T)] = o(T^\alpha)$ whereas in Definition 6.7 we make the choice $a_k^j = 1/M$. Our result could be extended to this case, but we chose to keep the notation simple and focus on *fair allocation* of the optimal arms between players.

Price of decentralized learning. Interestingly, our lower bound is exactly a multiplicative constant factor M away from the lower bound given by [AVW87a] for centralized algorithms (which is clearly a simpler setting). This intuitively suggests the number of players M as the (multiplicative) *"price of decentralized learning"*. However, to establish our regret bound, we lower bounded the number of collisions by zero, which may be too optimistic. Indeed, for an algorithm to attain the lower bound (6.11), the number of selections of each sub-optimal arm should match the lower bound (6.10) *and* term (b) and term (c) in the regret decomposition of Lemma 6.5 should be negligible compared to $\ln(T)$.

To the best of our knowledge, no algorithm has been shown to experience only $o(\ln(T))$ collisions so far, for every $M \in \{2, \dots, K\}$ and $\boldsymbol{\mu} \in \mathcal{P}_M$. Since our article [BK18a], we kept as a future work the following question. A lower bound on the minimal number of collisions experienced by any strongly uniformly efficient decentralized algorithm would thus be a nice complement to our Theorem 6.8.

6.3.3 Towards regret upper bounds

A natural approach to obtain an upper bound on the regret of an algorithm is to upper bound separately each of the three terms defined in Lemma 6.5. The following result shows that term (b) can be related to the number of sub-optimal selections and the number of collisions that occurs on the M best arms.

Lemma 6.9. *The term (b) in Lemma 6.5 is upper bounded as*

$$(b) \leq (\mu_1^* - \mu_M^*)\Big(\sum_{k\in M\text{-worst}} \mathbb{E}_\mu[N_k(T)] + \sum_{k\in M\text{-best}} \mathbb{E}_\mu[C_k(T)] \Big). \tag{6.13}$$

Proof. Recall that we want to upper bound (b) $:= \sum_{k\in M\text{-best}}(\mu_k - \mu_{M*})\,(T - \mathbb{E}_\mu[N_k(T)])$. First, we observe that, for all $k \in M$-best,

$$\begin{aligned} T - \mathbb{E}_\mu[N_k(T)] &\leq T - \mathbb{E}_\mu\left[\textstyle\sum_{t=1}^T \mathbb{1}(\exists j : A^j(t) = k)\right] \\ &= \mathbb{E}_\mu\left[\textstyle\sum_{t=1}^T \mathbb{1}(\forall j, A_j(t) \neq k)\right] = \mathbb{E}_\mu\left[\textstyle\sum_{t=1}^T \mathbb{1}(k \notin \widehat{S}_t)\right], \end{aligned}$$

where we denote by $\widehat{S}_t = \{A^j(t), j \in [M]\}$ the set of selected arms at time t (with no repetition). With this notation one can write

$$\begin{aligned} (b) \quad &\leq \quad (\mu_1 - \mu_{M^*}) \sum_{k\in M\text{-best}} (T - \mathbb{E}_\mu[N_k(T)]) \leq (\mu_1 - \mu_{M^*})\mathbb{E}_\mu\left[\sum_{k\in M\text{-best}} \sum_{t=1}^T \mathbb{1}(k \notin \widehat{S}_t)\right] \\ &= \quad (\mu_1 - \mu_{M^*})\mathbb{E}_\mu\left[\sum_{t=1}^T \sum_{k\in M\text{-best}} \mathbb{1}(k \notin \widehat{S}_t)\right]. \end{aligned}$$

The quantity $\sum_{k\in M\text{-best}} \mathbb{1}(k \notin \widehat{S}_t)$ counts the number of optimal arms that have not been selected at time t. For each mis-selection of an optimal arm, there either exists a sub-optimal arm that has been selected, or an arm in M-best on which a collision occurs. Hence

$$\sum_{k\in M\text{-best}} \mathbb{1}(k \notin \widehat{S}_t) = \sum_{k\in M\text{-best}} \mathbb{1}(C_k(t)) + \sum_{k\in M\text{-worst}} \mathbb{1}(\exists j : A^j(t) = k),$$

which yields

$$\mathbb{E}_\mu\left[\sum_{t=1}^T \sum_{k\in M\text{-best}} \mathbb{1}(k \notin \widehat{S}_t)\right] \leq \sum_{k\in M\text{-best}} \mathbb{E}_\mu\left[\mathcal{C}_k(T)\right] + \sum_{k\in M\text{-worst}} \mathbb{E}_\mu\left[N_k(T)\right]$$

and Lemma 6.9 follows. $\square$

This result can also be used to recover Proposition 1 from [AMTA11], giving an upper bound on the regret that only depends on the *expected number of sub-optimal selections* – $\mathbb{E}_\mu[N_k(T)]$ for $k \in M$-worst – and the *expected number of colliding players on the optimal arms* – $\mathbb{E}_\mu[\mathcal{C}_k(T)]$ for $k \in M$-best. Note that, in term (c) the number of colliding players on the sub-optimal arm k may be upper bounded as $\mathbb{E}_\mu[\mathcal{C}_k(T)] \leq M\mathbb{E}_\mu[N_k(T)]$.

In the next section, we present an algorithm that has a logarithmic regret, while ensuring that the number of sub-optimal selections is matching the lower bound of Theorem 6.8.

6.4 New algorithms for multi-players bandits

Regardless of whether sensing is or not possible, we start by presenting formally in Section 6.4.1 the Selfish heuristic, as it was used by all the IoT devices in the first model presented in Chapter 5. It does not use an orthogonalization strategy as the collisions are directly accounted for in the UCB-like indices that are used by each device to select its channel. Selfish can also be used under observation model (III) –*without sensing*–, and without the knowledge of M. It was conjectured in [BK18a] that Selfish can suffer linear regret, and later confirmed in [LM18, BP18]. When sensing is possible, that is under observation models (I) and (II), most existing strategies build on a *single-player bandit algorithm* (usually an *index policy*) that relies on the sensing information, together with an *orthogonalization strategy* to deal with collisions. Following this approach, we introduce two new algorithms, RandTopM and MCTopM, in Section 6.4.2.

Warning: We want to strengthen the fact that in all the proposed algorithms, the players do *not* know their numbers $j \in [M]$, and they do not need to know it to achieve low regret. We discuss the fact that this would be an unrealistic hypothesis in Section 6.7.1 below.

6.4.1 The Selfish heuristic, with or without "sensing"

Under observation model (III) no sensing information is available and the previous algorithms cannot be used, as the sum of sensing information $S_k^j(t)$ and thus the empirical mean $\widehat{\mu}_k^j(t)$ cannot be computed, hence neither the indices $U_k^j(t)$. However, one can still define a notion of *empirical reward* received from arm k by player j, by introducing

$$\widetilde{S_k}^j(t) \doteq \sum_{t=1}^{T} r^j(t)\mathbb{1}(A^j(t)=k) \quad \text{and letting} \quad \widetilde{\mu_k}^j(t) \doteq \widetilde{S_k}^j(t) \;/\; N_k^j(t). \tag{6.14}$$

Note that $\widetilde{\mu_k}^j(t)$ is no longer meant to be an unbiased estimate of μ_k as it also takes into account the collision information, that is present in the reward. Based on this empirical reward, one can similarly defined modified indices as

$$\widetilde{U_k}^j(t) \doteq \begin{cases} \widetilde{\mu_k}^j(t) + \sqrt{f(t)/(2N_k^j(t))} & \text{for UCB,} \\ \sup\left\{q \in [0,1] : N_k^j(t) \times \mathrm{kl}(\widetilde{\mu_k}^j(t), q) \leq f(t)\right\} & \text{for kl-UCB.} \end{cases} \tag{6.15}$$

Given any of these two index policies (UCB or kl-UCB), the Selfish algorithm is then playing like a single-player index policy (see Algorithm 2.2) $A^j(t) \in \mathcal{U}(\arg\max_{k\in[K]} \widetilde{U_k}^j(t-1))$. The name "selfish" comes from the fact that each player is targeting, in a "selfish" way, the arm that has the highest index, instead of accepting to target only one of the M best. The reason that this may work precisely comes from the fact that $\widetilde{U_k}^j(t)$ is no longer an upper-confidence on μ_k, but some hybrid index that simultaneously increases when a transmission occurs and decreases when a collision occurs. This behavior is easier to be understood for the case of Selfish-UCB in which, letting $N_k^{j,C}(t) = \sum_{s=1}^{t} \mathbb{1}(C^j(t))$ be the number of collisions on arm k, one can show that the hybrid Selfish index induces a penalty proportional to the fraction of collision on this arm and the quality of the arm itself:

$$\widetilde{U_k}^j(t) = U_k^j(t) - \underbrace{\left(\frac{N_k^{j,C}(t)}{N_k^j(t)}\right)}_{\text{fraction of collisions}} \underbrace{\left(\frac{1}{N_k^{j,C}(t)} \sum_{t=1}^{T} Y_{A^j(t),t} \mathbb{1}(C^j(t)) \mathbb{1}(A^j(t) = k)\right)}_{\text{estimate of } \mu_k}. \tag{6.16}$$

From a bandit perspective, it looks like each player is using a stochastic bandit algorithm (UCB or kl-UCB) when interacting with K arms that give a feedback (the reward, and not the sensing information) that is far from being *i.i.d.* from some distribution, due to the collisions. As such, the algorithm does not appear to be well justified, and one may rather want to use adversarial bandit algorithms like EXP3 [ACBFS02], that do not require a stochastic (*i.i.d.*) assumption on arms. However, we found out empirically that Selfish is doing surprisingly well when using UCB-like indexes, greatly outperforming Selfish based on EXP3, like what we found in Chapter 5 in harder settings.

We illustrate in Section 6.5.3 that Selfish does have a (very) small probability to fail (badly), for some problem with small K, which precludes the possibility of a logarithmic regret for any problem. In most cases, it empirically performs similarly to all the algorithms described before, and usually outperforms RhoRand, even if it neither exploits the sensing information, nor the knowledge of the number of players M. Practitioners may still be interested by the algorithm, especially for Cognitive Radio applications in which sensing is hard or cannot be considered. In fact, we used the Selfish heuristic in Chapter 5, in the different models, using UCB or Thompson sampling as the underlying policy. Thus we propose next our main contribution, the MCTopM algorithm, proved to be asymptotically optimal for the identification of suboptimal arms (when using kl-UCB), attaining order-optimal logarithmic regret, and outperforming all the other algorithms for the "sensing case".

6.4.2 Two new strategies based on indices and orthogonalization

The approache we now describe for multi-players bandits can be used in combination with any index policy (see Algorithm 2.2), but we restrict our presentation to UCB algorithms, for which strong theoretical guarantees can be obtained. In particular, we focus on two types of indices: UCB$_1$ [ACBF02] and kl-UCB [CGM+13]. For player $j \in [M]$ denote $S_k^j(t) \doteq \sum_{s=1}^{t} Y_{k,s} \mathbb{1}(A^j(t) = k)$ the current sum of rewards obtained for arm k, then $\widehat{\mu}_k^j(t) \doteq S_k^j(t)/N_k^j(t)$

(if $N_k^j(t) \neq 0$) is the empirical mean of arm k, and thus one can define the index

$$U_k^j(t) \doteq \begin{cases} \widehat{\mu}_k^j(t) + \sqrt{f(t)/(2N_k^j(t))} & \text{for UCB,} \\ \sup \left\{ q \in [0,1] : N_k^j(t) \times \mathrm{kl}(\widehat{\mu}_k^j(t), q) \leq f(t) \right\} & \text{for kl-UCB,} \end{cases} \tag{6.17}$$

where $f(t)$ is some *exploration function*, usually taken to be $\ln(t)$ in practice, and slightly larger in theory, which ensures that $\mathbb{P}(U_k^j(t) \geq \mu_k) \gtrsim 1 - 1/t$ (see [CGM+13]). A classical (single-player) UCB algorithm aims at the arm with largest index, as presented above in Section 2.4.2. However, if each of the M players selects the arm with largest UCB, all the players will end up colliding most of the time on the best arm. To circumvent this problem, several coordination mechanisms have emerged, that rely on *ordering* the indices and targeting *one of* the M-best indices.

Two ideas for orthogonalization. On the one hand, the TDFS algorithm [LZ10] relies on the player agreeing in advance on the time steps at which they will target each of the M best indices. Even though some alternative without pre-agreement are proposed, they are quite complicated and we prefer to focus on other approaches. On the other hand, the RhoRand algorithm [AMTA11] relies on randomly selected *ranks*. More formally, letting $\pi(k, \mathbf{g})$ be the index of the k-th largest entry in a vector $\mathbf{g}$, in RhoRand each player maintains at time t an internal rank $R^j(t) \in [M]$ and selects at time t, $A^j(t) \doteq \pi\left(R^j(t), [U_\ell^j(t)]_{\ell=1,\dots,K}\right)$. If a collision occurs, a new rank is drawn uniformly at random, $R^j(t+1) \sim \mathcal{U}([M])$.

Our two proposals. We now propose two alternatives to this strategy, that do not rely on ranks and rather randomly fix themselves on one *arm* in $\widehat{M^j}(t)$, that is defined as the set of arms that have the M largest indices (at the current time t and for player j),

$$\widehat{M^j}(t) \doteq \left\{ \pi\left(k, \{U_\ell^j(t)\}_{\ell=1,\dots,K}\right), k = 1, \dots, M \right\}. \tag{6.18}$$

The RandTopM algorithm. We precisely state our first proposal below in Algorithm 6.1.

RandTopM is essentially a refinement over RhoRand, to not use the indirection of ranks, and a simpler version of MCTopM. The difference with MCTopM is that the later introduces a concept of a "Chair", by considering a binary "being fixed" state $s^j(t)$, as presented in Algorithm 6.2 below. In RandTopM, player j is always considered "not fixed", and a *collision always forces a uniform sampling of the next arm* from $\widehat{M^j}(t)$.

The MCTopM algorithm. Our second proposal MCTopM is stated below as Algorithm 6.2, it is a slightly more complex extension of the RandTopM algorithm. From there on, we focus on MCTopM as it is easier to analyze and performs better. Both algorithms ensure that player j always selects at time $t+1$ an arm from $\widehat{M^j}(t)$. When a collision occurs for a player implementing the RandTopM algorithm, that player randomly switches arm within $\widehat{M^j}$, while MCTopM uses a more sophisticated mechanism, that is reminiscent of "Musical Chair" (MC)

```
Let $A^j(0) \sim \mathcal{U}([K])$ and $C^j(0) = \text{False}$
for $t = 1, \dots, T$ do
    if $A^j(t-1) \notin \widehat{M^j}(t)$ then
        if $C^j(t-1)$ then                                   // collision at previous step
            $A^j(t) \sim \mathcal{U}\left(\widehat{M^j}(t)\right)$                            // randomly switch
        else             // randomly switch on an arm that had smaller UCB
            $A^j(t) \sim \mathcal{U}\left(\widehat{M^j}(t) \cap \left\{k : U_k^j(t-1) \leq U_{A^j(t)}^j(t-1)\right\}\right)$
    else
        $A^j(t) = A^j(t-1)$                                  // stays on the same arm
    Play arm $A^j(t)$, get new observations (sensing and collision),
    Compute the indices $U_k^j(t+1)$ and set $\widehat{M^j}(t+1)$ for next step.
end
```
Algorithm 6.1: The RandTopM decentralized learning policy (for an index policy U^j).

and inspired by the work of [RSS16]: players tend to fix themselves on arms ("chairs") and ignore future collision when this happens.

```
Let $A^j(0) \sim \mathcal{U}([K])$ and $C^j(0) = \text{False}$ and $s^j(1) = \text{False}$
for $t = 1, \dots, T$ do
    if $A^j(t-1) \notin \widehat{M^j}(t)$ then                      // transition (3) or (5)
        $A^j(t) \sim \mathcal{U}\left(\widehat{M^j}(t) \cap \left\{k : U_k^j(t-1) \leq U_{A^j(t)}^j(t-1)\right\}\right)$      // not empty
        $s^j(t) = \text{False}$          // aim at an arm with a smaller UCB at $t-1$
    else if $C^j(t-1)$ and $\overline{s^j(t-1)}$ then              // collision and not fixed
        $A^j(t) \sim \mathcal{U}\left(\widehat{M^j}(t)\right)$                                  // transition (2)
        $s^j(t) = \text{False}$
    else                                                    // transition (1) or (4)
        $A^j(t) = A^j(t-1)$                                 // stay on the previous arm
        $s^j(t) = \text{True}$                          // become or stay fixed on a "chair"
    Play arm $A^j(t)$, get new observations (sensing and collision),
    Compute the indices $U_k^j(t+1)$ and set $\widehat{M^j}(t+1)$ for next step.
end
```
Algorithm 6.2: The MCTopM decentralized learning policy (for an index policy U^j).

More precisely, under MCTopM, if player j did not encounter a collision when using arm k at time t, then she marks her current arm as a "chair" ($s^j(t) = \text{True}$), and will keep using it even if collisions happen in the future (Lines 9-11). As soon as this "chair" k is no longer in $\widehat{M_j}(t)$, a new arm is sampled uniformly from a subset of $\widehat{M^j}(t)$, defined with the previous indices $U^j(t)$ (Lines 3-5). The subset enforces a certain inequality on indices, $U_{k'}^j(t-1) \leq U_k^j(t-1)$

and $U^j_{k'}(t) \geq U^j_k(t)$, when switching from $k = A^j(t-1)$ to $k' = A^j(t)$. This helps to control the number of such changes of arm, as shown in Lemma 6.13. The considered subset is never empty as it contains at least the arm replacing the $k \in \widehat{M^j}(t-1)$ in $\widehat{M^j}(t)$. Collisions are dealt with only for non-fixed player j, and when the previous arm is still in $\widehat{M^j}(t)$. In this case, a new arm is sampled uniformly from $\widehat{M^j}(t)$ (Lines 6-8). This stationary aspect helps to minimize the number of collisions, as well as the number of switches of arm. The five different transitions (1), (2), (3), (4), (5) refer to the notations used in the analysis of MCTopM, and they are illustrated in Figure 6.1 below.

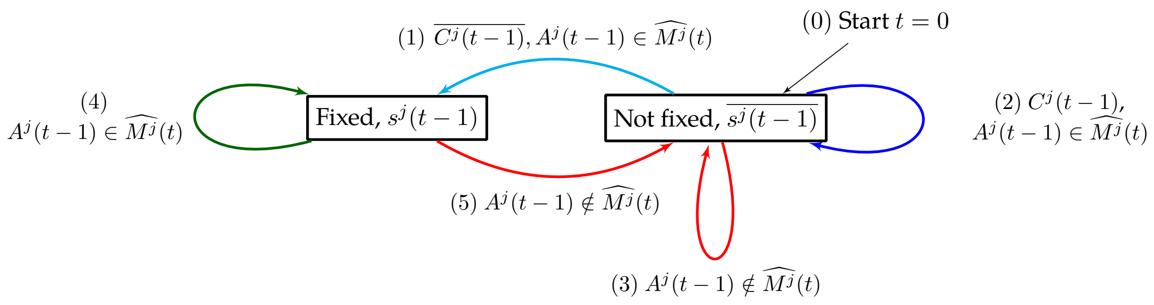

(0) Start $t = 0$
(1) $\overline{C^j(t-1)}, A^j(t-1) \in \widehat{M^j}(t)$
(4) $A^j(t-1) \in \widehat{M^j}(t)$
Fixed, $s^j(t-1)$
Not fixed, $\overline{s^j(t-1)}$
(2) $C^j(t-1)$, $A^j(t-1) \in \widehat{M^j}(t)$
(5) $A^j(t-1) \notin \widehat{M^j}(t)$
(3) $A^j(t-1) \notin \widehat{M^j}(t)$

Figure 6.1 – Player j using MCTopM, represented as "state machine" with 5 transitions. Taking one of the five transitions means playing one round of the Algorithm 6.2, to decide $A^j(t+1)$ using information of previous steps.

6.5 Theoretical elements, and regret upper bound for MCTopM

We now focus on obtaining positive results for the algorithms we proposed in Section 6.4 above. Section 6.5.1 gives an asymptotically optimal analysis of the expected number of sub-optimal draws for our two proposals RandTopM and MCTopM as well as for RhoRand, when they are combined with kl-UCB indices, and Section 6.5.2 proves that the number of collisions are logarithmic, and hence the regret of MCTopM is also logarithmic. Finally, Section 6.5.3 shortly discusses a disappointing result regarding Selfish.

6.5.1 Common analysis for RandTopM- and MCTopM

Lemma 6.10 gives a finite-time upper bound on the expected number of draws of a sub-optimal arm k for any player j, that holds for both RandTopM-kl-UCB and MCTopM-kl-UCB. Our improved analysis also applies to RhoRand. Explicit expressions for C_μ, D_μ can be found in the proof given below.

Lemma 6.10. *For any* $\boldsymbol{\mu} \in \mathcal{P}_M$*, let player* $j \in [M]$ *use the* RandTopM-, MCTopM- *or* RhoRand-kl-UCB *decentralized policy with exploration function* $f(t) \doteq \ln(t) + 3\ln(\ln(t))$. *Then for any* *sub-optimal arm* $k \in M\text{-worst}$ *there exists problem-dependent constants* $C_{\boldsymbol{\mu}}, D_{\boldsymbol{\mu}} > 0$ *such that*

$$\mathbb{E}_\mu[N_k^j(T)] \leq \frac{\ln(T)}{\mathrm{kl}(\mu_k, \mu_M^*)} + \underbrace{C_{\boldsymbol{\mu}}\sqrt{\ln(T)} + D_{\boldsymbol{\mu}} \ln(\ln(T)) + 3M + 1}_{=o(\ln(T))}. \tag{6.19}$$

It is important to notice that the leading constant in front of $\ln(T)$ is the same as in the constant featured in Equation (6.10) of Theorem 6.8. This result proves that the lower bound on sub-optimal selections is asymptotically matched for the three considered algorithms. This is a strong improvement in comparison to the previous state-of-the-art results [LZ10, AMTA11].

Proof. Fix $k \in M\text{-worst}$ and a player $j \in [M]$. The key observation is that for MCTopM, $\mathrm{RandTopM}$ as well as the $\mathrm{RhoRand}$ algorithm, it holds that

$$\left(A^j(t) = k\right) = \left(A^j(t) = k, \exists m \in M\text{-best} : U_m^j(t) < U_k^j(t)\right). \tag{6.20}$$

Indeed, for the three algorithms, an arm selected at time $t+1$ belongs to the set $\widehat{M^j}(t)$ of arms with M largest indices. If the sub-optimal arm k is selected at time t, it implies that $k \in \widehat{M^j}(t)$, and, because there are M arms in both $M\text{-best}$ and $\widehat{M^j}(t)$, one of the arms in $M\text{-best}$ must be excluded from $\widehat{M^j}(t)$. In particular, the index of arm k must be larger than the index of this particular arm m.

Thanks to (6.20) and if $\mathbb{P}_{\boldsymbol{\mu}}$ denote here the probability under model $\boldsymbol{\mu}$, it is easy to decompose the number of selections of arm k by user j up to round T as

$$\begin{aligned}\mathbb{E}_\mu[N_k^j(T)] = \mathbb{E}_\mu\left[\sum_{t=1}^T \mathbb{1}\left(A^j(t) = k\right)\right] &= \sum_{t=1}^T \mathbb{P}_{\boldsymbol{\mu}}\left(A^j(t) = k\right). \\ &= \sum_{t=1}^T \mathbb{P}_{\boldsymbol{\mu}}\left(A^j(t) = k,\ \ \exists m \in [M] :\ U_{m^*}^j(t) < U_k^j(t)\right).\end{aligned}$$

Considering the relative position of the upper-confidence bound $U_{m^*}^j(t)$ and the corresponding mean $\mu_m^* = \mu_{m^*}$, one can write the decomposition

$$\begin{aligned}\mathbb{E}_\mu[N_k^j(T)] \leq \sum_{t=1}^T \mathbb{P}_{\boldsymbol{\mu}}\left(\exists m_1 \in [M] :\ U_{m_1^*}(t) < \mu_m^*\right) + \\ \sum_{t=1}^T \mathbb{P}_{\boldsymbol{\mu}}\left(A^j(t) = k,\ \ \exists m_2 \in [M] :\ U_{m_2^*}(t) \leq U_k(t), \forall m_3 \in [M] :\ U_{m_3{}^*}(t) \geq \mu_{m_3}^*\right)\end{aligned}$$

$$\begin{aligned}\mathbb{E}_{\mu}[N^j_k(T)] &\leq \sum_{m_1=1}^{M}\sum_{t=1}^{T}\mathbb{P}_{\boldsymbol{\mu}}\left(U_{m_1^*}(t) < \mu^*_{m_1}\right) + \sum_{t=1}^{T}\mathbb{P}_{\boldsymbol{\mu}}\left(A^j(t) = k,\ \ \exists m_2 \in [M]:\ \mu^*_{m_2} \leq U_k(t)\right) \\ &\leq \sum_{m_1=1}^{M}\sum_{t=1}^{T}\mathbb{P}_{\boldsymbol{\mu}}\left(U_{m_1^*}(t) < \mu^*_{m_1}\right) + \sum_{t=1}^{T}\mathbb{P}_{\boldsymbol{\mu}}\left(A^j(t) = k,\ \mu_{M^*} \leq U_k(t)\right) \qquad (6.21)\end{aligned}$$

where the last inequality (for the first term) comes from the fact that μ_{M^*} is the smallest of the μ_{m^*} for $m \in [M]$.

Now each of the two terms in the right hand side of (6.21) can directly be upper bounded using tools developed by [CGM+13] for the analysis of kl-UCB. The leftmost term in (6.21) can be controlled using Lemma 6.11 below that relies on a self-normalized deviation inequality, whose proof exactly follows from the proof of Fact 1 in Appendix A of [CGM+13].

Lemma 6.11. *For any arm k, if $U^j_k(t)$ is the* kl-UCB *index with exploration function $f(t) = \ln(t) + 3\ln(\ln(t))$, then $\sum_{t=1}^{T}\mathbb{P}_{\boldsymbol{\mu}}\left(U^j_k(t) < \mu_k\right) \leq 3 + 4e\ln(\ln(T))$.*

The rightmost term in (6.21) can be controlled using Lemma 6.12, that is a direct consequence of the proof of Fact 2 in Appendix A of [CGM+13]. Denote $\mathrm{kl}'(x,y)$ the derivative of the function $x \mapsto \mathrm{kl}(x,y)$ (for any fixed $y \neq 0, 1$).

Lemma 6.12. *For any arms k and k' such that $\mu_{k'} > \mu_k$, if $U^j_k(t)$ is the kl-UCB index with exploration function $f(t)$,*

$$\sum_{t=1}^{T}\mathbb{P}_{\boldsymbol{\mu}}\left(A^j(t) = k, \mu_{k'} \leq U^j_k(t)\right) \leq \frac{f(T)}{\mathrm{kl}(\mu_k,\mu_{k'})} + \sqrt{2\pi}\sqrt{\frac{\mathrm{kl}'(\mu_k,\mu_{k'})^2}{\mathrm{kl}(\mu_k,\mu_{k'})^3}}\sqrt{f(T)} + 2\left(\frac{\mathrm{kl}'(\mu_k,\mu_{k'})}{\mathrm{kl}(\mu_k,\mu_{k'})}\right)^2 + 1.$$

Putting things together, one obtains the non-asymptotic upper bound

$$\begin{aligned}\mathbb{E}_{\mu}\left[N^j_k(T)\right] \leq{}& \frac{\ln(T) + 3\ln(\ln(T))}{\mathrm{kl}(\mu_k,\mu_{M^*})} + \sqrt{2\pi}\sqrt{\frac{\mathrm{kl}'(\mu_k,\mu_{M^*})^2}{\mathrm{kl}(\mu_k,\mu_{M^*})^3}}\sqrt{\ln(T) + 3\ln(\ln(T))} \\ &+ 2\left(\frac{\mathrm{kl}'(\mu_k,\mu_{M^*})}{\mathrm{kl}(\mu_k,\mu_{M^*})}\right)^2 + 4Me\ln(\ln(T)) + 3M + 1, \qquad (6.22)\end{aligned}$$

which yields Lemma 6.10, with explicit constants $C_{\boldsymbol{\mu}}$ and $D_{\boldsymbol{\mu}}$. □

As announced, Lemma 6.13 controls the number of switches of arm that are due to the current arm leaving $\widehat{M^j}(t)$, for both RandTopM and MCTopM. It essentially proves that

Lines 3-5 in Algorithm 6.2 (when a new arm is sampled from the non-empty subset of $\widehat{M^j}(t)$) happen a logarithmic number of times. The proof of this result is given below.

Lemma 6.13. *For any $\boldsymbol{\mu} \in \mathcal{P}_M$, any player $j \in [M]$ using* RandTopM- *or* MCTopM-kl-UCB, *and any arm k, it holds that*

$$\sum_{t=1}^{T} \mathbb{P}\left(A^j(t) = k, k \notin \widehat{M^j}(t)\right) = \left(\sum_{k', \mu_{k'} < \mu_k} \frac{1}{\mathrm{kl}(\mu_k, \mu_{k'})} + \sum_{k', \mu_{k'} > \mu_k} \frac{1}{\mathrm{kl}(\mu_{k'}, \mu_k)} \right) \ln(T) + o(\ln(T)).$$

Proof. We analyze the case when the current arm leaves the set $\widehat{M^j}$ (Line 4):

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbb{P}\left(A^j(t) = k, k \notin \hat{M}^j(t)\right) \\
&\leq \sum_{t=1}^{T} \mathbb{P}\left(A^j(t) = k, k \notin \hat{M}^j(t), A^j(t+1) \in \widehat{M^j}(t) \cap \{k' : U^j_{k'}(t-1) \leq U^j_k(t-1)\}\right) \\
&\leq \sum_{t=1}^{T} \sum_{k' \neq k} \mathbb{P}\left(A^j(t) = k, A^j(t+1) = k', U^j_{k'}(t) \geq U^j_k(t), U^j_{k'}(t-1) \leq U^j_k(t-1)\right) \\
&= \sum_{k' \neq k} \underbrace{\sum_{t=1}^{T} \mathbb{P}\left(A^j(t) = k, A^j(t+1) = k', U^j_{k'}(t) \geq U^j_k(t), U^j_{k'}(t-1) \leq U^j_k(t-1)\right)}_{\doteq N_{k'}}
\end{aligned}$$

Now, to control $N_{k'}$, we distinguish two cases. If $\mu_k < \mu_{k'}$, one can write

$$N_{k'} \leq \sum_{t=1}^{T} \mathbb{P}\left(U^j_{k'}(t) \leq \mu_{k'}\right) + \sum_{t=1}^{T} \mathbb{P}\left(A^j(t) = k, U^j_k(t-1) \geq \mu_{k'}\right)$$

The first sum is $o(\ln(T))$ by Lemma 6.11. To control the second sum, we apply the same trick that led to the proof of Lemma 6.12 in [CGM+13]. Letting $\mathrm{kl}^+(x,y) \doteq \mathrm{kl}(x,y)\mathbb{1}(x \geq y)$, and $\widehat{\mu}^j_{k,s}$ be the empirical mean of the s first observations from arm k by player j, one has

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbb{P}\left(A^j(t) = k, U^j_k(t-1) \geq \mu_{k'}\right) \\
&\quad = \mathbb{E}\left[\sum_{t=1}^{T} \sum_{s=1}^{t-1} \mathbb{1}\left(A^j(t) = k, N^j_k(t-1) = s\right) \mathbb{1}\left(s \times \mathrm{kl}^+\left(\widehat{\mu}^j_{k,s}, \mu_k\right) \leq f(t)\right)\right]
\end{aligned}$$

$$\leq \mathbb{E}\left[\sum_{s=1}^{T} \mathbb{1}\left(s \times \mathrm{kl}^+\left(\widehat{\mu}^j_{k,s}, \mu_k\right) \leq f(T)\right) \sum_{t=s-1}^{T} \mathbb{1}\left(A^j(t) = k, N^j_k(t-1) = s\right)\right]$$

$$\leq \sum_{s=1}^{T} \mathbb{P}\left(s \times \mathrm{kl}^{+}\left(\widehat{\mu}^{j}_{k,s}, \mu_k\right) \leq f(T)\right), \tag{6.23}$$

where the last inequality uses that for all s,

$$\sum_{t=s-1}^{T} \mathbb{1}\Big(A^j(t) = k, N^j_k(t-1) = s\Big) = \sum_{t=s-1}^{T} \mathbb{1}\Big(A^j(t) = k, N^j_k(t) = s+1\Big) \leq 1.$$

From (6.23), the same upper bound as that of Lemma 6.12 can be obtained using the tools from [CGM+13], which proves that for $T \to \infty$,

$$N_{k'} = \frac{\ln(T)}{\mathrm{kl}(\mu_k, \mu_{k'})} + o(\ln(T)).$$

If $\mu_k > \mu_{k'}$, we rather use that $N_{k'} \leq \sum_{t=1}^{T} \mathbb{P}\left(U^j_k(t) \leq \mu_k\right) + \sum_{t=1}^{T} \mathbb{P}\left(A^j(t+1) = k', U^j_{k'}(t) \geq \mu_k\right)$, and similarly Lemma 6.11 and a slight variant of Lemma 6.12 to deal with the modified time indices yields $N_{k'} = \frac{\ln(T)}{\mathrm{kl}(\mu_{k'}, \mu_k)} + o(\ln(T))$. Summing over k' yields the result. □

6.5.2 Regret analysis of MCTopM with kl-UCB indexes

For our proposal MCTopM, we are furthermore able to obtain a logarithmic regret upper bound, by proposing an original approach to control the number of collisions under this algorithm. First, we can bound the number of collisions by the number of collisions for players not yet "fixed on their arms" ($\overline{s^j(t)}$), that we can then bound by the number of changes of arms.

Lemma 6.14. *For any $\boldsymbol{\mu} \in \mathcal{P}_M$, if all players use the* MCTopM-kl-UCB *decentralized policy, and $M \leq K$, then the total average number of collisions (on all arms) is upper-bounded by*

$$\mathbb{E}_{\mu}\left[\sum_{k=1}^{K} \mathcal{C}_k(T)\right] \leq M^2 (2M+1) \left(\sum_{\substack{a,b=1,\dots,K \\ \mu_a < \mu_b}} \frac{1}{\mathrm{kl}(\mu_a, \mu_b)}\right) \ln(T) + o(\ln T). \tag{6.24}$$

Note that this bound is in $\mathcal{O}(M^3)$, which significantly improves the $\mathcal{O}\left(M\binom{2M-1}{M}\right)$ proven by [AMTA11] for RhoRand. It is worse than the $\mathcal{O}(M^2)$ proven by [RSS16] for Musical Chair. However, unlike Musical Chair, our algorithm does not need any prior knowledge on the problem complexity. It is indeed not very satisfying, from an applicative point of view, to require a prior knowledge of $\mu^*_M - \mu^*_{M+1}$ if one wants to run the Musical Chair algorithm.

Proof. A key feature of both the $\mathrm{RandTopM}$ and MCTopM algorithms is Lemma 6.13, that states that the probability of switching from some arm because this arm leaves $\widehat{M^j}(t)$ is small.

Figure 6.1 presented above provides a schematic representation of the execution of the MCTopM algorithm, that has to be exploited in order to properly control the number of collisions. The sketch of the proof is the following: by focusing only on collisions in the "not fixed" state, bounding the number of transitions (2) and (3) is enough. Then, we show that both the number of transitions (3) and (5) are small: as a consequence of Lemma 6.13, the average number of these transitions is $\mathcal{O}(\ln T)$. Finally, we use that the length of a sequence of consecutive transitions (2) is also small (on average smaller than M), and except for possibly the first one, starting a new sequence implies a previous transition (3) or (5) to arrive in the state "not fixed". This gives a logarithmic number of transitions (2) and (3), and so gives $\mathbb{E}_{\mu}[\sum_k \mathcal{C}_k(T)] = \mathcal{O}(\ln T)$, with explicit constants depending on $\boldsymbol{\mu}$ and M.

As in Algorithm 6.2, $s^j(t-1)$ is the event that player j decided to fix herself on an arm at the end of round $t-1$. Formally, $s^j(0)$ is false, and $s^j(t)$ is defined inductively from $s^j(t-1)$ as

$$s^j(t) = \left(s^j(t-1) \cup \left(\overline{s^j(t-1)} \cap \overline{C^j(t-1)}\right)\right) \cap \left(A^j(t) \in \widehat{M^j}(t-1)\right). \tag{6.25}$$

For the sake of clarity, we now explain Figure 6.1 in words. At step t, if player j is not fixed ($\overline{s^j(t-1)}$), she can have three behaviors when executing MCTopM. She keeps the same arm and goes to the other state $s^j(t)$ with transition (1), or she stays in state $\overline{s^j(t)}$, with two cases. Either she sampled $A^j(t)$ uniformly from $\widehat{M^j}(t) \cap \{m : U^j_m(t-1) \leq U^j_k(t-1)\}$ with transition (3), in case of collision and if $A^j(t-1) \in \widehat{M^j}(t)$, or she sampled $A^j(t)$ uniformly from $\widehat{M^j}(t)$ with transition (2), if $A^j(t-1) \notin \widehat{M^j}(t)$. In particular, note that if $\overline{C^j(t-1)}$, transition (3) is executed and not (2). Transition (3) is a uniform sampling from $\widehat{M^j}(t)$ (the "Musical Chair" step).

For player j and round t, we now introduce a few events that are useful in the proof. First, for every $x = 1,2,3,4,5$, we denote $I^j_x(t)$ the event that a transition of type (x) occurs for player j after the first t observations (*i.e.*, in round t, to decide $A^j(t)$). Formally they are defined by

$$I^j_1(t) \doteq \left(\overline{s^j(t-1)}, \overline{C_j(t-1)}, A^j(t-1) \in \widehat{M^j}(t)\right),$$
$$I^j_2(t) \doteq \left(\overline{s^j(t-1)}, C_j(t-1), A^j(t-1) \in \widehat{M^j}(t)\right), \quad \text{and} \quad I_3(t) \doteq \left(\overline{s^j(t-1)}, A^j(t-1) \notin \widehat{M^j}(t)\right),$$
$$I_3(t) \doteq \left(s^j(t-1), A^j(t-1) \in \widehat{M^j}(t)\right), \quad \text{and} \quad I_5(t) \doteq \left(s^j(t-1), A^j(t-1) \notin \widehat{M^j}(t)\right).$$

Then, we introduce $\widetilde{C^j}(t)$ as the event that a collision occurs for player j at round t if she is not yet fixed on her arm, that is $\widetilde{C^j}(t) \doteq \left(C^j(t), \overline{s^j(t)}\right)$.

A key observation is that $C^j(t)$ implies $\bigcup_{j'=1}^{M} \widetilde{C^{j'}}(t)$, as a collision necessarily involves at least one player $j' \in [K]$ not yet fixed on her arm ($\overline{s^{j'(t-1)}}$). Otherwise, if they are all fixed, *i.e.*, for all j', $s^{j'}(t-1)$, then by definition of $s^j(t-1)$, none of the player changed their arm from $t-1$ to t, and none experienced any collision at time $t-1$ so by induction there is no collision at time t. Thus, $\sum_{j=1}^{M}\mathbb{P}(C^j(t))$ can be upper bounded by $M\sum_{j=1}^{M}\mathbb{P}(\widetilde{C^j}(t))$ (union bound), and it follows that if $\mathcal{C}(T) \doteq \sum_{k=1}^{K}\mathcal{C}_k(T)$ then

$$\mathbb{E}_\mu[\mathcal{C}(T)] \leq M\sum_{j=1}^{M}\sum_{t=1}^{T}\mathbb{P}(\widetilde{C^j}(t)).$$

We can further observe that $\widetilde{C^j}(t)$ implies a transition (2) or (3), as a transition (1) cannot happen in case of collision. Thus another union bound gives

$$\sum_{t=1}^{T}\mathbb{P}(\widetilde{C^j}(t)) \leq \sum_{t=1}^{T}\mathbb{P}(I_2^j(t)) + \sum_{t=1}^{T}\mathbb{P}(I_3^j(t)). \tag{6.26}$$

In the rest of the proof we focus on bounding the number of transitions (2) and (3).

Let $N_x^j(T)$ be the random variable denoting the number of transitions of type (x). Neglecting the event $\overline{s^j(t-1)}$ for $x=3$ and $s^j(t)$ for $x=5$, one has

$$\mathbb{E}_\mu[N_x^j(t)] = \sum_{t=1}^{T}\mathbb{P}(I_x^j(t)) \leq \sum_{t=1}^{T}\mathbb{P}\left(A^j(t-1)\notin\widehat{M^j}(t)\right) \leq \sum_{t=1}^{T}\sum_{k=1}^{K}\mathbb{P}\left(A^j(t-1)=k, k\notin\widehat{M^j}(t)\right), \tag{6.27}$$

which is $\mathcal{O}(\ln T)$ (with known constants) by Lemma 6.13. In particular, this controls the second term in the right hand side of (6.26).

To control the first term $\sum_{t=1}^{T}\mathbb{P}(I_2^j(t))$ we introduce three sequences of random variables, the starting times $(\theta_i)_{i\geq 1}$ and the ending times $(\tau_i)_{i\geq 1}$ (possibly larger than T), of sequences during which $I_2(s)$ is true for all $s=\theta_i,\dots,\tau_i-1$ but not before and after, that is $\forall i\in\{1,\dots,n(T)\}, \overline{I_2^j(\theta_i-1)}\cap\bigcap_{t=\theta_i}^{\tau_i-1}I_2^j(t)\cap\overline{I_2^j(\tau_i)}$ with $n(T)$ the number of such sequences, *i.e.*, $n(T)\doteq\inf\{i\geq 1 : \min(\theta_i,\tau_i)\geq T\}$ (or 0 if θ_1 does not exist). If $\theta_i=1$, the first sequence does not have term $\overline{I_2^j(\theta_i-1)}$.

Now we can decompose the sum on $t=1,\dots,T$ with the use of consecutive sequences,

$$\mathbb{E}_\mu[N_2^j(t)] = \mathbb{E}_\mu\left[\sum_{t=1}^{T}\mathbb{1}\left(I_2^j(t)\right)\right] = \mathbb{E}_\mu\left[\sum_{i=1}^{n(T)}\left(\sum_{t=\theta_i}^{\tau_i-1}1+\sum_{t=\tau_i}^{\theta_{i+1}-1}0\right)\right] = \mathbb{E}_\mu\left[\sum_{i=1}^{n(T)}(\tau_i-\theta_i)\right].$$

Both $n(T)$ and $\tau_i-\theta_i\geq 0$ have finite averages for any i (as $\tau_i-\theta_i\leq T$), and $n(T)$ is a *stopping time* with respect to the past events (that is, $\mathcal{F}_T^j$), thus we can use Wald's Lemma [Wal45], to obtain a decomposition with two terms (α) and (β), $\mathbb{E}_\mu[N_2^j(t)]\leq\mathbb{E}_\mu\left[n(T)\right]\times\max_{i\in\mathbb{N}}\mathbb{E}_\mu\left[\tau_i-\theta_i\right]$.

(α) To control $\mathbb{E}_\mu[n(T)]$, we can observe that the number of sequences $n(T)$ is smaller than 1 plus the number of times when *a sequence begins* (1 plus because maybe the game starts in a sequence). And beginning a sequence at time θ_i implies $\overline{I_2^j(\theta_i - 1)} \cap I_2^j(\theta_i)$, which implies a transition of type (3) or (5) at time $\theta_i - 1$, as player j is in state "not fixed" at time θ_i (transitions (1) and (4) are impossible). As stated above, $\mathbb{E}_\mu[N_x^j(T)] = \mathcal{O}(\ln T)$ for both $x = 3$ and $x = 5$, and so $\mathbb{E}_\mu[n(T)] = \mathcal{O}(\ln T)$ also.

(β) To control $\mathbb{E}_\mu\left[\tau_i - \theta_i\right]$, a simple argument can be used. The union of events $\bigcup_{t=\theta_i}^{\tau_i - 1} I_2^j(t)$ implies $C^j(t)$ for $\tau_i - \theta_i$ consecutive times. The very structure of $\mathrm{RandTopM}$ gives that in this sequence of transitions (2), the successive collisions (*i.e.*, $C^j(t-1) \cap C^j(t)$) implies that each new arm $A^j(t)$ for $t \in \{\theta_i, \tau_i - 1\}$ is selected uniformly from $\widehat{M^j}(t)$, a set of size M with at least one available arm. Indeed, as there is $M-1$ other players, at time t *at least* one arm in $\widehat{M^j}(t)$ is not selected by any player $k' \neq k$, and so player j has *at least* a probability $1/M$ to select a free arm, which implies $\overline{C^j(t)}$, and so implies the end of the sequence. In other words, the average length of sequences of transitions (2), $\mathbb{E}_\mu\left[\tau_i - \theta_i\right]$, is bounded by the expected number of failed trial of a repeated Bernoulli experiment, with probability of success larger than $1/M$ (by the uniform choice of $A^j(t)$ in a set of size M with at least one available arm). We recognize the mean of a geometric random variable, of parameter $\lambda \geq 1/M$, and so $\mathbb{E}_\mu\left[\tau_i - \theta_i\right] = \frac{1}{\lambda} \leq \frac{1}{1/M} = M$.

This finishes the proof as $\mathbb{E}_\mu[N_2^j(T)] = \sum_{t=1}^T \mathbb{P}(I_2^j(t)) = \mathcal{O}(\ln T)$ and so $\sum_{t=1}^T \mathbb{P}(\widetilde{C^j}(t) \cap (A^j(t) = k)) = \mathcal{O}(\ln T)$ and finally $\mathbb{E}_\mu[\mathcal{C}(T)] = \sum_{k=1}^K \mathbb{E}_\mu[\mathcal{C}^k(T)] = \mathcal{O}(\ln T)$ also.

We can be more precise about the constants, all the previous arguments can be used successively:

$$\begin{aligned}
\mathbb{E}_\mu[\mathcal{C}(T)] &\leq M \sum_{j=1}^{M} \left(\sum_{t=1}^{T} \mathbb{P}(I_2^j(t)) + \sum_{t=1}^{T} \mathbb{P}(I_3^j(t)) \right) = M \left(\sum_{j=1}^{M} \mathbb{E}_\mu[N_2^j(T)] + \mathbb{E}_\mu[N_3^j(T)] \right) \qquad (6.28)\\
&\leq M^2 \left(\mathbb{E}_\mu[n(T)] \mathbb{E}_\mu[\theta_i - \tau_i] \right) + M^2 \mathbb{E}_\mu[N_3^1(T)]\\
&\\
&\leq M^2 (1 + \mathbb{E}_\mu[N_3^1(T)] + \mathbb{E}_\mu[N_5^1(T)]) M + M^2 \mathbb{E}_\mu[N_3^1(T)]\\
&\leq 2M^3 \mathbb{E}_\mu[N_3^1(T)] + o(\ln T) + M^2 \left(\sum_{\substack{a,b=1,\dots,K\\ \mu_a < \mu_b}} \frac{1}{\mathrm{kl}(\mu_a, \mu_b)} \right) \ln(T) + o(\ln T)\\
&\leq \left(2M^3 + M^2 \right) \left(\sum_{\substack{a,b=1,\dots,K\\ \mu_a < \mu_b}} \frac{1}{\mathrm{kl}(\mu_a, \mu_b)} \right) \ln(T) + o(\ln T) \,. \qquad (6.29)
\end{aligned}$$

Thus we prove the desired inequality, with explicit constants depending only on $\boldsymbol{\mu}$ and M.

$$\sum_{k=1}^{K} \mathbb{E}_\mu[\mathcal{C}^k(T)] = \mathbb{E}_\mu[\mathcal{C}(T)]$$

$$\leq M^2\,(2M+1)\left(\sum_{\substack{a,b=1,\dots,K\\ \mu_a<\mu_b}} \frac{1}{\mathrm{kl}(\mu_a,\mu_b)}\right)\ln(T) + o(\ln T)\,. \tag{6.30}$$

□

Logarithmic switching cost for MCTopM**.** While Lemma 6.14 bounds the number of collisions, another consequence of our proof is that we have also bounded the (expected) number of *switches of arms*, $\mathrm{SC}^{\mathcal{A}}(T) = \sum_{t=1}^{T-1} \mathbb{P}(A^j(t+1) \neq A^j(t))$. We controlled the total number of transitions (2), (3) and (5) (see Figure 6.1), which are the only transitions when a player can switch from arm k to arm $k' \neq k$. Thus, the total number of arm switches is also proven to be logarithmic, if all players uses the MCTopM-kl-UCB algorithm. This additional guarantee was never clearly stated for previous state-of-the-art works, like RhoRand. Even though minimizing the number of arms switching was not a goal, this guarantee is appealing, in particular for Cognitive Radio applications, where switching arms means re-configuring a radio hardware, an operation that costs energy. An algorithm guaranteeing a small number of switches is thus interesting, and for instance [DMNM16] studied the empirical impact of the number of hardware reconfigurations on the battery life of a dynamic end-devices. This work highlighted a tradeoff between minimizing regret and minimizing the number of switches of arms. On a more experimental note, it was shown in [DMNM16] that the previous state-of-the-art policy for multi-players bandits, RhoRand, could be tuned to run in batches, in order to reduce by a certain multiplicative factor its switching cost while only adding an additive factor on its regret. While such ideas can interest a practitioner, they does not change the asymptotic behavior of $\mathrm{SC}^{\mathcal{A}_1,\dots,\mathcal{A}_M}(T)$, which is $\Theta(\ln(T))$ for any efficient policy. We also note that introducing explicit *switching costs* in the regret, like it was done in previous works like [TRY17] for single-player bandits, could also be fruitful.

Logarithmic Regret for MCTopM**.** Now that the sub-optimal arms selections and the collisions are both proven to be at most logarithmic in Lemmas 6.10 and 6.14, it follows from our regret decomposition (Lemma 6.5) together with Lemma 6.9 that the regret of MCTopM-kl-UCB is logarithmic. More precisely, one obtains a finite-time problem-depend upper bound on the regret of this algorithm.

Theorem 6.15. *If all M players use* MCTopM-kl-UCB*, and $M \leq K$, then for any problem $\boldsymbol{\mu} \in \mathcal{P}_M$, there exists a problem dependent constant $G_{M,\boldsymbol{\mu}}$, such that the regret satisfies:*

$$R_T(\boldsymbol{\mu}, M, \mathcal{A}) \leq G_{M,\boldsymbol{\mu}} \ln(T) + o(\ln T)\,. \tag{6.31}$$

Moreover, the dependency of the constant regarding the number of players is $G_{M,\boldsymbol{\mu}} = \mathcal{O}(M^3)$.

Strong uniform efficiency. As soon as $R_T = \mathcal{O}(\ln T)$ for all problems, MCTopM is clearly proven to be uniformly efficient, as $\ln T$ is $o(T^\alpha)$ for any $\alpha \in (0,1)$. And as justified after Definition 6.7 (page 155), uniform efficiency and invariance under permutations of the users implies strong uniform efficiency, and so MCTopM satisfies Definition 6.7. This is a sanity check: the lower-bound of Theorem 6.8 indeed applies to our algorithm MCTopM, and finally this highlights that it is order-optimal for the regret, in the sense that it matches the lower-bound up-to a multiplicative constant, and optimal for the term (a).

6.5.3 Discussion on Selfish

The analysis of the Selfish algorithm is harder, but we obtained some understanding of the behavior of this algorithm, that seems to be doing surprisingly well in many contexts, as in our experiments with $K = 9$ arms and in extensive experiments not reported in this section. However, a disappointing result is that we found simple problems, usually with small number of arms, for which the algorithm may fail. For example with $M = 2$ or $M = 3$ players competing for $K = 3$ arms, with means $\boldsymbol{\mu} = [0.1, 0.5, 0.9]$, the histograms in Figure 6.2 suggest that with a small probability, the regret R_T of Selfish-kl-UCB can be very large.

In the Appendix E of our article [BK18a], we explained when such situations may happen, and we included a conjectured (constant, but small) lower bound on the probability that Selfish experience collision almost at every round. This result would then prevent Selfish from having a logarithmic regret. However, it is to be noted that the lower bound of Theorem 6.8 does not apply to the censored observation model (III) under which Selfish operates. The SIC-MMAB algorithm proposed in [BP18] answers the question we left open last year in [BK18a], of whether logarithmic regret is at all possible for the “no sensing” case (model (III)). They confirmed that Selfish-UCB can indeed suffer linear regret, and proposed an algorithm based on a “communication trick” to achieve logarithmic regret for this harder model (III).

6.6 Numerical simulations in presence of sensing

In all the numerical simulations, we focus on the model with sensing, *i.e.*, model (II) or (III). We illustrate here the empirical performances of the algorithms presented in Section 6.4, used in combination with the UCB or kl-UCB indices. The analysis given in Section 6.5 was focussing on kl-UCB, because it is well known that it is more efficient both in practice and in theory. To illustrate this, we first show below the result of some experiments comparing different algorithms that use either the UCB or the kl-UCB indexes. Our proposed algorithms, MCTopM, RandTopM and Selfish are benchmarked against the state-of-the-art RhoRand algorithm. We also include a (unrealistic) centralized multiple-play kl-UCB algorithm, as defined by [AVW87a], essentially to check that the *“price of decentralized learning”* is not too large.

First experiment: UCB **vs** kl-UCB**.** The first experiment is considering $K = 9$ arms, with means $\boldsymbol{\mu} = [0.1, 0.2, \dots, 0.9]$, and $M = 3$ then 6 then 9 players. Note that here and as in all this section we consider different algorithms that know the number of players M (see Section 6.7.1 below for a discussion on the case when M could be unknown). Performance is measured with the *expected* regret up to horizon $T = 10000$, estimated based on 1000 repetitions on the same bandit instance.

Algorithm	**Index policy**	$M = 3$ players	$M = 6$ players	$M = 9$ players
Centralized multiple-play	UCB	321 ± 30	233 ± 25	0
	kl-UCB	94 ± 17	68 ± 18	0
Selfish	UCB	1263 ± 157	3694 ± 387	12420 ± 404
	kl-UCB	243 ± 31	743 ± 113	3005 ± 492
RhoRand	UCB	1455 ± 208	4775 ± 463	11794 ± 1083
	kl-UCB	394 ± 96	2385 ± 412	7057 ± 1053
RandTopM	UCB	1020 ± 92	2899 ± 418	470 ± 346
	kl-UCB	258 ± 43	902 ± 234	551 ± 520
MCTopM	UCB	980 ± 76	1466 ± 132	43 ± 13
	kl-UCB	$\mathbf{248 \pm 40}$	$\mathbf{410 \pm 54}$	$\mathbf{42 \pm 10}$

Table 6.1 – No matter the orthogonalization policy and M the number of players, using kl-UCB is much more efficient than using UCB, for multi-players bandit (here in a simple problem with $K = 9$ arms).

We report in Table 6.1 above the results in terms of mean regret, plus or minus one standard deviation. The conclusions are three fold: first we observe for any of the compared algorithms, using kl-UCB is always (much) better than using UCB. Second we observe that our proposal MCTopM is outperforming the state-of-the-art RhoRand policy, and performs closely to the centralized (unrealistic) approach. Third we observe in the last column, when $M = K = 9$, that MCTopM is achieving constant regret, as we proved that the regret upper-bound of Theorem 6.15 is indeed $\mathcal{O}(1)$ if there is no arms in M-worst, and it shows that the orthogonalization scheme proposed for MCTopM is very efficient, especially in comparison to previous state-of-the-art approaches using ranks. Additional experiments and illustrations are given below. We illustrate below the (empirical mean) regret R_t as a function of time and not only results in terms of empirical mean regret R_T at the end of the bandit game.

The purpose of this work is not to optimize on the index policy, but rather propose new ways of using indices in a decentralized setting, and as we observed in Table 6.1 that using kl-UCB rather than UCB indices always yield better practical performance, from now on we only report results for kl-UCB.

Other experiments. We now present more results for two bandit instances. The first instance is a small problem with $K = 3$ arms and means $\boldsymbol{\mu} = [0.1, 0.5, 0.9]$, for the case of $M = 2$ players. It is used to illustrate that in some unlucky runs, Selfish can suffer a linear regret, as illustrated in Figure 6.2. The second instance considers $K = 9$ arms with means $\boldsymbol{\mu} = [0.1, 0.2, \dots, 0.9]$, three cases are presented: $M = 6$ in Figure 6.3, and for the two limit

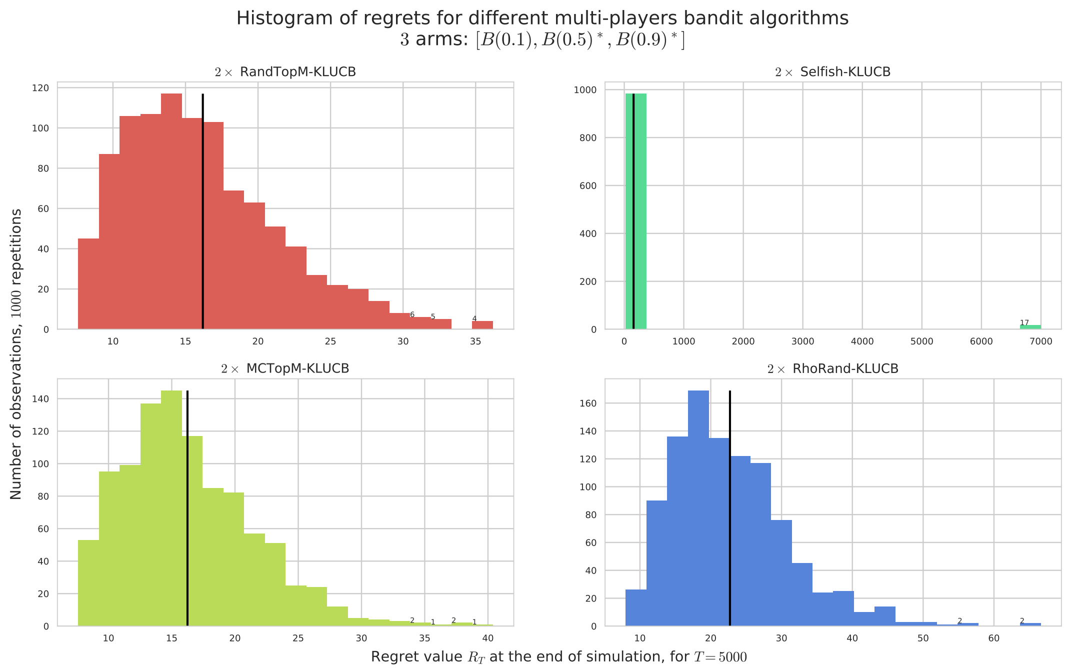


Figure 6.2 – Regret for $M = 2$ players, $K = 3$ arms, horizon $T = 5000$, 1000 repetitions and $\boldsymbol{\mu} = [0.1, 0.5, 0.9]$. Axis x is for regret (different scale for each part), and the green curve for Selfish shows a small probability of having a linear regret (17 cases of $R_T \geq T$, out of 1000). The regret for the three other algorithms is very small for this problem, always smaller than 100 here.

cases $M = 2$ and $M = 9 = K$ in Figure 6.4. We also include histograms showing the *distribution* of the final regret R_T, as this allows to check if the regret is indeed small for *each* run of the simulation. We also include our *asymptotic* lower bound from Theorem 6.8 on regret plots.

We also compared our algorithms to with MEGA [AM15] and Musical Chair [RSS16], in the presence of sensing, *i.e.*, observation model (II), for which they were developed. Yet these two algorithms were found hard to use efficiently in practice and we show in Figure 6.5 that they perform poorly in comparison to RhoRand, RandTopM and MCTopM. MEGA needs a careful tuning of *five* parameters (c, d, p_0, α and β) to attain reasonable performances. No good guideline for tuning them is provided and using *cross validation*, as suggested by [AM15], can be considered out of the scope of *online* sequential learning. Musical Chair consists of a random exploration phase of length T_0 after which the players quickly converge to orthogonal strategies targeting the M best arms. With probability at least $1 - \delta$, its regret is proven to be "constant" (of order $\ln(1/\delta)$). The theoretical minimal value for T_0 depends on δ, on the horizon T and on a lower bound ε on the gap $\Delta = \mu_M^* - \mu_{M+1}^*$, and the practical tuning is hard too.

Uniformly sampled problems. Experiments with a different problem for each repetition, that is uniformly sampled $\boldsymbol{\mu} \sim \mathcal{U}([0,1]^K)$, are also considered, in Figure 6.6 and 6.7. This helps to check that no matter the *complexity* of the considered problem (one measure of complexity being the constant in our lower bound), MCTopM performs similarly or better than all the

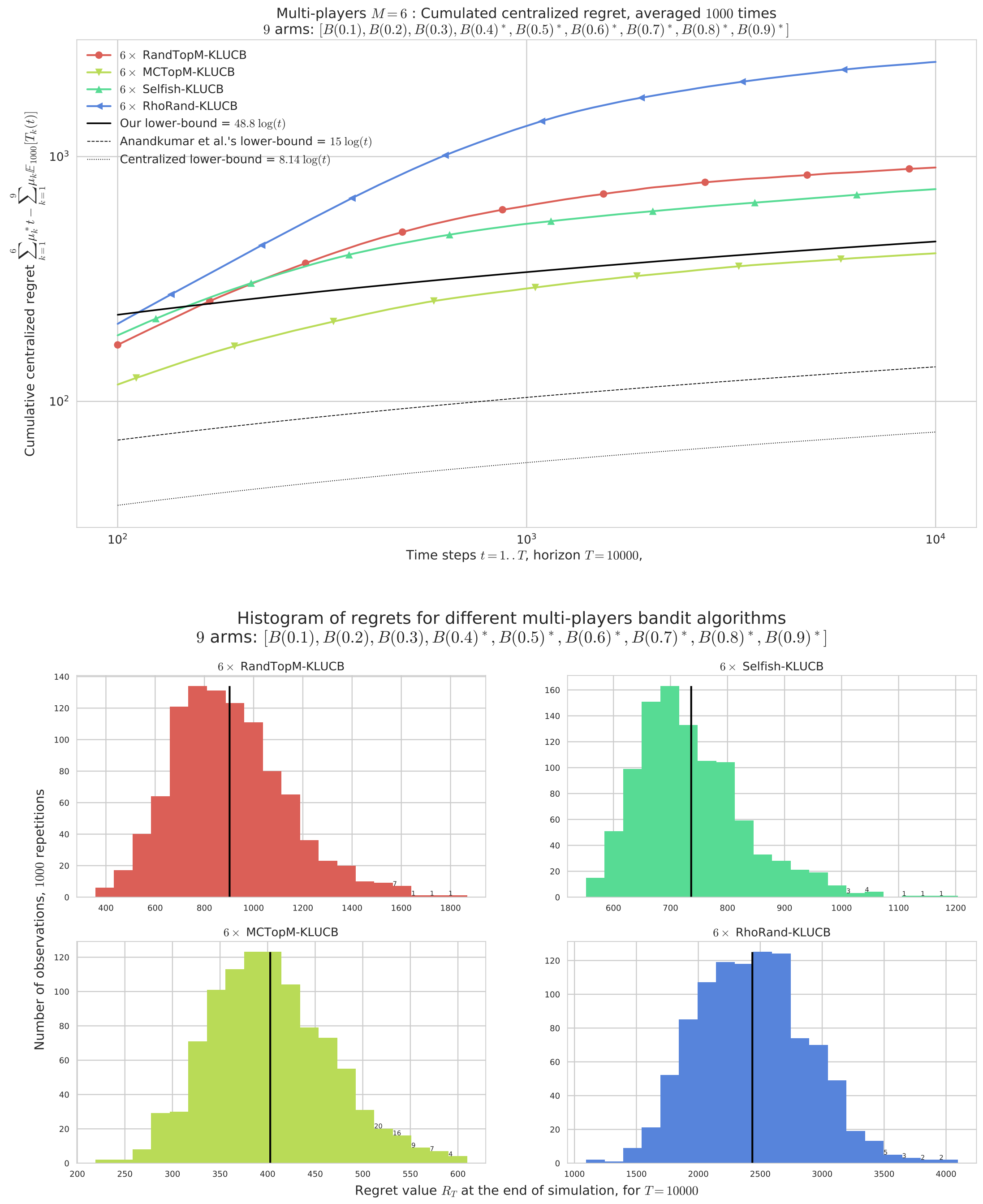
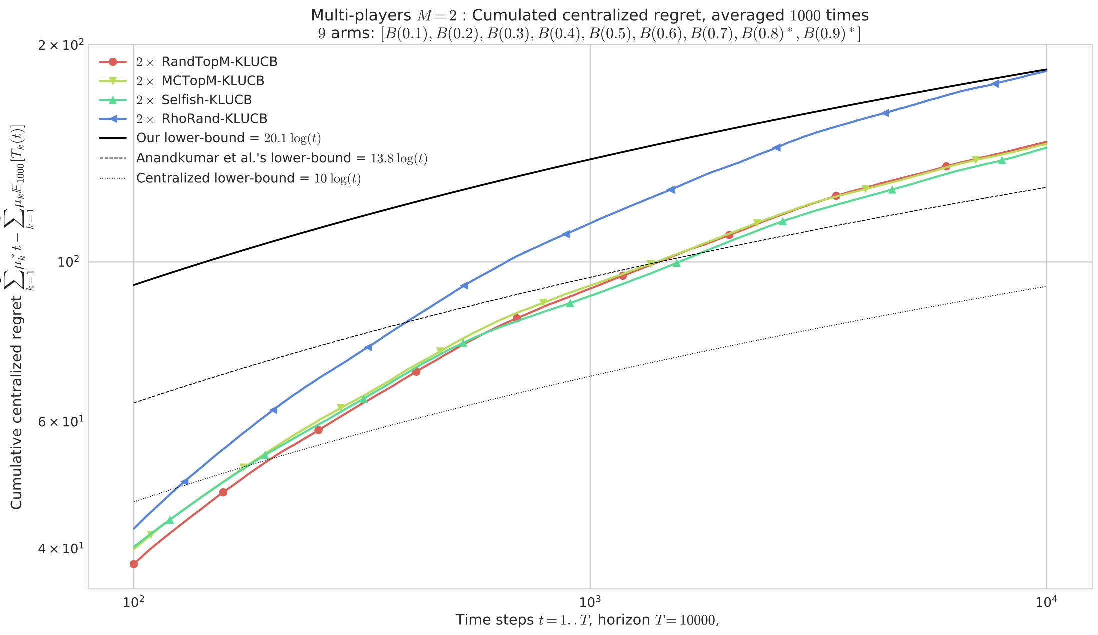


Figure 6.3 – Regret (in log-log scale), for $M = 6$ players for $K = 9$ arms, horizon $T = 5000$, for 1000 repetitions on problem $\boldsymbol{\mu} = [0.1, \dots, 0.9]$. $\mathrm{RandTopM}$ (yellow curve) outperforms $\mathrm{Selfish}$ (green), both clearly outperform $\mathrm{RhoRand}$. The regret of MCTopM is logarithmic, empirically with the same slope as the lower bound. The x axis on the regret histograms have different scale for each algorithm.



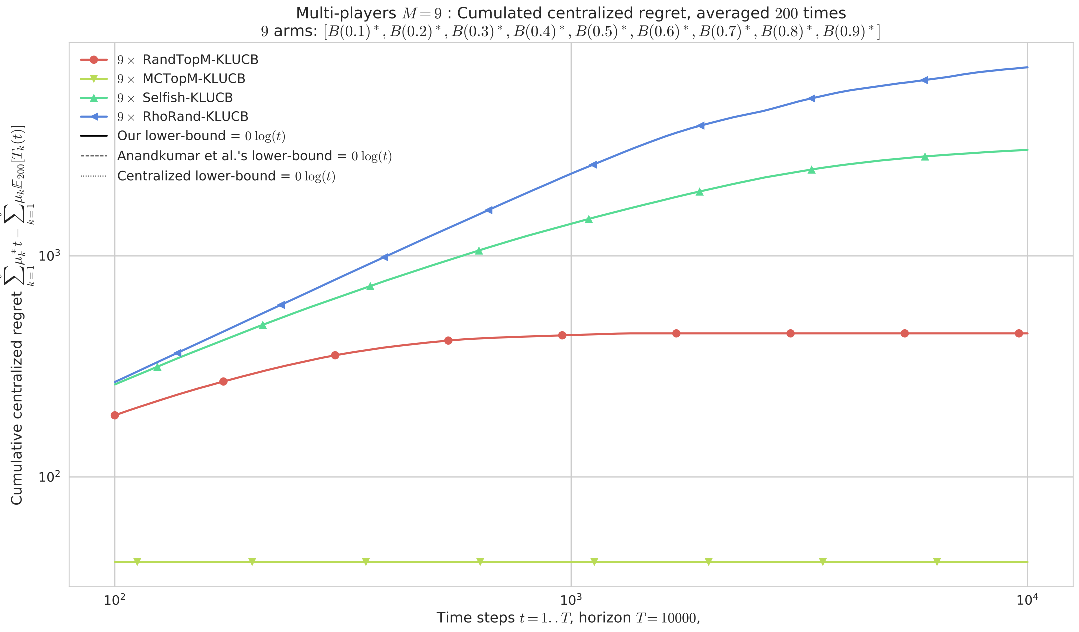
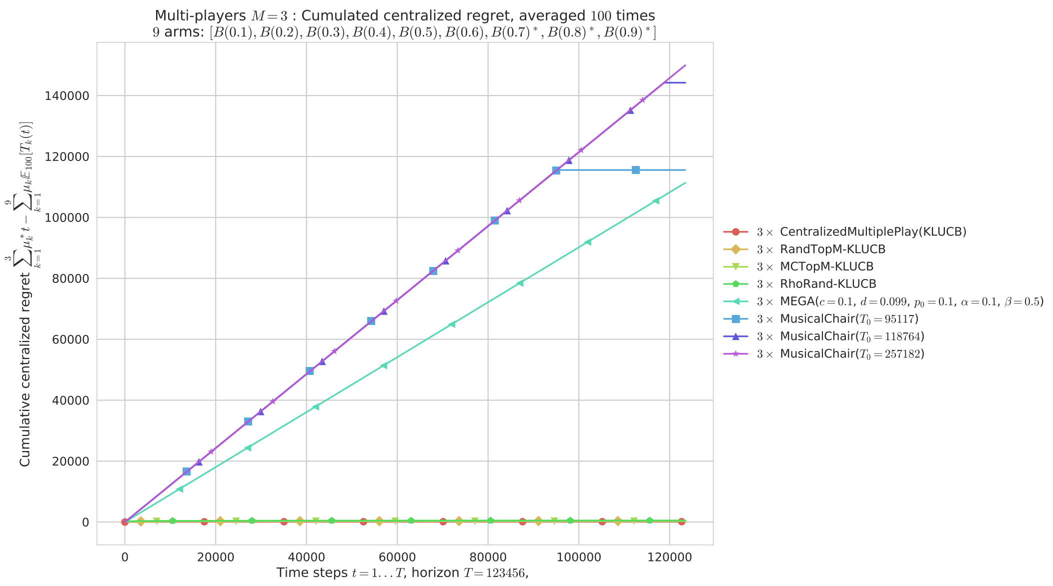


Figure 6.4 – Regret (in log-log scale), for $M = 2$ and 9 players for $K = 9$ arms, horizon $T = 5000$, for problem $\boldsymbol{\mu} = [0.1, \dots, 0.9]]$ for problem $\boldsymbol{\mu} = [0.1, \dots, 0.9]$. In different settings, RandTopM (yellow curve) and Selfish (green) can outperform each other, and always outperform RhoRand. MCTopM is always among the best algorithms, and for M not too small, its regret seems logarithmic with a constant matching the lower bound.



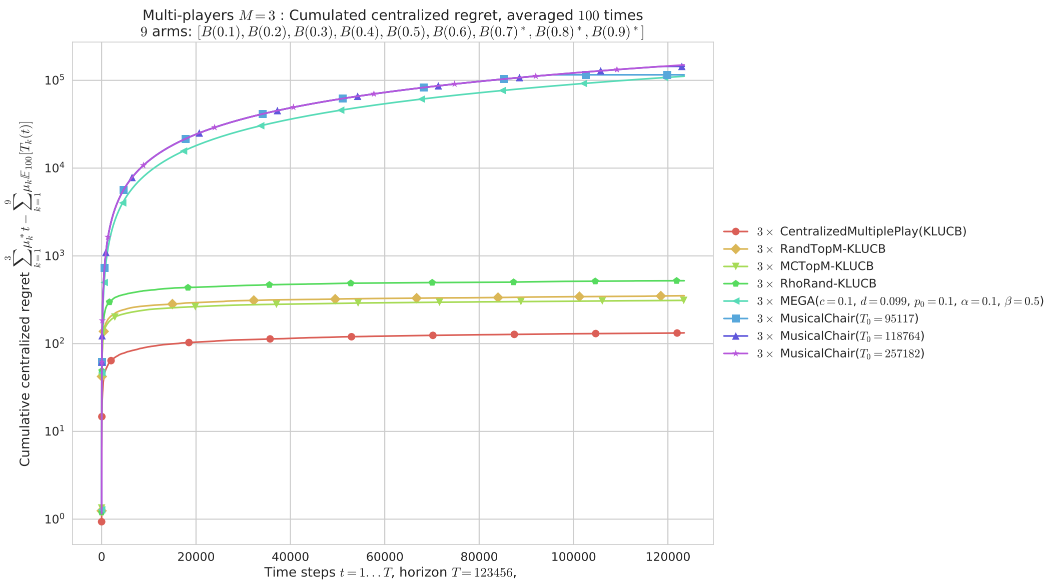


Figure 6.5 – Regret for $M = 3$ players for $K = 9$ arms, horizon $T = 123456$, for 100 repetitions on problem $\mu = [0.1, \dots, 0.9]$. With a perfect knowledge on the horizon and the gap ($\Delta = 0.1$ here) and by using the parameters suggested from their respective articles, MEGA and Musical Chair perform badly, even in this simple setting. The first two Musical Chair instances use the optimal T_0 value from [RSS16], with ε taken slightly smaller than the gap Δ ($\varepsilon = 0.99\Delta$), and respectively with $\delta = 0.5$ and $\delta = 0.1$, for which the regret can bounded with probability 0.5 and 0.9 respectively. The third instance uses the optimal T_0 corresponding to $\delta = 1/T$, that is guaranteed to have an expected regret of order $\ln(T)$. The log-y scale is used to easily differentiate between the different algorithms, and highlight that our proposal (in yellow ∇) outperform both MEGA and Musical Chair by three orders of magnitudes!

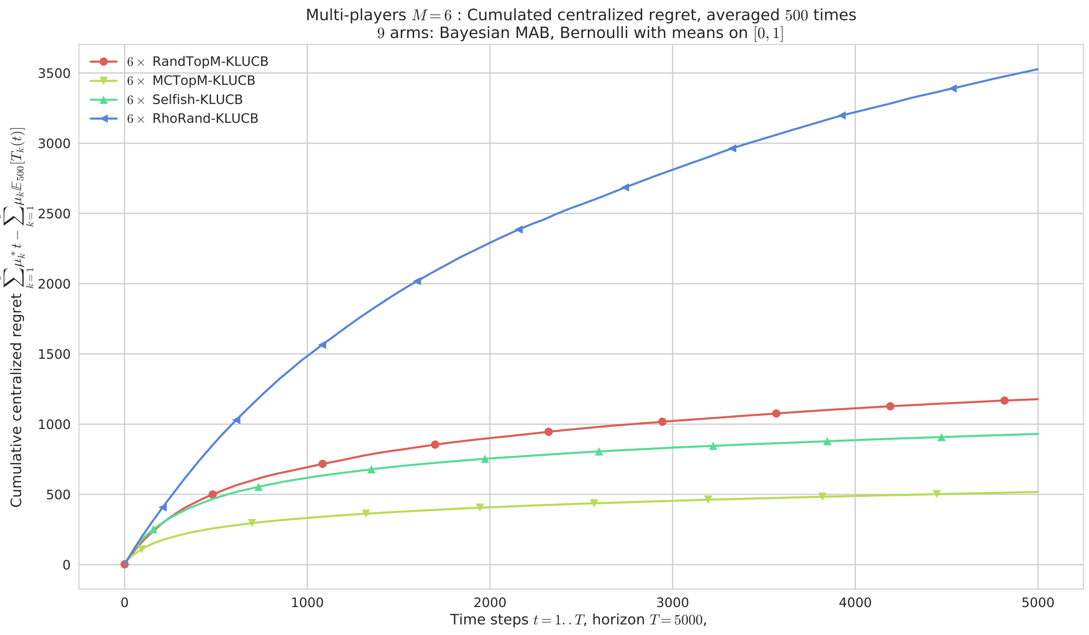


Figure 6.6 – Regret for $M = 6$ players, $K = 9$ arms, horizon $T = 5000$, against 500 problems $\boldsymbol{\mu}$ uniformly sampled in $[0,1]^K$. $\mathrm{RhoRand}$ (top blue curve) is outperformed by the other algorithms (and the gain increases with M). MCTopM (bottom yellow) outperforms all the other algorithms is most cases.

other algorithms, and $\mathrm{Selfish}$ outperforms $\mathrm{RhoRand}$ in most cases. Figure 6.6 is a good example of outstanding performances of MCTopM and $\mathrm{Selfish}$ in comparison to $\mathrm{RhoRand}$. Empirically, our proposals were found to always outperform $\mathrm{RhoRand}$, and except for $\mathrm{Selfish}$ that can fail badly on problems with small K, we verified that MCTopM outperforms the state-of-the-art algorithms in many different problems, and is more and more efficient as M and K grows.

Note that more numerical experiments were conducted for the article [BK18a], and in particular the last pages of the paper show more figures, in other settings.

6.7 Literature review of many extensions of the models

Before concluding this chapter, we review many extensions to the three models presented above in Section 6.2 that were introduced in recent literature. We wanted to highlight that the community has been quite active on research on multi-players bandits in the last 10 years, but especially active since last spring 2018. For instance, our article [BK18a] was the first to propose the "no sensing" case, and it was studied by (at least) two independent group of researchers since its publication in April 2018 [BP18, LM18]. Another example is the model

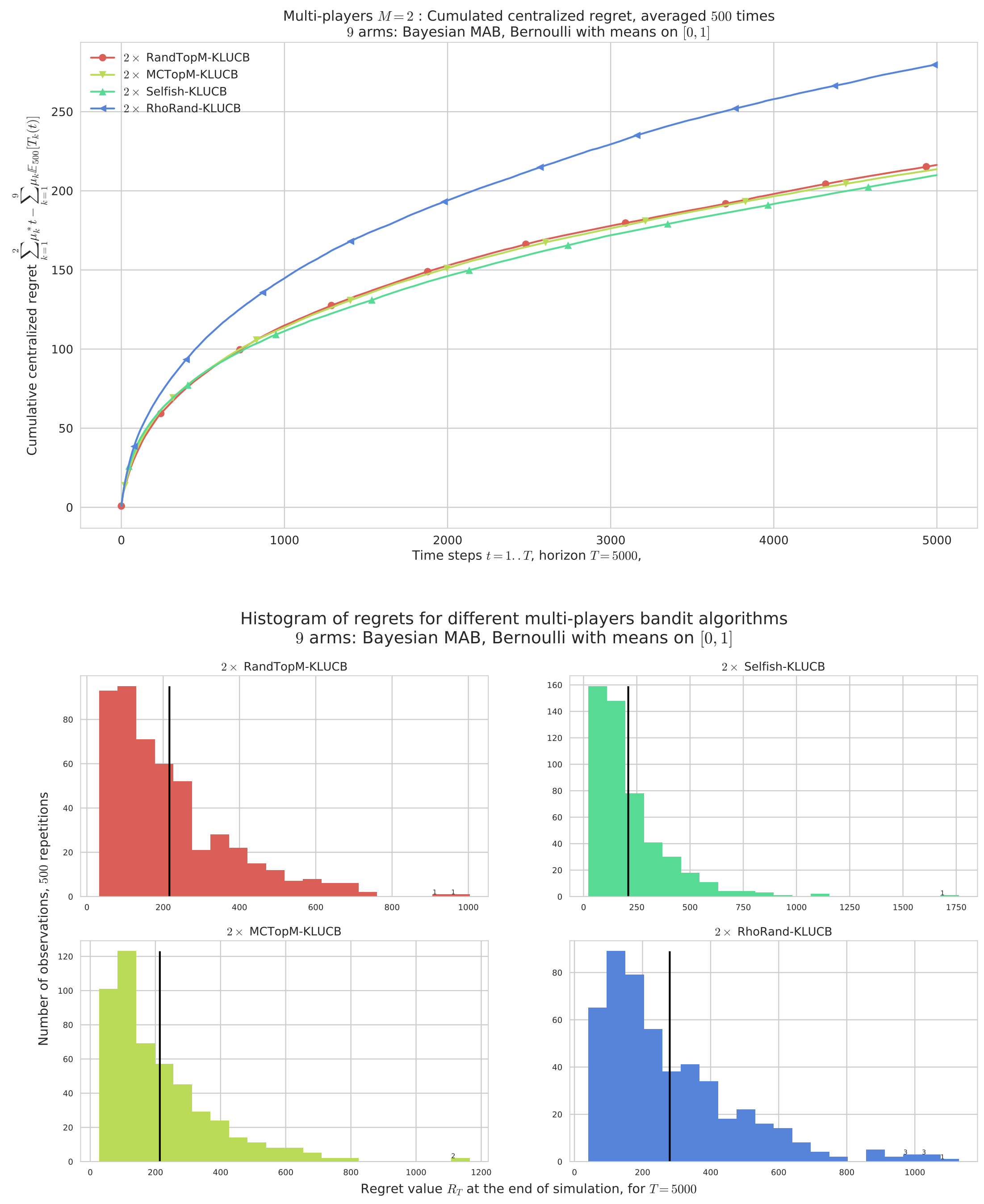
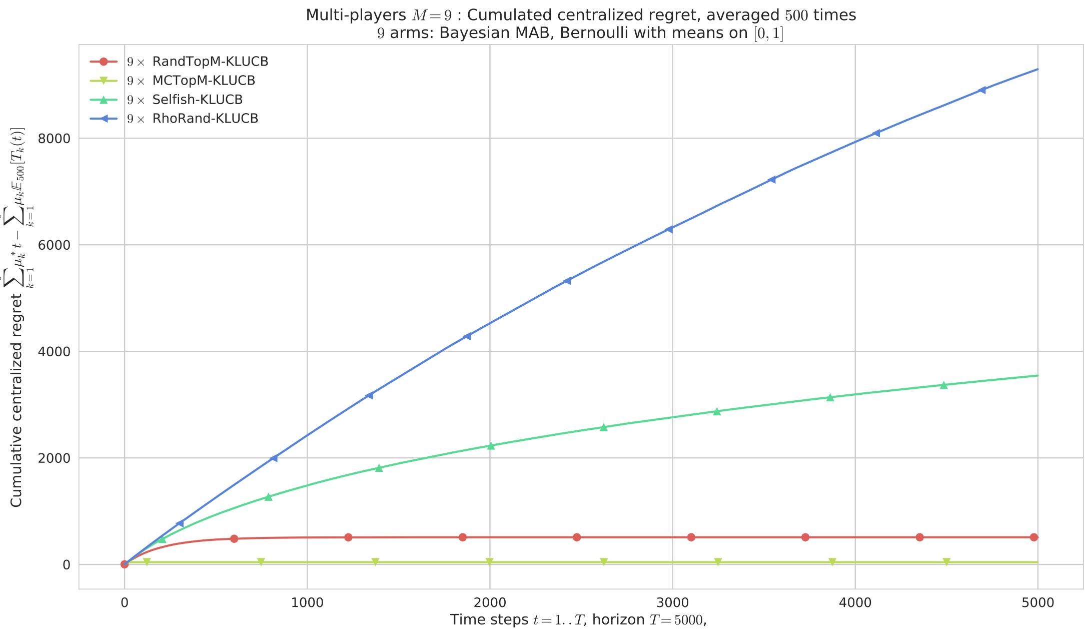


Figure 6.7 – Regret for $M = 2$ players, $K = 9$ arms, horizon $T = 5000$, against 500 problems $\boldsymbol{\mu}$ uniformly sampled in $[0,1]^K$. RhoRand (top blue) is outperformed by the other algorithms (and the gain increases when M increases), which all perform similarly in such configurations. Note that the (small) tail of the histograms come from complicated problems $\boldsymbol{\mu}$ and not failure cases.

with different arm means among players, who was initially studied in [AMT10, KNJ12] and then more recently in [BL18, KM19].



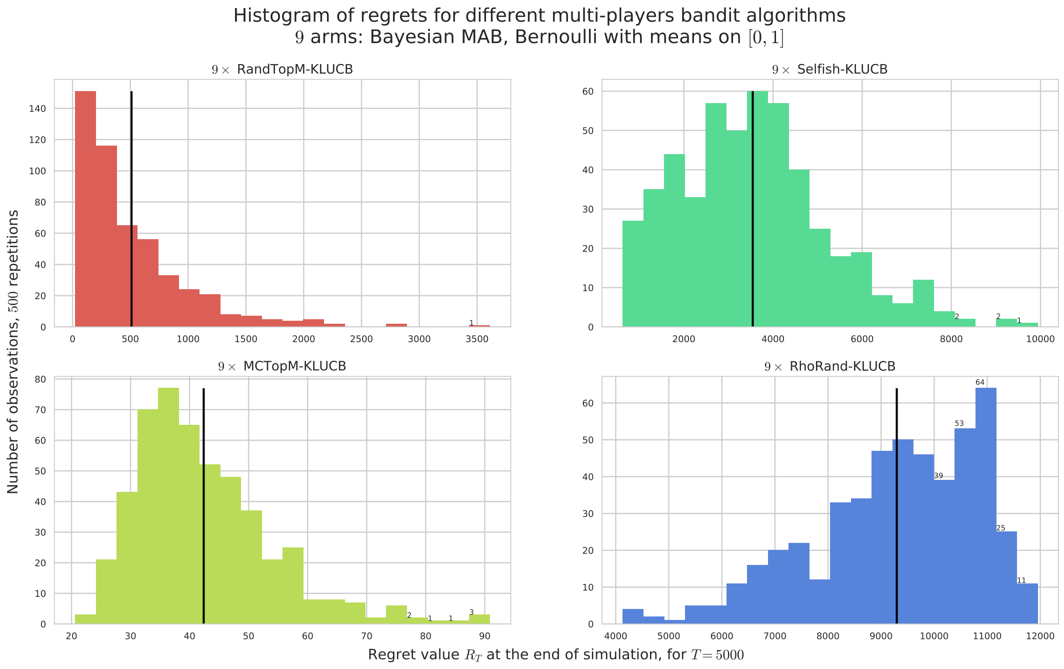


Figure 6.8 – Regret for $M = K = 9$, horizon $T = 5000$, against 500 problems $\boldsymbol{\mu}$ uniformly sampled in $[0,1]^K$. This extreme case $M = K$ shows the drastic difference of behavior between $\mathrm{RandTopM}$ and MCTopM, having constant regret, and $\mathrm{RhoRand}$ and $\mathrm{Selfish}$, having large regret.

Many extensions of the simpler model of multi-players bandits have been considered. The number of players M can be fixed but initially unknown to the players, the sensing information can be absent (like the model (III) presented above), there could be communications between players (happening at each time step or only occasionally), or M could evolve over time to model the possibility of arrivals or departures of players (*i.e.*, connexions and disconnections of devices in a wireless networks). Also inspired by real-world wireless networks, the arm means may vary among players, for instance to model players located at different distance of the gateway, and we can also consider networks with some "jammers" whose purpose is not to communicate to the gateway in a collaborative way, like the M devices, but to interfere with the communications of the M devices.

Finally, we can also be interested by models where M stays fixed, but the environment evolves, for instance with abrupt changes in the means of arms. This last model makes a good connexion between this chapter and the next one, and an exciting future work is to further study this model in order to tackle this kind of problems by merging our contributions for stationary multi-players bandits and for non-stationary single-player bandits.

6.7.1 Unknown (fixed) number of players

In the model we presented above, we assumed the number of players M to be fixed in time during the learning process. Without removing this hypothesis (we study this case below in Section 6.7.2), it is interesting to remark that we made no hypothesis on whether the players can know this value M or not. Our proposals, $\mathrm{RandTopM}$ and MCTopM, both assume to know M beforehand, like it was done for their main inspiration, $\mathrm{RhoRand}$.

Performance of our proposals for a wrong value of M. We can start by asking whether the most efficient algorithm, MCTopM-kl-UCB, is still efficient if it uses a value M' different than the real number of players M. Even though we did not include numerical simulations for this case in Section 6.6 above, we did some tests, which confirmed two disappointing results that were also proven analytically. First, the three algorithms ($\mathrm{RhoRand}$, $\mathrm{RandTopM}$ and MCTopM) give linear regret if they are using a value M' strictly smaller than M (and if $\mu^*_{M-1} > \mu^*_M$), as players will converge to play about $T - \mathcal{O}(\ln(T))$ times the $M-1$ best arms, leading to at least one collision most of the time, and thus a linear regret. Second, if the M players falsely use a value $M' > M$, then there is also a certain (fixed) probability to achieve a linear regret. Indeed, imagine one player ($M = 1$) running MCTopM with the false knowledge of $M' = 2$, and if it learned to accurately identify the set of the 2 best arms, then the MCTopM orthogonalization scheme can make it play the worst of the two arms, and as $M = 1$ this player will never encounter any collision, thus playing this suboptimal arm for about $T - \mathcal{O}(\ln(T))$ times, also leading to linear regret.

Interpretation of the hypothesis of knowning M for real-world networks. One could criticize our approach if we study a wireless networks with no central coordination from the gateway, in particular where the devices cannot be assigned to a channel or be assigned a unique ID by the gateway when they first log in the network. Then in such networks, if one wants to apply an algorithm like MCTopM-kl-UCB, the M devices need to know their number M, and have to receive it from the gateway, as it is the only part of this example of network which knowns M. It seems unrealistic to ask the gateway to send the fixed value M to each device, and not a unique ID to each device, for instance $\mathrm{id}^j \in [K]$.

If the M players each have a unique ID, then a simple explore-then-commit algorithm (see Section 2.4.1), running on top of a round-Robin phase can achieve order-optimal regret. If the players know the time horizon T, then they can fix a confidence level δ. For the first T_0 time steps, the M players will use a simple round-Robin game, using their (unique) ID to stay orthogonal: player j starts at arm j, then $j+1$, then cycle in $[K]$. They encounter no collision in these time steps, and then user j targets the j-th best arm among the set of M-best arm. If T_0 is large enough, all players have built the same estimate of the ranking of the arms (and not only correctly identified the set of M-best arms), with high probability (at least δ), and thus they will also encounter no collision and no regret from after time T_0. The mean regret of such approach is easily bounded by $R_T \leq MT_0 + (1-\delta)(T-T_0)$, so by using $\delta = 1 - 1/T$, $R_T = \mathcal{O}(T_0)$. By calibrating T_0 based on δ (which needs prior knowledge of T, see the proof we gave in Section 16) and the minimal gap Δ between M-best and M-worst, one can show, using similar arguments as used by [RSS16] for the Musical Chair policy, that with large probability a constant regret is obtained. A logarithmic regret can be obtained from the same bound, proving the order-optimality of this simple approach, if we assume that user j knows it is user number j. Without giving more details, we let the interested refer to what is explained for the algorithms presented in [DH18, H. 18, KDH+19]. These works assume a prior knowledge of Δ, or other measures of the difficulty of the problem, and as such we do not find them comparable with the approach chosen here.

Two ideas to estimate M. Some works studied the same model as our model (II) "with sensing", under the hypothesis that players do not know in the value of M. As illustrated above, if we consider algorithms building on the same ideas as RhoRand or MCTopM, it seems mandatory to first build an estimate of the value of M then run the initial algorithm that assumed a perfect knowledge of M. Two possible directions exist to estimate M on the fly.

The first idea comes from [AMTA11], and the intuition behind it is quite simple, even if the mathematical derivations are not. All players will build an estimate $\hat{M}^j(t)$ of the number of player, that start by $\hat{M}^j(0) = 1$. As soon as one collision is observed, a player knows that $M \geq 2$, so $\hat{M}^j(t+1) = 2$. Then, based on probabilistic computations on the expected number of collisions if there were m players, all following the same strategy (*e.g.*, RhoRand in the case of), and because the formula is simple to compute for different m, the authors proposed a statistical test of the hypothesis $M \leq \hat{M}^j(t)$ against $M > \hat{M}^j(t)$ which is expressed as a simple comparison of the current number of collisions (since last update of $\hat{M}^j(t)$) against a threshold. If a player observed "too many" collisions (that are unlikely to be caused by only $\hat{M}^j(t) - 1$ other players), then she increases her current estimate $\hat{M}^j(t+1) = \hat{M}^j(t) + 1$.

The second idea comes from [RSS16], and suppose to know beforehand both the horizon T and a certain measure of the difficulty of the problem (*i.e.*, a lower bound on the minimal gap between two means). If all the M players start to play for a long enough time T_0 uniformly at random among all the K arms, then the (expected) number of collisions observed $\mathbb{E}[\mathcal{C}^j_{T_0}]$ by any player j is a (relatively simple) function of M. By knowing K and T_0 and if all players use the same mechanism, then they can invert the formula to obtain the most likely estimate of M which explained the observations of $\mathcal{C}^j_{T_0}$ collisions. This second approach works empirically very fine, but it requires a fine tuning of the stopping time T_0. Rosenski et al. studies the performance of their algorithm with high-probability bounds [RSS16], and the tuning of T_0 they propose depends on prior knowledge on the problem. For this reason, we are not fond of this approach, as this hypothesis is quite unrealistic if one wants to apply this kind of algorithms for real-world wireless networks. We note that all the following works assume some sort of prior knowledge on the difficulty of the problem: [KDY$^+$17, KYDH18, SKHD18, H. 18, DH18, KDH$^+$19, TPHD19].

Extension of our proposals to learn the value of M. Similarly to what was proposed for the RhoEst algorithm in [AMTA11], we could have worked on proposing and analyzing an extension of our proposals that could efficiently learn the value of M the number of player. We implemented this RhoEst policy in our library SMPyBandits [Bes19], as well as this mechanism for RandTopM and MCTopM. Building from the theoretical analysis given for RhoEst in [AMTA11], and the analysis of MCTopM-kl-UCB, we believe it is possible to show that the aforementioned extension also achieves sub-linear regret without requiring players to know M in the beginning of the bandit game. More precisely, we believe that MCTopM-kl-UCB can still give order-optimal logarithmic regret if M players use it, and this extension of Theorem 6.15 is not included due to space constraints. Writing its proof and performing more simulations are left as possible future work.

Simulations. We consider a bandit problem with $K = 9$ arms, of means $0.1, \dots, 0.9$, and three different cases of $M = 3, 6, 9$ players, for 100 independent repetitions and an horizon of $T = 10000$. As before, we include the centralized multiple-play kl-UCB as an unachievably efficient baseline, the Selfish-kl-UCB algorithm as an heuristic which does not require to know M. Then we compare the RhoRand, RandTopM and MCTopM algorithms, using kl-UCB, which know M beforehand, with their extensions implementing the same algorithm as RhoEst, to estimate M on the fly.

Algorithms	**Hyp. on M**	$M = 3$ players	$M = 6$ players	$M = 9$ players
Centralized multiple-play	Known M	$\mathbf{92 \pm 20}$	$\mathbf{70 \pm 16}$	$\mathbf{0}$
Selfish	Don't need M	250 ± 34	735 ± 85	3010 ± 472
$\mathrm{RhoRand}$	Known M	417 ± 103	2481 ± 449	6639 ± 1035
	Estimate M	1422 ± 1051	9030 ± 1922	7264 ± 1009
$\mathrm{RandTopM}$	Known M	268 ± 45	941 ± 217	437 ± 367
	Estimate M	688 ± 614	4256 ± 2701	1155 ± 544
MCTopM	Known M	244 ± 39	401 ± 55	44 ± 13
	Estimate M	563 ± 546	1560 ± 1293	618 ± 29

Table 6.2 – Mean regret $\pm\, 1$ std-dev, for different algorithms on the same problem with $M = 3, 6, 9$, comparing algorithms which knows M against algorithms which estimate M on the fly. All use kl-UCB.

One can observe in Table 6.2 the empirical performances of different algorithms in an example problem, in each case of low, medium and maximum number of players ($M = 3, 6, 9$ for $K = 9$ arms). The three algorithms $\mathrm{RhoRand}$, $\mathrm{RandTopM}$ and MCTopM all suffer from similar increase on their regret when they have to estimate M on the fly (written "Estimate M" in red, in Table 6.2). Our proposals are still much more efficient than $\mathrm{RhoRand}$ when using the procedure to estimate M described from [AMTA11]. For the two non-extreme cases ($M = 3, 6$), the performance drop when having to estimate M is quite large, and consistent on the different algorithms. The extreme case of $M = K$ players is of highest interest, as MCTopM achieves a very small regret (proven to be $\mathcal{O}(1)$ as it is just the expected time of the orthogonalization process), while the same algorithm with unknown M suffers from a much larger regret. In this extreme case, $\mathrm{RhoRand}$ and its extension both perform very closely, as expected, and much worse than MCTopM. Additional simulations, for increasing time horizons T, confirmed the expected order-optimal regret of this extension of MCTopM-kl-UCB.

6.7.2 Arrival and departures of players: the "dynamic setting"

As reminded above, the model assumes that the number of players M remains fixed during all the bandit game. However, in real-world wireless networks, when players model communicating devices connected to a single gateway, devices can arrive or leave the network at any time. The existing previous work on multi-players bandit models are all motivated by possible applications to wireless networks, but most of them assume M to be fixed. The first work studying the relaxation of this hypothesis is [RSS16], where this case is called the "dynamic setting". If the arrival or departures of players is not random, but determined in advance while still being unknown to any player, the natural notion of regret is the following, where the expectation $\mathbb{E}[\bullet]$ is capturing the randomness in the sensing information, as well as in the players' decisions (*i.e.*, collisions):

$$R_T^{\mathrm{dyn}} \doteq \sum_{t=1}^{T} \sum_{k=1}^{M(t)} \mu_k^* - \mathbb{E}\left[\sum_{t=1}^{T} \sum_{j=1}^{M(t)} r^j(t) \right]. \tag{6.32}$$

In [RSS16], the authors explain that if the model allows arrival or departures of players at *any time step*, then the game is much harder, and sub-linear regret is most likely un-achievable, if we consider the natural extension of the definition of regret as in our model presented in Section 6.2 above. But this negative results depends on how arrival of players are modeled: if an arriving player has no prior memory of the previous observations, *i.e.*, if it model a new device, then if there is no restriction on the number or frequency of arrival or departures, we can most likely prove that sub-linear regret is not achievable in general. A simple but extreme example shows that regret has to be linear for any bandit strategy (in the observation model with sensing). Consider $K = 2$ Bernoulli-distributed arms of means $\mu_1 = \mu_2 = 1/2$, and one player is always active and play any bandit strategy (*e.g.*, MCTopM-kl-UCB). Every time step, another player is either arriving, without any knowledge of the problem (*e.g.*, it is a fresh and new IoT device). Then no matter the strategy of these new players, even if they all play MCTopM-kl-UCB for instance, if they play a uniformly efficient strategy there is a non-zero probability that player 1 suffers from a collision at each time step, thus resulting in linear centralized system regret.

The authors of [RSS16] also assume that the players all know a lower-bound L on the length of all the intervals during which arrival or departures of players are allowed. A naive idea, if L is large enough, is simply to restart the underlying algorithm every L time steps, in order to directly benefit from the theoretical guarantees of the "static setting" (where M stays constant). Unfortunately this idea yields a large regret if the length of "static" intervals L is too small. Applied to the Musical Chair algorithm, the authors in [RSS16] call this basic extension Dynamic Musical ChairUnder the simple hypothesis that the overall number of players entering and leaving the game is sub-linear in T (*i.e.*, a $o(T)$), they analyze the regret of Dynamic Musical Chair and prove it is also sub-linear (see Theorem 2 in Section 3.4). Under the same hypothesis, and if the lower-bound L is known before-hand and is large enough, then we could also apply the same idea as from [RSS16] to our approach. We believe that we could easily prove a sub-linear regret bound, if all players run MCTopM-kl-UCB, and if currently active players restart their memory of the past observations every L time steps. As the regret guarantee is stronger (*i.e.*, smaller regret upper-bound) for MCTopM than Musical Chair, we also believe that in this "slowly varying" dynamic setting, applying the idea of Dynamic Musical Chair from [RSS16] to MCTopM would also give a small regret upper-bound.

After being introduced in [RSS16], some more recent works studied the case of arrival or departures of players, usually referred to as "dynamic setting" or "dynamic case". For instance, [BP18] studies in Section 4 the first algorithm proposed for the dynamic case under the no-sensing model. They study the same notion of regret as the one proposed in [RSS16] and given above in (6.32), see Equation (10) in Section 4.2.1. With the assumption of a non-decreasing number of players, *i.e.*, if only arrivals of players are considered, they prove a regret upper-bound of the Dyn-MMAB algorithm in Theorem 3 . If all arriving players are using their Dyn-MMAB algorithm, the "dynamic" regret is bounded by $\mathcal{O}(M^2 K \ln(T)/\mu_M^*)$, if $M = M(T)$ is the total number of players involved in the problem.

6.7.3 Without sensing information

We presented in Section 6.2 the model (III), without sensing information. Players can only observe $Y_{k,t}$, the product of the *i.i.d.* random sensing information from arm k and of the no-collision indicator. This model seems harder than the model with sensing information, and even though the proposal Selfish works fine in simulations (in terms of average regret), we proved that it can yield linear regret. In our paper [BK18a] as well as in the previous sections, it was left as a future work to know if an algorithm can achieve sub-linear regret in this harder model without sensing information.

Inspired by [BK18a], Boursier and Perchet studied this question in the summer 2018 following its publication [BP18]. Their answer is that the model without sensing information is essentially not harder than the model with sensing information. Similarly, Lugosi and Mehrabian studied the same problem [LM18]. In both works, the authors detail algorithms that give a logarithmic regret if $M \leq K$ players all independently implement the proposed algorithm.

The "communication trick". The recent work [BP18] proposed an idea they called the "communication trick". It essentially allow any player to send one bit to all the others, at some pre-agreed time steps, with no modification on the model. Thanks to this "communication trick", recent research efforts have been more focus on studying the basic model –without explicit communications between players–, while proposing algorithms that can rely on some communication between players. This trick essentially use the fact that the players share a synchronized time, thus they can use a collision as a way to directly exchange one bit of information between two players. This process is slow and not efficient, as all but two players must do nothing when player i is sending a bit to player j, but it does build a communication protocol in the model without explicit communication between players. This "communication trick" is used for instance in [KM19]. We let the interested reader refer to these last two works [BP18, KM19] as they are both solid, and well explained.

Estimating collisions through uniform exploration. [LM18] gives a first algorithm with expected regret of $\mathcal{O}(MK \ln(T)/\Delta^2)$, if $\Delta = \mu_M^* - \mu_{M+1}^* \neq 0$, achieving a better dependency regarding M when compared to our result (our bound is $\mathcal{O}(M^3)$) but at the cost of (much) larger constants hidden in the $\mathcal{O}$ notation, and a non-fully explicit algorithms which usually obtain bad (or worse) empirical performance. Then they study an interesting extension that works also if $\Delta = 0$ or if Δ is so close than the bound in $1/\Delta^2$ becomes useless for "small horizons". This behavior is well known for classical bandits, where bounds of the form $\ln(T)/\Delta^2$ become worse than a linear regret T if Δ is very small (*i.e.*, $\Delta \ll \sqrt{\ln(T)/T}$). For this extension, their proposed algorithm is proven to achieve $\mathcal{O}\Big(K^2 M (\ln(T))^2/\mu + KM \min(\sqrt{T \ln(T)}, \ln(T)/\Delta')\Big)$ regret, if μ is a lower-bound on μ_M^* and $\Delta' = \max(\Delta, \min\{|\mu_M^* - \mu_i^*| : \mu_M^* > \mu_i^*\})$ (that both have to be known beforehand by the algorithm). The proposed algorithms in [LM18] are all based on the Musical Chair algorithm from [RSS16], and the curious reader should read for instance their Algorithm 2 (page 15) for more details.

The results presented in [LM18] are based on some hypotheses, for instance the tuning they propose for the length g of the uniform exploration phase in the beginning of their "Musical Chair"-like algorithm is based on a prior knowledge of the horizon T. In Section 4 they explain how to relax the assumptions of their results. For the same example, the "doubling trick" technique can be used to obtain a regret upper bound for an algorithm unaware of the value of T, within a constant multiplicative factor of the upper bound given for the algorithm aware of T (Section 4.1). Because the regret bound is of the form $\mathcal{O}\Big(\sqrt{T}\ln(T)\Big)$, a simple "doubling trick" of increasing horizons of lengths $T_i = 2^i$ works well, as proposed in [CBL06] and as studied in depth in our article [BK18b], and quickly presented in Appendix A. They also study in Section 4.3 an extension of the model for the case with more players than arms, but we do not give more details on this aspect. Their definition of a centralized regret for this case is an interesting and natural generalization of the definition, and they also proposed an algorithm achieving $\mathcal{O}(MK \ln(T) \exp(4M/K)/\Delta^2)$ regret in this case. Finally, they also proposed in Section 4.4 an extension of their algorithm to estimate the number of players M, which is analyzed as for the Musical Chair algorithm in [RSS16], and also achieve logarithmic regret. Note that it is significantly harder to estimate M without collision information, and Algorithm 3 in page 24 is quite complex. We did not implement it, and it would be interesting to run some numerical simulations in order to validate it empirically.

About Selfish-UCB **inefficiency.** It is also proven in Appendix A of [BP18] that Selfish-UCB has a linear regret, in the theoretical case, with a very neat argument from number theory (using Lindemann-Weierstrass theorem). As we conjectured, there is a gap between the theoretical result and its practical consequence, as the Theorem 4 they gave is only valid for real-valued number, and not for hardware-represented floating point number. Their proof is supporting what we illustrated in Figure 4 in [BK18a]. Their argument is essentially to prove that for Selfish-powered players using the simple UCB-indexes, with a probability p at time t (both independent from T), two players might have the same number of pulls and the same observed rewards for each arm. In that case, the two players would pull the same arms and thus collide for a long time, until they reach a "tie breaking point" where they could choose different arms thanks to a random tie breaking rule (*e.g.*, if two values of their UCB indexes are the same, the arg max is a uniform random choice among the two arms). They prove that it is unlikely to encounter any of these "tie breaking points", in theory if the UCB indexes are real-valued number (that can be rational or irrational).

Additionnal numerical simulations. To illustrate the difficulty of the "no sensing" case, we illustrate here the performances of different algorithms designed specifically for this setting. As above, we consider a bandit problem with $K = 9$ arms, of means $0.1, \dots, 0.9$, and three different cases of $M = 3, 6, 9$ players, for $N = 100$ independent repetitions, and horizon $T = 10000$. We include the Selfish-kl-UCB algorithm as an heuristic, as well as the Improved Musical Chair algorithm from [LM18] and SIC-MMAB from [BP18]. It is also interesting to add the centralized multiple-play kl-UCB is included as an unrealistic efficient baseline, as it does not use the sensing information but only the joint information (the reward) $r^j(t)$, because it directly affects the player in an orthogonal configuration and thus never encounters any collision (thus $r^j(t) = Y_{t,A^j(t)}$ for each j, t). For the two other algorithms, we use the advised tuning of their parameters: for SIC-MMAB, we used $T_0 = \lceil K \ln(T) \rceil$, for Improved-MC, we used $c = 1$ which gives $g = 235$, whereas in the paper the authors use $c = 128$ for their analysis. We found empirically no difference when using different values of the constant c or g, and unfortunately the regret of Improved-MC was always found to be linear[3].

We give in Table 6.3 the mean regret obtained for these different algorithms, and we observe a drastic difference between the unrealistic centralized algorithm, which achieves a very small regret, the heuristic Selfish-kl-UCB which achieve small mean regret (but is small to fail in theory and in some instances), and the two other algorithms. We were unable to find any bug in our implementation of Improved Musical Chair from [LM18] and all the different tuning of c (or g) explored gave the same disappointing result (linear regret). The SIC-MMAB algorithm performs better than the Selfish heuristic for the extreme case of $M = K$, but its large regret in this case shows that the orthogonalization protocol developed by [BP18] is not fast to converge (for illustration, for the same problem for $M = K = 9$ for the "sensing" case, MCTopM-kl-UCB achieves a mean regret of about 40, two orders of magnitude smaller!). Further empirical evaluation would be needed to fully understand the situation, and a first future work would be to either fix our implementation of the algorithm proposed by [LM18] or propose an efficient modification, and illustrate in some problems that it can indeed achieve sub-linear regret (maybe it does but only for large horizons, even if we did try larger values of T like up-to $T = 200000$).

[3]For instance Improved-MC obtained a mean regret 12150 for $T = 10000$, for $M = 3, K = 9$, but seeing one value for one horizon does not mean anything, and we also experimented with larger values of T and found a similar linear behavior.

Algorithms \ Number of players	$M = 3$	$M = 6$	$M = 9$
Centralized Multiple play kl-UCB	$\mathbf{91 \pm 21}$	$\mathbf{70 \pm 16}$	$\mathbf{0 \pm 0}$
Selfish-kl-UCB	249 ± 35	728 ± 94	2953 ± 501
Sic-MMAB	1705 ± 340	3915 ± 300	1713 ± 52
Improved Musical Chair	12149 ± 60	22341 ± 77	27462 ± 85

Table 6.3 – Comparison of the mean regret $\pm$ 1 std-dev, for different algorithms, on the same problem with $M = 3, 6, 9$ players, for the "no sensing" case. More work is needed on our implementation on Improved Musical Chair. The results on Sic-MMAB confirm the numerical experiments of [BP18].

6.7.4 With communication or coordination between players

In the models of IoT networks considered in this thesis, we assume that *the different IoT devices cannot communicate with each other*, and can only communicate with their gateway. This hypothesis is realistic, mostly for energy consumption and spectrum efficiency reasons. Of course, we can relax this hypothesis, and in practice in some families of wireless networks, communication between devices are possible. Note that this extension is *not* considering a graph of distributed agents all playing cooperatively to solve a unique bandit game, like it is studied in the "*graph bandit*" problem [Val16]. We are interested here by an extension of the model where players can send some bits to one or all the other players, at some or every time steps.

Clearly, the problem is easier by allowing communication, and it is quite immediate to see that the communication capacity between players must be limited otherwise the problem is already known and solved. Indeed, imagine that at each time step t, all the M players could share an unbounded number of bits with the other players, at no cost (even if this hypothesis is clearly unrealistic for wireless networks). Then they can share all their observations, and they can all run the same multiple-plays MAB algorithm [AVW87a], like for instance the extension of Thompson sampling for multiple-plays studied in [KHN15], or extensions of KL-UCB from [LKC16]. In this setting, a logarithmic regret is easily obtained, and the regret upper-bound of the two aforementioned algorithms asymptotically achieve the lower-bound from [AVW87a].

A more interesting extension is thus to limit the communication between players, either to a small number of bits or just one bit, at every or only some time steps. We identified that the state-of-the-art on this direction of research consists in the two very recent works [TZZ19] and [WHCW19]. Distributed pure exploration is studied in [TZZ19], where the proposed new lower bounds for the regret of any algorithm for the distributed best arm identification problem, under the fixed time or fixed confidence settings. The also propose effective algorithms that asymptotically match their lower-bounds (up-to logarithmic factors, in some cases). The second work studies distributed learning for (not necessarily stochastic) multi-armed bandits as well as linear bandits [TZZ19]. For a distributed K-armed bandit with M agents, they developed two protocols achieving near-optimal regret $\mathcal{O}(\sqrt{MKT \ln(T)})$, and requiring little communication cost, one is independent of the time horizon T and use $\mathcal{O}(M \ln(T))$ communications, and the other is independent of the number of arms K and use $\mathcal{O}(MK \ln(M))$ communications. Their model and algorithms fit in the distributed, decentralized framework we advertise in all this thesis, and this last work is very interesting.

Additionally, after the recent work [BP18] introduced the "communication trick", some recent research efforts have been more focus on studying the basic model –without explicit communications between players–, while proposing algorithms that can rely on some communication between players. This "communication trick" is used for instance in [KM19]. Due to space constraint, we let the interested reader refer to these last two works [BP18, KM19] as they are both solid, and well explained.

Another line of research is to consider models that are closer to realistic wireless communication networks, as it is done in [AM16, AM18]. Explaining in details their model and proposed solutions would be quite lengthy, and we rather prefer to let the interested reader refer to the later work [AM18]. We sum-up its contributions quickly. They address several aspects of the challenge of communication networks shared by many users simultaneously: learning unknown stochastic network characteristics, sharing resources with other users while keeping coordination overhead to a minimum. The solution they proposed combines Multi-Armed Bandit learning with a lightweight signalling-based coordination scheme, and ensures convergence to a stable allocation of resources. Their work considers single-user level algorithms for two scenarios: an unknown fixed number of users, and a dynamic number of users, both for different arms means for each user (players). Analytic performance guarantees, proving convergence to stable marriage configurations, are presented for both setups. The algorithms are based on a system-wide perspective, rather than focusing on single user welfare.

6.7.5 With different arm utilities among players

In the multi-players model presented in this chapter, we assume that the arm distributions are the same for all the players. For cognitive radio applications, where arms model channels and players are radio devices, it can be unrealistic to consider that two players, maybe located at different distances from the gateway or equipped with different hardwares, encounter the same mean quality when accessing the same channel. Motivated by this weakness, some researchers studied an interesting extension of the model of interest, considering multi-players MAB models with different arms distributions among players. Starting in 2012 by [KNJ12] where arms are Markov chain, this model was studied more actively recently, [DH18, BL18, KM19, TPHD19].

In such models, instead of considering K arms characterized by a vector of distributions, $(\nu_k)_{1\leq k\leq K}$, if there is M players we consider a *matrix of distributions*, $(\nu_k^j)_{1\leq k\leq K, 1\leq j\leq M}$. Two users j and j' can experience different utilities for the same arm k, *i.e.*, $\nu_k^j \neq \nu_k^{j'}$. The goal stays the same, each player wants to maximize its cumulated reward, with or without explicit communications between players, with or without sensing information, but always without central supervision. Like before, maximizing the rewards of each player simultaneously is maybe not possible. As soon as the matrix is not invariant under permutation of the users, the problem nature changes fundamentally: instead of finding an optimal orthogonal assignment of the M players to the $M\text{-best}$ arms, the goal of the system is now to reach an equilibrium position, also referred to as a stable marriage. Assignment are also called matching, and total (mean) reward of an assignment is its utility. Such equilibrium position means that the utility obtained by the M player cannot be increased by swapping two users. Indeed imagine just $M = 3$ players and $K = 3$ arms, and Bernoulli distributions of means $[0.1, 0.5, 0.5]$, $[0.1, 0.5, 0.5]$ and $[0.9, 0.5, 0.1]$ for player 1, 2 and 3. Then two optimal affectations of players to arms are $[3, 2, 1]$ or $[2, 3, 1]$, both giving the same utility of 1.9.

In this extension, performance is still evaluated by a system regret, now defined as the difference between the sum of the cumulated rewards by the M players, and the utility of any matching. The question is to know if it is possible to obtain a logarithmic –or even sub-linear– regret in this problem is more difficult that for the model presented in Section 6.2. This question was first answered in [BL18], and proposed an algorithm based on alternating three phases, and increasing their lengths after the end of each epoch (2^p for the p-th epoch). First, players explore in order to estimate the expectations of the arm rewards ; then players use their “Game of Thrones” dynamics (GoT, inspired by Musical Chair [RSS16]) and play the optimal solution most of the time ; finally players play the action they played most of the time in the recent GoT phases. They analyze their GoT algorithm and proven that it achieves a regret of $\mathcal{O}((\ln(T))^{2+\kappa})$ for any positive constant κ, as small as possible, if κ is known by the algorithm.

Until the very recent work of [KM19], it was unknown if a logarithmic regret was possible. They proposed an algorithm that is based on two phases: first, players will learn M and learn orthogonal ranks, using the “communication trick” of [BP18], and then one player is elected as a leader and the others are followers. This second step, after initialization, is also using epoch of increasing lengths 2^p. Until the optimal solution, each epoch is alternating three phases. Followers start by a Round-Robin uniform exploration with no collision, then they use the “communication trick” to communicate their samples to the leader, and finally they start an exploitation phase until the end of the game if the leader player tells them so. The leader can thus collect enough samples from all arms and all players, and is able to solve iteratively the stable marriage problem, and sends back the successive estimate of the solution to the players. In the easier case when their is a unique optimal matching, their algorithm Multiplayer Explore-Then-Commit (M-ETC) is the first one to achieve logarithmic regret, in the form of $R_T = \mathcal{O}(MK\ln(KT) + KM^3\ln(MKT)/\Delta) + KM^2(\ln(M\ln(KT)/\Delta))^2$, if Δ

is defined as the gap between the utility of the best matching and the utility of that of the matching with second best utility. In the generic case, their algorithm M-ETC with Elimination achieves a regret of $\mathcal{O}((\ln(T))^{1+\kappa})$ for any positive constant κ.

Finally, we note that previous works all focused on the "sensing" case, but most likely sub-linear regret can be achieve by decentralized algorithms that leverage the same techniques as introduced by [BP18, LM18] for the "no sensing case". We conclude by noting that this model was recently studied by two very recent other works, [DH18, TPHD19], who obtained results comparable to the results from the two works presented above. They both also study the case of dynamic settings, with arrival or departures of players, as presented in Section 6.7.2. Both articles [DH18, TPHD19] use the terminology of ad-hoc networks, and they compare empirically their proposal with some previous works, while [KM19] do not include numerical simulations, and while [BL18] illustrate the performance of their GoT algorithm on a simple example, they do not compare with other algorithms. It would be interesting to compare empirically all the different approaches. Another interesting directions are to study the possible extension of this model with different arm utilities by players to the non-stationary case, or real-world validation of such decentralized algorithms in real IoT networks.

6.7.6 Modeling more closely a real wireless network

We simply quote here three recent articles which proposed a similar approach of using decentralized reinforcement learning algorithms on the device-side on wireless networks, but proposed models closer to the reality of wireless networks. The first work is [NC17], which proposes to use a decentralized learning based on deep learning, in the "no sensing" case, but in a model where the users have to learn not only the channels to use for their uplink messages (only from the feedback through the acknowledgements, like in Section 5.2), but where they have to learn the whole "spectrum access actions". The work of [AM18] is discussed above for the case with communicating players, and their model is very interesting from the point-of-view of real-world wireless communication protocols.

Finally, the very recent work of [ZBLN19] is the first one to present experiments of reinforcement learning done on simulated LTE and 5G channels. The mathematics behind their model are actually quite close to the model, but they explain it in terms of OFDMA and Quality-of-Service (QoS). Their algorithm is essentially based on a pre-agreement of the M players, that will deterministically run an alternance of exploration phase, auction phase and exploitation phase, of (exponentially) increasing durations. By diving into the details of the modulation and giving explicitly the form of the uplink messages sent by the devices, the authors are able to set-up two different uplink packets, to efficiently perform the auction phase (see Figure 2). They prove a regret upper-bound of the order $R_T = \mathcal{O}(\ln(T))$, with no special care regarding the constants, but we can also note that their algorithm require a prior knowledge of $\Delta_{\min}$ a problem-dependent constant (Theorem 3).

6.7.7 Can MAB learning also be used to learn the rank for $\mathrm{RhoRand}$?

Similarly to our demonstration of MAB learning in an IoT network that we presented in the previous Chapter 5 in Section 5.3, we list here some related works, dating back from 2016, who proposed a similar approach for OSA. All the similar works that we are aware of also used USRP boards, and implemented the demonstration using either the Simulink and MATLAB softwares, or the GNU Radio software like we did [BBM18]. Indeed, the demonstrations and the works presented below are coming from our team experiments, and have been extended by the team of S. Darak at IIIT Delhi after his postdoctoral fellowship in our SCEE team. In [DNMP16, DMP16], the authors study the same model and focused on the $\mathrm{RhoRand}$ policy, which was the state-of-the-art back in 2016, combined with UCB or kl-UCB, as well as an extension using Bayes-UCB from [KCG12].

Their idea is to use the same skeleton as $\mathrm{RhoRand}$ [DNMP16], that is to assign a *rank* to each player, and make players change their rank after any collision. But instead of selecting a new rank uniformly at random among $[M]$, they proposed to use a “second-stage” learning policy, based on Bayes-UCB, to (try to) learn the best rank while still exploring each rank from time to time. This is a very natural idea: use a bandit algorithm to balance the exploration/exploitation aspect of rank selection, instead of a random hoping. This algorithm is named $\mathrm{RhoLearn}$, and it is implemented in SMPyBandits, where it can use Bayes-UCB, or any of the 65 or more bandit algorithms available in the library. Even if no theoretical guarantee backs up this idea of second-stage learning with Bayes-UCB, empirical simulations found that it can be efficient. We illustrate this in Table 6.4 below, where we consider the same problem as described above in Section 6.7.3 (Table 6.3). We include different $\mathrm{RhoLearn}$ algorithms, that uses kl-UCB, Bayes-UCB or Exp3 for the second-stage learning, and we include both the centralized multiple-play version of kl-UCB and MCTopM-kl-UCB for comparison.

Algorithm $\setminus$ Number of players	$M = 3$	$M = 6$	$M = 9$
Centralized multiple-play kl-UCB	$\mathbf{92 \pm 20}$	$\mathbf{70 \pm 16}$	$\mathbf{0}$
$\mathrm{RhoRand}$-kl-UCB	417 ± 103	2481 ± 449	6639 ± 1035
$\mathrm{RhoLearn}$-kl-UCB + kl-UCB	546 ± 190	1172 ± 295	1416 ± 347
$\mathrm{RhoLearn}$-kl-UCB + Bayes-UCB	561 ± 219	1204 ± 394	1363 ± 331
$\mathrm{RhoLearn}$-kl-UCB + Exp3	529 ± 175	1659 ± 331	5134 ± 976
MCTopM-kl-UCB	244 ± 39	401 ± 55	44 ± 13

Table 6.4 – Comparing $\mathrm{RhoRand}$ and $\mathrm{RhoLearn}$ on a simple MP-MAB problem with $K = 9$ arms.

6.7.8 With malicious jammers

In all this chapter and the previous research literature, another common hypothesis is that all the M players are pursuing the same goal, and are all behaving nicely by following the same algorithm. Distributed algorithms proposed in the literature aim to maximize the network throughput by ensuring orthogonal channel allocation for the SUs. However, these algorithms work under the assumption that all the SUs faithfully follow the algorithms which may not always hold due to the decentralized nature of the network. In the paper [SKHD18], the authors study for the first time distributed algorithms that are robust against malicious behavior, also called *jamming attack*. They consider both the cases of *jammers* launching coordinated and uncoordinated attacks, and consider a set of J jammers. In the coordinated attack, the jammers select non-overlapping channels to attack in each time slot and can significantly increase the number of collisions for SUs. They setup the problem in each scenario as a multi-players bandit and develop algorithm, and their analysis shows that when the SUs faithfully implement proposed algorithms, the regret is constant with high probability. They validate their claims through exhaustive synthetic experiments and also through a realistic USRP. Their synthetic experiments consider different cases, for different values of K the number of channels, M the number of players and J the number of jammers.

6.7.9 Towards non-stationary multi-players MAB models

Since the beginning of this thesis, all the studied bandit problems were stationary: the rewards coming from choosing arm k are *i.i.d.* and follow the same distribution ν_k. The next Chapter 7 is focused on the piece-wise stationary bandit model, a relaxation of this hypothesis.

When we were working on multi-players bandits in Autumn 2017 [BK18a], we left the study of the piece-wise stationary case as a future work, and shortly after we were excited to see that three independent works tackling this question were published, in December 2018 [WS18a] and in February 2019 [ALK19, BV19]. Without diving too much into the details, we review here these three works. Without a more serious analysis, we conjecture that it should not be difficult to merge the regret lower-bound for stationary *multi-players* MAB (Theorem 6.8) and the lower-bound for *piece-wise stationary* single-player MAB from [GM11]. We conjecture that any reasonable decentralized bandit algorithm must suffer a regret at least $\Omega(M\sqrt{K\Upsilon_T T})$ for $M \leq K$ players, K arms, and Υ_T stationary intervals.

On the one hand, piece-wise stationary multi-players MAB can be tackled by extending algorithms developed for the single-player case. In [WS18a], the authors study exactly the same multi-players bandit model as our model, and they propose two distributed algorithms that can efficiently be used by $M \leq K$ players to achieve sub-linear regret for piece-wise stationary problems, essentially by considering it as a harder case of a stationary problem.

The proposed the RR-SW-UCB# algorithm, which combines their SW-UCB# algorithm, previously proposed in [WS18b], and a Round-Robin hoping, under the assumption that each user $j \in [M]$ knows its ID j (we criticize this unrealistic assumption in Section 6.7.1 above). The SW-UCB# algorithm is discussed more in details in the literature review of Chapter 7 below. If Υ_T is bounded by $\Upsilon_T = \mathcal{O}(T^{\gamma})$, for a known γ but an unknown Υ_T, they prove for instance in Theorem 2 [WS18a] that the expected cumulative regret of their RR-SW-UCB# algorithm is bounded by $R_T = \mathcal{O}\Big(T^{\frac{1+\gamma}{2}} \ln(T)\Big)$ (what we call the centralized system regret). This bound is of the same order as the bound obtained by the same authors for the SW-UCB# algorithm in [WS18b] for the single-player case. If $\Upsilon_T = \mathcal{O}(T^{\gamma})$, this bound is comparable to the results obtained for most of the research literature on piece-wise stationary bandits (earlier works like D-UCB in [GM11] have the $\ln(T)$ outside the square root, while more recent works all improved this aspect and have the $\ln(T)$ in the square root, like for instance our algorithm GLR-klUCB in [BK19b] and presented in Chapter 7). Even if their analysis is not explicit regarding the constant, in the case where $\Upsilon_T = \mathcal{O}(T^{\gamma})$ for a known γ, their regret upper-bound actually matches the conjectured lower-bound, up to a logarithmic factor $\ln(T)$.

And finally, even though numerical simulations in their papers [WS18b, WS18a] are interesting and confirm the regret upper-bounds, they do not compare with any other algorithm, and it is left as an interesting future work to study this direction in more details. Sadly, a major drawback of their work is that assume that player $j \in [M]$ knows its index j, and as we explained above this small hypothesis is actually quite strong, as it allows players to be already orthogonal, and it reduces greatly the difficulty of the decentralized bandit problem.

On the other hand, another possibility is to extend ideas developed for the adversarial setting. In [ALK19], the authors essentially consider the piece-wise stationary as an easier case of an adversarial problem. The recent work [BV19] also proposes a decentralized algorithm that can achieve sub-linear regret ($\mathcal{O}(T^{3/4})$) under an adversarial multi-players bandit model.

In [ALK19], the authors build on the $\mathrm{Musical\,Chair}$ algorithm from [RSS16], to let the M players converge to a ranking in an efficient and decentralized way (cf. Algorithm 1), and they apparently discovered the "communication trick" independently from [BP18] to use (virtual) communications between players to set up a "coordinator" player and $M-1$ "followers" (running respectively their Algorithms 2 and 3). Their key algorithmic technique is to imitate the idealized case where there is full communication between the players. Then, to address the no-communication constraint, we enforce the players to keep the same decisions (arms) within long periods of time (blocks). This gives them the chance to coordinate between themselves via a simple protocol that uses collisions as a primitive, yet effective manner of communication.

Their proposal is using the Exp3 algorithm from [ACBFS02] on a combinatorial problem: they consider "meta" arms that are affectations of the M players to M distinct arms among the K arms. As soon as the "coordinator" can effectively use one algorithm to decide the arms played by all the M players, they show that the regret will grow as $R_T = \mathcal{O}\Big(M^{4/3}K^{2/3}(\ln(K))^{1/3}T^{2/3}\Big)$ as shown in Theorem 4.1 (they used K for M the number of players, and N for K the number of arms). This bound is much worse than the one obtained for the first article [WS18a], but is more general, and it is much worse than the bound of $\mathcal{O}\Big(\sqrt{KT\ln(K)}\Big)$ obtained for Exp3 for the single-player adversarial setting in [ACBFS02]. Note that this bound in $T^{2/3}$ is of the same order as the one given in [AM15], for the MEGA algorithm. However, we note that the dependency in M is surprisingly better for their result than for the regret upper-bound for MCTopM-kl-UCB (while scales in M^3 in Theorem 6.15).

Moreover, at first sight, this idea of "meta" arms implies an exponential blow up in terms of computational and storage cost, as there is $\binom{M}{K}$ such "meta arms", but the authors provide in Section 4.1 and Lemma 4.1 an interesting discussion regarding the time efficiency of their proposed algorithm, and they show it can stay polynomial in M and K by leveraging techniques from the Determinental Point Processes (DPP, see [GBV18] for a good review). In this second work also, we can note the poor experimental section, as the proposed algorithm is only compared against the Musical Chair algorithm, in "easy" problems (small number of breakpoints Υ_T). Despite being more complicated than other approaches, their algorithm "`C & P`" should not be too hard to implement by ourselves, and studying its empirical performance is an interesting future work.

6.8 Conclusion

To sum up, we presented in this chapter three variants of Multi-Players Multi-Arm Bandits, with different levels of feedback being available to the decentralized players, under which we proposed efficient algorithms. The two easiest models are the ones with sensing information (*i.e.*, for OSA), for which our theoretical contribution improves both the state-of-the-art upper and lower bounds on the regret. In the absence of sensing, we also provide some motivation for the practical use of the interesting Selfish heuristic, a simple index policy based on hybrid indices that are directly taking into account the collision information. We also reviewed various variants of this model, and for some interesting variants we discussed the related literature, which has proven to be very active in the last two years. For some models, we explained why our approach does not work efficiently without modifications, but we detailed and illustrated how to adapt MCTopM to other settings. For example, it assumes to know the number of players M before-hand, but we illustrated that previously introduced technique to estimate M can also be applied to our proposal and give satisfactory empirical performances. Further works would be required to adapt the theoretical analysis to these various extensions.

This chapter suggests several interesting further research directions. First, one could want to investigate the notion of *optimal algorithms* in the decentralized multi-players model with sensing information. So far we provided the first matching upper and lower bound on the expected number of sub-optimal arms selections, which suggests some form of (asymptotic) optimality. However, sub-optimal draws turn out to not be the dominant term in the regret, both in our upper bound and in practice, thus an interesting future work is to identify some notion of *minimal number of collisions*. Similarly to what was done very recently in [WHCW19], for communications between players who collaborate with each other, it would be interesting to characterize the (minimal) number of collisions needed to achieve logarithmic regret.

Additionally, we have implemented in SMPyBandits [Bes19] the following extensions: evaluating M on the fly (Section 6.7.1), and piece-wise stationary multi-players bandits (Section 6.7.9). We have not yet implemented the other extensions, but we are interested to do it, and it is one of the major future works left on our library. For more details, see the issue tickets at `GitHub.com/SMPyBandits/SMPyBandits/issues/`, for tickets number 120, 124, 185.

Reproducility of the experiments. The experiments in this chapter use our library SMPyBandits, as presented in Chapter 3, and we refer to the following page for details on how to reproduce them, `SMPyBandits.GitHub.io/MultiPlayers.html`

6.9 Appendix

Illustrating the lower bound

We proved in Theorem 6.8 that the normalized regret, *i.e.*, R_T divided by $\ln T$, is asymptotically lower bounded by a constant $C_{\boldsymbol{\mu},M}$ depending on the problem $\boldsymbol{\mu}$ and the number of players M, for any $\mathcal{A}$.

$$\liminf_{T\to+\infty} \frac{R_T(\boldsymbol{\mu}, M, \mathcal{A})}{\ln T} \geq C_{\boldsymbol{\mu},M}. \tag{6.33}$$

For an example problem with $K = 9$ arms, we display below on the x axis is the number of player, from 1 player to 9 players, and on the y axis is the value of this constant $C_{\boldsymbol{\mu},M}$, from the initial theorem and from our theorem. We chose a simple problem, with Bernoulli distributed arms, with $\boldsymbol{\mu} = [0.1, 0.2, \dots, 0.9]$. Figure 6.9 clearly shows that our improved lower bound is indeed larger than the initial one by [LZ10], and both become uninformative when $M = K$ (*i.e.*, it gives $R_T/\ln(T) \geq 0$ which is obvious anyway).

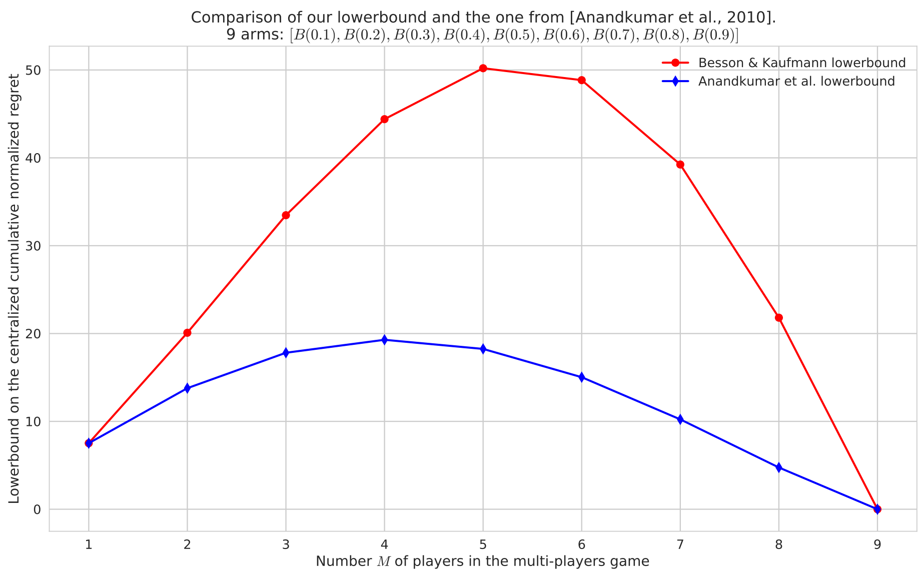
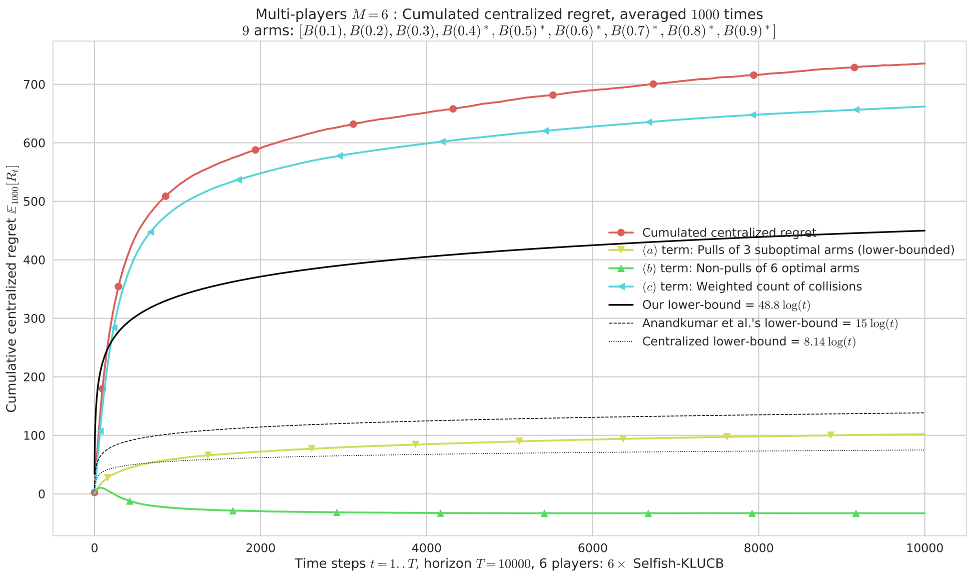


Figure 6.9 – Comparison of our lower bound against the one from [LZ10], on a simple problem with 9 Bernoulli arms, of means $\boldsymbol{\mu} = [0.1, 0.2, \dots, 0.9]$, as a function of the number of players M.

Illustrating the regret decomposition

The Figure 6.10 below shows the regret $R_T(\boldsymbol{\mu}, M, \mathcal{A})$ on the same example problem $\boldsymbol{\mu}$, with $K = 9$ arms and respectively $M = 6$, or 9 players, for Selfish-kl-UCB. It is just a simple way to check that the two lower bounds on the regret indeed appear as valid lower bounds empirically, and are moreover lower bounds on the count of selections ((a), displayed in cyan).

The lower bounds (in black) are $C(\boldsymbol{\mu}, M) \ln t$, the dashed line for the lower bound from [LZ10], and the continuous line is our lower bound. These plot show the regret (in red), and the three terms (a), (b), (c) in the decomposition of the regret. As explained in Lemma 6.5, term (b) is not always non-negative. For $M = 9$ and Selfish, (c) is actually larger than the regret, and term (a) is zero, as well as the lower bounds.



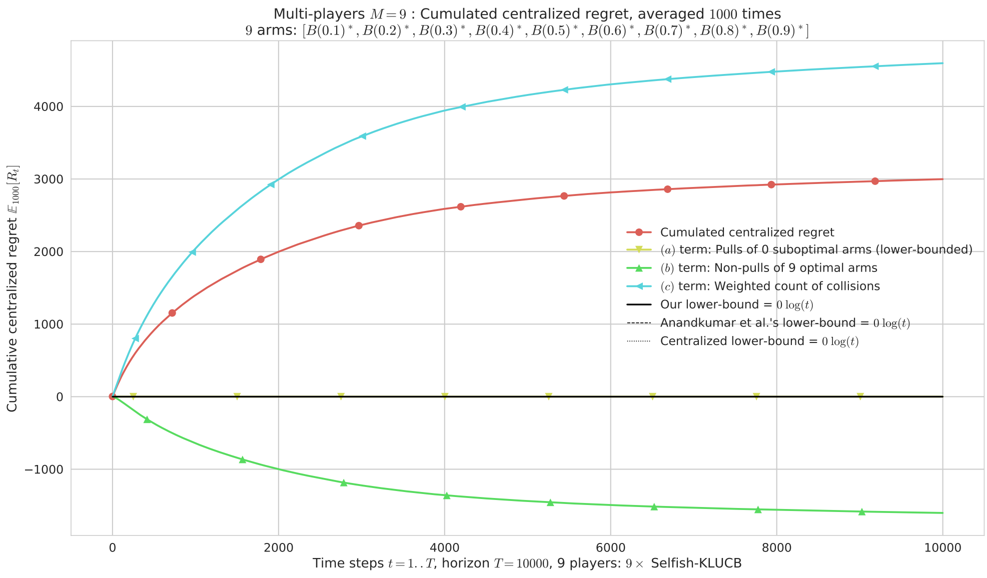


Figure 6.10 – The three terms (a), (b), (c) of the multi-players regret, and lower bounds (6.11) and (6.12) in **black**, for Selfish-kl-UCB: $M = 6$ and $M = 9$ players, $K = 9$ arms, horizon $T = 10000$ (for 1000 runs).

Chapter 7

Piece-Wise Stationary Multi-Armed Bandits

In this last chapter, we study a generalization of the stationary multi-armed bandit model, to account for possible non-stationarity of the rewards. We review existing works on piece-wise stationary MAB models. We study the Generalized Likelihood Ratio Test (GLRT) for bounded or sub-Bernoulli distributions. Our test enjoys finite-time guarantees of its *false alarm probability* and *detection delay*, and can be combined with an efficient bandit policy (kl-UCB), to propose an efficient algorithm for piece-wise stationary problems, GLR-klUCB. We analyze its regret, and show that it achieves state-of-the-art non-asymptotic regret bounds. Finally, we compare our approach with other state-of-the-art solutions on large numerical experiments.

Contents

7.1 Motivations for non stationary MAB models

For cognitive radio applications, as well as other applications such as recommender systems, the assumption that the arms distribution *do not change over time* may be a big limitation. Indeed, in cognitive radio or IoT networks, new devices can enter or leave the network, which impacts the availability of the radio channel they use to communicate, making it non stationary. And for instance in online recommendation, the popularity of items is also subject to trends. Hence, there has been some interest on how to take those *non-stationary* aspects into account within a multi-armed bandit model.

A first possibility to cope with non-stationary is to model the decision making problem as an *adversarial bandit problem* [ACBFS02]. Under this model, rewards are completely arbitrary and are not assumed to follow any probability distribution. For adversarial environments, the pseudo-regret, which compares the accumulated reward of a given strategy with that of the best fixed-arm policy, is often studied. The pseudo regret of the EXP3 algorithm has been shown to be $\mathcal{O}(\sqrt{KT})$, which matches the lower bound given by [ACBFS02]. However, this model is a bit too general for the considered applications, where reward distributions do not necessarily vary at every round. For these reasons, an intermediate model, called the *piece-wise stationary MAB*, has been introduced in Section 8 of the seminal paper [ACBFS02]. They propose a simple extension of the EXP3 policy, referred to as Exp3.S, that uses exponential weights like EXP3 tweaked with an adaptive forced exploration probability to passively adapt to changes. It was shown that Exp3.S attains a regret of $\mathcal{O}(\sqrt{KTS\ln(KT)})$, when comparing with an arbitrary sequence of comparators that does not switch for more than $S-1$ times. Running their algorithm over the non-stationary problem, and picking their comparators sequence as the best arm in each stationary interval, one can already get $\sqrt{\Upsilon_T T\ln(T)}$ regret, and the only prior knowledge needed to run the Exp3.S algorithm is T and Υ_T. Moreover, Exp3.S is efficient both in term of storage and computation time, and simple to implement.

It could seem that the problem is considered as solved by the Exp3.S algorithm, but it was actively studied since the seminal paper [ACBFS02]. Two main reasons explain the dynamic research on this problem: first, it is well known that despite their qualities, exponential-weights algorithms like Exp3 can usually be greatly outperformed by UCB-based algorithms, for stochastic problems, and so it is expected that Exp3.S can be outperformed by other algorithms, thus any practical application might benefit from algorithms more efficient than Exp3.S. The second reason is that passively adaptive algorithms function as black-box models: they do not try to detect changes and do not explain why they changed from one arm to another. A lot of the recent research has been focused on actively adaptive algorithms, precisely because they allow to interpret their decisions, by detecting changes on arms with statistical tests. The piece-wise stationary MAB model was then studied by [KS06] and [YM09]. This model is sometimes referred to as the abruptly changing [WS18b], or switching environments

[MS13]. In this model, described in full details in Section 7.2, the (random) reward of arm k at round t has some mean $\mu_k(t)$, that is constant on intervals between two *breakpoints*, and the regret is measured with respect to the *current* best arm $k^*_t \doteq \arg\max_i \mu_k(t)$.

Outline. We introduce the model in Section 7.2, and review related works in Section 7.3. In Section 7.4 we study the Generalized Likelihood Ratio test for sub-Bernoulli distributions (B-GLRT) as a Change-Point Detector (CPD) algorithm. We introduce the two variants of GLR-klUCB algorithm in Section 7.5, where we also present upper bounds on the regret of both variants. The unified regret analysis is given in Section 7.6, we prove one result in Section 7.7. We discuss numerical experiments in Section 7.8, with more details in the Appendix 7.10.

Publications. This chapter is based on our articles [BK19b, BK19a].

7.2 The piece-wise stationary bandit model

A *piece-wise stationary bandit model* is characterized by a set of K arms. A (random) stream of rewards $(Y_{k,t})_{t\in\mathbb{N}^*}$ is associated to each arm $k \in [K]$. We assume that the rewards are bounded, and without loss of generality we assume that $Y_{k,t} \in [0,1]$. We denote by $\mu_k(t) \doteq \mathbb{E}[Y_{k,t}]$ the mean reward of arm k at round t. At each round t, a decision maker has to select an arm $A(t) \in [K]$, based on past observation and receives the corresponding reward $r(t) \doteq Y_{A(t),t}$. At time t, we denote by k^*_t an arm with maximal expected reward, *i.e.*, $\mu_{k^*_t}(t) \doteq \max_k \mu_k(t)$, called an optimal arm (possibly not unique).

A policy $\mathcal{A}$ chooses the next arm to play based on the sequence of past plays and obtained rewards. Like for stationary problems (see Definition 2.3 in Chapter 2), the performance of $\mathcal{A}$ is measured by its (piece-wise stationary) *regret*, the difference between the expected reward obtained by an oracle policy playing an optimal arm k^*_t at time t, and that of the policy $\mathcal{A}$ We use the same notation as the regret for stationary problem without ambiguity, as in this chapter we only consider the following definition,

$$R_T^{\mathcal{A}} \doteq \mathbb{E}\left[\sum_{t=1}^{T} \left(\mu_{k^*_t}(t) - \mu_{A(t)}(t)\right)\right]. \tag{7.1}$$

In the piece-wise *i.i.d.* model, we furthermore assume that there is a (relatively small) *number of breakpoints* Υ_T, defined by

$$\Upsilon_T \doteq \sum_{t=1}^{T-1} \mathbb{1}\left(\exists k \in [K] : \mu_k(t) \neq \mu_k(t+1)\right). \tag{7.2}$$

If we denote $\tau^{(0)} \doteq 0$, we define the i-th breakpoint recursively by $\tau^{(i)} \doteq \inf\{t > \tau^{(i-1)} : \exists k : \mu_k(t) \neq \mu_k(t+1)\}$. Hence for $t \in [\tau^{(i)} + 1, \tau^{(i+1)}]$, that is on any stationary segments, the rewards $(Y_{k,t})$ associated to each arm k are *i.i.d.* (hence the name *piece-wise stationary*).

Note than when a breakpoint occurs, we do not assume that all the arms means change, but that *there exists* an arm whose mean has changed. Depending on the application, many scenario can be meaningful: changes occurring on all arms simultaneously (due to some exogenous event), or only one or a few arms change at each breakpoint. For instance, for a cognitive radio application in a IoT-like network, we can imagine that all the objects of one company (or network provider) use a fixed (or a subset of) channel(s), and on a particular day in a city, if the company deploys a lot of new objects, then one (or some) channel(s) see their mean occupancy rates abruptly change.

We define NC_k the *number of change-points* on arm k by $\mathrm{NC}_k \doteq \sum_{t=1}^{T-1} \mathbb{1}\left(\mu_k(t) \neq \mu_k(t+1)\right)$, for which it clearly holds that $\mathrm{NC}_k \leq \Upsilon_T$, but there can be an arbitrary difference between these two quantities for some arms. Letting $C_T \doteq \sum_{k=1}^{K} \mathrm{NC}_k$ be the total number of change-points on the arms, one can have $C_T \in \{\Upsilon_T, \dots, K\Upsilon_T\}$. We illustrate the two extreme cases in problems 1 and 2 presented in Figures 7.1 and 7.2 below.

An interesting interpolation. The piece-wise stationary bandit model can be viewed as an interpolation between stationary and adversarial models, as the stationary model corresponds to $\Upsilon_T = 0$, while some adversarial models can be considered as a special (worst) case, with $\Upsilon_T = T - 1$ (when considering an adaptive or an oblivious adversary). However, analyzing an algorithm for the piece-wise stationary bandit model requires to assume a small number of changes, typically $\Upsilon_T = o(\sqrt{T})$. Note that the adversarial model of [ACBFS02] is quite powerful, and the authors presented in Section 8 of their paper the EXP3.S algorithm for the piece-wise stationary problem, as explained in Section 7.1 above.

Two example of problems. To illustrate the model, we give here two examples of piece-wise stationary problems. used for the numerical experiments presented in Section 7.8.

Problem 1. We consider $K = 3$ arms changing $\Upsilon_T = 4$ times until $T = 5000$. The arm means $(\mu_k(t))_{1 \leq i \leq K, 1 \leq t \leq T}$ are shown in Figure 7.1 below. Note that changes happen on only one arm (*i.e.*, $C_T = \Upsilon_T = 4$), and the optimal arm changes once at $\tau_2^{(1)} = 2000$.

Problem 2. This problem is close to Problem 1, with a minimum optimality gap of 0.1 (at any time, the smallest difference between two means is at least 0.1), and shown in Figure 7.2. However, all arms change at every breakpoint (*i.e.*, $C_T = K\Upsilon_T = 12$), with identical gaps of 0.1 for arms 0 and 1, and of 0.2 for arm 2 (between two breakpoints, the mean change of $+0.1$ for arm 0 and -0.1 for arm 1 and -0.2 for arm 2). The first optimal arm decreases at every change (2 with ∇ markers), and one arm stays the worst (1 with $\diamond$ markers).

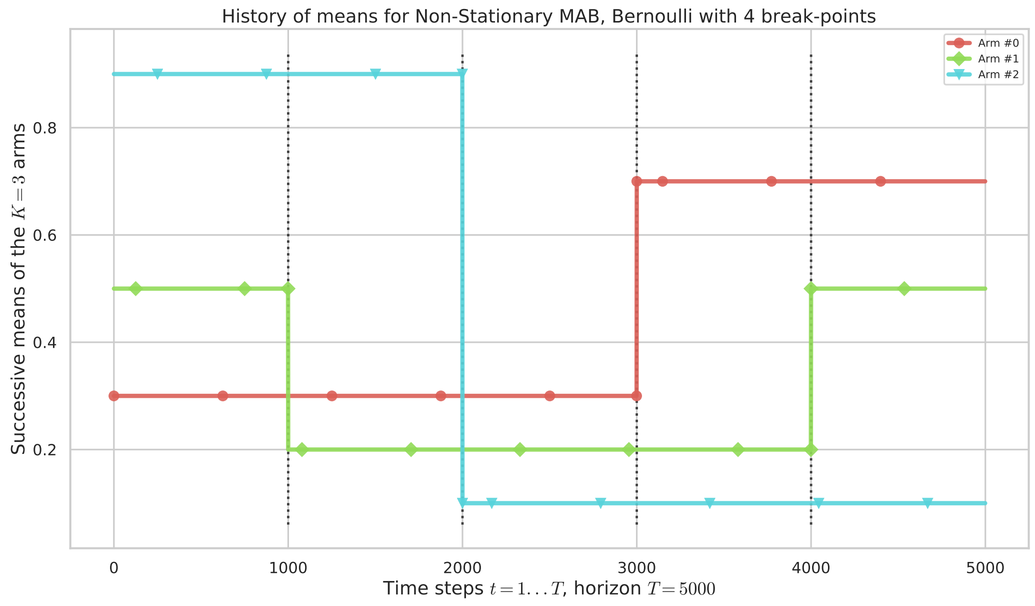


Figure 7.1 – Problem 1: $K = 3$ arms with $T = 5000$, and $\Upsilon = 4$ changes occur on only one arm at a time (*i.e.*, $C = 4$). The means are in $[0, 1]$, and there are $C + 1 = 5$ stationary intervals of equal lengths. Some changes do not modify the optimal arm (*e.g.*, at $T = 1000$ and $T = 4000$) and others do.

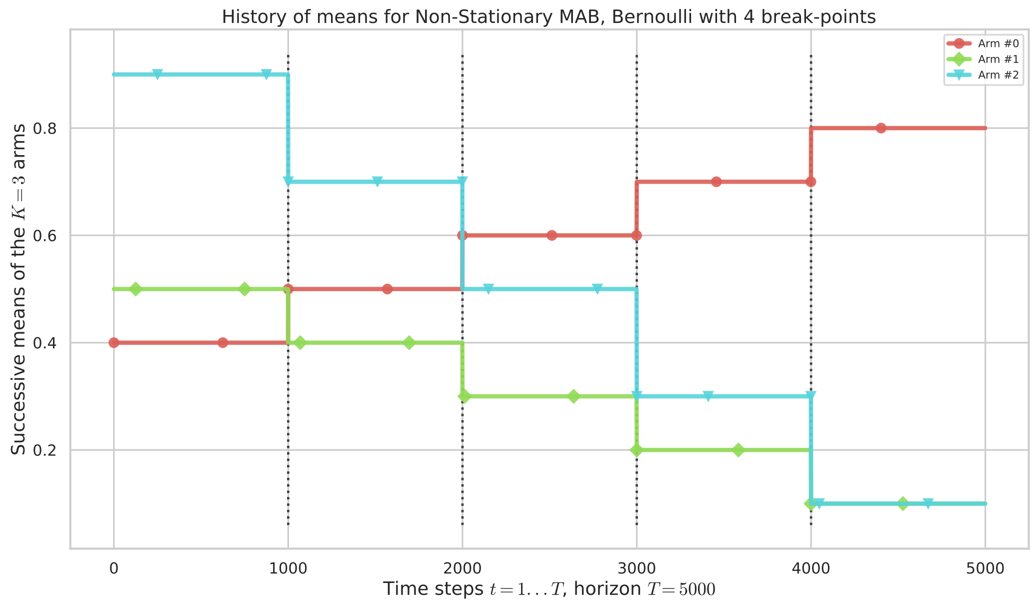


Figure 7.2 – Problem 2: $K = 3$ arms with $T = 5000$, and $\Upsilon = 4$ changes occur on all arms (*i.e.*, $C = 12$). The means are again in $[0, 1]$, and there are also $C + 1 = 5$ stationary intervals of equal lengths.

7.3 Review of related works

We review previous works that studied the piece-wise stationary bandit model, or variants of this model. To the best of our knowledge, all the previous works are based on the idea of combining a classical bandit policy, such as Thompson sampling, UCB or EXP3, with a strategy to account for changes in arms' distributions. Following the vocabulary used in previous works, we make the distinction between *passively* and *actively* adaptive strategies. On the one hand, Actively adaptive strategies monitor the arms' rewards, by using change detection algorithms [BN93], and reset the history of pulls and rewards of one or all the arms as soon as a change is detected. On the other hand, Passively adaptive strategies use a (fixed) discount factor or a limited memory size, while most active strategies use a growing memory.

Passively adaptive algorithms. Their common idea is to adapt a classical policy into forgetting old rewards, . If the forgetting behavior is done efficiently, then the policy can efficiently focus mostly on the most recent rewards, and passively adapt to changes when they happen. The Discounted UCB (D-UCB) algorithm was first introduced in [KS06], and it is an adaptation of the UCB algorithm, with a discount factor $\gamma \in (0,1)$. It works by decreasing all the past rewards by a multiplicative factor, when receiving a new reward from an arm, so that the recent rewards weight more in the (discounted) empirical mean used for the computation of the UCB indexes. The regret of D-UCB was proven to be upper-bounded by $\mathcal{O}(\sqrt{\Upsilon_T T}\ln(T))$ in [GM11], with a tuning $\gamma = 1 - \sqrt{\Upsilon_T / T}/4$, dependent on Υ_T. The Sliding-Window UCB (SW-UCB) uses a sliding window of a fixed size τ, to store only the most τ recent rewards for each arm, and it was proposed by [GM11]. They prove that tuning its window-size to $\tau = 2\sqrt{T\ln(T)/\Upsilon_T}$, gives a bound on the regret of SW-UCB of the form $\mathcal{O}(\sqrt{\Upsilon_T T \ln(T)})$.

Both D-UCB and SW-UCB build on a stationary policy, but for example EXP3.S from [ACBFS02] builds on the EXP3 policy, which is designed for the adversarial case. EXP3.S actually achieves a good regret upper-bound, of the form $\mathcal{O}(\sqrt{K\Upsilon_T T})$, with no additional prior knowledge except that of T and Υ_T. It constitues a good baseline for our numerical experiments in Section 7.8, even if we did find that EXP3.S performs worse that the most of the other approaches based on extensions of stationary bandit algorithms (*e.g.*, SW-UCB). Similarly, previous works showed that older algorithms have no or weaker regret guarantees, and have been proven to be less efficient empirically, or are designed for more specific settings.

The idea of using a simple discount factor, as for D-UCB, was recently adapted to a Bayesian policy, with the Discounted Thompson sampling (DTS) algorithm presented by [RK17]. Even if no theoretical guarantee was given, it can be empirically very efficient, but we found that DTS is not robust in the choice of if its discount factor $\gamma \in (0,1)$, contrarily to what was highlighted in the paper. The DTS algorithm can perform well in practice, for instance with $\gamma = 0.75$ on Problems 1 and 2 (see Section 7.8). However, no theoretical guarantees are given for this strategy, and our experiments did not really confirm the robustness to γ. In general, we found that passively adaptive approaches can be efficient when their parameters are well tuned, but our experiments show that actively adaptive algorithms perform significantly better.

Actively adaptive algorithms. We distinguish two families, the first line of research uses frequentist change-point detectors [BN93], combined with stationary policies, usually index policies like UCB. When using an efficient Change-point Detector (CD) algorithm with an efficient index policy, these approaches usually perform more efficiently than the passively algorithms. The Adapt-EVE algorithm from [HGB+06] uses a Page-Hikley (PH) Test and the UCB policy, but no theoretical guarantee was given. The Windowed-Mean Shift algorithm from [YM09] is more generic and combines any efficient bandit policy with a CD test based on a sliding window, but their approach is not applicable to the bandit setting of interest in this chapter, as they consider side observations. The EXP3.R algorithm from [AF15, AFM17] combines a CD algorithm with EXP3, and the history of all arms are reset as soon as a suboptimal arm is detecting to become optimal. A regret bound of $\mathcal{O}(\Upsilon_T\sqrt{T\ln(T)})$ was proven, even when Υ_T is known.

Two recent and related works use the two-sided CUSUM CD algorithm for CUSUM-UCB in [LLS18] or a specific and simpler CD algorithm for the Monitored UCB (M-UCB) algorithm introduced in [CZKX19]. The CUSUM test is rather complicated and parametric: it uses the first M samples (for a fixed M) from one arm to compute an average $\hat{u_0}$, and then builds two random walks, using the remaining observations for this arm. A change is detected when either random walks cross a threshold h. It requires the tuning of three parameters, M and h as well as a drift correction parameter $\varepsilon \in (0,1)$. The PH test, also studied in [LLS18], is similarly complex, and we found that PHT-UCB perform very similarly to CUSUM-UCB in our experiments. Even if it has the advantage of not requiring a parameter M, it has no theoretical guarantee, so we do not include PHT-UCB in the experiments presented below. In comparison, M-UCB uses a much simpler test, based on the w most recent observations on one arm (for a fixed and even number w), and compares with a fixed threshold h the difference of the sum of rewards for the first half and the second half. It is numerically much simpler, and has the advantage of using only a bounded memory (of the order $\mathcal{O}(Kw)$ for K arms). Both approaches also introduced a mechanism to force a uniform exploration of each arm, parametrized by $\omega \in (0,1)$ (it is called α in both papers, but to avoid clutter with the α parameter in UCB_1, see (2.7)), we rename it ω in this chapter).

When all their parameters are tuned correctly, both approaches are proven to achieve a good regret upper-bound, of order $\mathcal{O}(K\sqrt{\Upsilon_T T\ln(KT)})$. As both results are valid under different assumptions, we consider the two results to be the current state-of-the-art. As for previous works, tuning their parameters requires to know both the horizon T and the number of breakpoints Υ_T. But most importantly it requires a prior knowledge of the problem difficulty, by assuming to know a lower bound on both the length of stationary segments (*e.g.*, L for $\mathrm{M\text{-}UCB}$) and on the changes on the means of arms (*e.g.*, ε for CUSUM). CUSUM-UCB achieves a better regret upper bound, of the order of $\mathcal{O}(\sqrt{\Upsilon_T T \ln(T/\Upsilon_T)})$, but *only* for Bernoulli distributions, and when its 4 parameters $(\omega, \varepsilon, M, h)$ are tuned based on problem-dependent knowledge. $\mathrm{M\text{-}UCB}$ achieves a regret bounded by $\mathcal{O}(\sqrt{\Upsilon_T T \ln(T)})$, for bounded distributions, and when its 4 parameters (ω, w, b, γ) are also tuned based on a problem-dependent knowledge of $\tilde{\delta}$ and L (see Assumption 1 and Remark 1 in [CZKX19]).

On the one hand, CUSUM-UCB performs *local restarts* using this test, to reset the history of *one arm* for which the test detects a change. On the other hand, $\mathrm{M\text{-}UCB}$ performs *global restarts* using this test, to reset the history of *all arms* whenever the test detects a change on one of the arms. Compared to CUSUM-UCB, note that $\mathrm{M\text{-}UCB}$ is numerically much simpler as it only uses a bounded memory, of order $\mathcal{O}(Kw)$ for K arms.

Another interesting recent work is [AGO18], where the authors propose an efficient algorithm, AdSwitch, for the two-armed case ($K = 2$). It proceeds in episodes, starting a new episode when the algorithm detects a change in one of the arms. Each episode consists of three phases, estimation, exploitation then exploration. First, an estimation phase where both arms are played alternatingly until their means can be distinguished. Then follows an exploitation phase, and finally with some low probability, an exploration phase is started that checks whether a change has occurred. Similarly to what previous approaches did (*e.g.*, CUSUM uses a threshold ε), this phase checks for changes of a certain minimum magnitude, but the innovation in this work is that this algorithm does not need any prior knowledge of the minimum magnitude. Instead, it uses geometrically decreasing magnitudes $d_k = 2^{-i}$, meaning that even difficult changes should be detected, possibly in a later episode, as each episode only considers a few different values of d_k. They prove that their algorithm achieves a regret bound of $\mathcal{O}(\ln(T)\sqrt{\Upsilon_T T})$, without a prior knowledge of Υ_T. However, this work as two drawbacks: first, the algorithm currently only applies to two arms, and the generalization does not seem straightforward. Then, the theoretical result is valid for "sufficiently large constants C_1 and C_2", with a large constant C hiding in the $\mathcal{O}(\bullet)$ notation. We did some experiments with AdSwitch, even if it is not included in the benchmark presented below in Section 7.8. It uses two parameters C_1, C_2 that are set at large values for the theoretical analysis, and while we explored numerically different choices, it seemed that setting $C_1 = C_2 = 1$ gave the best performance (which is outside of the comfort zone of the theoretical analysis). Even for this tuning, and for problems with just $K = 2$ arms, empirically this policy is performing poorly in comparison to other policies based on active CD detection and kl-UCB indexes.

Expert aggregation. Another of research on actively adaptive algorithms uses a Bayesian point of view. A Bayesian CD algorithm is combined with Thompson Sampling in [MS13], and more recently [AF17] introduced the Memory Bandit algorithm. It is presented as efficient empirically, but no theoretical guarantee are given for these two works, and due to its complexity, we do not include it in our experiments. The idea behind Memory Bandit is to use an expert aggregation algorithm, like Exp4 from [ACBF02] or our $\mathrm{Aggregator}$ algorithm from Chapter 4 [BKM18], modified to efficiently aggregate a growing number of experts from techniques presented in [MM17]. At each time step, a new expert is introduced, and experts correspond to different Thompson sampling algorithms, each using a different history of pulls and rewards. Intuitively, after a change-point the newest experts will soon become the most efficient, as they are learning by using rewards drawn from the new distribution(s). Note that the regret bound for these methods are still open-problems, and they are computationally more costly, therefore we choose to not include them in our experiments.

Slowly-varying model. A different setting where the quantity of interest is not Υ_T but the total variational budget V_T, was introduced by [BGZ14], for which they proposed the RExp3 algorithm. The total variation budget V_T is defined as $\sum_{t=1}^{T-1}\sum_{k=1}^{K} D(\nu_k(t), \nu_k(t+1))$, for a certain measure of difference D, which measures how much the two distributions of any arm k have changed from time t to time $t+1$. Two examples can be the total variation distance $\|\cdot\|_{TV}$ or the Kullback-Leibler divergence. The RExp3 algorithm can also be qualified as passively adaptive: it is based on (non-adaptive) restarts of the EXP3 algorithm. Note that this algorithm is introduced for a different setting, where the quantity of interest is not Υ_T but a quantity V_T called the total variational budget (satisfying $\Delta^{\text{change}}\Upsilon_T \leq V_T \leq \Upsilon_T$ with Δ^{change} the minimum magnitude of a change-point). A regret bound of the order of $\mathcal{O}(V_T^{1/3} T^{2/3})$ is proven, which is weaker than existing results in our setting.

This setting is quite different from the setting we are considering in this chapter, and even if the most recent works on slowly-varying MAB models are very interesting, we preferred to focus on the piece-wise stationary (or abruptly changing) model, hence we do no include any of these algorithms in the experiments presented in Section 7.8.

Another point-of-view on prior knowledge of Υ_T. In this chapter, like in our two main inspirational works [CZKX19, LLS18], we assume that the algorithm knows in advance the number of break-points Υ_T. Another interesting point-of-view is the one presented by [WS18b, WS18a], where the authors assume that there exists a number $\nu \in (0,1)$ for which $\Upsilon_T = o(T^{\nu})$, and they also assume that any algorithm tackling such problems can know in advance the value of ν (or an upper-bound on ν), and tweak its parameters from this value ν, but cannot know the exact value of Υ_T. This interpretation is interesting too, but empirically we found that the SW-UCB# algorithm proposed in [WS18b] performs comparably to SW-UCB, and thus we did not include it in the experiments presented below in Section 7.8.

Adversarial or non-stationary contextual bandits. For contextual bandits, two very recent works are [LWAL18] and [CLLW19]. While both works target a more general setting, their algorithms are also applicable to non-contextual bandits, *i.e.*, classical bandits like the model in this chapter. From Table 1 in [LWAL18], one can see that their two algorithms Exp4.S and Ada-ILTCB recover the same regret guarantees as we do, in the non-contextual case: that is, a $\mathcal{O}(\Upsilon_T\sqrt{T})$ regret without knowing the number of changes and $\mathcal{O}(\sqrt{\Upsilon_T T})$ with this knowledge. We discuss both these algorithms: first, Exp4.S actually reduces to Exp3.S in the non-contextual case, and Exp3.S is discussed above. Then, the Ada-ILTCB algorithm is however a less appealing candidate for our setting, for the following reason. This algorithm requires as input a parameter L which needs to be an upper bound on the largest stationary sequence in the problem, in order for its regret to scale as $\Upsilon_T\sqrt{L}$ (neglecting K and $\ln$ factors). It is claimed that the tuning $L = T/\Upsilon_T$ yields the optimal bound $\sqrt{\Upsilon_T T}$. However, this tuning is only possible in a "balanced" instance without stationary sequence longer than T/Υ_T. In a piece-wise stationary instance with a stationary sequence of length $T/2$, one cannot get better than $\mathcal{O}(\Upsilon_T\sqrt{T})$ regret, by setting $L = T/2$ (see for instance the problem 4 in our benchmark below in Section 7.8), and as such the regret bound of Ada-ILTCB appears much worse than the announced $\mathcal{O}(\sqrt{\Upsilon_T T})$ bound. Besides, no experiments are reported by [LWAL18], and the Ada-ILTCB algorithm seems much more complicated to implement than other alternatives, so we prefer to not include it in the experiments presented later.

Positionning of our results. CUSUM-UCB and M-UCB are both analyzed under some reasonable assumptions on the problem parameters –the means $(\mu_k(t))$– mostly saying that the breakpoints are sufficiently far away from each other. However, the proposed guarantees only hold for parameters *tuned using some prior knowledge of the means*. Indeed, while in both cases the threshold h can be set as a function of the horizon T and the number of breakpoints Υ_T (also needed by previous approaches to obtain the best possible bounds), the parameter ε for CUSUM and w for M-UCB require the knowledge of Δ^{change} the smallest magnitude of a change-point. In this chapter, we propose *the first algorithm that does not require this knowledge*, and still attains a $\mathcal{O}(\sqrt{\Upsilon_T T \ln(T)})$ regret. Moreover we propose the first comparison of the use of local and global restarts within an adaptive algorithm, by studying two variants of our algorithm. In others words, if we want to apply our proposal to piece-wise stationary problems such as problem 1 or 2 presented above (see Figures 7.1, 7.2), it only needs to know before hand that $T = 5000$ and $\Upsilon_T = 4$ for these examples, but it does not need to know the amplitude of changes $\max_{k,t} |\mu_k(t) - \mu_k(t+1)|$ nor the optimality gap or a bound on the optimality gap at any time step $\max_{k,k'} |\mu_k(t) - \mu_{k'}(t)|$.

Moreover, we can use the two problems 1 and 2 to illustrate the expected empirical behavior of our proposal GLR-klUCB. Problem 1 has only *local changes*, meaning that at any change-point, only one arm sees its distribution change, while problem 2 has only *global changes*, meaning that all arms see their distribution change at each change-point. They illustrate both the extreme cases of having $C_T = \Upsilon_T$ (for problem 1) and $C_T = K\Upsilon_T$ (for problem 2).

- On the one hand, we expect the *local restart* variant of GLR-klUCB to outperform the *global restart* variant for problem 1, as the intuition suggests that it is sub-optimal to reinitialize the memory of the observations of the $K-1$ arms whose distributions did not change after a change-point is detected (as the *global restart* variant does).

- On the other hand, we expect the *global restart* variant to outperform the *local restart* variant for problem 2, as detecting a change on any arm is enough to know that all arms changed and to reinitialize the memory of all arms, and thus the intuition suggests that it is sub-optimal to reinitialize only the memory of the observation of the arm on which the change-point was detected (as the *local restart* variant does).

Of course, on a given problem, without additional prior knowledge on the difficulty of the problem at hand, the algorithm cannot know which situation is more likely to happen, only local changes ($C_T = \Upsilon_T$) or only global changes ($C_T = K\Upsilon_T$) or any intermediate setting between the two extreme cases. Thus we do not believe to be able to design a policy that could be uniformly better than the two variants of GLR-klUCB. Surprisingly, numerical experiments show that the *local* variant of GLR-klUCB actually outperforms the *global* variant for both problem 1 and 2, as shown below in Table 7.2, and in other problems as shown in Table 7.3. Understanding this difference in terms of numerical performance of the two variants is left as future work. Finally, on the practical side, while we can note that the proposed B-GLRT test is more complex to implement than the test used by M-UCB, we propose two heuristics to speed it up while not losing much in terms of regret, in Appendix 7.10.4.

About kl-UCB. Our proposal GLR-klUCB is inspired by both the M-UCB and CUSUM-UCB algorithms. Previous works focused on using UCB [LLS18, CZKX19], but we propose to use kl-UCB instead, as it is known to be more efficient for Bernoulli rewards as well as for a more generic case of bounded or one-dimensional exponential families [CGM+13]. Theoretical results for these works were only given for UCB, but they both suggest that extending the results to kl-UCB (or other efficient stationary policies) should not be difficult. For a fair comparison, we therefore chose to compare the different change-point detector algorithms combined with kl-UCB. We also include in Table 7.1 a comparison of the performance of different algorithms when using UCB or kl-UCB indexes, in order to illustrate that using a more efficient index policy always improves performance, as predicted.

7.4 The Bernoulli GLRT Change-Point Detector

Sequential change-point detection has been extensively studied in the statistical community. We refer to the book [BN93] for a survey. In this section, we are interested in detecting changes on the mean of a probability distribution with bounded support.

Assume that we collect independent samples $X_1, X_2, \dots$ all from some distributions supported in $[0, 1]$. We want to discriminate between two possible scenarios: all the samples come from distributions that have a common mean μ_0, or there exists a *change-point* $\tau > 1$ such that $X_1, \dots, X_\tau$ have some mean μ_0 and $X_{\tau+1}, X_{\tau+2}, \dots$ have a different mean $\mu_1 \neq \mu_0$. A sequential change-point detector is a *stopping time*[1] $\hat{\tau}$ with respect to the filtration $\mathcal{F}_t \doteq \sigma(X_1, \dots, X_t)$ such that $(\hat{\tau} < \infty)$ means that we reject the hypothesis $\mathcal{H}_0 : (\exists \mu_0 \in [0, 1] : \forall t \in \mathbb{N}^*, \mathbb{E}[X_t] = \mu_0)$.

Generalized Likelihood Ratio tests date back to the seminal work of [Wil38], and were for instance studied for change-point detection by [Bar59, SV95]. Exploiting the fact that bounded distribution are $(1/4)$-sub Gaussian (*i.e.*, their moment generating function is dominated by that of a Gaussian distribution with the same mean and a variance $1/4$), the (Gaussian) GLRT, recently studied in depth by [Mai19], can be used for this problem. We propose instead to exploit the fact that bounded distributions are also dominated by Bernoulli distributions.

Definition 7.1 (Sub-Bernoulli distributions)**.** *We call a* sub-Bernoulli distribution *any distribution ν that satisfies $\ln \mathbb{E}_{X\sim\nu}\left[e^{\lambda X}\right] \leq \phi_\mu(\lambda)$ with $\mu \doteq \mathbb{E}_{X\sim\nu}[X]$ and $\phi_\mu(\lambda) \doteq \ln(1 - \mu + \mu e^\lambda)$ is the log moment generating function of a Bernoulli distribution with mean μ, for any $\mu \in [0, 1]$.*

Lemma 1 of [CGM+13] establishes that any bounded distribution supported in $[0, 1]$ is a sub-Bernoulli distribution. As explained in Section 2.1.2, our work can be applied to any bounded distribution without loss of generality, as a reward in $r \in [a, b]$ can clearly be mapped to $[0, 1]$ simply by using $r' = (r - a)/(b - a)$ (we assume that the decision maker or the algorithm knows beforehand the values of a and b).

7.4.1 Presentation of the test

If the samples (X_t) were all drawn from a Bernoulli distribution, this change-point detection problem would reduce to a parametric sequential test of $\mathcal{H}_0 : (\exists \mu_0 : \forall t \in \mathbb{N}^*, X_t \overset{\text{i.i.d.}}{\sim} \mathcal{B}(\mu_0))$, against the alternative $\mathcal{H}_1 : (\exists \mu_0 \neq \mu_1, \tau \in \mathbb{N}^* : \; X_1, \dots, X_\tau \overset{\text{i.i.d.}}{\sim} \mathcal{B}(\mu_0)$ and $X_{\tau+1}, X_{\tau+2}, \dots \overset{\text{i.i.d.}}{\sim} \mathcal{B}(\mu_1))$. The Generalized Likelihood Ratio statistic for this test is defined by

$$\mathrm{GLR}(n) \doteq \frac{\sup\limits_{\mu_0,\mu_1,\tau<n} \ell(X_1, \dots, X_n; \mu_0, \mu_1, \tau)}{\sup\limits_{\mu_0} \ell(X_1, \dots, X_n; \mu_0)}, \tag{7.3}$$

[1] Stopping times are for instance presented formally in Chapter 3 of [LS19].

where $\ell(X_1, \dots, X_n; \mu_0)$ and $\ell(X_1, \dots, X_n; \mu_0, \mu_1, \tau)$ respectively denote the *likelihoods* of the first n observations under a model in $\mathcal{H}_0$ and $\mathcal{H}_1$. High values of this statistic tend to indicate rejection of $\mathcal{H}_0$. By using the form of the likelihood for Bernoulli distributions, this statistic can be written with the binary relative entropy kl, defined for $x, y \in [0,1]$ (with the usual convention that $t\ln(t) \doteq 0$ if $t = 0$), $\mathrm{kl}(x,y) \doteq x \ln\left(\frac{x}{y}\right) + (1-x)\ln\left(\frac{1-x}{1-y}\right)$ Indeed, we show below that for Bernoulli distributions, we have

$$\ln \mathrm{GLR}(n) = \sup_{s \in [2, n-1]} \left[s \times \mathrm{kl}\left(\widehat{\mu}_{1:s}, \widehat{\mu}_{1:n}\right) + (n-s) \times \mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \widehat{\mu}_{1:n}\right)\right]. \tag{7.4}$$

where for $k \leq k'$, $\widehat{\mu}_{k:k'} \doteq \frac{1}{k-k'+1} \sum_{t=k}^{k'} X_t$ denotes the average of the observations X_t collected between the instants k and k'. This result is proven in Appendix 7.10.2.

This motivates the following definition of the Bernoulli GLRT change point detector.

Definition 7.2. *The Bernoulli GLRT (B-GLRT) change point detector with threshold function $\beta(n,\delta)$ is the stopping time $\widehat{\tau}_\delta$ defined by*

$$\widehat{\tau}_\delta \doteq \inf \left\{ n \in \mathbb{N}^* : \sup_{s \in [2, n-1]} \left[s \times \mathrm{kl}\left(\widehat{\mu}_{1:s}, \widehat{\mu}_{1:n}\right) + (n-s) \times \mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \widehat{\mu}_{1:n}\right)\right] \geq \beta(n,\delta) \right\}. \tag{7.5}$$

with the convention that we enforce $\left[s \times \mathrm{kl}\left(\widehat{\mu}_{1:s}, \widehat{\mu}_{1:n}\right) + (n-s) \times \mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \widehat{\mu}_{1:n}\right)\right] = 0$ *in the case of $\widehat{\mu}_{1:s} = \widehat{\mu}_{s+1:n}$ (in particular, if $\widehat{\mu}_{1:n} \in \{0,1\}$, then $\widehat{\mu}_{1:s} = \widehat{\mu}_{s+1:n} = 0$ or $\widehat{\mu}_{1:n}$).*

Asymptotic properties of the GLRT for change-point detection have been studied by [LX10] for Bernoulli distributions and more generally for one-parameter exponential families, for which the GLR test is defined as in (7.5), but with $\mathrm{kl}(x,y)$ replaced by the Kullback-Leibler divergence $d(x,y)$ between two elements in that exponential family that have mean x and y. For example, the Gaussian GLR studied by [Mai19] corresponds to the stopping time (7.5) with the choice $d(x,y) = 2(x-y)^2$, when the variance is set to $\sigma^2 = 1/4$, and non-asymptotic properties of this test are given for any $(1/4)$-subGaussian samples. Note that Pinsker's inequality gives that $\mathrm{kl}(x,y) \geq 2(x-y)^2$, hence the B-GLRT may stop earlier that the Gaussian GLR, based on the quadratic divergence $d(x,y) = 2(x-y)^2$. In the next section, we provide new non-asymptotic results about the B-GLRT under the assumption that the samples (X_t) come from a sub-Bernoulli distribution, which holds for any distribution supported in $[0,1]$.

7.4.2 Non-asymptotic properties of the B-GLRT

When used for a bandit problem, the two main properties of a sequential change-point detection test are its false alarm probability and its detection delay. A small false alarm probability ensures that no detection occurs before it should (*i.e.*, no useless detection occur on stationary segments), and a small detection delay ensures that, if the stationary segments are long enough, then every change-points on every arm will be detected after a short enough amount of samples from that arm. We give below two lemmas, Lemma 7.3 which bounds the false alarm probability for a choice of threshold function, and Lemma 7.5 which bounds the detection delay if there are enough samples before a change.

False alarm probability. In Lemma 7.3 below, we propose a choice of the threshold function $\beta(n, \delta)$ under which the probability that there exists a *false alarm* under *i.i.d.* data is small. To define β, we need to introduce the function $\mathcal{T}$, defined for $x > 0$ by

$$\mathcal{T}(x) \doteq 2\widetilde{h}\left(\frac{h^{-1}(1+x)+\ln(2\zeta(2))}{2}\right), \tag{7.6}$$

where for $u \geq 1$ we define $h(u) \doteq u - \ln(u)$ and its inverse $h^{-1}(u)$, we define $\widetilde{h}(x) \doteq e^{1/h^{-1}(x)}h^{-1}(x)$ if $x \geq h^{-1}(1/\ln(3/2))$ and $\widetilde{h}(x) \doteq (3/2)(x - \ln(\ln(3/2)))$ otherwise, for any $x \geq 0$, and with the value $\zeta(2) = \pi^2/6$. Even if it does not have a closed form expression, the function $\mathcal{T}$ is easy to compute numerically. The inverse h^{-1} can be computed using $\mathcal{W}$, the Lambert $\mathcal{W}$ function [CGH+96], which is the inverse of $x \mapsto x\exp(x)$, as $h^{-1}(x) = -\mathcal{W}(-\exp(-x))$.

The use of $\mathcal{T}$ for the construction of concentration inequalities that are uniform in time is detailed in [KK18], where tight upper bound on this function $\mathcal{T}$ are also given: $\mathcal{T}(x) \simeq x + 4\ln\left(1 + x + \sqrt{2x}\right)$ for $x \geq 5$ and $\mathcal{T}(x) \sim x$ when x is large.

Lemma 7.3. *Let be $\mathbb{P}_{\mu_0}$ a probabilistic model under which $X_t \in [0,1]$, and X_t has an average of μ_0 for all t. Then the B-GLRT test given in Definition 7.2 satisfies $\mathbb{P}_{\mu_0}(\widehat{\tau}_\delta < \infty) \leq \delta$, with the threshold function*

$$\beta(n,\delta) \doteq 2\mathcal{T}\left(\frac{\ln(3n\sqrt{n}/\delta)}{2}\right) + 6\ln(1+\ln(n)). \tag{7.7}$$

Proof. Lemma 7.3 is presented for bounded distributions and is actually valid for any sub-Bernoulli distribution. It could also be presented for more general distributions satisfying

$$\mathbb{E}[e^{\lambda X}] \leq e^{\phi_\mu(\lambda)} \quad \text{with} \quad \mu = \mathbb{E}[X], \tag{7.8}$$

where $\phi_\mu(\lambda)$ is the $\log$ moment generating of some one-dimensional exponential family. The Bernoulli divergence $\mathrm{kl}(x, y)$ would be replaced by the corresponding divergence in that exponential family (which is the Kullback-Leibler divergence between two distributions of means x and y). Let us go back to the Bernoulli case with divergence $\mathrm{kl}(x, y)$. We first have

$$s \times \mathrm{kl}\left(\widehat{\mu}_{1:s}, \widehat{\mu}_{1:n}\right) + (n-s) \times \mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \widehat{\mu}_{1:n}\right) = \inf_{\lambda \in [0,1]} \left[s \times \mathrm{kl}\left(\widehat{\mu}_{1:s}, \lambda\right) + (n-s) \times \mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \lambda\right)\right].$$

Hence the probability of a false alarm occurring is upper bounded as

$$\begin{aligned}
\mathbb{P}_{\mu_0}\left(T_\delta < \infty\right) &\leq \mathbb{P}_{\mu_0}\left(\exists s \in \mathbb{N}^*, n \in \mathbb{N}^*, s < n : s\,\mathrm{kl}\left(\widehat{\mu}_{1:s}, \widehat{\mu}_{1:n}\right) + (n-s)\,\mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \widehat{\mu}_{1:n}\right) > \beta(n,\delta)\right) \\
&\leq \mathbb{P}_{\mu_0}\left(\exists s \in \mathbb{N}^*, n \in \mathbb{N}^*, s < n : s\,\mathrm{kl}\left(\widehat{\mu}_{1:s}, \mu_0\right) + (n-s)\,\mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \mu_0\right) > \beta(n,\delta)\right) \\
&\leq \sum_{s=1}^{\infty} \mathbb{P}_{\mu_0}\left(\exists n > s : s\,\mathrm{kl}\left(\widehat{\mu}_{1:s}, \mu_0\right) + (n-s)\,\mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \mu_0\right) > \beta(n,\delta)\right) \\
&= \sum_{s=1}^{\infty} \mathbb{P}_{\mu_0}\left(\exists r \in \mathbb{N}^* : s\,\mathrm{kl}\left(\widehat{\mu}_{s}, \mu_0\right) + r\,\mathrm{kl}\left(\widehat{\mu}'_{r}, \mu_0\right) > \beta(s+r,\delta)\right),
\end{aligned}$$

where $\widehat{\mu}_s$ and $\widehat{\mu}'_r$ are the empirical means of respectively s and r *i.i.d.* observations with mean μ_0 and distribution ν, that are independent from the previous ones. As ν is sub-Bernoulli, the conclusion follows from Lemma 7.4 below and from the definition of $\beta(n,\delta)$, if we denote $F(x) = \ln(1 + \ln(x))$,

$$\begin{aligned}
&\mathbb{P}_{\mu_0}\left(T_\delta < \infty\right) \\
&\leq \sum_{s=1}^{\infty} \mathbb{P}_{\mu_0}\left(\exists r \in \mathbb{N}^* : s\,\mathrm{kl}\left(\widehat{\mu}_{s}, \mu_0\right) + r\,\mathrm{kl}\left(\widehat{\mu}'_{r}, \mu_0\right) > 6F(s+r) + 2\mathcal{T}\left(\frac{\ln(3(s+r)^{3/2}/\delta)}{2}\right)\right) \\
&\leq \sum_{s=1}^{\infty} \mathbb{P}_{\mu_0}\left(\exists r \in \mathbb{N}^* : s\,\mathrm{kl}\left(\widehat{\mu}_{s}, \mu_0\right) + r\,\mathrm{kl}\left(\widehat{\mu}'_{r}, \mu_0\right) > 3F(s) + 3F(r) + 2\mathcal{T}\left(\frac{\ln(3s^{3/2}/\delta)}{2}\right)\right)
\end{aligned}$$

And so we have $\mathbb{P}_{\mu_0}\left(T_\delta < \infty\right) \leq \sum\limits_{s=1}^{\infty} \frac{\delta}{3s^{3/2}} \leq \delta$. □

Lemma 7.4. *Consider a one-dimensional exponential family $\mathcal{E}$, and let ν_μ be the unique distribution in this family that has mean μ, with moment generating function $\phi_\mu(\lambda) = \mathbb{E}_{X\sim\nu_\mu}[e^{\lambda X}]$. Let $\mathrm{kl}_{\mathcal{E}}(\mu,\mu') \doteq \mathrm{KL}(\nu_\mu, \nu_{\mu'})$ be the divergence function associated to $\mathcal{E}$. Let $(X_i)_{i\in\mathbb{N}^*}$ and $(Y_k)_{i\in\mathbb{N}^*}$ be two independent* i.i.d. *processes, with respective means μ and μ', such that $\mathbb{E}[e^{\lambda X_1}] \leq e^{\phi_\mu(\lambda)}$ and $\mathbb{E}[e^{\lambda Y_1}] \leq e^{\phi_{\mu'}(\lambda)}$. Denote $\widehat{\mu}_s \doteq \frac{1}{s}\sum\limits_{i=1}^{s} X_i$ and $\widehat{\mu}'_r \doteq \frac{1}{r}\sum\limits_{i=1}^{r} Y_k$. Then for every $s, r \in \mathbb{N}^*$ we have,*

$$\mathbb{P}\left(\exists r \in \mathbb{N}^* : s\,\mathrm{kl}_{\mathcal{E}}\left(\widehat{\mu}_s, \mu\right) + r\,\mathrm{kl}_{\mathcal{E}}\left(\widehat{\mu}'_r, \mu'\right) > 3\ln(1+\ln(s)) + 3\ln(1+\ln(r)) + 2\mathcal{T}\left(\frac{x}{2}\right)\right) \leq e^{-x}, \tag{7.9}$$

where $\mathcal{T}$ is the function defined in (7.6).

Proof. This Lemma 7.4 is proven in Appendix 7.10.2. □

Detection delay. Another key feature of a change-point detector is its *detection delay* under a model in which a change from μ_0 to μ_1 occurs at time τ. We already observed that from Pinsker's inequality, the B-GLRT stops earlier than a Gaussian GLR. Hence, one can leverage some techniques from [Mai19] to upper bound the detection delay of the B-GLRT. Letting $\Delta = |\mu_0 - \mu_1|$, one can essentially establish that for τ larger than $(1/\Delta^2)\ln(1/\delta)$ (*i.e.*, enough samples before the change), the delay can be of the same magnitude (*i.e.*, enough samples after the change). In the bandit analysis to follow, the detection delay will be crucially used to control the probability of the good event (in Lemma 7.13 and the corresponding Lemma not included here, but given in Appendix of [BK19b]).

Lemma 7.5. *Let be $\mathbb{P}_{\mu_0,\mu_1,\tau}$ a probabilistic model under which $X_t \in [0,1]$, and X_t has an average of μ_0 for all $t \leq \tau$, and μ_1 for all $t > \tau$, with $\mu_0 \neq \mu_1$, and let $\Delta = |\mu_0 - \mu_1|$.*
Then the B-GLRT test given in Definition 7.2 satisfies

$$\mathbb{P}_{\mu_0,\mu_1,\tau}(\widehat{\tau}_\delta \geq \tau + u) \leq \exp\left(-\frac{2\tau u}{\tau+u}\left(\max\left[0, \Delta - \sqrt{\frac{\tau+u}{2\tau u}\beta(\tau+u,\delta)}\right]\right)^2\right). \tag{7.10}$$

Proof. It is actually not used as such in the proofs of the regret upper bounds given below, because a similar but more specific result is used and proven in the proofs. For instance, see below in Section 7.7.2 (more specifically, in page 228). □

For a threshold β chosen as in the Lemma 7.3, we can show that a consequence of the Lemma 7.5 is that if the break-point τ takes place after about $\Delta^{-2}\ln(1/\delta)$ samples, the detection time may be of the same magnitude (with high probability). In other words, if the B-GLRT test observes enough samples before a change-point, its delay $\widehat{\tau}_\delta$ is in the order of $O(\Delta^{-2}\ln(1/\delta))$, with high probability.

7.4.3 Practical considerations

Lemma 7.3 provides the first non-asymptotic control of false alarm for the B-GLRT employed for bounded data. However, the threshold (7.7) is not fully explicit as the function $\mathcal{T}(x)$ can only be computed numerically. Note that for sub-Gaussian distributions, results from [Mai19] show that the smaller and more explicit threshold $\beta(n,\delta) = \left(1+\frac{1}{n}\right)\ln\left(\frac{3n\sqrt{n}}{\delta}\right)$, can be used to prove an upper bound of δ for the false alarm probability of the GLR, with quadratic divergence $\mathrm{kl}(x,y) = 2(x-y)^2$. For the B-GLRT, numerical simulations suggest that the threshold (7.7) is a bit conservative (see Appendix 7.10.7), and in practice we recommend to keep only the leading term and use $\beta(n,\delta) = \ln(3n^{3/2}/\delta) = \ln(3) + 3/2\ln(n) - \ln(\delta)$.

Also note that, as any test based on scan-statistics, the B-GLRT can be costly to implement as at every time step, it considers all previous time steps as a possible position for a change-point. Thus, down-sampling the possible time steps is an interesting adaptation in practice:

$$\widetilde{\tau}_\delta = \inf \left\{ n \in \mathcal{N} : \sup_{s \in \mathcal{S}_n} \left[s \times \mathrm{kl}\left(\widehat{\mu}_{1:s}, \widehat{\mu}_{1:n}\right) + (n-s) \times \mathrm{kl}\left(\widehat{\mu}_{s+1:n}, \widehat{\mu}_{1:n}\right) \right] \geq \beta(n,\delta) \right\}, \tag{7.11}$$

for subsets $\mathcal{N}$ and $\mathcal{S}_n$. Following the proof of Lemma 7.3, we can easily see that this variant enjoys the same false-alarm control. However, the detection delay may be slightly increased. In Appendix 7.10.4 we show that using these practical speedups has little impact on the regret of the bandit strategy introduced below.

7.5 The GLR-klUCB algorithm

We start by giving the pseudo-code of our algorithm, by explaining every part, then we give its finite-time regret upper bounds, for the two variants of using local or global restarts. The analysis in both cases is using a unified proof technique, that we present before giving more details about the proof of half of the results (the other proof can be found in [BK19b]).

We now present the GLR-klUCB algorithm, which combines a bandit algorithm with a change-point detector running on each arm. It also needs a third ingredient, some forced exploration parameterized by $\omega \in (0,1)$ to ensure each arm is sampled enough and changes can also be detected on arms currently under-sampled by the bandit algorithm. GLR-klUCB combines the kl-UCB algorithm as it is given and analyzed by [CGM⁺13], known to be optimal for Bernoulli bandits, with the B-GLRT change-point detector introduced in Section 7.4. This algorithm, formally stated as Algorithm 7.1, can be used in any bandit model with bounded rewards, and we expect it to be very efficient for Bernoulli distributions, which are relevant for practical applications.

When does GLR-klUCB **restart?** The GLR-klUCB algorithm can be viewed as a kl-UCB algorithm allowing for some *restarts* on the different arms. A restart happens when the B-GLRT change-point detector detects a change on the arm that has been played (line 9). To be fully specific, $\mathrm{GLR}_\delta(Z_1, \dots, Z_n) = \mathrm{True}$ if and only if

$$\sup_{1<s<n} \left[s \times \mathrm{kl}\left(\frac{1}{s}\sum_{i=1}^{s} Z_k, \frac{1}{n}\sum_{i=1}^{n} Z_k \right) + (n-s) \times \mathrm{kl}\left(\frac{1}{n-s}\sum_{i=s+1}^{t} Z_k, \frac{1}{n}\sum_{i=1}^{n} Z_k \right) \right] \geq \beta(n,\delta), \tag{7.12}$$

with $\beta(n,\delta)$ defined in (7.7), or $\beta(n,\delta) = \ln(3n^{3/2}/\delta)$, as recommended in practice, see Section 7.4.3. We remind in this notation, if $\widehat{Z}_{1:s} = \widehat{Z}_{s+1:n}$, *i.e.*, if all observations $Z_1, \dots, Z_n$ are equal, we set the left-hand side of (7.12) to 0 and the B-GLRT cannot detect a change-point. We define the (kl-UCB like) index used by our algorithm, by denoting $\tau_k(t)$ the last restart that

```
1  Input: Parameters: exploration rate $\omega \in (0,1)$, confidence level $\delta > 0$
2  Input: Option: Local or Global restart
3  initialization: $\forall k \in [K]$, $\tau_k = 0$ and $n_k = 0$;
4  for $t = 1, 2, \ldots, T$ do
5      if $t \mod \lfloor \frac{K}{\omega} \rfloor \in [K]$ then                    // forced exploration
6          $A(t) = t \mod \lfloor \frac{K}{\omega} \rfloor$;
7      else
8          $A(t) \in \mathcal{U}\left(\arg\max_{k\in[K]} \mathrm{UCB}_k(t)\right)$, with $\mathrm{UCB}_k(t)$ defined in (7.13);
9      Play arm $A(t) : n_{A(t)} = n_{A(t)} + 1$;
10     Observe the reward $Y_{A(t),t} : Z_{A(t),n_{A(t)}} = Y_{A(t),t}$;
11     if $\mathrm{GLR}_\delta(Z_{A(t),1}, \ldots, Z_{A(t),n_{A(t)}}) = True$ then   // Change point is detected
12         if Global restart then
13             $\forall k \in [K], \tau_k = t$ and $n_k = 0$;                      // restart all arms
14         else
15             $\tau_{A(t)} = t$ and $n_{A(t)} = 0$;                              // restart only this arm
16 end
```

Algorithm 7.1: GLR-klUCB, with **Local** or **Global** restarts

happened for arm k before time t, $n_k(t) = \sum_{s=\tau_k(t)+1}^{t} \mathbb{1}(A(s) = k)$ the number of selections of arm k, and $\widehat{\mu}_k(t) = \frac{1}{n_k(t)} \sum_{s=\tau_k(t)+1}^{t} Y_{k,s}\mathbb{1}(A(s) = i)$ their empirical mean (if $n_k(t) \neq 0$). With the exploration function $f(t) = \ln(t) + 3\ln(\ln(t))$ if $t > 1$ and $f(t) = 0$ otherwise, the index is defined using the binary relative entropy kl as

$$\mathrm{UCB}_k(t) \doteq \max\Big\{q \in [0,1] : n_k(t) \times \mathrm{kl}\left(\widehat{\mu}_k(t), q\right) \leq f(t - \tau_k(t))\Big\}. \tag{7.13}$$

Two options for restarts. For this algorithm, we simultaneously investigate two possible behaviors: *global restart* (reset the history of all arms once a change was detected on one of them, line 11), and *local restart* (reset only the history of the arm on which a change was detected, line 13), which are the two different options in Algorithm 7.1. Under local restart, in the general case the times $\tau_k(t)$ are not equal for all arms, hence the index policy associated to (7.13) is *not* a standard UCB algorithm, as each index uses a *different exploration rate*. One can highlight that in the CUSUM-UCB algorithm, which is the only existing algorithm based on local restart, the UCB index are defined differently[2]: $f(t - \tau_k(t))$ is replaced by $f(n_t)$ with $n_t \doteq \sum_{k=1}^{K} n_k(t)$.

Threshold function β. We present in Appendix 7.10.7 numerical simulations that compare different choices of threshold functions β. We compare the non-explicit function used in

[2] This alternative is currently not correctly supported by theory, as we found mistakes in the analysis of CUSUM-UCB: the main problem resides in the use of Hoeffding's inequality with a *random* number of observations and a *random* threshold to obtain Eq. (31)-(32) in the paper [LLS18].

Lemma 7.3 (that makes use of a numerical approximation of the function $\mathcal{T}$), with the simpler value $\beta(n,\delta) = \ln(3n\sqrt{n}/\delta)$, as well as two other choices. To sum-up these simulations, they validate the use of a simpler and more explicit threshold, thus we recommend in practice to use $\beta_1(n,\delta) = \ln(3n\sqrt{n}/\delta)$.

Forced exploration. The forced exploration scheme used in GLR-klUCB (lines 3-5) generalizes the deterministic exploration proposed for M-UCB by [CZKX19], whereas CUSUM-UCB performs a uniform random exploration. Both approaches are parameterized by $\omega \in (0,1)$. More precisely, CUSUM-UCB samples arm k with probability ω/K at each time step, and M-UCB uses a deterministic scheme, to force exploring the K arms: when the time since the last restart, $t-\tau$, is found to be in $[K]$ modulo $\lceil 1/\omega \rceil$. Both solutions ensure that all arms are sampled enough on each stationary sequence, so that the CD algorithm has enough *i.i.d.* samples to detect changes. A consequence of this forced exploration is given in Proposition 7.6 below.

Proposition 7.6. *For every pair of instants $s \leq t \in \mathbb{N}^*$ between two restarts on arm k* (i.e., *for a $i \in [NC_k]$, one has $\tau_k^{(i)} = \tau_k(t) < s \leq t < \tau_k^{(i+1)}$) it holds that $n_k(t) - n_k(s) \geq \lfloor \frac{\omega}{K}(t-s) \rfloor$.*

Proof. We consider one arm $k \in [K]$, and when the GLR-klUCB algorithm is running, we consider two time steps $s \leq t \in \mathbb{N}^*$, chosen between two restart times for that arm k. Lines 3-4 state that $A(u) = u \mod \lceil \frac{K}{\omega} \rceil$ if $u \mod \lceil \frac{K}{\omega} \rceil \in [K]$, thus we directly find

$$\begin{aligned}
n_k(t) - n_k(s) &= \sum_{u=s+1}^{t} \mathbb{1}(A(u) = k) \\
&\geq \sum_{u=s+1}^{t} \mathbb{1}\left(A(u) = k, A(u) = u \mod \left\lceil \frac{K}{\omega} \right\rceil\right) \\
&\geq \sum_{u=s+1}^{t} \mathbb{1}\left(k = u \mod \left\lceil \frac{K}{\omega} \right\rceil\right) \\
&= (t - (s+1) + 1) / \left\lceil \frac{K}{\omega} \right\rceil \geq \left\lfloor \frac{\omega}{K}(t-s) \right\rfloor.
\end{aligned}$$

□

Other forced exploration schemes? We present in Appendix 7.10.8 numerical simulations that compare three different options of forced exploration schemes. We compare the uniformly random exploration used by CUSUM-UCB, against the deterministic scheme used in Algorithm 7.1, and against a more complicated scheme based on *tracking*. To sum-up these simulations, the deterministic scheme gives an efficient change-point detection algorithm, and an efficient GLR-klUCB policy. As it is the simplest one to handle in our proofs, this is the one we chose for Algorithm 7.1.

7.6 Finite-time upper-bounds on the regret of GLR-klUCB

This section gives the finite-time regret bounds for the two variants, and interpretations of the different results. Although we only prove one bound, the other proof can be found in [BK19b].

7.6.1 Results for GLR-klUCB using *global restarts*

Recall that $\tau^{(i)}$ denotes the position of the i-th break-point and let $\mu_k^{(i)}$ be the mean of arm k on the segment between the i- and $(i+1)$-th breakpoint: $\forall t \in \{\tau^{(i-1)}+1,\dots,\tau^{(i)}\}, \mu_k(t) = \mu_k^{(i)}$. Let $i^* = \arg\max_k \mu_k^{(i)}$ and the largest gap at break-point i as $\Delta^{(i)} \doteq \max_{k\in[K]} |\mu_k^{(i)} - \mu_k^{(i-1)}| > 0$.

Assumption 7.7. *Define* $d^{(i)} \doteq d^{(i)}(\omega,\delta) \doteq \left\lceil \frac{4K}{\omega(\Delta^{(i)})^2}\beta(T,\delta) + \frac{K}{\omega} \right\rceil$. *Then* we assume that there are enough samples between two global change-points. *In other words, we assume for all change* $i \in [\Upsilon_T]$, $\tau^{(i)} - \tau^{(i-1)} \geq 2\max(d^{(i)}, d^{(i-1)})$.

Assumption 7.7 is easy to interpret and standard in non-stationary bandits. It requires that the distance between two consecutive breakpoints is large enough: how large depends on the magnitude of the largest change that happen at those two breakpoints. Under this assumption, we provide in Theorem 7.8 a finite time problem-dependent regret upper bound. It features the parameters ω and δ, the KL-divergence terms $\mathrm{kl}(\mu_k^{(i)}, \mu_{k^*}^{(i)})$ expressing the hardness of the (stationary) MAB problem between two breakpoints, and the $\Delta^{(i)}$ terms expressing the hardness of the change-point detection problem.

Theorem 7.8. *For ω and δ for which Assumption 7.7 is satisfied, the regret of* GLR-klUCB *with parameters ω and δ based on* ***Global*** *Restart satisfies*

$$\begin{aligned} R_T \;\leq\; & 2\sum_{k=1}^{\Upsilon_T} \frac{4K}{\omega\left(\Delta^{(i)}\right)^2}\beta(T,\delta) + \omega T + \delta(K+1)\Upsilon_T \\ & + \sum_{k=1}^{\Upsilon_T} \sum_{\substack{i=1,\dots,\Upsilon_T \\ \mu_k^{(i)} \neq \mu_{k^*}^{(i)}}} \frac{\left(\mu_{k^*}^{(i)} - \mu_k^{(i)}\right)}{\mathrm{kl}\left(\mu_k^{(i)}, \mu_{k^*}^{(i)}\right)} \ln(T) + \mathcal{O}\left(\sqrt{\ln(T)}\right). \end{aligned} \tag{7.14}$$

Warning: We highlight that *this result is finite-time* and not asymptotic, even if it uses the notation $\mathcal{O}\left(\sqrt{\ln(T)}\right)$. This last term does not mean the whole inequality is only valid for $T \to \infty$, but it rather means that at finite time, the inequality is valid with the last term being a function $g(T)$, which is upper-bounded by a constant times $\sqrt{\ln(T)}$ from a certain time T_0.

Corollary 7.9. *For "easy" problems satisfying the corresponding Assumption 7.7, let Δ^{opt} denote the smallest value of a sub-optimality gap on one of the stationary segments, and Δ^{change} be the smallest magnitude of any change point on any arm.*

1. *Choosing $\omega = \sqrt{\ln(T)/T}$, $\delta = 1/\sqrt{T}$ gives*

$$R_T = \mathcal{O}\left(\frac{K}{\left(\Delta^{change}\right)^2} \Upsilon_T \sqrt{T \ln(T)} + \frac{(K-1)}{\Delta^{opt}} \Upsilon_T \ln(T) \right), \tag{7.15}$$

2. *Choosing $\omega = \sqrt{\Upsilon_T \ln(T)/T}$, $\delta = 1/\sqrt{\Upsilon_T T}$ gives*

$$R_T = \mathcal{O}\left(\frac{K}{\left(\Delta^{change}\right)^2} \sqrt{\Upsilon_T T \ln(T)} + \frac{(K-1)}{\Delta^{opt}} \Upsilon_T \ln(T) \right). \tag{7.16}$$

7.6.2 Results for GLR-klUCB using Local Restarts

Some new notations are needed to state a regret bound for GLR-klUCB using *local restarts*, and to distinguish notations between the results for the two variants (local and global restarts), we denote ℓ instead of k for the indexes of change-points. We let $\tau_k^{(\ell)}$ denote the position of the ℓ-th change point *for arm k*: $\tau_k^{(\ell)} = \inf\{t > \tau_k^{(\ell-1)} : \mu_k(t) \neq \mu_k(t+1)\}$, with the convention $\tau_k^{(0)} = 0$, and let $\overline{\mu}_k^{(\ell)}$ be the ℓ-th value for the mean of arm k, such that $\forall t \in [\tau_k^{(\ell-1)}+1, \tau_k^{(\ell)}],\ \ \mu_k(t) = \overline{\mu}_k^{(\ell)}$. We also introduce the gap $\Delta_k^{(\ell)} = \overline{\mu}_k^{\ell} - \overline{\mu}_k^{\ell-1} > 0$.

Assumption 7.10. *Define* $d_k^{(\ell)} \doteq d_k^{(\ell)}(\omega,\delta) \doteq \left\lceil \frac{4K}{\omega\left(\Delta_k^{(\ell)}\right)^2}\beta(T,\delta) + \frac{K}{\omega} \right\rceil$. *Then* we assume that there are enough samples between two local change-points. *In other words, we assume that for all arm k and all change-point of that arm $\ell \in [\mathrm{NC}_k]$, $\tau_k^{(\ell)} - \tau_k^{(\ell-1)} \geq 2\max(d_k^{(\ell)}, d_k^{(\ell-1)})$.*

Assumption 7.10 is again easy to interpret, like Assumption 7.7, but it is non standard in non-stationary bandits, and to the best of or knowledge, our analysis is the first one to be given for such assumption on the problem difficulty. It requires that any two consecutive change-points *on a given arm* are sufficiently spaced (relatively to the magnitude of those two change-points). Under that assumption, Theorem 7.11 provides a regret upper bound that scales with similar quantities as that of Theorem 7.8, except that the number of breakpoints Υ_T is replaced with the *total* number of change points $\mathrm{C}_T \doteq \sum_{k=1}^K \mathrm{NC}_k \leq K\Upsilon_T$.

Theorem 7.11. *For ω and δ for which Assumption 7.10 is satisfied, the regret of* GLR-klUCB *with parameters ω and δ based on Local Restart satisfies*

$$R_T \leq 2\sum_{k=1}^{K}\sum_{\ell=1}^{\mathrm{NC}_k} \frac{4K}{\omega\left(\Delta_k^{(\ell)}\right)^2}\beta(T,\delta) + \omega T + 2\delta \mathrm{C}_T + \sum_{k=1}^{K}\sum_{\ell=1}^{\mathrm{NC}_k} \frac{\ln(T)}{\mathrm{kl}(\overline{\mu}_k^{(\ell)}, \mu_{i,\ell}^*)} + \mathcal{O}\left(\sqrt{\ln(T)}\right), \tag{7.17}$$

where $\mu^*_{i,\ell} \doteq \inf \left\{ \mu_{k^*_t}(t) : \mu_{k^*_t}(t) \neq \overline{\mu}_k^{(\ell)}, t \in [\tau_k^{(\ell)} + 1, \tau_k^{(\ell+1)}] \right\}$.

Like for the first Theorem 7.8, we highlight that *this result is finite-time*. The proof of GLR-klUCB with *local restarts* is not included, but it can be found in the Appendix of [BK19b], as it is quite similar to the proof for *global restarts* given above. The main difficulty relies in defining the "good event" $\mathcal{E}_T$, and proving that it happens with high probability (by showing that the complementary event $\mathcal{E}_T^c$ is highly unlikely). We can directly obtain different regret upper-bounds for different choices of the two parameters ω and δ. We prefer to state the four cases, to highlight the modularity of the result given by Theorem 7.11.

Corollary 7.12. *For "easy" problems satisfying the corresponding Assumption 7.10, with* Δ^{opt} *and* Δ^{change} *defined as in Corollary 7.9, the following regret bounds hold.*

1. *Choosing* $\omega = \sqrt{\ln(T)/T}$, $\delta = 1/\sqrt{T}$ *(with no prior knowledge of* Υ_T *or* C_T*) gives*

$$R_T = \mathcal{O}\left(\frac{K}{(\Delta^{change})^2} \mathrm{C}_T \sqrt{T \ln(T)} + \frac{\mathrm{C}_T}{(\Delta^{opt})^2} \ln(T) \right), \tag{7.18}$$

2. *Choosing* $\omega = \sqrt{\Upsilon_T \ln(T)/T}$, $\delta = 1/\sqrt{\Upsilon_T T}$ *(with prior knowledge of* Υ_T *and "optimist" guess* $\Upsilon_T \simeq \mathrm{C}_T \ll K\Upsilon_T$*) gives*

$$R_T = \mathcal{O}\left(\frac{K^2}{(\Delta^{change})^2} \sqrt{\Upsilon_T T \ln(T)} + \frac{K\Upsilon_T}{(\Delta^{opt})^2} \ln(T) \right), \tag{7.19}$$

3. *Choosing* $\omega = \sqrt{\mathrm{C}_T \ln(T)/T}$, $\delta = 1/\sqrt{\mathrm{C}_T T}$ *(with prior knowledge of* C_T*) gives*

$$R_T = \mathcal{O}\left(\frac{K}{(\Delta^{change})^2} \sqrt{\mathrm{C}_T T \ln(T)} + \frac{\mathrm{C}_T}{(\Delta^{opt})^2} \ln(T) \right), \tag{7.20}$$

4. *Choosing* $\omega = \sqrt{K\Upsilon_T \ln(T)/T}$, $\delta = 1/\sqrt{K\Upsilon_T T}$ *(with prior knowledge of* Υ_T *and "pessimist" guess* $\mathrm{C}_T \simeq K\Upsilon_T$*) gives*

$$R_T = \mathcal{O}\left(\frac{K}{(\Delta^{change})^2} \sqrt{\mathrm{C}_T T \ln(T)} + \frac{\mathrm{C}_T}{(\Delta^{opt})^2} \ln(T) \right). \tag{7.21}$$

7.6.3 Interpretation and comparison of the results

The regret bounds we obtain for the two variants of our proposal GLR-klUCB, Theorems 7.8 and 7.11 respectively, both show that there exists a tuning of ω and δ as a function and T *and the number of changes* such that the regret is of order $\mathcal{O}_h(K\sqrt{\Upsilon_T T \ln(T)})$ and $\mathcal{O}_h(K\sqrt{\mathrm{C}_T T \ln(T)})$

respectively, where the $\mathcal{O}_h$ notations ignore the gap terms. For very particular instances such that $\Upsilon_T = C_T$, *i.e.*, at each break-point only one arm changes (*e.g.*, problem 1 from Figure 7.1), the theory advocates the use of local restarts. Indeed, while the regret guarantees obtained are similar, those obtained for local restarts hold for a wider variety of problems as Assumption 7.10 is less stringent than Assumption 7.7. Besides those specific instances, our results are essentially worse for *local* than *global restarts*. However, we only obtain regret upper bounds – thus providing a theoretical safety net for both variants of our algorithm, and the practical story is different, as discussed in Section 7.8. We find that GLR-klUCB performs better with local restarts, uniformly on all problems.

One can note that with the tuning of ω and δ prescribed by Corollaries 7.9 and 7.12, the regret bounds of GLR-klUCB hold for problem instances for which two consecutive break-points (or change-points on an arm) are separated by more than (about) $\sqrt{T\ln(T)}/(\Delta^{\text{change}})^2$ rounds. Hence those guarantees are valid on "easy" problems only, with few changes of large magnitudes, in particular they do not hold for the harder problems 3 or 5 presented in Section 7.8. However, this does not prevent our algorithms from performing well on more realistic instances, as shown by the numerical experiments, for instance with problem 3 presented in the next Section 7.8). And M-UCB [CZKX19] is also analyzed for the same type of unrealistic assumptions, while its practical performance is illustrated beyond those.

7.7 Proof of the regret upper-bounds

This Section starts by giving a unified analysis of regret of the two variants of our algorithm, then we give the proof of the finite-time regret bound for the variant using *global restarts*. The proof of the other variant using *local restarts* follows the skeleton of the unified analysis, and it can be found in Appendix E of [BK19b].

7.7.1 Sketch of the unified regret analysis

In this section, we sketch a unified proof for the two Theorems 7.8 and 7.11 given above. We emphasize that our approach is significantly different from those proposed by [CZKX19] for M-UCB and by [LLS18] for CUSUM-UCB. Recall that in this Chapter, the regret is defined as $R_T \doteq \mathbb{E}\left[\sum_{t=1}^{T}(\mu_{k_t^*}(t) - \mu_{A(t)}(t))\right]$. First, we introduce $\mathcal{D}(T,\omega)$ the (deterministic) set of time steps at which the forced exploration is performed before time T (see lines 3-4 in Algorithm 7.1), and as we observe that $\mu_{k_t^*} - \mu_{A(t)} \leq 1$ because rewards are assumed to be bounded in $[0,1]$, we can write a first decomposition of the regret,

$$\sum_{t=1}^{T}(\mu_{k_t^*}(t) - \mu_{A(t)}(t))$$

$$\leq \sum_{t=1}^{T} \mathbb{1}(t \in \mathcal{D}(T,\omega)) + \sum_{t=1}^{T} (\mu_{k_t^*}(t) - \mu_{A(t)}(t)) \mathbb{1}\left(t \notin \mathcal{D}(T,\omega), \mathrm{UCB}_{A(t)}(t) \geq \mathrm{UCB}_{k_t^*}(t)\right)$$
$$\leq \omega T + \sum_{t=1}^{T} \mathbb{1}\left(\mathrm{UCB}_{k_t^*}(t) \leq \mu_{k_t^*}(t)\right) + \sum_{k=1}^{K}\sum_{t=1}^{T} (\mu_{k_t^*}(t) - \mu_k(t)) \mathbb{1}\left(A(t) = k, \mathrm{UCB}_k(t) \geq \mu_{k_t^*}(t)\right).$$

Introducing some *good event* $\mathcal{E}_T$, to be specified in each case, and its complementary $\mathcal{E}_T^c$, one can write the following decomposition, that highlights two important terms (A) and (B)

$$R_T \leq T\mathbb{P}(\mathcal{E}_T^c) + \omega T + \underbrace{\mathbb{E}\left[\mathbb{1}(\mathcal{E}_T) \sum_{t=1}^{T} \mathbb{1}\left(\mathrm{UCB}_{k_t^*}(t) \leq \mu_{k_t^*}(t)\right)\right]}_{(A)} \tag{7.22}$$
$$+ \underbrace{\mathbb{E}\left[\mathbb{1}(\mathcal{E}_T) \sum_{t=1}^{T} (\mu_{k_t^*}(t) - \mu_{A(t)}(t)) \mathbb{1}\left(\mathrm{UCB}_{A(t)}(t) \geq \mu_{k_t^*}(t)\right)\right]}_{(B)}.$$

Each analysis requires to define an *appropriate good event*, stating that *some* change-points are detected within a reasonable delay. Each regret bound then follows from upper bounds on term (A), term (B), and on the failure probability $\mathbb{P}(\mathcal{E}_T^c)$. To control (A) and (B), we split the sum over consecutive segments $[\tau^{(i)} + 1, \tau^{(i+1)}]$ for *global restarts* and $[\tau_k^{(i)} + 1, \tau_k^{(i+1)}]$ for each arm k for *local restarts*, and use elements from the analysis of kl-UCB of [CGM+13].

The tricky part of each proof, which crucially exploits Assumption 7.7 or 7.10, is actually to obtain an upper bound on $\mathbb{P}(\mathcal{E}_T^c)$. For example for *local restarts* (Theorem 7.11), the good event is defined as $\mathcal{E}_T(\omega,\delta) \doteq \left(\forall k \in [K], \forall \ell \in [\mathrm{NC}_k], \widehat{\tau}_k^{(\ell)} \in \left[\tau_k^{(\ell)} + 1, \tau_k^{(\ell)} + d_k^{(\ell)}\right]\right)$, where $\widehat{\tau}_k^{(\ell)}$ is defined as the ℓ-th change detected by the algorithm on arm k and $d_k^{(\ell)} = d_k^{(\ell)}(\omega,\delta)$ is defined in Assumption 7.10. Introducing the event $\mathcal{C}_k^{(\ell)} \doteq \left\{\forall j \leq \ell, \widehat{\tau}_k^{(\ell)} \in \left[\tau_k^{(j)} + 1, \tau_k^{(j)} + d_k^{(j)}\right]\right\}$, that all the changes up to the ℓ-th have been detected, a union bound yields this decomposition

$$\mathbb{P}(\mathcal{E}_T(\omega,\delta)^c) \leq \sum_{k=1}^{K}\sum_{\ell=1}^{\mathrm{NC}_k} \underbrace{\mathbb{P}\left(\widehat{\tau}_k^{(\ell)} \leq \tau_k^{(\ell)} \mid \mathcal{C}_k^{(\ell-1)}\right)}_{(a)} + \sum_{k=1}^{K}\sum_{\ell=1}^{\mathrm{NC}_k} \underbrace{\mathbb{P}\left(\widehat{\tau}_k^{(\ell)} \geq \tau_k^{(\ell)} + d_k^{(\ell)} \mid \mathcal{C}_k^{(\ell-1)}\right)}_{(b)}. \tag{7.23}$$

- Term (a) is related to the control of probability of false alarm, which is given by Lemma 7.3 for a change-point detector run in isolation. Observe that under the bandit algorithm, the change point detector associated to arm k is based on (possibly much) less than $t - \tau_k(t)$ samples from arm k, which makes false alarm even less likely to occur. Hence, it is easy to show that $(a) \leq \delta$.

- Term (b) is related to the control of the detection delay, which is more tricky to obtain under the GLR-klUCB adaptive sampling scheme, when compared to a result like Theorem 6 in [Mai19] for the change-point detector run in isolation. More precisely, we need to leverage the forced exploration (Proposition 7.6) to be sure we have enough samples for detection. This explains why delays defined in Assumption 7.10 are scaled by $1/\omega$. Using some elementary calculus and a concentration inequality given in Lemma 7.14, we can finally prove that $(b) \leq \delta$.

Finally, the "bad event" is unlikely, $\mathbb{P}(\mathcal{E}_T^c) \leq 2\mathrm{C}_T\delta$. By putting together the three pieces, that are a bound on (A), on (B) and on $\mathbb{P}(\mathcal{E}_T^c)$, we obtain the desired finite-time upper-bound on the regret of our proposal GLR-klUCB.

7.7.2 Proof for GLR-klUCB with *global restarts*

The proof uses these notations: let $\widehat{\tau}^{(i)}$ be the k-th change detected by the algorithm, leading to the k-th (full) restart and let $\widehat{\tau}(t)$ be the last time before t that the algorithm restarted. We denote $n_k(t) \doteq \sum_{s=\tau(t)+1}^{t} \mathbb{1}(A(s) = i)$ the number of selections of arm k since the last (global) restart, and $\widehat{\mu}_k(t) \doteq \frac{1}{n_k(t)} \sum_{s=\tau(t)+1}^{t} Y_{k,s} \mathbb{1}(A(s) = i)$ their empirical average (if $n_k(t) \neq 0$).

As explained before, our analysis relies on the general regret decomposition (7.22), with the following appropriate good event. Let d^k be defined as in Assumption 7.7, we define

$$\mathcal{E}_T(\delta) \doteq \left(\forall k \in [\Upsilon_T], \widehat{\tau}^{(i)} \in \left[\tau^{(i)} + 1, \tau^{(i)} + d^{(i)}\right]\right). \tag{7.24}$$

Under the good event, all the change points are detected within a delay at most $d^(k)$. From Assumption 7.7, as the period between two changes are long enough, if $\mathcal{E}_T(\delta)$ holds, then for all change k, we have $\tau^{(i)} \leq \widehat{\tau}^{(i)} \leq \tau^{(i+1)}$. So we can prove the following.

Lemma 7.13. *With $\mathcal{E}_T(\delta)$ defined as in (7.24), the "bad event" is unlikely $\mathbb{P}(\mathcal{E}_T^c(\delta)) \leq \delta(K + 1)\Upsilon_T$.*

We now turn our attention to upper bounding the two terms (A) and (B) in (7.22).

Upper bound on term (A).

$$\begin{aligned}
(A) &\leq \mathbb{E}\left[\mathbb{1}(\mathcal{E}_T) \sum_{t=1}^{T} \mathbb{1}\left(n_{k_t^*}(t)\, \mathrm{kl}\left(\widehat{\mu}_{k_t^*}(t), \mu_{k_t^*}(t)\right) \geq f(t - \tau(t))\right)\right] \\
&\leq \sum_{k=1}^{\Upsilon_T} \mathbb{E}\left[\mathbb{1}(\mathcal{E}_T) \sum_{t=\tau^{(i)}+1}^{\tau^{(i+1)}} \mathbb{1}\left(n_{k^*}(t)\, \mathrm{kl}\left(\widehat{\mu}_{k^*}(t), \mu_{k^*}^{(i)}\right) \geq f(t - \widehat{\tau}(t))\right)\right] \\
&\leq \sum_{k=1}^{\Upsilon_T} \mathbb{E}\left[d^{(i)} + \mathbb{1}(\mathcal{E}_T) \sum_{t=\widehat{\tau}^{(i)}+1}^{\tau^{(i+1)}} \mathbb{1}\left(n_{k^*}(t)\, \mathrm{kl}\left(\widehat{\mu}_{k^*}(t), \mu_{k^*}^{(i)}\right) \geq f(t - \widehat{\tau}^{(i)})\right)\right]
\end{aligned}$$

$$\leq \sum_{k=1}^{\Upsilon_T} d^{(i)} + \sum_{k=1}^{\Upsilon_T} \mathbb{E}\left[\mathbb{1}(\mathcal{C}^{(i)}) \sum_{t=\widehat{\tau}^{(i)}}^{\tau^{(i+1)}} \mathbb{1}\left(n_{k^*}(t)\,\mathrm{kl}\left(\widehat{\mu}_{k^*}(t), \mu_{k^*}\right) \geq f(t-\widehat{\tau}^{(i)})\right)\right],$$

where we introduce the event $\mathcal{C}^{(i)}$ that all the changes up to the k-th have been detected:

$$\mathcal{C}^{(i)} = \left\{\forall j \leq k, \widehat{\tau}^{(j)} \in \{\tau^{(j)}+1, \ldots, \tau^{(j)}+d^{(j)}\}\right\}. \tag{7.25}$$

Clearly, $\mathcal{E}_T \subseteq \mathcal{C}^{(i)}$ and $\mathcal{C}^{(i)}$ is $\mathcal{F}_{\widehat{\tau}^{(i)}}$-measurable. Observe that conditionally to $\mathcal{F}_{\widehat{\tau}^{(i)}}$, when $\mathcal{C}^{(i)}$ holds, $\widehat{\mu}_{k^*}(t)$ is the average of samples that have all mean $\mu_{k^*}^{(i)}$. Thus, introducing $\widehat{\mu}_s$ as a sequence of *i.i.d.* random variables with mean $\mu_{k^*}^{(i)}$, one can write

$$\begin{aligned}
&\mathbb{E}\left[\mathbb{1}(\mathcal{C}^{(i)}) \sum_{t=\widehat{\tau}^{(i)}}^{\tau^{(i+1)}} \mathbb{1}\left(n_{k^*}(t)\,\mathrm{kl}\left(\widehat{\mu}_{k^*}(t), \mu_{k^*}^{(i)}\right) \geq f(t-\widehat{\tau}^{(i)})\right) \middle| \mathcal{F}_{\widehat{\tau}^{(i)}}\right] \\
&= \mathbb{1}(\mathcal{C}^{(i)}) \sum_{t=\widehat{\tau}^{(i)}}^{\tau^{(i+1)}} \mathbb{E}\left[\mathbb{1}\left(n_{k^*}(t)\,\mathrm{kl}\left(\widehat{\mu}_{k^*}(t), \mu_{k^*}^{(i)}\right) \geq f(t-\widehat{\tau}^{(i)})\right) \mid \mathcal{F}_{\widehat{\tau}^{(i)}}\right] \\
&\leq \mathbb{1}(\mathcal{C}^{(i)}) \sum_{t'=1}^{\tau^{(i+1)}-\widehat{\tau}^{(i)}} \mathbb{P}\left(\exists s \leq t' : s\,\mathrm{kl}(\widehat{\mu}_s, \mu_{k^*}^{(i)}) \geq f(t')\right) \\
&\leq \sum_{t=1}^{T} \frac{1}{t\ln(t)} \leq \ln(\ln(T)),
\end{aligned}$$

where the last but one inequality relies on the concentration inequality given in Lemma 2 of [CGM+13] and the choice $f(t) = \ln(t) + 3\ln(\ln(t))$. Finally, using the law of total expectation yields

$$(A) \leq \sum_{k=1}^{\Upsilon_T} \left[d^{(i)} + \ln(\ln(T))\right]. \tag{7.26}$$

Upper bound on term (B). We let $\widetilde{\mu}_{k,s}^{(i)}$ denote the empirical mean of the first s observations of arm k made after time $t = \widehat{\tau}^{(i)} + 1$. Rewriting the sum in t as the sum of consecutive intervals $[\tau^{(i)}+1, \tau^{(i+1)}]$, we obtain

$$\begin{aligned}
(B) &\leq \mathbb{E}\Big[\mathbb{1}(\mathcal{E}_T) \sum_{k=1}^{\Upsilon_T} \sum_{t=\tau^{(i)}}^{\tau^{(i+1)}} \left(\mu_{k^*}^{(i)} - \mu_{A(t)}^{(i)}\right) \mathbb{1}\left(\mathrm{UCB}_{A(t)}(t) \geq \mu_{k^*}^{(i)}\right)\Big] \\
&\leq \sum_{k=1}^{\Upsilon_T} \mathbb{E}\Big[\mathbb{1}(\mathcal{E}_T)\widehat{\tau}^{(i)} + \mathbb{1}(\mathcal{E}_T) \sum_{t=\widehat{\tau}^{(i)}+1}^{\tau^{(i+1)}} \left(\mu_{k^*}^{(i)} - \mu_{A(t)}^{(i)}\right) \mathbb{1}\left(\mathrm{UCB}_{A(t)}(t) \geq \mu_{k^*}^{(i)}\right)\Big] \\
&\leq \sum_{k=1}^{\Upsilon_T} d_k^{(i)} + \sum_{k=1}^{K} \mathbb{E}\Big[\mathbb{1}(\mathcal{E}_T) \sum_{t=\widehat{\tau}^{(i)}+1}^{\tau^{(i+1)}} \left(\mu_{k^*}^{(i)} - \mu_{k}^{(i)}\right) \mathbb{1}\left(A(t) = i, \mathrm{UCB}_k(t) \geq \mu_{k^*}^{(i)}\right)\Big]
\end{aligned}$$

$$\leq \sum_{k=1}^{\Upsilon_T} d_k^{(i)} + \sum_{k=1}^{K} \sum_{k=1}^{\Upsilon_T} \left(\mu_{k^*}^{(i)} - \mu_k^{(i)}\right) \times$$
$$\mathbb{E}\Big[\mathbb{1}(\mathcal{E}_T) \sum_{t=\widehat{\tau}^{(i)}+1}^{\tau^{(i+1)}} \sum_{s=1}^{t-\widehat{\tau}^{(i)}} \mathbb{1}\left(A(t)=i, n_k(t)=s\right) \mathbb{1}\left(s\,\mathrm{kl}(\widetilde{\mu}_{k,s}^{(i)}, \mu_{k^*}^{(i)}) \leq f(\tau^{(i+1)} - \widehat{\tau}^{(i)})\right)\Big]$$

$$\leq \sum_{k=1}^{\Upsilon_T} d_k^{(i)} + \sum_{k=1}^{K} \sum_{k=1}^{\Upsilon_T} \left(\mu_{k^*}^{(i)} - \mu_k^{(i)}\right) \mathbb{E}\Big[\mathbb{1}(\mathcal{E}_T) \sum_{s=1}^{n_k(\tau^{(i+1)})} \mathbb{1}\left(s\,\mathrm{kl}(\widetilde{\mu}_{k,s}^{(i)}, \mu_{k^*}^{(i)}) \leq f(\tau^{(i+1)} - \tau^{(i)})\right)\Big]$$
$$\leq \sum_{k=1}^{\Upsilon_T} d_k^{(i)} + \sum_{k=1}^{K} \sum_{k=1}^{\Upsilon_T} \left(\mu_{k^*}^{(i)} - \mu_k^{(i)}\right) \mathbb{E}\Big[\mathbb{1}(\mathcal{C}^{(i)}) \sum_{s=1}^{n_k(\tau^{(i+1)})} \mathbb{1}\left(s\,\mathrm{kl}(\widetilde{\mu}_{k,s}^{(i)}, \mu_{k^*}^{(i)}) \leq f(\tau^{(i+1)} - \tau^{(i)})\right)\Big].$$

Conditionally to $\mathcal{F}_{\widehat{\tau}^{(i)}}$, when $\mathcal{C}^{(i)}$ holds, for $s \in [n_k(\tau^{(i+1)})]$, $\widetilde{\mu}_{k,s}^{(i)}$ is the empirical mean from *i.i.d.* observations of mean $\mu_k^{(i)}$. Therefore, introducing $\widehat{\mu}_s$ as a sequence of *i.i.d.* random variables with mean $\mu_k^{(i)}$, it follows from the law of total expectation that

$$(B) \leq \sum_{k=1}^{\Upsilon_T} d^{(i)} + \sum_{k=1}^{K} \sum_{k=1}^{\Upsilon_T} \left(\mu_{k^*}^{(i)} - \mu_k^{(i)}\right) \sum_{s=1}^{\tau^{(i+1)}-\tau^{(i)}} \mathbb{P}\left(s \times \mathrm{kl}(\widehat{\mu}_s, \mu_{k^*}^{(i)}) \leq f(\tau^{(i+1)} - \tau^{(i)})\right).$$

If $\mu_{k^*}^{(i)} > \mu_k^{(i)}$ by definition, we can use the analysis of kl-UCB from [CGM+13] to further upper bound the right-most part, and we obtain

$$(B) \leq \sum_{k=1}^{\Upsilon_T} d^{(i)} + \sum_{k=1}^{K} \sum_{k=1}^{\Upsilon_T} \mathbb{1}\left(\mu_k^{(i)} \neq \mu_{k^*}^{(i)}\right) \left[\frac{\left(\mu_{k^*}^{(i)} - \mu_k^{(i)}\right)}{\mathrm{kl}(\mu_k^{(i)}, \mu_{k^*}^{(i)})} \ln(T) + \mathcal{O}\Big(\sqrt{\ln(T)}\Big)\right]. \tag{7.27}$$

Combining the regret decomposition (7.22) with Lemma 7.13 and the two upper bounds of (A) in (7.26) and of (B) in (7.27),

$$R_T \leq 2\sum_{k=1}^{\Upsilon_T} \frac{4K}{\omega\left(\Delta^{(i)}\right)^2} \beta(T,\delta) + \omega T + \delta(K+1)\Upsilon_T + \sum_{k=1}^{\Upsilon_T} \sum_{i:\mu_k^{(i)} \neq \mu_{k^*}^{(i)}} \frac{\left(\mu_{k^*}^{(i)} - \mu_k^{(i)}\right)}{\mathrm{kl}\left(\mu_k^{(i)}, \mu_{k^*}^{(i)}\right)} \ln(T) + \mathcal{O}\Big(\sqrt{\ln(T)}\Big),$$

which concludes the proof. □

Controlling the probability of the good event: proof of Lemma 7.13

Recall that $\mathcal{C}^{(i)}$ defined in (7.25) is the event that all the breakpoints up to the k-th have been correctly detected. Using a union bound, one can write

$$\mathbb{P}(\mathcal{E}_T^c) \leq \sum_{k=1}^{\Upsilon_T} \mathbb{P}\left(\widehat{\tau}^{(i)} \notin \{\tau^{(i)}+1, \dots, \tau^{(i)}+d^{(i)}\} \middle| \, \mathcal{C}^{(i-1)}\right)$$

And thus we have

$$\mathbb{P}(\mathcal{E}_T^c) \leq \sum_{k=1}^{\Upsilon_T} \underbrace{\mathbb{P}\left(\widehat{\tau}^{(i)} \leq \tau^{(i)} \mid \mathcal{C}^{(i-1)}\right)}_{(a)} + \sum_{k=1}^{\Upsilon_T} \underbrace{\mathbb{P}\left(\widehat{\tau}^{(i)} \geq \tau^{(i)} + d^{(i)} \mid \mathcal{C}^{(i-1)}\right)}_{(b)}.$$

The final result follows by proving that $(a) \leq K\delta$ and $(b) \leq \delta$, as detailed below.

Upper bound on (a): controlling the false alarm. We have $\widehat{\tau}^{(i)} \leq \tau^{(i)}$ and it implies that there exists an arm whose associated change point detector has experienced a false-alarm:

$$\begin{aligned}(a) &\leq \mathbb{P}\left(\exists i, \exists s < t \leq n_k(\tau_k^{(i)}) : s \operatorname{kl}\left(\widetilde{\mu}_{i,1:s}^{(i-1)}, \widetilde{\mu}_{i,1:t}^{(i-1)}\right) + (t-s)\operatorname{kl}\left(\widetilde{\mu}_{i,s+1:t}^{(i-1)}, \widetilde{\mu}_{i,1:t}^{(i-1)}\right) > \beta(t,\delta) \mid \mathcal{C}^{(i-1)}\right) \\ &\leq \sum_{k=1}^{K} \mathbb{P}\left(\exists s < t : s \operatorname{kl}(\widehat{\mu}_{1:s}, \mu_k^{(i-1)}) + (t-s)\operatorname{kl}(\widehat{\mu}_{s+1:t}, \mu_k^{(i-1)}) > \beta(t,\delta)\right),\end{aligned}$$

with $\widehat{\mu}_{s:s'} = \sum_{r=s}^{s'} Z_{i,r}$ where $Z_{i,r}$ is an *i.i.d.* sequence with mean $\mu_k^{(i-1)}$. Indeed, conditionally to $\mathcal{C}^{(i-1)}$, the $n_k(\tau^{(i)})$ successive observations of arm k arm starting from $\widehat{\tau}^{(i)}$ are *i.i.d.* with mean $\mu_k^{(i-1)}$. Using Lemma 7.4, term (a) is upper bounded by $K\delta$. □

Upper bound on term (b): controlling the delay. From the definition of $\Delta^{(i)}$, there exists an arm k such that $\Delta^{(i)} \doteq |\mu_k^{(i)} - \mu_k^{(i-1)}|$. We shall prove that it is unlikely that the change-point detector associated to i doesn't trigger within the delay $d^{(i)}$.

First, it follows from Proposition 7.6 that there exists $\overline{t} \in \{\tau^{(i)}, \dots, \tau^{(i)} + d^{(i)}\}$ such that $n_k(\overline{t}) - n_k(\tau^{(i)}) = \overline{r}$, where $\overline{r} \doteq \lfloor \frac{\omega}{K} d^{(i)} \rfloor$ (as the mapping $t \mapsto n_k(t) - n_k(\tau^{(i)})$ is non-decreasing, is 0 at $t = \tau^{(i)}$ and its value at $\tau^{(i)} + d^{(i)}$ is larger than $\overline{r}$). Using that $\left(\widehat{\tau}^{(i)} \geq \tau^{(i)} + d^{(i)}\right) \subseteq \left(\widehat{\tau}^{(i)} \geq \overline{t}\right)$ the event $\left(\widehat{\tau}^{(i)} \geq \tau^{(i)} + d^{(i)}\right)$ further implies that

$$n_k(\tau^{(i)}) \operatorname{kl}\left(\widetilde{\mu}_{i,n_k(\tau^{(i)})}^{k-1}, \widetilde{\mu}_{i,n_k(\overline{t})}^{k-1}\right) + \overline{r} \operatorname{kl}\left(\widetilde{\mu}_{i,n_k(\tau^{(i)}):n_k(\overline{t})}^{k-1}, \widetilde{\mu}_{i,n_k(\overline{t})}^{k-1}\right) \leq \beta(n_k(\tau^{(i)}) + \overline{r}, \delta),$$

where $\widetilde{\mu}_{k,s}^{k-1}$ denotes the empirical mean of the s first observation of arm k since the $(k-1)$-th restart $\widehat{\tau}^{(i-1)}$ and $\widetilde{\mu}_{i,s:s'}^{k-1}$ the empirical mean that includes observation number s to number s'.

Conditionally to $\mathcal{C}^{(i-1)}$, $\widetilde{\mu}^{k-1}_{i,n_k(\tau^{(i)})}$ is the empirical mean of $n_k(\tau^{(i)})$ *i.i.d.* replications of mean μ^{k-1}, whereas $\widetilde{\mu}^{k-1}_{i,n_k(\tau^{(i)}):n_k(\bar{t})}$ is the empirical mean of $\bar{r}$ *i.i.d.* replications of mean μ^k_k.

Moreover, Proposition 7.6 shows that $n_k(\tau^{(i)})$ is bounded in a certain interval, $n_k(\tau^{(i)}) \in \left[\left\lfloor \frac{\omega}{K}\left(\tau^{(i)} - \widehat{\tau}^{(i-1)}\right)\right\rfloor, \left(\tau^{(i)} - \widehat{\tau}^{(i-1)}\right)\right]$. Conditionally to $\mathcal{C}^{(i-1)}$, by using Assumption 7.7, we can prove that $d^{(i-1)} \leq (\tau^{(i)} - \tau^{(i-1)})/2$, and thus we obtain furthermore that

$$n_k(\tau^{(i)}) \in \left\{\left\lfloor \frac{\omega}{2K}(\tau^{(i)} - \tau^{(i-1)})\right\rfloor, \ldots, \tau^{(i)} - \tau^{(i-1)}\right\} \doteq \mathcal{I}_k.$$

Introducing $\widehat{\mu}_{a,s}$ (resp. $\widehat{\mu}_{b,s}$) the empirical mean of s *i.i.d.* observations with mean $\widehat{\mu}^{(i-1)}_k$ (resp. $\widehat{\mu}^{(i)}_k$), such that $\widehat{\mu}_{a,s}$ and $\widehat{\mu}_{b,r}$ are independent, it follows that

$$(b) \leq \mathbb{P}\left(\exists s \in \mathcal{I}_k : s\,\mathrm{kl}\left(\widehat{\mu}_{a,s}, \frac{s\widehat{\mu}_{a,s} + \bar{r}\widehat{\mu}_{b,\bar{r}}}{s+\bar{r}}\right) + \bar{r}\,\mathrm{kl}\left(\widehat{\mu}_{b,\bar{r}}, \frac{s\widehat{\mu}_{a,s} + \bar{r}\widehat{\mu}_{b,\bar{r}}}{s+\bar{r}}\right) \leq \beta(s+\bar{r},\delta)\right),$$

where we have also used that $\widetilde{\mu}^{k-1}_{i,n_k(\bar{t})} = \left(n_k(\tau^{(i)})\widetilde{\mu}^{k-1}_{i,n_k(\tau^{(i)})} + \bar{r}\widetilde{\mu}^{k-1}_{i,n_k(\tau^{(i)}):n_k(\bar{t})}\right)/(n_k(\tau^{(i)}) + \bar{r})$.

Using Pinsker's inequality, and introducing the gap $\Delta^{(i)}_k \doteq \mu^{(i-1)}_k - \mu^{(i)}_k$ (which is such that $\Delta^{(i)} = |\Delta^k_k|$), one can write

$$\begin{aligned}(b) &\leq \mathbb{P}\left(\exists s \in \mathcal{I}_k : \frac{2s\bar{r}}{s+\bar{r}}\left(\widehat{\mu}_{a,s} - \widehat{\mu}_{b,\bar{r}}\right)^2 \leq \beta(s+\bar{r},\delta)\right)\\ &\leq \mathbb{P}\left(\exists s \in \mathbb{N} : \frac{2sr}{s+r}\left(\widehat{\mu}_{a,s} - \widehat{\mu}_{b,s} - \Delta^{(i)}_k\right)^2 \geq \beta(s+r,\delta)\right)\\ &+ \mathbb{P}\left(\exists s \in \mathcal{I}_k : \frac{2s\bar{r}}{s+\bar{r}}\left(\widehat{\mu}_{a,s} - \widehat{\mu}_{b,\bar{r}} - \Delta^{(i)}_k\right)^2 \leq \beta(s+\bar{r},\delta), \frac{2s\bar{r}}{s+\bar{r}}\left(\widehat{\mu}_{a,s} - \widehat{\mu}_{b,\bar{r}}\right)^2 \leq \beta(s+\bar{r},\delta)\right)\end{aligned}$$

Using Lemma 7.14 stated in Appendix 7.10.2, and a union bound, the first term in the right hand side is upper bounded by δ (as $\beta(r+s,\delta) \geq \beta(r,\delta) \geq \ln(3s\sqrt{s}/\delta)$). For the second term, we use the observation

$$\frac{2s\bar{r}}{s+\bar{r}}\left(\widehat{\mu}_{a,s} - \widehat{\mu}_{b,\bar{r}} - \Delta^{(i)}_k\right)^2 \leq \beta(s+\bar{r},\delta) \;\Rightarrow\; |\widehat{\mu}_{a,s} - \widehat{\mu}_{b,\bar{r}}| \geq |\Delta^{(i)}_k| - \sqrt{\frac{s+\bar{r}}{2\bar{r}s}\beta(s+\bar{r},\delta)}$$

and, using that $\Delta^{(i)} = |\Delta^k_k|$, one obtains

$$(b) \leq \delta + \mathbb{P}\left(\exists s \in \mathcal{I}_k : \Delta^{(i)} \leq 2\sqrt{\frac{s+\bar{r}}{2s\bar{r}}\beta(s+\bar{r},\delta)}\right). \tag{7.28}$$

Define $s_{\min} \doteq \left\lfloor \frac{\omega}{K}(\tau^{(i)} - \tau^{(i-1)})/2\right\rfloor$. Using that the mappings $s \mapsto (s+\bar{r})/s\bar{r}$ and $s \mapsto \beta(s+\bar{r},\delta)$ are respectively decreasing and increasing in s, one has, for all $s \in \mathcal{I}_k$,

$$2\frac{s+\bar{r}}{s\bar{r}}\beta\left(s+\bar{r},\delta\right) \quad\leq\quad 2\frac{s_{\min}+\bar{r}}{s_{\min}\bar{r}}\beta\left(T,\delta\right) \leq \frac{4\beta(T,\delta)}{\left\lfloor\frac{\omega}{K}d^{(i)}\right\rfloor},$$

where the last inequality follows from the fact that $\overline{r} \leq s_{\min}$ as $d^{(i)} \leq (\tau^{(i)} - \tau^{(i-1)})/2$ by Assumption 7.7. Now the definition of $d^{(i)}$ readily implies that $\left\lfloor \frac{\omega}{K} d^{(i)} \right\rfloor > \frac{4\beta(T,\delta)}{\left(\Delta^{(i)}\right)^2}$, which yields

$$\forall s \in \mathcal{I}_k, \;\; 2\frac{s+\overline{r}}{s\overline{r}}\beta\left(s+\overline{r},\delta\right) \leq \left(\Delta^{(i)}\right)^2.$$

Hence, the probability in the right-hand side of (7.28) is zero, which yields $(b) \leq \delta$. □

7.8 Experimental results for piece-wise stationary bandit problems

In this section we report results of numerical simulations performed on synthetic data to compare the performance of GLR-klUCB against other state-of-the-art approaches, on some piece-wise stationary bandit problems. For simplicity, we restrict to rewards generated from Bernoulli distributions, even if GLR-klUCB can be applied to any bounded distributions.

We report results obtained on five different piece-wise stationary bandit problems, illustrated in Figures 7.1 and 7.2 above, and Figures 7.5, 7.6 and 7.7 below. We present regret tables and regret plots, for which the regret was estimated using 1000 independent runs.

Algorithms and parameters tuning. We include in this study two algorithms designed for the classical MAB, kl-UCB [GC11], and Thompson sampling [AG12, KKM12], as well as an "oracle" version of kl-UCB, that we call Oracle-Restart. This algorithm knows the exact locations of the breakpoints, and restarts kl-UCB at those locations (without any delay).

Then, we compare our algorithms to several competitor designed for a piece-wise stationary model. For a fair comparison, all algorithms that use UCB as a sub-routine were adapted to use kl-UCB instead, which yields better performance[3]. Moreover, all the algorithms are tuned as described in the corresponding paper, using in particular the knowledge of the number of breakpoints Υ_T and the horizon T. We first include three *passively adaptive algorithms*: Discounted kl-UCB (D-kl-UCB, [KS06]), with discount factor $\gamma = 1 - \sqrt{\Upsilon_T/T}/4$; Sliding-Window kl-UCB (SW-kl-UCB, [GM11]) using window-size $\tau = 2\sqrt{T\ln(T)/\Upsilon_T}$ and Discounted Thompson sampling (DTS, [RK17]) with discount factor $\gamma = 0.95$. For this last algorithm, the discount factor $\gamma = 0.75$ suggested by the authors was performing significantly worse on the problem instances we tried. More precisely, we found that $\gamma \leq 0.95$ gives better performances for short-term problems (pbs $1, 2, 4$) and $\gamma \geq 0.95$ is better suited for long experiments (pbs $3, 5$). Additionally, we include the Exp3.S algorithm from [ACBFS02], setting its parameters as in Corollary 8.3 of the paper, based on a prior knowledge of T and Υ_T, using $\omega = 1/T$ and $\gamma = \min(1, \sqrt{K(\Upsilon_T\ln(KT)+\mathrm{e})/((\mathrm{e}-1)T)})$.

[3][LLS18, CZKX19] both mention that extending their analysis to the use of kl-UCB should not be too difficult.

Our main goal is to compare against *actively adaptive algorithms*. We include CUSUM-klUCB, tuned with $M = 150$ and $\varepsilon = 0.1$ for easy problems $(1, 2, 4)$ and $\varepsilon = 0.001$ for hard problems $(3, 5)$, and with $h = \ln(T/\Upsilon_T)$, $\omega = \sqrt{\Upsilon_T \ln(T/\Upsilon_T)/T}$, as suggested in the paper [LLS18]. Finally, we include M-klUCB, tuned with $w = 150$, based on a prior knowledge of the problems as the formula using $\delta_{\min}$ given in [CZKX19] is too large for small horizons (on all our problem instances), a threshold $b = \sqrt{w \ln(2KT)}$ and $\gamma = \sqrt{\Upsilon_T K(2b + 3\sqrt{w})/(2T)}$ as suggested by Remark 4 in the paper [CZKX19].

For GLR-klUCB, we explore the two different options with **Local** and **Global** restarts, using respectively $\delta = 1/\sqrt{\Upsilon T}, \omega = \omega_0\sqrt{\Upsilon_T \ln(T)/T}$ and $\delta = 1/\sqrt{K\Upsilon_T T}, \omega = \omega_0\sqrt{K\Upsilon_T \ln(T)/T}$ from Corollaries 7.9 and 7.12. Both choices of ω appear to be too large empirically, as they come from minimizing the regret upper-bound rather than the regret itself, thus the constant is set to $\omega_0 = 0.05$. We show in Appendix 7.10.5 a certain robustness, with similar regret as soon as $\omega \leq 0.1$. We do not use a constant δ_0 to change the confidence level to $\delta = \delta_0\delta_T$, as we show in Appendix 7.10.6 that $\delta_0 = 1$ is uniformly better for different problems. To speed up the simulations, two optimizations are used, with $\Delta n = \Delta s = 10$, and CUSUM also uses the first trick with $\Delta n = 10$ (see Appendix 7.10.4 for more details).

Examples of detection delays *on one run* for two simple problems. Before giving larger results that compare our approach against other algorithms, we consider the two problems 1 and 2 presented above in Figures 7.1 and 7.2 (for $T = 5000$). *For one (random) simulation*, we give below examples of the efficiency of the change-point detection algorithm used by GLR-klUCB as well as CUSUM-klUCB and M-klUCB, by showing the times at which they detect a change and reinitialize their memory of observations of one or all the arms. We also display the number of observations (since the last restart) of the arm on which the change was detected. As shown below, in the experiments summary, the three change-point detection algorithm give similar performances in terms of regret, but they have different behaviors. The test in M-UCB usually detects less changes, while the test in CUSUM-UCB usually detects more changes and even unnecessary changes. On the examples presented below as well as large scale examples, all the tests appear to have no false alarm, and always a small detection delay for changes that are "easy enough" to be efficiently detected. The four algorithms are presented in the order of decreasing performance (*i.e.*, the first one is empirically the most efficient, that is, it obtains a smaller regret in average).

Problem 1 has only local changes, and important changes, that denote here changes on the current optimal arm, happen only at $t = 2000$ when arm 2 becomes sub-optimal (μ_2 goes from 0.9 to 0.1) and arm 0 becomes optimal, and at $t = 3000$ when arm 0 stays optimal but sees its mean change from 0.3 to 0.7. Other changes concern arm 1 at $t = 1000$ and $t = 4000$.

Figure 7.3 above displays the locations of the successively detected change-points by four algorithms, 1) M-klUCB, 2) CUSUM-klUCB, 3) GLR-klUCB with *local restart*, 4) GLR-klUCB

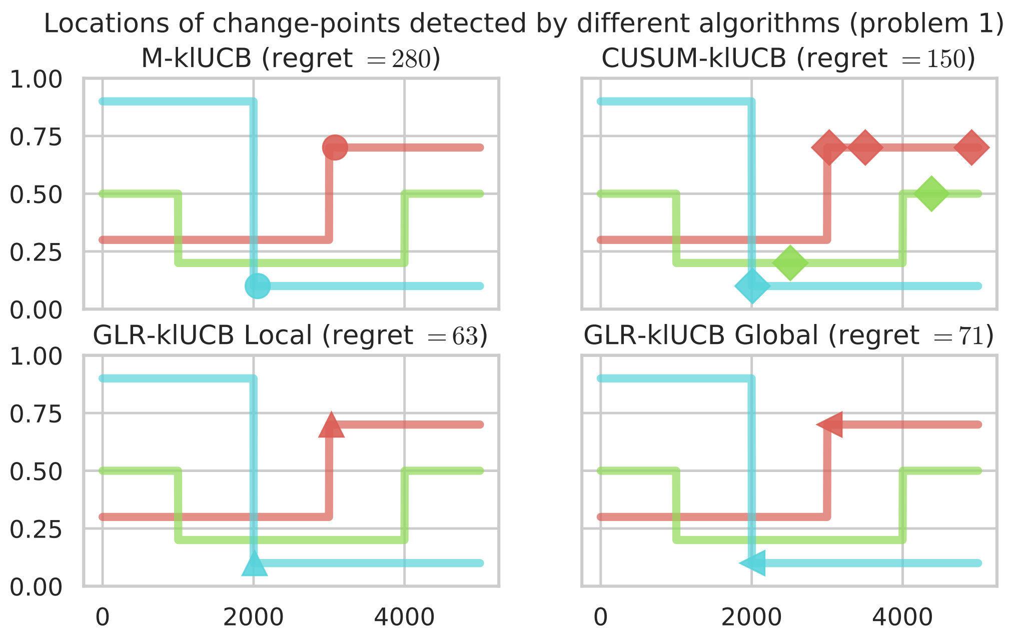


Figure 7.3 – Locations of the detected change-points for four algorithms on Problem 1.

with *global restart*. In this "easy" problem, all algorithms correctly detect the important changes, except CUSUM which detected a change on arm 1 (in red) twice after the change-point located at $t = 3000$. The three other algorithms have a very small delay, for instance only 9 samples from arm 0 (in blue) after its change at time $t = 2000$ are enough for the GLR test to detect a change. Very small delays are possible only after having collected a lot of samples of the arm which changed, and for instance the same algorithm obtains a delay of 32 samples from arm 1 for the next change at $t = 3000$, because this arm was sampled less. On this problem, while M-klUCB detects the same changes as GLR-klUCB, with larger but comparable delays, it is seen to obtain a larger regret than the two GLR variants, because the tuning of its forced exploration probability makes it explore suboptimal arms more often.

Problem 2 has only global changes, and important changes happen only at $t = 1000$ when arm 2 sees its mean change from 0.9 to 0.7, then at $t = 2000$ when it becomes sub-optimal (μ_2 goes from 0.7 to 0.5) and arm 0 becomes optimal, and at $t = 3000$ and $t = 4000$ when arm 0 stays optimal but sees its mean change from 0.6 to 0.7 and then from 0.7 to 0.8.

Like for the first problem, we illustrate the behavior of the same four algorithms in Figure 7.4 above. In this other problem, the difference between M-klUCB and CUSUM-klUCB is clear: the first algorithm fails to detect any change, leading to a large regret, while the second one detects the changes that happen on the currently optimal arm and some other changes. Drawing conclusions on their behavior from a single simulation is tricky, as this example of behavior is counter intuitive: On the one hand, CUSUM uses local restarts, and it should be less efficient than global restarts for this problem, as changes are all global. On the other hand,

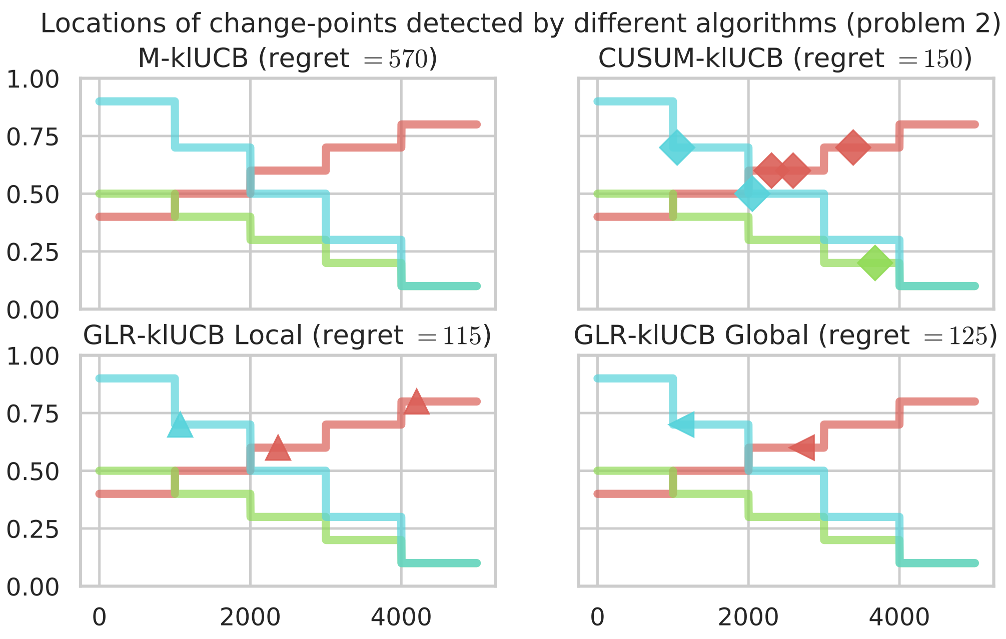


Figure 7.4 – Locations of the detected change-points for four algorithms on Problem 2.

M-klUCB uses global restarts, but here it fails to detect any changes. The two GLR-klUCB variants correctly detect the important changes, and we observe that their delays to detect a change can vary, mainly due to the randomness (we remind that we illustrate only one repetition of the simulation here). For instance, after the change on arm 1 (in red), the *local restart* detects it at time $t = 2376$ and the *global restart* detects it at time $t = 2701$.

In these two examples, we compare the two variants of our proposal with the two state-of-the-art policies CUSUM-klUCB and M-klUCB, that are all based on combining kl-UCB with a change-point detection algorithm. The results given above should be taken carefully, they only have the purpose of being an illustration of the possible behaviors of these algorithms, as they are the results of *only one (random) simulation*! But it is still interesting to compare the final regret in one run, as given above, with the mean regret for 1000 independent runs, as given below in Table 7.2. The values are not the same, but the ranking is: our proposal outperforms both CUSUM-klUCB and M-klUCB, and the test based on CUSUM seems more efficient than the test based of M-klUCB in the problems at hand.

Illustrations of the rest of the benchmark. We continue here the presentation of the other problems of our benchmark, see Figures 7.1 and 7.2 above for the two problems 1 and 2.

Problem 3. (see Figure 7.5) This problem is harder, with $K = 6$, $\Upsilon_T = 8$ and $T = 20000$. At every break-points, almost all arms change, and means are bounded in $[0.01, 0.07]$. The gaps Δ are much smaller than for the first problems, with amplitudes ranging from 0.02 to

0.001. Note that the assumptions of the regret upper bounds for GLR-klUCB are violated, as well as the assumptions for the analysis of M-UCB and CUSUM-UCB. It is interesting to say that this problem is inspired from Figure 3 of [CZKX19], where the synthetic data was obtained from manipulations on a real-world database of clicks from *Yahoo!*.

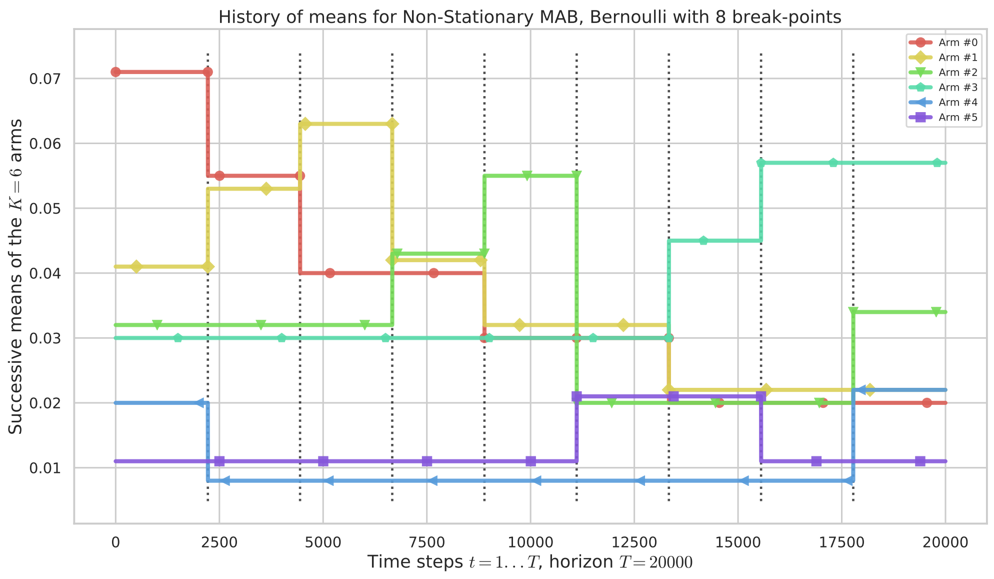


Figure 7.5 – Problem 3: $K = 6, T = 20000, C = 19$ changes occur on most arms at $\Upsilon = 8$ break-points.

Problem 4. Like problem 1, it uses $K = 3$ arms, $\Upsilon = 4$ change-points and $T = 5000$, but the stationary sequences between successive change-points no longer have the same length, as illustrated in Figure 7.6. Classical (stationary) algorithms such as kl-UCB can be "tricked" by large enough stationary sequences, as they learn with a large confidence the optimal arm, and then fail to adapt to a new optimal arm after a change-point. We observe below in Table 7.3 that they can suffer higher regret when the change-points are more spaced out, as this problem starts with a longer stationary sequence of length $T/2$.

Problem 5. Like problem 3, this harder problem is also inspired from synthetic data obtained from a real-world database of clicks from *Yahoo!*, but from another competitor paper, see Figure 3 from [LLS18]. It is much harder, with $\Upsilon = 81$ change-points on $K = 5$ arms for a longer horizon of $T = 100000$. Some arms change at almost every time steps, for a total number of breakpoints $C = 179$, but the optimal arm is almost always the same one (arm 0, with ●). It is a good benchmark to see if the actively adaptive policies do not detect *too many changes*, as the Oracle-Restart policy suffers higher regret in comparison to kl-UCB. Means are also bounded in $[0.01, 0.07]$, with small gaps of amplitude in $[0.001, 0.02]$, as shown in Figure 7.7.

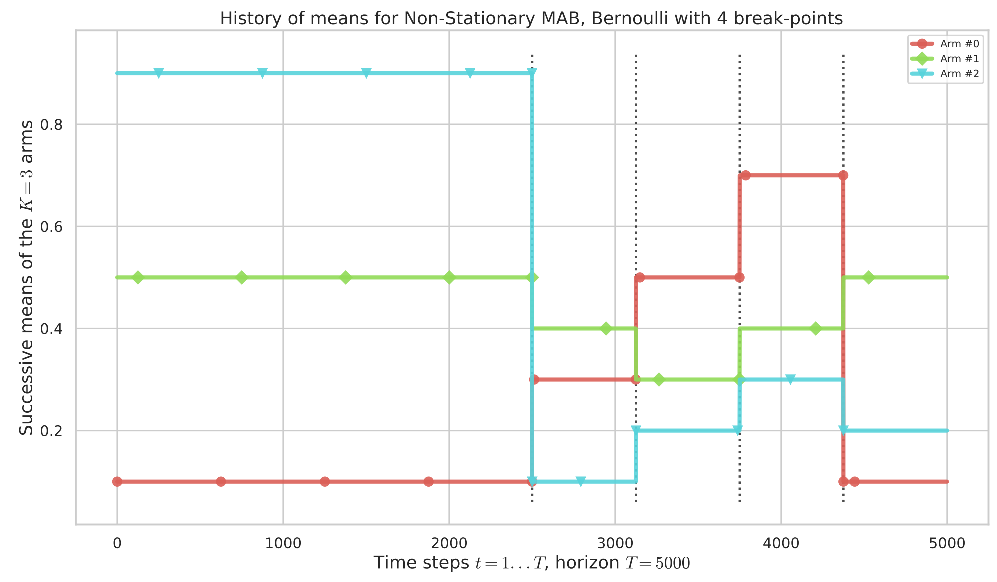


Figure 7.6 – Problem 4: $K = 3$, $T = 5000$, $C = 12$ changes occur on all arms at $\Upsilon = 4$ break-points.

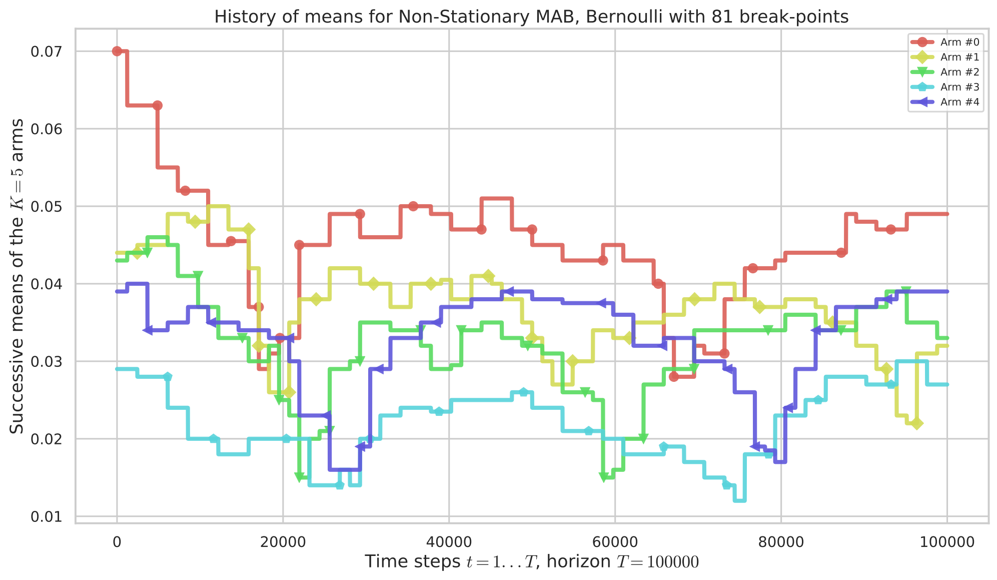


Figure 7.7 – Pb 5: $K = 5$, $T = 100000$, $C = 179$ changes occur on some arms at $\Upsilon = 81$ break-points.

First experiment: UCB **vs** kl-UCB**.** Similarly to what is presented in Chapter 6 in Table 6.1, we start by some experiments that justify the focus on the kl-UCB index policy. The purpose of this work is not to optimize on the index policy, but rather propose new ways of using indices for piece-wise stationary problems. Consider two experiments on **problems 1** and **2**, with a horizon of $T = 20000$ and 1000 independent repetitions, for which we compare CUSUM-UCB against CUSUM-klUCB, M-UCB against M-klUCB, and the oracle policy using UCB against kl-UCB. We report in Table 7.1 below the results (in terms of mean regret), and we observe that using kl-UCB rather than UCB indices always yields better practical performance. Thus it is fair to compare our proposal against the modified versions of the two algorithms CUSUM and M- that use the kl-UCB index policy instead of UCB, as they perform uniformly better with kl-UCB than with UCB (the same tendency was observed on all other problems). We also note that, without surprise, the oracle policy also performs (much) better with kl-UCB than with UCB. Consequently, from now on we only report results for kl-UCB.

Algorithm	**Index policy**	**Problem 1**	**Problem 2**
Oracle-Restart	UCB	216	219
	kl-UCB	**54**	**67**
M-	UCB	878	2040
	kl-UCB	817	1485
CUSUM	UCB	439	381
	kl-UCB	304	316
GLR (Local)	UCB	191	232
	kl-UCB	**132**	**186**

Table 7.1 – Mean regret $\pm$ 1 std-dev, on problems 1 and 2 with $T = 5000$. We conclude that using kl-UCB is much more efficient than using UCB, for non-stationary bandit.

Results. The two Tables 7.2 and 7.3 show the final regret R_T obtained for each algorithm. Results highlighted in **bold** show the best non-oracle algorithm for each experiment, with our proposal being the best non-oracle strategy for problems 1 and 2. Thompson sampling and kl-UCB are efficient, and better than Discounted-kl-UCB which is inefficient. DTS and SW-kl-UCB can sometimes be more efficient than their stationary counterparts, but perform worse than the Oracle and most actively adaptive algorithms. For kl-UCB indexes, M- and CUSUM- outperform the previous algorithms, but GLR- is often better. On these problems, GLR-klUCB with **Local** restarts is always more efficient than with **Global** restarts. Note that on problem 2, all means change at every breakpoint, hence one could expect the Global variant to be more efficient, yet the experiments show the superiority of Location variant on every instance. The Exp3.S algorithm was found to outperform other algorithms based on Exp3, including recent variants like Exp3++ [SL17] or Exp3.R from [AFM17]. Exp3.S usually performs similarly to M-klUCB, but it is greatly outperformed by the oracle algorithm, by GLR-klUCB and by CUSUM-klUCB.

We highlight that the best non-oracle strategies are actively adaptive, thus the experiments confirm that an efficient bandit algorithm (*e.g.*, kl-UCB) combined with an efficient change point detector (*e.g.*, GLR) provides efficient strategies for the piece-wise stationary model.

Algorithms \ **Problems**	Pb 1	Pb 2 ($T = 5000$)	Pb 3 ($T = 20000$)
Oracle-Restart kl-UCB	$\mathbf{37 \pm 37}$	$\mathbf{45 \pm 34}$	$\mathbf{257 \pm 86}$
Exp3.S	352 ± 51	310 ± 62	665 ± 93
kl-UCB	270 ± 76	162 ± 59	529 ± 148
Discounted-kl-UCB	1456 ± 214	1442 ± 440	1376 ± 37
SW-kl-UCB	177 ± 34	182 ± 34	1794 ± 71
Thompson sampling	493 ± 175	388 ± 147	1019 ± 245
DTS	209 ± 38	249 ± 39	2492 ± 52
M-klUCB	290 ± 29	534 ± 93	645 ± 141
CUSUM-klUCB	148 ± 32	152 ± 42	$\mathbf{490 \pm 133}$
GLR-klUCB(Local)	$\mathbf{74 \pm 31}$	$\mathbf{113 \pm 34}$	513 ± 97
GLR-klUCB(Global)	97 ± 32	134 ± 33	621 ± 103

Table 7.2 – Mean regret $\pm$ 1 std-dev, on problems 1, 2 (with $T = 5000$) and 3 ($T = 20000$).

Algorithms \ **Problems**	Pb 4 ($T = 5000$)	Pb 4 ($T = 10000$)	Pb 5
Oracle-Restart kl-UCB	$\mathbf{68 \pm 40}$	$\mathbf{86 \pm 50}$	126 ± 54
Exp3.S	551 ± 63	860 ± 109	723 ± 121
kl-UCB	615 ± 74	1218 ± 123	106 ± 36
SW-kl-UCB	202 ± 33	322 ± 47	228 ± 27
Discounted-kl-UCB	911 ± 210	1741 ± 200	2085 ± 910
Thompson sampling	756 ± 65	1476 ± 137	$\mathbf{88 \pm 39}$
DTS	250 ± 39	481 ± 58	238 ± 24
M-klUCB	337 ± 46	544 ± 47	116 ± 36
CUSUM-klUCB	267 ± 69	343 ± 94	117 ± 34
GLR-klUCB(Local)	$\mathbf{99 \pm 32}$	$\mathbf{128 \pm 42}$	149 ± 34
GLR-klUCB(Global)	128 ± 32	185 ± 47	152 ± 32

Table 7.3 – Mean regret $\pm$ 1 std-dev. Problem 4 use $K = 3$ arms, and a first long stationary sequence. Problem 5 use $K = 5$, $T = 100000$ and is much harder with $\Upsilon = 82$ breakpoints and $C = 179$ changes.

Regret plots. We show below on Figure 7.8 and additional in Appendix 7.10.1 the simulation results for the five problems. The results in terms of mean regret R_T are given above in Tables 7.2 and 7.3, but it is also interesting to observe two plots for each experiment. First, we show the mean regret as a function of time (*i.e.*, R_t for $1 \leq t \leq T$), for 9 of the considered algorithms (as they are outperformed by the others, Exp3.S and Discounted-kl-UCB are not included to avoid clutter). Efficient stationary algorithms, like TS and kl-UCB, typically suffer a linear regret after a change on the optimal arm changes if they had "too" many samples before the change-points (*e.g.*, on Figure 7.8 and even more on Figures 7.11 and 7.13). This illustrates the conjecture that classical algorithms can suffer linear regret even on simple piece-wise stationary problems.

On simple problems, like problem 1, all the algorithms being designed for piece-wise stationary environments perform similarly, but as soon as the gaps are smaller or there are more changes, we clearly observe that our approach GLR-klUCB can outperform the two other

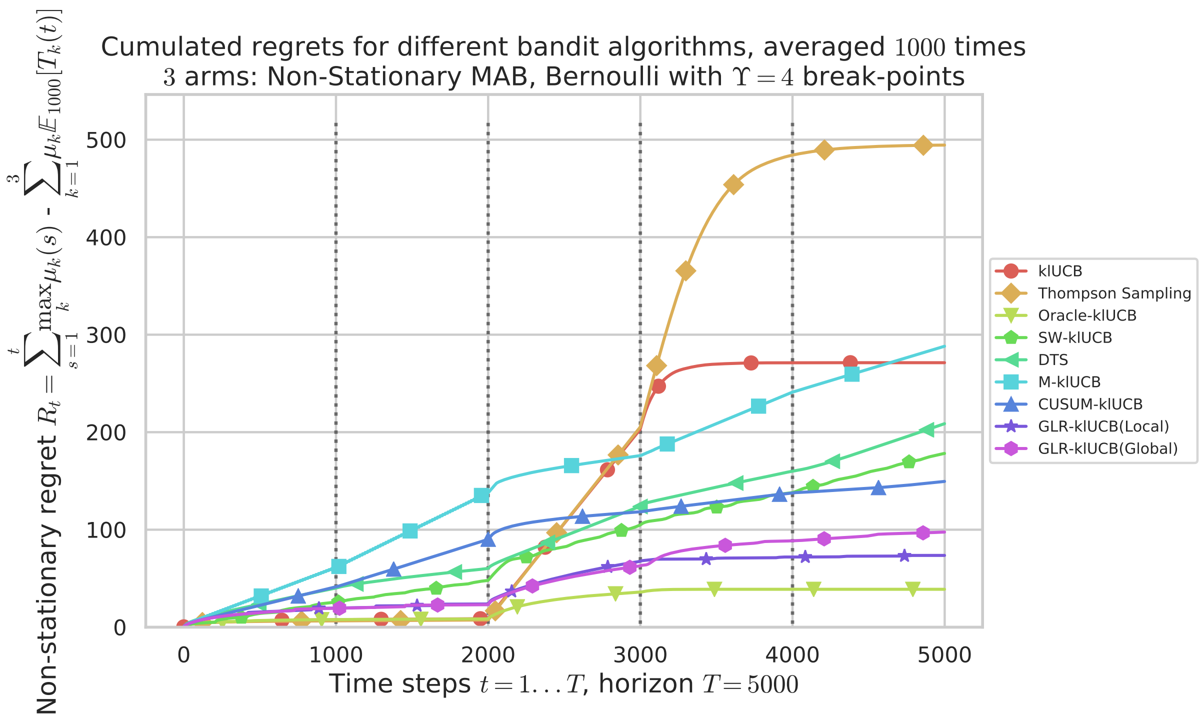


Figure 7.8 – Mean regret as a function of time, R_t ($1 \leq t \leq T = 5000$) for problem 1.

actively adaptive algorithms CUSUM-klUCB and M-klUCB (*e.g.*, on Figure 7.10), and performs much better than passively adaptive algorithms DTS and SW-kl-UCB (*e.g.*, on Figure 7.12). Our approach is the algorithm which performs the closer to the oracle for problem 4.

Finally, in the case of hard problems, like problems 3 and 5, that have a lot of changes but where the optimal arm barely changes, we verify in Figure 7.13 that kl-UCB and TS can outperform the oracle policy. Indeed the oracle policy is suboptimal as it restarts as soon as one arm change but is unaware of the meaningful changes, and stationary policies which quickly identify the best arm will play it most of the times, achieving a smaller regret. We note that, sadly, all actively adaptive policies fail to outperform stationary policies on such hard problems, because they do not observe enough rewards from each arm between two restarts (*i.e.*, the Assumptions 7.7 and 7.10 for the two Theorems 7.8 and 7.11 are not satisfied). We can also verify that the two options, **Local** and **Global** restart, for GLR-klUCB, give close results, and that the **Local** option is always better.

We also show the empirical distribution of the regret R_T, on Figure 7.9. It shows that all algorithms have a rather small variance on their regret, except Thompson Sampling which has a large tail due to its large mean regret on this (easy) non-stationary problems.

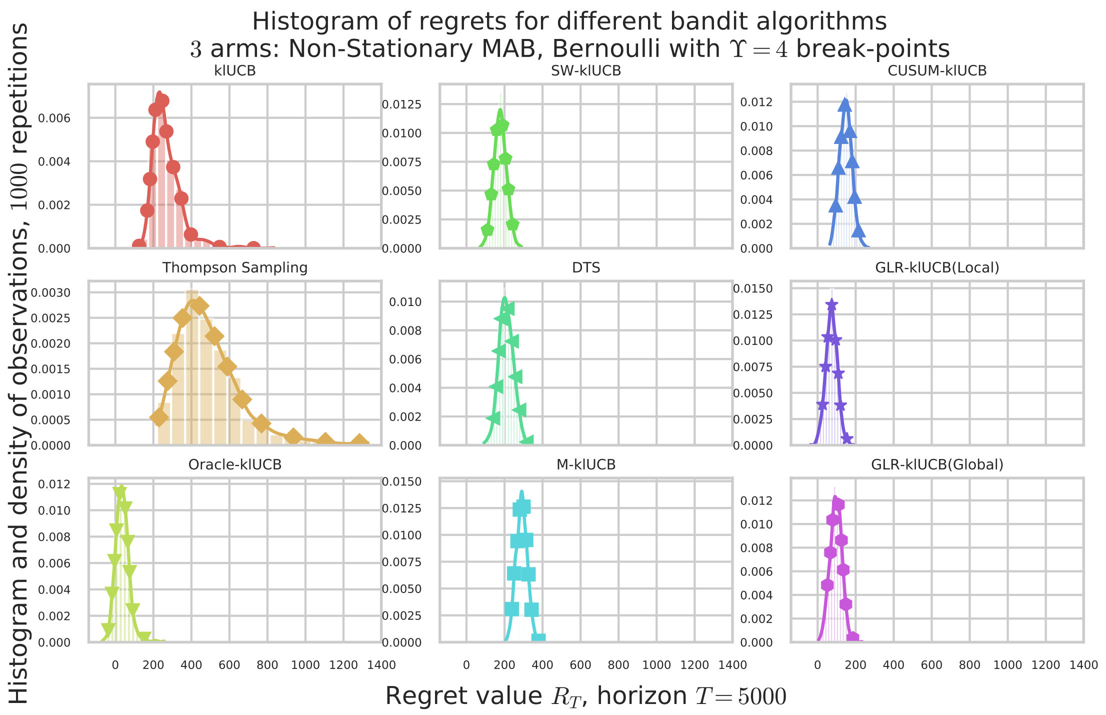


Figure 7.9 – Histograms of the distributions of regret R_T ($T = 5000$) for problem 1.

7.9 Conclusion

In this chapter, we studied and presented the piece-wise stationary bandit model, and reviewed existing work. We proposed a new algorithm for this problem, GLR-klUCB, which combines the kl-UCB algorithm with the Bernoulli GLR change-point detector. This actively adaptive method attains state-of-the-art regret upper-bounds when tuned with a prior knowledge of the number of changes Υ_T, but *without any other prior knowledge on the problem*, unlike CUSUM-UCB and M-UCB that require to know a lower bound on the smallest magnitude of a change. We also gave numerical evidence of the efficiency of our proposal.

We believe that our new proof technique could be used to analyze GLR-klUCB under less stringent assumptions than the one made in this chapter (and in previous work), that would require only a few "meaningful" changes to be detected. This interesting research direction is left for future work, but the hope is that the regret would be expressed in term of this number of meaningful changes instead of Υ_T. We shall also investigate whether actively adaptive approaches can attain a $\mathcal{O}(\sqrt{\Upsilon_T T})$ regret upper-bound without the knowledge of Υ_T. We also believe that combining change-point *localization* with an efficient change-point detection algorithm, such as GLR-klUCB, could lead to an interesting class of algorithms, as suggested by [Mai19]. Finally, we would like to study in the future possible extension of our approach to the slowly varying model, as studied in [BGZ14, WS18b].

As mentioned at the end of Chapter 6, another interesting direction of future work is to study non-stationary distributed multi-players bandits. A natural extension of the non-stationary model presented in this Chapter is to consider non-communicating players cooperating in a decentralized way to play the same bandit game, as it was proposed recently in [WS18a]. The authors build on their recent work [WS18b] and show that a regret bounded by $\mathcal{O}(\sqrt{T^{\frac{1+\nu}{2}}\ln(T)})$ can still be achieved by M players, in the number of breaking points is $\Upsilon_T = \mathcal{O}(T^{\nu})$. Their algorithm assume that player j knows its ID $j \in [M]$ as well as ν, and removing these hypotheses is an interesting direction of future work. Furthermore, a promising direction is to directly try to join our contributions from Chapters 6 and 7, and propose an efficient algorithm using three parts: kl-UCB indexes for arm selection, MCTopM for orthogonalization (*i.e.*, dealing with collisions), and GLR-klUCB for non-stationarity (*i.e.*, dealing with abrupt changes).

Reproducility of the experiments. The experiments in this chapter use our library SMPyBandits, as presented in Chapter 3, and we refer to the following page for details on how to reproduce them, `SMPyBandits.GitHub.io/NonStationaryBandits.html`

7.10 Appendix

We start by including figures as complementary illustrations to the numerical results presented in Section 7.8 above. We then give some additional useful results, along with their proofs, and then we give complementary numerical experiments to justify some choices made in the presentation of our proposal GLR-klUCB.

7.10.1 Additional figures

We give here plots showing the mean regret R_t as a function of time, and histograms of the distributions of the final regret R_T, for the different problems of our benchmark. Figures 7.8 and 7.9 above show that our two proposals are efficient against problem 1. Then we show similar results for problem 2 in Figure 7.10 and for problem 4 in Figure 7.11.

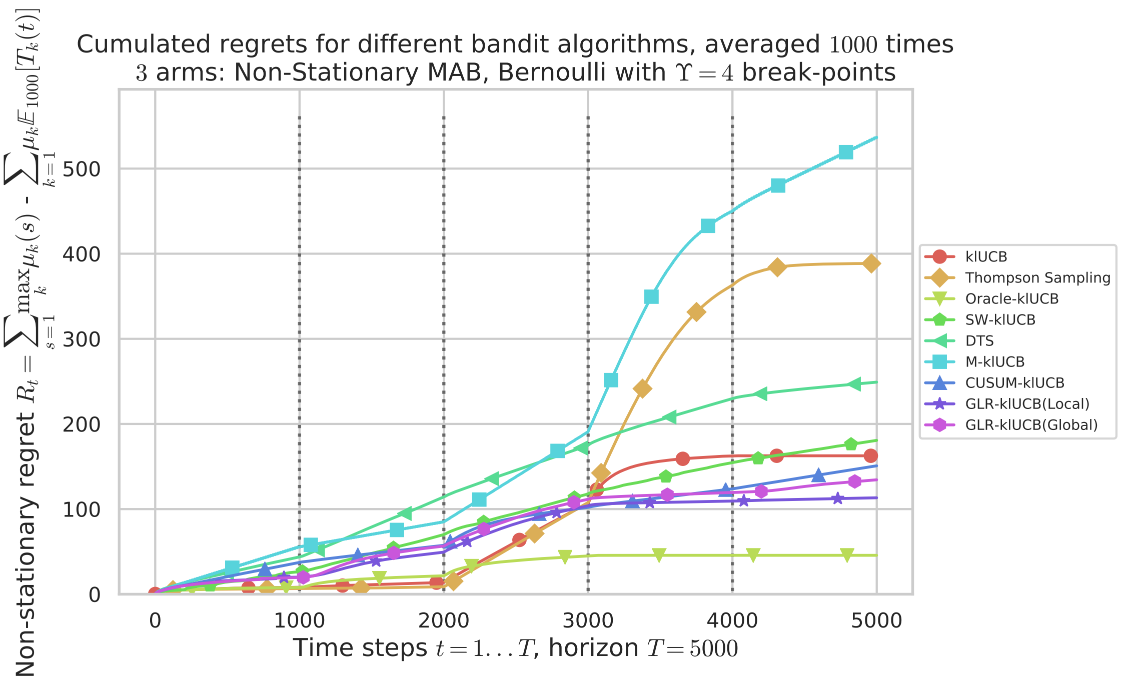


Figure 7.10 – Mean regret as a function of time, R_t ($1 \leq t \leq T = 5000$) for problem 2.

For harder problems, like problems 3 and 5, the stationary policy kl-UCB can outperform actively adaptive strategies, if the stationary intervals are too short or if the gap between arms are too small. In other words, we illustrate in Figures 7.12 and 7.13 that while the actively adaptive strategies can be very efficient when applied to problems that are not too difficult to track, they can become sub-optimal for difficult problems.

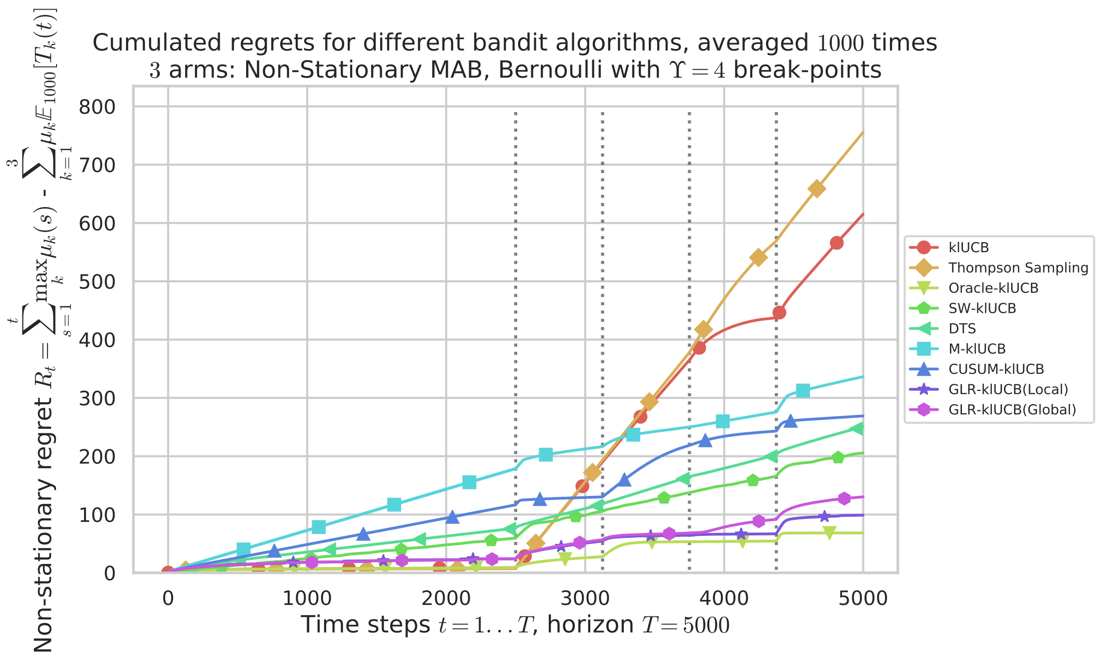


Figure 7.11 – Mean regret as a function of time, R_t ($1 \leq t \leq T = 5000$) for problem 4. We see that after a "long enough" stationary interval, the algorithms designed for stationary problems lose track of the best arm, and suffer from linear regret for a long period (*e.g.*, Thompson sampling in orange $\diamond$).

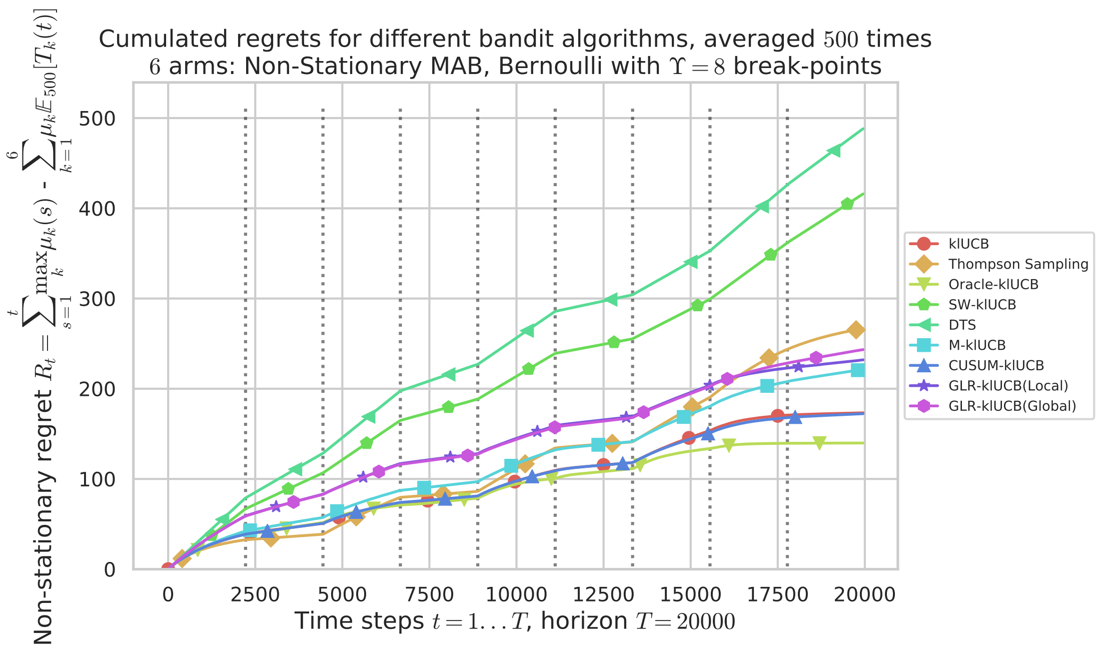


Figure 7.12 – Mean regret as a function of time, R_t ($1 \leq t \leq T = 20000$) for problem 3.

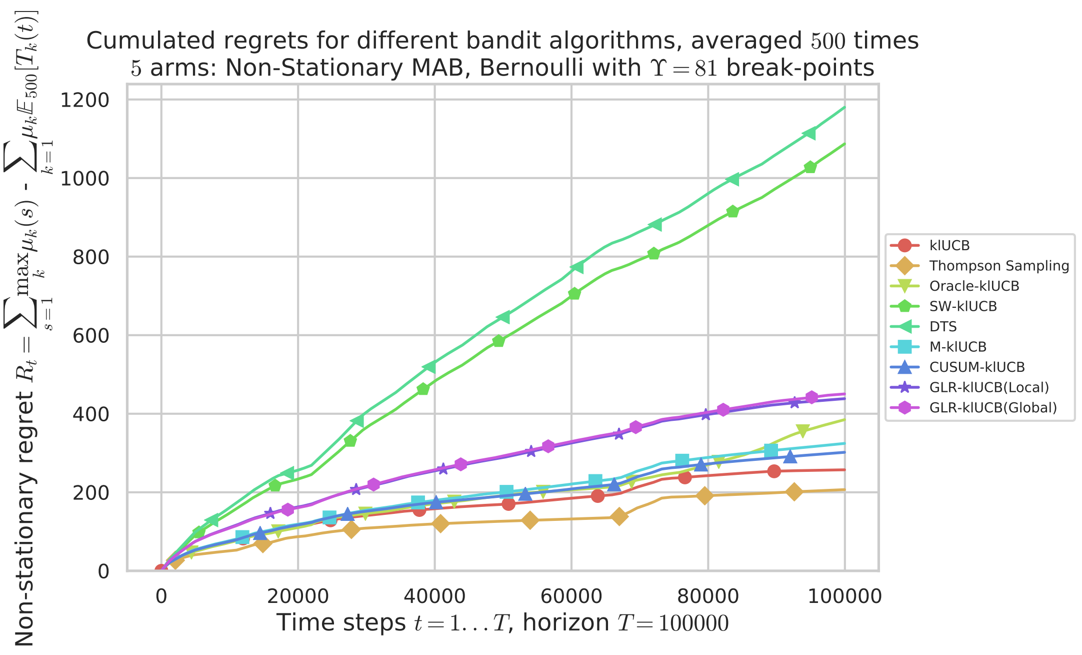


Figure 7.13 – Mean regret as a function of time, R_t $(1 \leq t \leq T = 100000)$ for problem 5.

7.10.2 Omitted proofs

This Appendix gives some proofs omitted in the main text of this chapter. Missing proofs can be found in the article [BK19b].

Simplified expression for the GLR statistic

First, we consider the denominator in the expression of $\mathrm{GLR}(n)$ (7.3), that is the $\sup$ on μ_0. We have $\ell(X_1, \dots, X_n; \mu_0) = \prod_{i=1}^{n} \ell(X_i; \mu_0)$ by independence of the observations X_i, and $\ell(X_i; \mu_0) = \mu_0^{X_i}(1-\mu_0)^{1-X_i}$ for Bernoulli distributions. Therefore, taking the logarithm gives

$$\begin{aligned}
\ln\left[\ell(X_1, \dots, X_n; \mu_0)\right] &= \sum_{i=1}^{n} X_i \ln(\mu_0) + (1 - X_i)\ln(1-\mu_0) \\
&= \ln(\mu_0) \times \left(\sum_{i=1}^{n} X_i\right) + \ln(1-\mu_0) \times \left(n - \sum_{i=1}^{n} X_i\right) \\
&= n\left(\widehat{\mu}_{1:n} \ln(\mu_0) + (1 - \widehat{\mu}_{1:n})\ln(1-\mu_0)\right).
\end{aligned}$$

For a constant $a \in [0,1]$, let $h(x) \doteq a\ln(x) + (1-a)\ln(1-x)$ on $(0,1)$, and we are trying to solve $\sup_{x\in[0,1]} h(x)$. If $a = 0$ or $a = 1$, h is maximum at $x = a$. Now if $a \neq 0$, As h is of class $\mathcal{C}^1$, we can just differentiate and find the root of its derivative, $h'(x) = a/x - (1-a)/(1-x)$, so

$h'(x) = 0$ if and only if $x = a$. Thus in all cases, $\sup_{x\in[0,1]} h(x) = h(a)$. Here, we have $a = \widehat{\mu}_{1:n}$, and thus we solve the $\sup_{\mu_0}$ optimization problem found in the denominator of the $\mathrm{GLR}(n)$ expression. By replacing $\mu_0 = \widehat{\mu}_{1:n}$, we obtained the following (unique) solution,

$$\sup_{\mu_0} \ln\left[\ell(X_1,\dots,X_n;\mu_0)\right] = n\left(\widehat{\mu}_{1:n}\ln(\widehat{\mu}_{1:n}) + (1-\widehat{\mu}_{1:n})\ln(1-\widehat{\mu}_{1:n})\right). \tag{7.29}$$

Now let us consider the nominator of the $\mathrm{GLR}(n)$ expression. We can again work with log-likelihoods, and so we have

$$\begin{aligned}
&\ln\left[\ell(X_1,\dots,X_n;\mu_0,\mu_1,\tau)\right] \\
&= \sum_{i=1}^{\tau} X_i\ln(\mu_0) + (1-X_i)\ln(1-\mu_0) + \sum_{i=\tau+1}^{n} X_i\ln(\mu_1) + (1-X_i)\ln(1-\mu_1) \\
&= s\big(\widehat{\mu}_{1:s}\ln(\mu_0) + (1-\widehat{\mu}_{1:s})\ln(1-\mu_0)\big) \\
&\quad + (n-s)\big(\widehat{\mu}_{s+1:n}\ln(\mu_1) + (1-\widehat{\mu}_{s+1:n})\ln(1-\mu_1)\big).
\end{aligned}$$

Because we solved in (7.29) the $\sup$ for the denominator, we have

$$\begin{aligned}
\mathrm{GLR}(n) &= \frac{\sup_{\mu_0,\mu_1,\tau<n} \ell(X_1,\dots,X_n;\mu_0,\mu_1,\tau)}{\exp\left[n\left(\widehat{\mu}_{1:n}\ln(\mu_0) + (1-\widehat{\mu}_{1:n})\ln(1-\mu_0)\right)\right]}, \\
&= \sup_{\mu_0,\mu_1,\tau<n} \frac{\ell(X_1,\dots,X_n;\mu_0,\mu_1,\tau)}{\exp\left[n\left(\widehat{\mu}_{1:n}\ln(\widehat{\mu}_{1:n}) + (1-\widehat{\mu}_{1:n})\ln(1-\widehat{\mu}_{1:n})\right)\right]}, \\
&= \exp\Big[\sup_{\mu_0,\mu_1,\tau<n}\Big[\ln\Big[\frac{\ell(X_1,\dots,X_n;\mu_0,\mu_1,\tau)}{\exp\left[n\left(\widehat{\mu}_{1:n}\ln(\widehat{\mu}_{1:n}) + (1-\widehat{\mu}_{1:n})\ln(1-\widehat{\mu}_{1:n})\right)\right]}\Big]\Big]\Big],
\end{aligned}$$

When the last equation comes from the fact that $\exp$ is increasing. Thus by taking the logarithm of both sides, we obtain

$$\begin{aligned}
\ln\mathrm{GLR}(n) &= \sup_{\mu_0,\mu_1,\tau<n}\Big[\ln\Big[\frac{\ell(X_1,\dots,X_n;\mu_0,\mu_1,\tau)}{\exp\left[n\left(\widehat{\mu}_{1:n}\ln(\widehat{\mu}_{1:n}) + (1-\widehat{\mu}_{1:n})\ln(1-\widehat{\mu}_{1:n})\right)\right]}\Big]\Big], \\
&= \sup_{\mu_0,\mu_1,1<s<n}\Big[s\big(\widehat{\mu}_{1:s}\ln(\mu_0) + (1-\widehat{\mu}_{1:s})\ln(1-\mu_0)\big) \\
&\qquad + (n-s)\big(\widehat{\mu}_{s+1:n}\ln(\mu_1) + (1-\widehat{\mu}_{s+1:n})\ln(1-\mu_1)\big) \\
&\qquad - n\big(\widehat{\mu}_{1:n}\ln(\widehat{\mu}_{1:n}) + (1-\widehat{\mu}_{1:n})\ln(1-\widehat{\mu}_{1:n})\big)\Big].
\end{aligned}$$

By linearity and independence, we can separate the joint optimization problem on μ_0, μ_1, s in two optimizations problems for μ_0, s and μ_1, s, that can first be solved explicitly for μ_0 (resp. μ_1) and then left to be solved for s. By definition, $n\ \widehat{\mu}_{1:n} = \sum_{i=1}^{n} X_i = s\ \widehat{\mu}_{1:s} + (n-s)\ \widehat{\mu}_{s+1:n}$, so the right hand side (negative) part involving $\widehat{\mu}_{1:n}$ can be distributed in the two left hand side (positive) terms, which are both handled similarly. For instance for μ_0, we use the same computation as above with the function h to find the optimum: $\widehat{\mu}_{1:s}\ln(\mu_0) + (1-\widehat{\mu}_{1:s})\ln(1-\mu_0)$

is optimum for $\mu_0 = \widehat{\mu}_{1:s}$. Similarly, the term for μ_1 gives that $\sup_{\mu_1} \widehat{\mu}_{s+1:n} \ln(\mu_1) + (1 - \widehat{\mu}_{s+1:n}) \ln(1-\mu_1)$ is attained for $\mu_1 = \widehat{\mu}_{s+1:n}$. Finally, by replacing the two expressions of the solutions for μ_0 and μ_1, we obtain

$$\begin{aligned}\ln \mathrm{GLR}(n) = \sup_{1<s<n} \Big[s \times \Big(& \widehat{\mu}_{1:s} \ln(\widehat{\mu}_{1:s}) + (1-\widehat{\mu}_{1:s}) \ln(1-\widehat{\mu}_{1:s}) \\ & - \widehat{\mu}_{1:s} \ln(\widehat{\mu}_{1:n}) + (1-\widehat{\mu}_{1:s}) \ln(1-\widehat{\mu}_{1:n}) \Big) \\ +(n-s) \times \Big(& \widehat{\mu}_{s+1:n} \ln(\widehat{\mu}_{s+1:n}) + (1-\widehat{\mu}_{s+1:n}) \ln(1-\widehat{\mu}_{s+1:n}) \\ & - \widehat{\mu}_{s+1:n} \ln(\widehat{\mu}_{1:n}) + (1-\widehat{\mu}_{s+1:n}) \ln(1-\widehat{\mu}_{1:n}) \Big) \Big].\end{aligned}$$

We conclude by recognizing the expressions of $s \times \mathrm{kl}(\widehat{\mu}_{1:s}, \widehat{\mu}_{1:n})$ and $(n-s) \times \mathrm{kl}(\widehat{\mu}_{s+1:n}, \widehat{\mu}_{1:n})$.

Proof of Lemma 7.4.

Using the same construction as in the proof of Theorem 14 in [KK18], one can prove that for every $\lambda \in I$ (for an interval I), there exists a non-negative super-martingale $M^\lambda(s)$ with respect to the filtration $\mathcal{F}_t \doteq \sigma(X_1, \dots, X_t)$ that satisfies $\mathbb{E}[M^\lambda(s)] \leq 1$ and

$$\forall s \in \mathbb{N}^*, \quad M^\lambda(s) \geq \mathrm{e}^{\lambda[s\,\mathrm{kl}(\widehat{\mu}_s, \mu) - 3\ln(1+\ln(s))] - g(\lambda)}$$

for some function $g : I \to \mathbb{R}$. This super-martingale is of the form $M^\lambda(s) \doteq \int \mathrm{e}^{\eta \sum_{i=1}^s X_i - \phi_\mu(\lambda) s} d\pi(\eta)$, for a well-chosen probability distribution π, and the function g can be chosen to be any

$$\begin{aligned} g_\xi : [0; 1/(1+\xi)] &\longrightarrow \mathbb{R} \\ \lambda &\mapsto \lambda(1+\xi) \ln\left(\frac{\pi^2}{3(\ln(1+\xi))^2}\right) - \ln(1-\lambda(1+\xi)) \end{aligned}$$

for a parameter $\xi \in [0, 1/2]$.

Similarly, if we denote $\mathcal{F}'_r$ the filtration $\sigma(Y_1, \dots, Y_r)$, there exists an independent super-martingale $W^\lambda(r)$ w.r.t. the filtration $\mathcal{F}'_r$, such that

$$\forall r \in \mathbb{N}^*, \quad W^\lambda(r) \geq \mathrm{e}^{\lambda[r\,\mathrm{kl}(\widehat{\mu}'_r, \mu) - 3\ln(1+\ln(r))] - g(\lambda)},$$

for the same function $g(\lambda)$. In the terminology of [KK18], the two following processes are g-DCC (for Doob-Cramér-Chernoff), $\boldsymbol{X}(s) \doteq s\,\mathrm{kl}(\widehat{\mu}_s, \mu) - 3\ln(1+\ln(s))$ and $\boldsymbol{Y}(s) \doteq r\,\mathrm{kl}(\widehat{\mu}_r, \mu) - 3\ln(1+\ln(r))$, as for both processes the Doob's inequality can be applied in combination with the Cramér-Chernoff method to obtain deviation inequalities that are uniform in time.

Here we have to modify the technique used in the Lemma 4 of [KK18] in order to take into account the two stochastic processes, and the presence of super-martingales instead of martingales (for which Doob inequality still works). One can write

$$
\begin{aligned}
&\mathbb{P}(\exists r \in \mathbb{N}^* : s \, \mathrm{kl}\left(\widehat{\mu}_s, \mu\right) + r \, \mathrm{kl}\left(\widehat{\mu}'_r, \mu'\right) > 3\ln(1+\ln(s)) + 3\ln(1+\ln(r)) + u) \\
&\leq \mathbb{P}\left(\exists r \in \mathbb{N}^* : M^\lambda(s) W^\lambda(r) > \mathrm{e}^{\lambda u - 2g(\lambda)}\right) \\
&= \lim_{n\to\infty} \mathbb{P}\left(\exists r \in [n] : M^\lambda(s) W^\lambda(r) > \mathrm{e}^{\lambda u - 2g(\lambda)}\right) \\
&= \lim_{n\to\infty} \mathbb{P}\left(\sup_{r\in[n]} M^\lambda(s) W^\lambda(r) > \mathrm{e}^{\lambda u - 2g(\lambda)}\right).
\end{aligned}
$$

Using that $\widetilde{M}(r) \doteq M^\lambda(s)W^\lambda(r)$ is a super-martingale with respect to the filtration $\widetilde{\mathcal{F}}_r \doteq \sigma(X_1, \dots, X_s, Y_1, \dots, Y_r)$, one can apply Doob's maximal inequality to obtain

$$
\begin{aligned}
\mathbb{P}\left(\sup_{r\in[n]} M^\lambda(s) W^\lambda(r) > \mathrm{e}^{\lambda u - 2g(\lambda)}\right) &\leq \mathrm{e}^{-(\lambda u - 2g(\lambda))}\mathbb{E}[\widetilde{M}(1))] \\
&= \mathrm{e}^{-(\lambda u - 2g(\lambda))}\mathbb{E}[M^\lambda(s)W^\lambda(1)] \\
&\leq \mathrm{e}^{-(\lambda u - 2g(\lambda))},
\end{aligned}
$$

using that $M^\lambda(s)$ and $W^\lambda(1)$ are independent and have an expectation smaller than 1.

Putting things together yields

$$
\mathbb{P}\left(\exists r \in \mathbb{N}^* : s \, \mathrm{kl}\left(\widehat{\mu}_s, \mu\right) + r \, \mathrm{kl}\left(\widehat{\mu}'_r, \mu'\right) > 3\ln(1+\ln(s)) + 3\ln(1+\ln(r)) + u\right) \leq \mathrm{e}^{-\left(\lambda u - 2g_\xi(\lambda)\right)},
$$

for any function g_ξ defined above. The conclusion follows by optimizing for both λ and ξ, using Lemma 18 in [KK18].

A Concentration Result Involving Two Arms

The following result is useful to control the probability of the good event in our two regret analyzes. Its proof follows from a straightforward application of the Cramér-Chernoff method [BLM13], and is given below.

Lemma 7.14. *Let $\widehat{\mu}_{i,s}$ be the empirical mean of $s \in \mathbb{N}^*$ i.i.d. observations with mean μ_i, for $i \in \{a, b\}$, that are σ^2-sub-Gaussian. Define $\Delta \doteq \mu_a - \mu_b$. Then for any $s, r > 0$, we have*

$$
\mathbb{P}\left(\frac{s\, r}{s+r}\left(\widehat{\mu}_{a,s} - \widehat{\mu}_{b,r} - \Delta\right)^2 \geq u\right) \leq 2\exp\left(-\frac{u}{2\sigma^2}\right). \tag{7.30}
$$

Proof of Lemma 7.14. We first note that

$$\begin{aligned}
&\mathbb{P}\left(\frac{s\,r}{s+r}\Big(\widehat{\mu}_{a,s}-\widehat{\mu}_{b,r}-\Delta\Big)^2 \geq u\right) \\
&\leq \mathbb{P}\left(\widehat{\mu}_{a,s}-\widehat{\mu}_{b,r} \geq \Delta + \sqrt{\frac{s+r}{sr}u}\right) + \mathbb{P}\left(\widehat{\mu}_{b,r}-\widehat{\mu}_{a,s} \geq -\Delta + \sqrt{\frac{s+r}{sr}u}\right), \qquad (7.31)
\end{aligned}$$

and those two quantities can be upper-bounded similarly using the Cramér-Chernoff method.

Let (X_i) and (Y_i) be two *i.i.d.* sequences that are σ^2 sub-Gaussian with mean μ_1 and μ_2 respectively. Let n_1 and n_2 be two integers and $\widehat{\mu}_{1,n_1}$ and $\widehat{\mu}_{2,n_2}$ denote the two empirical means based on n_1 observations from X_i, and n_2 observations from Y_i respectively. Then for every $\lambda > 0$, we have

$$\begin{aligned}
\mathbb{P}\left(\widehat{\mu}_{1,n_1}-\widehat{\mu}_{2,n_2} \geq \mu_1-\mu_2+x\right) &\leq \mathbb{P}\left(\frac{1}{n_1}\sum_{i=1}^{n_1}(X_i-\mu_1) - \frac{1}{n_2}\sum_{i=1}^{n_2}(Y_i-\mu_2) \geq x\right) \\
&\leq \mathbb{P}\left(\mathrm{e}^{\lambda\left(\frac{1}{n_1}\sum_{i=1}^{n_1}(X_i-\mu_1)-\frac{1}{n_2}\sum_{i=1}^{n_2}(Y_i-\mu_2)\right)} \geq \mathrm{e}^{\lambda x}\right)
\end{aligned}$$

And so thanks to Markov's inequality, we obtain

$$\begin{aligned}
\mathbb{P}\left(\widehat{\mu}_{1,n_1}-\widehat{\mu}_{2,n_2} \geq \mu_1-\mu_2+x\right) &\leq \mathrm{e}^{-\lambda x}\mathbb{E}\left[\mathrm{e}^{\lambda\frac{1}{n_1}\sum_{i=1}^{n_1}(X_i-\mu_1)}\right]\mathbb{E}\left[\mathrm{e}^{-\lambda\frac{1}{n_2}\sum_{i=1}^{n_2}(Y_i-\mu_2)}\right] \\
&= \exp\left(-\lambda x + n_1\phi_{X_1-\mu_1}\left(\frac{\lambda}{n_1}\right) + n_2\phi_{Y_1-\mu_2}\left(-\frac{\lambda}{n_2}\right)\right) \\
&\leq \exp\left(-\lambda x + \frac{\lambda^2\sigma^2}{2n_2} + \frac{\lambda^2\sigma^2}{2n_1}\right),
\end{aligned}$$

where the last inequality uses the sub-Gaussian property. Choosing the value

$$\lambda \doteq \frac{x}{2\left[\sigma^2/(2n_1)+\sigma^2/(2n_2)\right]}$$

which minimizes the right-hand side of the inequality yields

$$\mathbb{P}\left(\widehat{\mu}_{1,n_1}-\widehat{\mu}_{2,n_2} \geq \mu_1-\mu_2+x\right) \leq \exp\left(-\frac{n_1 n_2}{n_1+n_2}\frac{x^2}{2\sigma^2}\right).$$

Using this inequality twice in the right hand side of (7.31) concludes the proof. □

7.10.3 Time and memory costs of GLR-klUCB

As demonstrated in Section 7.8, our proposal is empirically efficient in terms of regret, but it is important to also evaluate its cost in terms of both *time* and *memory*. Remember that Υ_T denotes the number of change-points. If we denote $d_{\max}$ the longest duration of a stationary sequence, the worst case is $d_{\max} = T$ for a stationary problem, and the easiest case is $d_{\max} \simeq T/\Upsilon_T$ typically for Υ_T evenly spaced change-points. We begin by reviewing the costs of the algorithms designed for stationary problems and then of other approaches.

Classical algorithms. For a stationary bandit problem, almost all *classical algorithms* (*i.e.*, designed for stationary problems) need and use a storage proportional to the number of arms, *i.e.*, $\mathcal{O}(K)$, as most of them only need to store a number of pulls and the empirical mean of rewards for each arm. They also have a time complexity $\mathcal{O}(K)$ at every time step $t \in [T]$, hence an optimal total time complexity of $\mathcal{O}(KT)$. In particular, this case includes UCB, Thompson sampling and kl-UCB.

Oracle algorithms. Most algorithms designed for abruptly changing environments are more costly, both in terms of storage and computation time, as they need more storage and time to test for changes. The *oracle algorithm* presented in Section 7.8, combined with any efficient index policy, needs a storage at most $\mathcal{O}(K\Upsilon_T)$ as it stores the change-points, and have an optimal time complexity of $\mathcal{O}(KT)$ too[4].

Passively adaptive algorithms. Passively adaptive algorithms should intuitively be more efficient, but as they use a non-constant storage, they are actually as costly as the oracle. For instance SW-UCB uses a storage of $\mathcal{O}(K\tau)$, increasing as T increases, and similarly for other passively adaptive algorithms. We highlight that to the best of our knowledge, the Discounted Thompson sampling algorithm (DTS) is the only algorithm tailored for abruptly changing problems that is both efficient in terms of regret (see the simulations results, even though it has no theoretical guarantee), and optimal in terms of both computational and storage costs. Indeed, it simply needs a storage proportional to the number of arms, $\mathcal{O}(K)$, and a time complexity of $\mathcal{O}(KT)$ for a horizon T (see the pseudo-code in [RK17]). Note that the discounting scheme in Discounted-UCB (D-UCB) from [KS06] requires to store the whole history, and not only empirical rewards of each arm, as after observing a reward, all previous rewards must be multiplied by γ^n if that arm was not seen for $n > 0$ times. So the storage cannot be simply proportional to K, but needs to grows as t grows. Therefore, D-UCB costs $\mathcal{O}(KT)$ in memory and $\mathcal{O}(KT^2)$ in time.

[4]Note that if implemented naively, that is if testing is t is a change-points takes a time $\mathcal{O}(\Upsilon_T)$, its performance is sub-optimal, as it is of the order of $\mathcal{O}(KT\Upsilon_T)$. However, if the change-points are stored in a *dynamic linked list*, and the head is removed when it is found to be equal to the current time step, then the test at a time t costs only a constant time $\mathcal{O}(1)$, hence the total time complexity is bounded by $\mathcal{O}(KT)$ and the oracle algorithm is indeed optimal in terms of time complexity.

Actively adaptive algorithms. Limited memory actively adaptive algorithms, like M-UCB, are even more costly. For instance, M-UCB would have the same cost of $\mathcal{O}(KTd_{\max})$, except that [CZKX19] introduces a window-size w and run their CPD algorithm only using the last w observations of each arm. If w is constant w.r.t. the horizon T, their algorithm has a storage cost bounded by $\mathcal{O}(Kw)$ and a running time of $\mathcal{O}(KTw)$, being comparable to the cost $\mathcal{O}(KT)$ of the oracle approach. However in practice, as well as in the theoretical results, the window size should depend on T and a prior knowledge of the minimal change size $\widehat{\delta}$ (see Remark 1 in [CZKX19]), and $w = \mathcal{O}(\ln(T)/\widehat{\delta}^2)$. Hence it makes more sense to consider that M-UCB has a time cost bounded by $\mathcal{O}(KT\ln(T))$ and a memory costs bounded by $\mathcal{O}(K\ln(T))$, which is better than our proposal but more costly than the oracle or DTS or stationary algorithms.

Time and memory cost of GLR-klUCB**.** On the other hand, actively adaptive algorithms are more efficient (when tuned correctly) but at the price of being more costly, both in terms of time and memory. The two algorithms using the CUSUM or GLR tests (as well as PHT), when used with an efficient and low-cost index policy (that is, choosing the arm to play only costs $\mathcal{O}(K)$ at any time t), are found to be efficient in terms of regret. However, they need to store all past rewards and pulls history, to be able to reset them when the CPD algorithm tells to do, so they have a memory cost of $\mathcal{O}(Kd_{\max})$, that is $\mathcal{O}(KT)$ in the worst case (compared to $\mathcal{O}(K)$ for algorithms designed for the stationary setting). They are also costly in terms of computation time, as at current time t, when trying to detect a change with n_i observations of arm i (*i.e.*, $(Z_{i,n})_{1\leq n\leq n_i}$ in Algorithm 7.1), the CPD algorithm (CUSUM or GLR) costs a time $\mathcal{O}(n_i)$. Indeed, it needs to compute sliding averages for every s in an interval of size n_i (*i.e.*, $\mu_{\text{left}} = \mu_{1:s}$ and $\mu_{\text{right}} = \mu_{s+1:n_i}$) and a test for each s which costs a constant time $\mathcal{O}(1)$ (*e.g.*, computing two kl and checking for a threshold for our GLR test). So for every s, the running time is $\mathcal{O}(1)$, if the sliding averages are computed iteratively based on a simple scheme: first, one compute the total average $\mu_{t_0:t}$ and set $\mu_{\text{left}} = 0$ and $\mu_{\text{right}} = \mu_{t_0:t}$. Then for every successive values of s, both the left and right sliding window means can be updated with a single memory access and two computations (*i.e.*, $\mathcal{O}(1)$):

$$z \leftarrow Z_{i,s+1}, \;\; \mu_{\text{left}} \leftarrow \frac{s\mu_{\text{left}} + z}{s+1}, \;\; \mu_{\text{right}} \leftarrow \frac{(n_i+1-s)\mu_{\text{right}} - z}{n_i - s}, \;\; s \leftarrow s+1. \tag{7.32}$$

To sum up, at every time step the CPD algorithm needs a time $\mathcal{O}(n_i) = \mathcal{O}(d_{\max})$, and at the end, the time complexity of CUSUM-klUCB as well as GLR-klUCB is $\mathcal{O}(KTd_{\max})$, which can be up-to $\mathcal{O}(KT^2)$, much more costly than $\mathcal{O}(KT)$ for kl-UCB for instance.

Our proposal GLR-klUCB requires a storage of the order of $\mathcal{O}(Kd_{\max})$ and a running time of the order of $\mathcal{O}(KTd_{\max})$, and the two bounds are validated experimentally, see Table 7.4.

Empirical measurements of computation times and memory costs. A theoretical analysis shows that there is a large gap between the costs of stationary or passively adaptive algorithms, and the costs of actively adaptive algorithms, for both computation time and memory consumption. We include here an extensive comparison of memory costs of the different algorithms. For instance on the same experiment as the one used for Table 7.6, that is problem 1 with $T = 5000$, and then with $T = 10000$ and $T = 20000$, and 100 independent runs, in our Python implementation (using the SMPyBandits library, [Bes19]), we can measure the (mean) real memory cost[5] of the different algorithms. The Tables 7.4 and 7.5 included below give the mean ($\pm$ 1 standard-deviation) of real computation time and memory consumption, used by the algorithms. The computation time is normalized by the horizon, to reflect the (mean) time used for each time steps $t \in [T]$. We also found that our two optimizations described below in 7.10.4 do not reduce the memory, thus we used $\Delta n = \Delta s = 20$ to speed-up the simulations. The conclusions to draw for these two Tables 7.4 and 7.5 are twofold.

Algorithms	$T = 5000$	$T = 10000$	$T = 20000$
Thompson sampling	**55 µs ± 11 µs**	**51 µs ± 6 µs**	**49 µs ± 4 µs**
DTS	62 µs ± 11 µs	60 µs ± 8 µs	59 µs ± 7 µs
kl-UCB	122 µs ± 13 µs	125 µs ± 13 µs	128 µs ± 11 µs
Discounted-kl-UCB	94 µs ± 10 µs	97 µs ± 12 µs	103 µs ± 12 µs
SW-kl-UCB	162 µs ± 21 µs	169 µs ± 18 µs	167 µs ± 12 µs
Oracle-Restart kl-UCB	159 µs ± 31 µs	157 µs ± 21 µs	149 µs ± 15 µs
M-klUCB	202 µs ± 25 µs	220 µs ± 26 µs	230 µs ± 19 µs
CUSUM-klUCB	264 µs ± 53 µs	227 µs ± 31 µs	270 µs ± 33 µs
GLR-klUCB	314 µs ± 50 µs	399 µs ± 33 µs	920 µs ± 180 µs

Table 7.4 – Normalized computation time, for each time step $t \in [T]$, for different horizons.

Algorithms	$T = 5000$	$T = 10000$	$T = 20000$
Thompson sampling	813 B ± 63 B	819 B ± 27 B	818 B ± 35 B
DTS	946 B ± 38 B	946 B ± 38 B	946 B ± 39 B
kl-UCB	937 B ± 164 B	931 B ± 172 B	933 B ± 164 B
Discounted-kl-UCB	1 KiB ± 79 B	1 KiB ± 94 B	1 KiB ± 78 B
SW-kl-UCB	6 KiB ± 976 B	8 KiB ± 1 KiB	12 KiB ± 2 KiB
Oracle-Restart kl-UCB	11 KiB ± 3 KiB	19 KiB ± 7 KiB	31 KiB ± 17 KiB
M-klUCB	4 KiB ± 1 KiB	6 KiB ± 2 KiB	9 KiB ± 4 KiB
CUSUM-klUCB	8 KiB ± 4 KiB	11 KiB ± 6 KiB	15 KiB ± 10 KiB
GLR-klUCB	19 KiB ± 7 KiB	32 KiB ± 15 KiB	75 KiB ± 26 KiB

Table 7.5 – Non normalized memory costs, for the same problem (Pb 1) with different horizons.

First, we verify the results stated above for the time complexity of different algorithms, implemented in Python using SMPyBandits [Bes19]. The methodology of time and memory measurements is discussed in Section 3.4. On the one hand, stationary and passively adaptive

[5]We used one core of an Intel $i5$ Core CPU, with GNU/Linux Ubuntu 18.04, Python v3.6, and 8 Gb of RAM.

algorithms all have a time complexity scaling as $\mathcal{O}(T)$, as their normalized computation time is almost constant w.r.t. T. We check that TS and DTS are the fastest algorithms, 2.5 to 3 times faster than kl-UCB-based algorithms, due to the fact that sampling from a Beta posterior is typically faster than doing a (small) numerical optimization step to compute the $\sup$ in the kl-UCB indexes. We also check that the passively adaptive algorithms add a non-trivial but constant overhead on the computation times of their based algorithm, *e.g.*, SW-kl-UCB compared to kl-UCB, or DTS compared to TS. On the other hand, we also check that actively adaptive algorithms are most costly. M-klUCB normalized computation time is not increasing much when the horizon is doubled, as the window-size M was set to a constant w.r.t. the horizon T for this experiment. The complexity of GLR-klUCB follows the $\mathcal{O}(KT^2)$ bound we presented above.

Second, we also verify the results for the memory costs of the different algorithms. Similarly, stationary and passively adaptive algorithms based on a discount factor (D-UCB, DTS) have a memory cost constant w.r.t. the horizon T, as stated above, while algorithms based on a sliding-window have a memory cost increasing w.r.t. the horizon T. The Oracle-Restart and the actively adaptive algorithms see their memory costs increase similarly. These measurements validate the upper-bound we gave on their memory costs, $\mathcal{O}(Kd_{\max})$, as $d_{\max} \simeq T/\Upsilon$ for this problem with evenly spaced change-points.

7.10.4 Two numerical optimization tricks for GLR-klUCB

As the main weakness of GLR-klUCB is its numerical efficiency, as shown in Table 7.4, we suggest here two simple ideas to drastically speed-up its computation time.

1. The *first optimization*, parametrized by a constant $\Delta n \in \mathbb{N}^*$, is the following idea. We can test for statistical changes not at all time steps $t \in [T]$ but only every Δn time steps (*i.e.*, for t satisfying $t \mod \Delta n = 0$). In practice, instead of sub-sampling for the *time* t, we propose to sub-sample for the number of samples of arm i before calling GLR to check for a change on arm i, that is, $n_i(t)$ in Algorithm 7.1. Note that the first heuristic using Δn can be applied to M-UCB as well as CUSUM-UCB and PHT-UCB, with similar speed-up and typically leading to similar consequences on the algorithm's performance.

2. The *second optimization* is in the same spirit, and uses a parameter $\Delta s \in \mathbb{N}^*$. When running the GLR test with data $Z_1, \dots, Z_t$, instead of considering every splitting time steps $s \in [t]$, in the same spirit, we can skip some and test not at all time steps s but only every Δs time steps.

The new GLR test is using the stopping time $\widetilde{T_\delta}$ defined in (7.11), with $\mathcal{T} = \{t \in [T], t \mod \Delta n = 0\}$ and $\mathcal{S}_t = \{s \in [t], s \mod \Delta s = 0\}$. The goal is to speed up the computation time of every call to the GLR test (*e.g.*, choosing $\Delta s = 10$, every call should be about 10 times faster), and to speed up the overhead cost of running the tests on top of the index policy (kl-UCB), by testing for changes *less* often (*e.g.*, choosing $\Delta n = 10$ should speed up the all computation by a factor 10).

Empirical validation of these optimization tricks. We consider the problem 1 presented above (Figure 7.1), with $T = 5000$ and 100 repetitions, and we give the means ($\pm$ 1 standard-deviation) of both regret and computation time of GLR-klUCB with **Local** restarts, for different parameters Δn and Δs, in Table 7.6 below. The other parameters of GLR-klUCB are chosen as $\delta = 1/\sqrt{K \Upsilon_T T}$ and $\omega = 0.1\sqrt{K \ln(T)/T}$ (from Corollary 7.12). The algorithm analyzed in Section 7.6 corresponds to $\Delta n = \Delta s = 1$.

On the same problem, the Oracle-Restart kl-UCB obtained a mean regret of 37 for a running time of 711 ms, while kl-UCB obtained a regret of 270 for a time of 587 ms. In comparison with the two other efficient approaches, M-kl-UCB obtained a regret of 290 for a time of 943 ms, and CUSUM-klUCB obtained a regret of 148 for a time of 46 s. This shows that our proposal is very efficient compared to stationary algorithms, and comparable to the state-of-the-art actively adaptive algorithm. Moreover, this shows that two heuristics efficiently speed-up the computation times of GLR-klUCB. Choosing small values, like $\Delta n = 20, \Delta s = 20$, can speed-up GLR-klUCB, making it fast enough to be comparable to recent efficient approaches like M-UCB and even comparable to the oracle policy. It is very satisfying to see that these optimizations do not reduce much the regret of GLR-klUCB, as it still outperforms most state-of-the-art algorithms, and significantly reduces the computation time as wanted. With such numerical optimization, GLR-klUCB is not significantly slower than kl-UCB while being much more efficient for piece-wise stationary problems.

$\Delta n \ \backslash \ \Delta s$	1	5	10	20
1	44 ± 29	44 ± 28	50 ± 31	53 ± 28
5	48 ± 29	41 ± 30	44 ± 28	47 ± 31
10	51 ± 32	43 ± 26	47 ± 28	46 ± 29
20	46 ± 31	46 ± 34	46 ± 31	49 ± 31

$\Delta n \ \backslash \ \Delta s$	1	5	10	20
1	**50 s ± 4.5 s**	11.1 s ± 1.2 s	5.8 s ± 0.5 s	3.3 s ± 0.3 s
5	17.9 s ± 1.6 s	5.08 s ± 3.3 s	2.5 s ± 0.3 s	1.7 s ± 0.2 s
10	14.9 s ± 1.9 s	3.47 s ± 0.4 s	2.1 s ± 0.2 s	1.4 s ± 0.2 s
20	12.1 s ± 1.1 s	3.02 s ± 0.3 s	1.9 s ± 0.2 s	**0.2 s ± 0.1 s**

Table 7.6 – Effects of the two optimizations parameters Δn and Δs, on the mean regret R_T (top) and mean computation time (bottom) for GLR-klUCB on a simple problem. Using the optimizations with $\Delta n = \Delta s = 20$ does not reduce the regret much but speeds up the computations by about a **factor 50**.

7.10.5 Sensitivity analysis of the exploration probability ω

As demonstrated in our experiments in Section 7.8, the choice of $\omega = \omega_0\sqrt{\Upsilon_T \ln(T)/T}$ for the exploration probability is a good choice for GLR-klUCB to be efficient (for the **local restarts** option). The dependency w.r.t. the horizon T comes from Corollary 7.12 when Υ_T is unknown, and we observe in the Table 7.7 below that the value of ω_0 does not influence much the performance, as long as $\omega_0 \leq 1$. Different values of ω_0 are explored, for problems 1, 2 and 4, and we average the results over 100 independent runs. Other parameters are set to **Local** restarts, $\delta_T = 1/\sqrt{\Upsilon_T T}$, and $\Delta n = \Delta s = 10$ to speed-up the experiments. In the three experiments, GLR-klUCB performs closely to the Oracle-Restart kl-UCB, and outperforms all or almost all the other approaches, for all choices of ω_0. We observe in Table 7.7 that the parameter ω_0 does not have a significative impact on the performance, and that surprisingly, choosing $\omega_0 = 0$ does not reduce the empirical performance of GLR-klUCB, which means that on some problems there is no need of a forced exploration. However, the analysis of GLR-klUCB is based on the forced exploration, and we found that for a larger number of arms, or for problems when the optimal arm change constantly, the forced exploration is required. The same observations can be made for other variants of our algorithm, for instance with **Global** restarts, or other values of Δn, Δs, or other values of δ.

Choice of ω_0	Problem 1	Problem 2	Problem 4
$\omega_0 = 1$	51 ± 29	79 ± 35	82 ± 45
$\omega_0 = 0.5$	38 ± 29	70 ± 35	76 ± 39
$\omega_0 = 0.1$	33 ± 29	69 ± 31	68 ± 32
$\boldsymbol{\omega_0 = 0.05}$	$\mathbf{36 \pm 29}$	$\mathbf{65 \pm 33}$	$\mathbf{67 \pm 36}$
$\omega_0 = 0.01$	38 ± 33	66 ± 33	71 ± 37
$\omega_0 = 0.005$	40 ± 27	69 ± 30	73 ± 56
$\omega_0 = 0.001$	38 ± 30	69 ± 34	67 ± 34
$\omega_0 = 0$	36 ± 32	66 ± 36	67 ± 33

Table 7.7 – Mean regret $\pm$ 1 std-dev, for different choices of scaling factor for the forced exploration probability $\omega_0 \in [0,1]$ (*i.e.*, $\omega = \omega_0\omega_T$), on three problems of horizon $T = 5000$, for GLR-klUCB with **local restarts**, with $\omega_T = \omega_0\sqrt{\Upsilon_T \ln(T)/T}$.

7.10.6 Sensitivity analysis of the confidence level δ

Similarly, the choice of $\delta = \delta_0/\sqrt{\Upsilon_T T}$ for the confidence level is a good choice for GLR-klUCB to be efficient (for the **local restarts** option). The dependency w.r.t. the horizon T comes from Corollary 7.12 when Υ_T is unknown, and we observe in the Table 7.8 below that the value of δ_0 does not influence much the performance of our proposal, as long as $\delta_0 \leq 1$. Different values of δ_0 are explored, for problems 1, 2 and 4, and we average the results over 100 independent runs. Other parameters are set to **Local** restarts, $\omega_T = 0.1\sqrt{\Upsilon_T \ln(T)/T}$, and $\Delta n = \Delta s = 10$ to speed-up the experiments. We observe in Table 7.8 that the parameter δ_0 has no significative impact on the performance, and that choosing $\delta_0 = 1$ is indeed a valid choice. Inspired by Occam's razor, we recommend this choice of $\delta_0 = 1$ in practice. The same observations can be made for other variants of our algorithm, for instance with **Global** restarts, or other values of Δn, Δs, or other values of ω.

Choice of δ_0	Problem 1	Problem 2	Problem 4
$\delta_0 = 1000$	$\mathbf{65 \pm 31}$	106 ± 35	$\mathbf{93 \pm 34}$
$\delta_0 = 500$	70 ± 28	104 ± 35	94 ± 33
$\delta_0 = 100$	72 ± 32	105 ± 34	95 ± 32
$\delta_0 = 50$	72 ± 32	112 ± 31	94 ± 31
$\delta_0 = 10$	75 ± 32	107 ± 31	97 ± 33
$\delta_0 = 5$	76 ± 31	$\mathbf{103 \pm 30}$	97 ± 32
$\delta_0 = 1$	72 ± 32	111 ± 30	96 ± 31
$\delta_0 = 0.5$	72 ± 30	117 ± 35	97 ± 31
$\delta_0 = 0.1$	70 ± 29	111 ± 29	99 ± 32
$\delta_0 = 0.05$	68 ± 29	116 ± 37	101 ± 30
$\delta_0 = 0.01$	74 ± 29	114 ± 32	101 ± 32
$\delta_0 = 0.005$	76 ± 26	114 ± 34	101 ± 31
$\delta_0 = 0.001$	76 ± 29	117 ± 34	102 ± 32

Table 7.8 – Mean regret $\pm$ 1 std-dev, for different choices of scaling factor for the confidence level δ_0 (*i.e.*, $\delta = \delta_0 \delta_T$), on three problems of horizon $T = 5000$, for GLR-klUCB with $\delta = \delta_0 / \sqrt{\Upsilon_T T}$.

7.10.7 Comparison of different threshold functions β

We compare the following threshold functions, $\beta(n, \delta)$ for $n \in \mathbb{N}^*$ and $\delta > 0$, that can be used in the B-GLRT test used for the GLR-klUCB algorithm (see the details in equation (7.12) and in Algorithm 7.1). In the B-GLRT test, there is a $\sup$ optimization problem on $s \in [2, n-1]$, and this $\sup$ is compared with the threshold $\beta(n, \delta)$. The threshold function given in (7.7) was chosen to have Lemma 7.3, that is a false alarm probability bounded by δ. We note that we also considered the possibility of using a threshold that could be a function of both n the sample size as well as $s \in [2, n-1]$ the "splitting index" between means $\widehat{\mu}_{1:s}$ and $\widehat{\mu}_{s+1:n}$. We did some preliminary experiments to explore this direction and did not find a significant difference, in terms of numerical efficient of the resulting GLR-klUCB algorithm, and mathematically the analysis presented in Section 7.4.2 is simpler to follow if the threshold are uniform on s.

1. The first variant is the one we advised to use in practice for GLR-klUCB, it is very simple to compute numerically:

$$\beta_1(n, \delta) \doteq \ln\left(\frac{3n^{3/2}}{\delta}\right) = -\ln(\delta) + \ln(3) + 3/2 \ln(n).$$

2. The second variant is smaller and does not use this power $3/2$:

$$\beta_2(n,\delta) \doteq \ln\left(\frac{1}{\delta}\right) + \ln(1+\ln(n)).$$

3. The third variant is using the function $\mathcal{T}$, as introduced by (7.6). The function $\mathcal{T}$ is computed with a numerical approximation[6] of the Lambert function $\mathcal{W}$, as explained in Section 7.4.2,

$$\beta_3(n,\delta) \doteq 2\mathcal{T}\left(\frac{\ln(2n^{3/2})/\delta}{2}\right) + 6\ln(1+\ln(n)).$$

4. The forth variant is using the function $\widetilde{\mathcal{T}}(x) = x + 4\ln(1+x+\sqrt{2x})$, as a simple approximation of $\mathcal{T}(x)$, which is valid and quite accurate as soon as $x \geq 5$,

$$\beta_4(n,\delta) \doteq 2\widetilde{\mathcal{T}}\left(\frac{\ln(2n^{3/2})/\delta}{2}\right) + 6\ln(1+\ln(n)).$$

As before, we consider the three problems 1, 2 and 4, with time horizon $T = 5000$, and we present in Table 7.9 the mean results of 100 independent runs. We only consider the variant of GLR-klUCB based on **Local restarts**, and we used the parameters ω_T, δ_T as given by the Corollary 7.12, and with the deterministic exploration scheme.

Threshold function	Problem 1	Problem 2	Problem 4
$\beta_1(n,\delta)$	$\mathbf{70 \pm 30}$	109 ± 32	99 ± 29
$\beta_2(n,\delta)$	73 ± 28	99 ± 32	$\mathbf{88 \pm 32}$
$\beta_3(n,\delta)$	77 ± 27	$\mathbf{89 \pm 32}$	134 ± 35
$\beta_4(n,\delta)$	77 ± 30	101 ± 30	135 ± 30

Table 7.9 – Mean regret $\pm\, 1$ standard-deviation, for different choices of threshold function $\beta(n,\delta)$, on three problems of horizon $T = 5000$, for GLR-klUCB.

As we could expect, the two choices β_1 and β_2 give similar results, as well as β_3 and β_4. The first two choices give better performance in most cases, and they are computationally less costly. The threshold β_1 is closer mathematically to the threshold β_3, that was used in the analysis, and thus it is the one we advise to use in practice. The sum-up of these experiments is that it sufficient to use the simplest and most explicit threshold β_1, instead of the more complicated one that was used for the analysis. Therefore, the choice of threshold function $\beta_1(n,\delta) \doteq \ln\left(3n^{3/2}/\delta\right)$ for our proposal GLR-klUCB presented in Algorithm 7.1 is validated by these experiments, and this choice gives good empirical performance.

[6] The Lambert $\mathcal{W}$ function is available in Python as `scipy.special.lambertw` from `scipy` [JOP+01].

7.10.8 Comparison of mechanisms used to enforce uniform exploration

We compare the following exploration mechanisms that can be used to enforce a sufficient exploration of all arms in the GLR-klUCB algorithm. All options are parametrized by a constant $\omega \in (0, 1)$, which represents the fraction of time steps spent in the forced exploration, either in average or in total.

1. The **deterministic exploration** corresponds to one described in Algorithm 7.1. At time t, if $t \mod \left\lfloor \frac{K}{\omega} \right\rfloor \in [K]$, then the arm $A_t \leftarrow t \mod \left\lfloor \frac{K}{\omega} \right\rfloor$ is played. It is the simplest, both the compute numerically and to handle mathematically, as the proof of Proposition 7.6 is short and simple. Its deterministic nature makes it the easiest choice for the proof skeleton given in Section 7.7.1, as the set $\mathcal{D}(T, \omega)$ used in the decomposition (7.22) of the regret is deterministic and thus it greatly simplifies the manipulation of expectations and random events. Note that this mechanism is also the one used by M-UCB [CZKX19].

2. The **uniform random exploration** is the one proposed for CUSUM-UCB [LLS18]. At time t, a random arm is played with probability ω/K. That is, first we sample a boolean variable from a Bernoulli law of mean ω, so that 0 indicates a random play, and 1 indicates a play using the UCB indexes. Then if it is a random play, arm $i \in [K]$ is selected uniformly at random (with probability $1/K$), and arm $A_t = i$ is played. Proving a result like Proposition 7.6 is not much harder for this second mechanism, but the difficulty lies in extending the proof skeleton we give in Section 7.7.1 to have a random set $\mathcal{D}(T, \omega)$.

3. The **tracking-based exploration** mechanism is inspired by the tracking trick used in [GK16], and it is actually quite intuitive. At any time t, instead of having a uniform probability of sampling forcing an exploration of every arm, it can make sense to force exploring arms that are not explored enough. This way, we actively enforce that each arm have enough samples. At time t, the goal is for any arm i to have been sampled more than $\omega(t - \tau_i)$ time, if $t - \tau_i$ represents the number of time steps since the last restart on this arm. So the tracking-based exploration samples an arm i uniformly at random among the set of arms i such that $n_i(t) < \omega(t - \tau_i)$, if it is not empty, otherwise it plays according to the UCB indexes. Numerically, it is more complicated than the two previous solutions but it stays reasonably efficient. It was the first direction we pursued for our analysis, but we dropped it since mathematically, it was harder to prove a result like Proposition 7.6, and it was also harder to incorporate the randomness of this exploration scheme in the regret decomposition (7.22).

Note that the analysis we gave in Section 7.6 is based on the deterministic exploration, but with a careful handling of random events and if we prove a result similar to Proposition 7.6, we believe our analysis could also be extended to another exploration mechanism.

As before, we consider the three problems 1, 2 and 4, with time horizon $T = 5000$, and we present in Table 7.10 the mean results of 100 independent runs. We include the two variants of GLR-klUCB, based on **Local restarts** or **Global restarts**, for which we used the parameters ω_T, δ_T as given by the two Corollaries 7.9 and 7.12.

Exploration mechanism	Variant	Problem 1	Problem 2	Problem 4
Deterministic exploration	Local	$\mathbf{68 \pm 33}$	116 ± 36	99 ± 33
	Global	97 ± 28	134 ± 36	131 ± 32
Uniform random exploration	Local	74 ± 30	$\mathbf{108 \pm 33}$	106 ± 31
	Global	91 ± 30	134 ± 33	129 ± 33
Tracking-based exploration	Local	73 ± 32	104 ± 33	$\mathbf{89 \pm 29}$
	Global	96 ± 26	133 ± 32	120 ± 30

Table 7.10 – Mean regret $\pm$ 1 standard-deviation, for different choices of exploration mechanisms, on three problems of horizon $T = 5000$, for GLR-klUCB, with local or global restarts.

As expected, all options give similar results, and each of the three options was found to outperform the two others in one of the three problems considered for this experiments (problems 1, 2 and 4). The result highlighted in **bold** in Table 7.10 shows the best algorithm in each problem (*i.e.*, the algorithm and its option that obtains the smaller mean regret in the considered column). We note that in terms of its average regret on the different problems, the tracking-based exploration is the best choice. The sum-up of these experiments is that it sufficient to use the simplest exploration scheme based on a deterministic exploration, rather than a more complicated exploration scheme based on tracking. Therefore, our choice of the deterministic exploration scheme for our proposal GLR-klUCB presented in Algorithm 7.1 is validated by these experiments, and this choice gives good empirical performance.

Part III

Conclusion and Appendixes

Chapter 8

General Conclusion and Perspectives

8.1 General Conclusion

This manuscript was organized in two parts, Part I and Part II. It presented the problems we considered for my PhD research, and we discussed about the main contributions produced during the last three years.

- The first half, Part I, started by motivating our work and situating the context of this research. In Chapter 2, we presented the multi-armed bandit models, the hypotheses we usually make, and the most important algorithms that solve different kinds of MAB problems. Then in Chapter 3 we discussed in details about our open-source Python library, SMPyBandits, developed to run numerical simulations of MAB problems. Finally, we presented in Chapter 4 a first mathematical contribution, our algorithm Aggregator, as an efficient answer to the online algorithm selection problem.

- The second half of this thesis, Part II, presented our main contributions. In Chapter 5, we first proposed some models for realistic IoT wireless networks, in discrete times and with no central coordination between end-devices. In two different models, with or without retransmissions, we showed that end-devices can learn independently to increase their successful transmission rates in the network, by using low-cost MAB algorithms. We also presented a proof-of-concept that implements such network on real radio hardware and validates our model and proposed algorithms. Because the models studied for realistic IoT networks were found to be intractable to analyze from a theoretical point of view, mainly due to the large number of algorithms interacting at random times in an unpredictable environment, in Chapter 6, we studied a weaker but still interesting model. We presented different variants of the multi-players MAB model, and we proposed the state-of-the-art algorithm for the variant with collision information (MCTopM-kl-UCB). Finally, in Chapter 7, we removed the stationary hypothesis on the single-player MAB problem, and we analyzed a new actively adaptive algorithm, for which we also proved state-of-the-art results (GLR-klUCB). For the last two chapters, our proposed algorithms attain state-of-the-art performances both on numerical simulations and on the regret bounds obtained in our theoretical analyses.

A more precise summary of each chapter is given below.

Conclusion about Chapter 2. We started to present the multi-armed bandit model, with a finite number of arms and real-valued rewards. Our main focus is on one-dimensional exponential families of distributions, and on stochastic and stationary problems. The notations used in the manuscript were introduced by showcasing an interactive demonstration[1].

Conclusion about Chapter 3. Then we presented our Python library SMPyBandits [Bes19, Bes18], by detailing its purpose and qualities, its organization, and an overview of its usage. SMPyBandits allows any researcher to easily implement numerical simulations of stochastic or piece-wise stochastic problems of single- or multi-players multi-armed bandits. SMPyBandits is distributed on GitHub and Pypi freely, under an open-source license, and it is extensively documented (at `SMPyBandits.GitHub.io`).

We detailed how SMPyBandits is implementing arms, problems, algorithms, and how the library uses these components to implement a simulation loop, with various visualizations being performed after a simulation. Even if SMPyBandits is still restricted to the finite-arm case, it supports a wide range of arm distributions (as well as rested or restless Markov models). Different kinds of models are implemented, from stationary single-player to piece-wise stationary multi-players with different collision models. One of the main qualities of the library is that it is quite exhaustive, as all the main families of algorithms covering these different models have been implemented, even very recent algorithms from the literature. More than 65 algorithms (or variants) are implemented for the single-player case, 5 for the experts aggregation problem, about 15 for the multi-players case, and about 20 for the piece-wise stationary case. The entire codebase is fully documented, and the library is using continuous integration to run automated tests on the code after every modification. When comparing algorithms on a problem, the main performance measure is the regret, but the library also computes, stores and visualizes other measures, such as best-arm selection rate or mean cumulated reward, as well as real time and memory costs. Our library is used to run numerical simulations on Chapters 2 6 and 7, and in other contributions such as our article [BK18b].

[1]See `perso.crans.org/besson/phd/MAB_interactive_demo/`

Conclusion about Chapter 4. This first part ends by presenting one of our first contributions [BKM18]. We tackle the question of how to select a particular bandit algorithm when a practitioner is facing a particular (unknown) bandit problem. Instead of always choosing her favorite algorithm, or running costly benchmarks before real-world deployment of the chosen algorithm, a third solution for a practitioner can be to select a few candidate algorithms, where at least one is expected to be very efficient for the given problem, and use online algorithm selection to automatically and dynamically decide the best candidate. We proposed an extension of the Exp4 algorithm for this problem, Aggregator, and illustrate its performance on some bandit problems.

Conclusion about Chapter 5. This chapter started the second part of the manuscript, Part II, and presents three main contributions.

First in Section 5.2, we proposed an evaluation of the performance of applying MAB learning algorithms in IoT networks, with a focus on the convergence of algorithms, in terms of successful transmission rates. We restrict to the case of a protocol slotted in both time and frequency, and we assume a certain clock pre-agreement and time synchronisation between devices and the base station. We studied the behavior of the network when the proportion of intelligent dynamic devices changes. Concretely, increasing this probability allows to insert more devices in the same network, while maintaining a good Quality of Service. Similarly, if the number of devices remain constant, increasing the successful transmission rate directly improves the network QoS. We show that both UCB and Thompson sampling algorithms have near-optimal performance, even when their underlying *i.i.d.* assumption (see Chapter 2) is violated by the presence of many "intelligent" end-devices which follow a random activation process. This is both surprising and encouraging, it shows that applying bandit algorithms tailored for a stochastic model is still useful in broader settings. The fully *decentralized* application of classic stochastic MAB algorithms are almost as efficient as the best possible centralized policy in this setting, after a short learning period, even though the dynamic devices *cannot* communicate with each other, and *do not* know the system parameters.

We presented in Section 5.3 a proof-of-concept, demonstrated in 2018 at the ICT conference [BBM18], and further detailed in the companion paper [BBM19]. We gave all the necessary details on both the PHY and the MAC layer, as well as details on the User Interface developed for the demo. Results obtained in practice were discussed, to highlight the interest of using learning algorithms for radio online optimization problem, and especially multi-armed bandit learning algorithms. By using such low-cost algorithms, we demonstrated empirically that a dynamically re-configurable device can learn on its own to favor a certain channel, if the environment traffic is not uniform between the K different channels, by using onlt the acknowledgement (*Ack*) feedback sent from the base station.

Finally in Section 5.4, we presented an extension of our model of LPWA networks based on an ALOHA protocol, again under the hypothesis of a protocol slotted in time and frequency and with perfect synchronisation. If the priority is the Quality of Service (QoS), for instance with renewable energy capabilities, this second model is more appropriate. We presented and evaluated several heuristics that try to learn how to transmit and retransmit in a smarter way, by using the UCB algorithm for channel selection for first transmission (first-stage), and different proposals based on UCB for the retransmissions upon collisions (second-stage). The main novelty of this model is to also learn the most efficient way to retransmit packets upon radio collision, by also using a multi-armed bandit algorithm for this second-stage optimization problem. We showed that incorporating learning for the transmission is needed to achieve optimal performance, with significant gain in terms of successful transmission rate in networks with a large number of devices (up-to 30% in the example network). Our empirical simulations show that each of our proposed heuristic outperforms a naive random access scheme. Surprisingly, the main take-away message is that a simple UCB learning approach (that simply retransmits in the same channel), turns out to perform almost as well as more complicated heuristics.

Thus in Chapter 5, we proposed two models of Internet of Things (IoT) networks, composed of many independent IoT end-devices, that can use low-cost RL (Reinforcement Learning) algorithms to learn to improve their spectrum access. Decentralized RL for IoT lets the devices use acknowledgements sent by their Base Station as a reward, instead of sensing feedback like for OSA. We consider many independent "dynamic" devices, communicating with a small probability at every instant. The first model (without retransmission of packets), is interesting for its simplicity, and because considering no retransmission can improve the battery life of the IoT devices, at the cost of a lower Quality of Service (QoS). However, in case of failed transmission of a message, a dynamic device can also retransmit it up-to a fixed number of retransmission (*e.g.*, 10 times). The second model is more interesting if the main target is an improvement of the QoS, rather than a longer battery life.

Conclusion about Chapter 6. The second chapter of Part II of this manuscript studied Multi-Players Multi-Arm Bandits models. We present three variants of this model, with different level of feedback being available to the decentralized players, under which we proposed efficient algorithms. For the two easiest models –with sensing–, our theoretical contribution improves the state-of-the-art upper bounds on the regret. In the absence of sensing, we also provide some motivation for the practical use of the Selfish heuristic, a simple index policy based on hybrid indices that are directly taking into account the collision information. We also reviewed various variants of this model, and for some interesting variants we discussed the related literature, which has proven to be very active in the last two years. For some of these variants, we explained why our approach does not work efficiently without modifications, but we detailed and illustrated how to adapt MCTopM to other settings. For example, it assumes to know the number of players M before-hand, but we illustrated that previously introduced technique to estimate M can also be applied to our proposal and give satisfactory empirical performances.

Conclusion about Chapter 7. In the last chapter, we were interested in relaxing the stationary hypothesis made in Chapter 2. We first describe the break-point detection problem, that we solve by analyzing at finite-time the properties of the Bernoulli GLR change-point detector. This sequential break-point detection test can be applied for bounded rewards, and we bound its false alarm probability as well as its detection delay. For the piece-wise stationary bandit model, we proposed a new efficient algorithm, GLR-klUCB, which combines the kl-UCB algorithm with the Bernoulli GLR change-point detector. This actively adaptive method attains state-of-the-art regret upper-bounds when tuned with a prior knowledge of the number of changes Υ_T, but without any other prior knowledge on the problem, unlike CUSUM-UCB and M-UCB that require to know a lower bound on the smallest magnitude of a change. We also gave numerical evidence of the efficiency of our proposal, which performs usually much better than all the passively adaptive approaches as well as better or comparably to the previous actively adaptive ones.

8.2 Perspectives

As discussed at the end of each chapter, our works suggest possible directions of future studies.

Perspectives about Part I. As detailed in Chapter 2, we focused in this thesis on finite-arms and one-dimensional bandit problems, and thus two possible directions of future research could be to extend our works to MAB models with either multidimensional rewards, like contextual bandits, or infinite arms, like Lipschitz bandits. The hypotheses we made were motivated by the target applications, but these other models have also started to be used for cognitive radio applications.

A few tasks are left on our library SMPyBandits, a first one could be to implement new variants of the single-player stochastic models, as well as variants for the multi-players or the non-stationary cases. For more details, see the issue tickets at `GitHub.com/SMPyBandits/SMPyBandits/issues/`. A second interesting task could be to extend the library for problems with non-finite arms, *e.g.*, linear or contextual bandit (ticket 117), or to add support for the "dynamic case" of multi-players bandits to allow arrivals or departures of players (ticket 124). A third task that I would have liked to complete is to interface the library with a web-based interactive demonstration, in order to allow anyone to launch simulations without any knowledge of programming. It is already possible to reproduce some of the experiments presented in this thesis, by using the Jupyter notebooks [K+16], hosted on `perso.crans.org/besson/PhD/notebooks/`, which can be used locally if you install the library, but can also be used on cloud platforms, such as Binder[2] or Google Colab[3].

Perspectives about Chapter 5. The first model presented in Section 5.2 could easily be generalized with two probabilities p_S and p_D, if static and dynamic devices have different transmission pattern, and less easily with one probability per device. Also, other emission patterns could be considered, instead of a Bernoulli process for each user. In this whole Chapter 5, we prefer to consider that all devices have the same activation probability, to keep the notation as simple as possible. Moreover, for sake of simplicity we supposed that all devices use the same standard. Future works could consider more realistic interference scenarios and IoT networks, with, *e.g.*, non-slotted time, more than one base station etc.

Another extension could be to not have a Bernoulli process (or any random process), but a fixed rate of transmission, *e.g.*, one transmission a day. So additionally to deciding the channel for communication (*i.e.*, *where* to communicate), each device has to also decide *when* to communicate. However, this clearly leads to a much larger action space, as there are many time slots in one day (for example), and thus we believe that as soon as the action space becomes too large in this extension, the simple MAB-based learning approach could be no longer appropriate. It is well-known in the MAB literature that the larger the action space, the slower is the convergence speed of any stationary MAB algorithms. It could be exciting to study the possible application of *contextual* MAB [LCLS10, LWAL18] or structured MAB [CMP17] models and algorithms for this extension.

In order to validate our results in a realistic experimental setting and not only with simulations, future works should include a hardware implementation of the analyzed models, in order to complete our demonstration presented in Section 5.3 [BBM18, BBM19]. A hardware demonstrator could be also benefit to study other settings by relaxing some hypotheses, for instance by studying a similar model in non-slotted time.

Perspectives about Chapter 6. Our study on multi-players bandits suggests several further research directions. First, one could want to investigate the notion of *optimal algorithms* in the decentralized multi-players model with sensing information. So far we provided the first matching upper and lower bound on the expected number of sub-optimal arms selections, which suggests some form of (asymptotic) optimality. However, sub-optimal draws turn out not be the dominant terms in the regret, both in our upper bounds and in practice, thus a promising future work is to identify some notion of *minimal number of collisions*.

[2] You can try on `mybinder.org/v2/gh/SMPyBandits/SMPyBandits/master` from your browser.
[3] See `colab.research.google.com/github/SMPyBandits/SMPyBandits/tree/master/notebooks/`.

We also presented many extensions of the multi-players bandit model, and even if some have already been implemented, a major future work is to implement the most interesting ones in our library SMPyBandits (see tickets 120, 124, 185).

Perspectives about Chapter 7. We believe that our new proof technique could be used to analyze GLR-klUCB under less stringent assumptions than the one made in this chapter (and in previous work), that would require only a few "meaningful" changes to be detected. This promising research direction is left for future work, but the hope is that the regret would be expressed in term of this number of meaningful changes instead of Υ_T. We shall also investigate whether actively adaptive approaches can attain a $\mathcal{O}(\sqrt{\Upsilon_T T})$ regret upper-bound without the knowledge of Υ_T. We also believe that combining change-point *localization* with an efficient change-point detection algorithm, such as GLR-klUCB, could lead to an interesting class of algorithms, as suggested by [Mai19].

We would also like to study in the future possible extension of our approach to the slowly varying model, as studied in [BGZ14, LRC+16, WS18b], or the rotting bandit model [SLC+19].

Another interesting direction of future work is to study non-stationary distributed multi-players bandits, to unify the models from Chapter 6 and 7. A natural extension of the non-stationary model is to consider non-communicating players cooperating in a decentralized way to play the same bandit game, as it was proposed recently in [WS18a]. We could also build on the proof technique used in [WS18b], even if we are interested in removing the hypothesis that player j knows its ID $j \in [M]$ before starting to play. A promising direction is to directly try to join our contributions from Chapter 6 and 7, and propose an efficient algorithm using three parts: kl-UCB indexes for arm selection, MCTopM for orthogonalization (*i.e.*, dealing with collisions), and the Bernoulli GLR break-point detector for non-stationarity (*i.e.*, dealing with abrupt changes). The three parts should be connected, the same way we built GLR-klUCB by connecting both components. We propose the idea of incorporating the detected change-points by B-GLR test in the orthogonalization procedure: after any change-point is detected, the player is not sitting anymore. Analyzing this strategy is left as a future work.

Other directions of future works. Another very interesting direction of research is to propose a unique "unified" algorithm that can be used in all the different settings studied in this thesis, and automatically adapt to the setting at hand. For instance, even if GLR-klUCB is very efficient for piece-wise stationary problems, its forced exploration makes it sub-optimal for stationary problems. Another example is for multi-players bandits, where MCTopM-kl-UCB performs sub-optimally on piece-wise stationary problems. One approach to obtain a unified "master" algorithm could be to use expert aggregation on the different state-of-the-art algorithms presented in this thesis, and as each one is efficient in a different setting, the resulting aggregated algorithm could also be efficient in each of the different settings. But as we illustrated in Section 4.3.1, this approach does not seem theoretically promising.

Appendix A

About Doubling Tricks for Multi-Armed Bandits

This appendix quickly presents the contributions of another publication that was not presented in the main text of this thesis. We studied doubling tricks and their possible uses for multi-armed bandits, between fall 2017 and spring 2018, and we wrote an article [BK18b] which was rejected at the COLT 2018 conference, and unfortunately we lacked the time to improve it and resubmit it to another conference.

Motivations for anytime algorithms. As introduced in Chapter 2, an online reinforcement learning algorithm is *anytime* if it does not need to know in advance the horizon T of the experiment. Depending on the context of the practical application of interest, it might be unrealistic to assume to know in advance T. We give two examples to illustrate where this prior knowledge can be realistic or not. On the one hand, consider clinical trials: it is likely that the number of patients is known before starting the trial, and for instance we refer to all the research that studies the setting of fixed-time best-arm identification [ABM10, GK16]. On the second hand, consider cognitive radio and decentralized MAB learning implemented on IoT devices (like we present it in Chapter 5): usually a learning step corresponds to an uplink and then downlink message sent and received by the IoT device, and so the time horizon T denotes the total number of such messages. While it can be assumed that the device will run for instance for 10 years (as it is advertised by some companies), many kinds of application such as medical sensors or connected fields cannot predict the number of total communications before setting up the device.

For this later range of applications, it is of highest interest to be able to use low-cost MAB algorithms that do not rely on prior knowledge of the problem for which they will be applied, and especially do not need to know the horizon T. In this thesis, while we proposed in Chapter 6 any-time algorithms solving the presented problem of stationary multi-players bandit, our solution for the problem of non-stationary MAB problem studied in Chapter 7 does rely on a prior knowledge of the horizon T. It is interesting to know what is the best approach to fix this weakness: work more and design algorithms that are inherently any-time, or use a generic technique to avoid depending on a prior knowledge of T, and automatically transform our non-anytime approach to make it anytime.

Solution to "patch" a non-anytime algorithm. A well-known technique to obtain an any-time algorithm from any non-anytime algorithm is the "Doubling Trick", as first introduced in [CBL06] and used for instance in [AO10, AGO18]. In the context of adversarial or stochastic multi-armed bandits, the performance of an algorithm is measured by its regret, and we study two families of sequences of growing horizons (geometric and exponential) to generalize previously known results that certain doubling tricks can be used to conserve certain regret bounds. In a broad setting, we prove that a geometric doubling trick can be used to conserve (minimax) bounds in $R_T = \mathcal{O}(\sqrt{T})$ but *cannot* conserve (distribution-dependent) bounds in $R_T = \mathcal{O}(\ln T)$. We give insights as to why exponential doubling tricks may be better, as they conserve bounds in $R_T = \mathcal{O}(\ln T)$, and are close to conserving bounds in $R_T = \mathcal{O}(\sqrt{T})$. Interestingly, we prove that they conserve bounds of the mixed form $R_T = \mathcal{O}(T^{\gamma}(\ln(T))^{\delta})$, for $0 < \gamma < 1$ and $\delta > 0$, and so an exponential doubling trick could be used to obtain an anytime version of the algorithm we propose for non-stationary bandits, GLR-klUCB in Chapter 7. However, our study was only focussing on stationary bandit, and it is left as a future work to explore the harder case of piece-wise stationary problems.

In our article [BK18b], we also present numerical experiments in the case of stationary MAB problems, to compare the performance of efficient anytime algorithms, like kl-UCB, against efficient non-anytime algorithms, like kl-UCB^{++} from [MG17], made anytime with different choices of doubling-trick. We conclude that for such problems, if T is not known before, it is always more efficient to use policies that were designed to be anytime that to use a doubling trick. For example, an applicative paper written in 2018, [LLL19], only tested the use of an exponential doubling trick combined with kl-UCB^{++}, but most surely the kl-UCB algorithm would have given better empirical performance as well as more robust results…

To conclude this work, we would need to complete the study of the doubling tricks, and instead of focussing on two families (geometric and exponential), we need to completely characterize the doubling tricks that allow to preserve certain regret bound. Such doubling is given either by the function mapping the current time t to the current estimate of the horizon $T(t)$, or the sequence $(T_i)_{i\in\mathbb{N}^*}$ of successive estimates of the horizons. We are interested in pursuing this work, in the hope of finding an intermediate sequence, growing faster than a geometric but slower than an exponential doubling sequence, that can preserve both worlds, problem-dependent bounds in $\mathcal{O}(\ln(T))$ and problem-independent bounds in $\mathcal{O}(\sqrt{T})$.

Abbreviations and Notations

Acronyms and Abbreviations

Ack	*Acknowledgement*
ALOHA	*ALOHA* (not an acronym)
BTS	*Base Tranceiver Station* (or *gateway*)
CPU	*Central Processing Unit*
CR	*Cognitive Radio*
DSA	*Dynamic Spectrum Access*
GLR, GLRT	*Generalized Likelihood Ratio*, *Generalized Likelihood Ratio Test*
i.i.d.	*identically and independently distributed* (variables, observations or samples)
IoT	*Internet of Things*
ISM	*Industrial, Scientific and Medical* (bands)
KL	*Kullback-Leibler* (divergence)
klUCB	*Kullback-Leibler Upper Confidence Bound* (algorithm)
LAN	*Local Area Network*
LPWAN	*Low-Power Wide-Area Network*
MAB	*Multi-Armed Bandit*
MAC	*Medium Access Control* (layer)
MCTopM	*Musical-Chair on Top-M* (algorithm)
ML	*Machine Learning*
NB-IoT	*Narrow-Band Internet of Things*
NOMA	*Non-Orthogonal Multiple Access*
OSA	*Opportunistic Spectrum Access*
PHY	*PHYsical* (layer)
PLR	*Packet Loss Ratio*
PU	*Primary User*
QPSK	*Quadrature Phase-Shift Keying*
RAM	*Random Access Memory*

RandTopM	*Random Hoping on Top-M* (algorithm)
RF	*Radio Frequency*
RL	*Reinforcement Learning*
SCEE	*Signal, Communication et Électronique Embarquée* (research team in CentraleSupélec, campus of Rennes)
SNR	*Signal to Noise Ratio*
SU	*Secondary User*
TS	*Thompson Sampling* (algorithm)
UCB	*Upper Confidence Bound* (object or algorithm)
USRP	*Universal Software Radio Peripheral*
WLAN	*Wireless Local Area Network*
wlog	*with loss of generality*
wrt	*with respect to*

Exponents

y^j	Usually denotes a variable depending on a player in Chapter 6 (for $j \in [M]$), or to distinguish between different independent runs in numerical simulations in Chapter 2 (for $j \in [N]$)
$y^{(i)}$, $y^{(\ell)}$	Usually the superscript index (i) or (ℓ) denotes a variable in the i-th stationary interval, under the point-of-views of *global* or *local* changes, in Chapter 7

Greek symbols

α	Denotes the parameter of a UCB algorithm in Chapters 2 and 5
α_0, δ_0	Usually denotes a constant scaling of a parameter of an algorithm, for instance $\varepsilon_t = \varepsilon_0/t$ is used in Chapter 2, or $\alpha = \alpha_0 \alpha_T$ is used in Chapter 7
$\beta(n, \delta)$	Usually denotes a threshold for the statistical (GLR) tests in Chapter 7
$\boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\lambda}$,	Vector of means $(\mu_k) = \mu_1, \dots, \mu_K$, or vector of distributions $(\nu_k)_k$ or $(\lambda_k)_k$ (characterizing a problem)
Δn, Δs	Parameters of numerical optimization tricks in Chapter 7
Δ	Usually denotes the gap in terms of means of arms, usually between the best and second best arms, in Chapters 2, 6. In Chapter 5 in Algorithm 5.5 it denotes a delay. In Chapter 7, it can denote different gaps (Δ^{opt} and Δ^{change})
δ	Usually denotes a lower-bound on the gap Δ, known before hand by an algorithm, in Chapters 6, 7. Also denote a confidence level of an algorithm in Section 7.4
γ	Usually denotes a discount factor, *e.g.*, in D-UCB or D-TS in Chapter 7
μ^*, μ_k^*	Mean of the optimal arm (*i.e.*, $\mu^* = \max_k \mu_k$), and mean of the k-th best arm
μ_k, $\mu_k(t)$, μ_k^j, $\mu_k^{(i)}$	Mean of arm k, arm k at time t (in Chapter 7), arm k for user j (in Chapter 6), arm k in the i-th stationary interval (in Chapter 7)
ν_k, $\nu_k(t)$, ν_k^j	Distribution of arm k, arm k at time t, arm k for user j
ω	Denotes a parameter controlling the forced exploration mechanism in Chapter 7

π	It usually denotes the number $\pi \simeq 3.14\dots$, but it also denotes a probability distribution in Chapter 4 or a permutation in Section 6.4.2
τ	Length of a sliding-window, *e.g.*, in SW-UCB in Chapter 7
τ_k^j, $\tau_k^{(i)}$	Location of a change-point, *e.g.*, the j-th change-point on arm k, in Chapter 7
Υ_T	Number of break-points in a piece-wise stationary MAB problem in Chapter 7. NC_i denotes the number of change-points on arm i, and $C_T = \sum_{i=1}^K \mathrm{NC}_i$ the number of change-points on the arms
ε	Usually denotes a small positive real value, *e.g.*, the parameter for the ε-greedy algorithm, or the drift correction parameter for CUSUM in Chapter 7

Indices

x_k	Usually denotes a variable depending on an arm, for $k \in [K]$
$Y_{k,t}$	Usually denotes a variable depending on an arm $k \in [K]$ and on time $t \in [T]$

Roman symbols

$\mathcal{A}$	An algorithm, also referred to as a policy or a strategy. $\mathcal{A}_1, \dots, \mathcal{A}_N$ denote the N aggregated algorithms in Chapter 4 and $\mathcal{A}_1, \dots, \mathcal{A}_M$ denotes the algorithms of the M players in Chapter 6
$\mathcal{H}_0$, $\mathcal{H}_1$, $\mathcal{H}_2$	Hypothesis, in Chapters 5 and 7
$\mathcal{T}$	In Chapter 7, denote a function in (7.6) and a set of time steps in Section 7.10.4
MaxBackOff	Maximum number of retransmission of a packet in the ALOHA protocol in Chapter 5
$\mathrm{UCB}_k(t)$	Upper-Confidence Bound (UCB) of arm k at time t for an index policy
$\widehat{\mu}_k(t)$	Empirical mean of rewards obtained for arm k at time t
$A(t)$, $A^j(t)$	Decision of algorithm $\mathcal{A}$ at time $t \in [T]$, $A(t) \in [K]$ (from algorithm $\mathcal{A}$), decision for user $j \in [M]$ in Chapter 6 (from algorithm $\mathcal{A}^j$)
$C_k(t)$, $C^j(t)$	In Chapter 6, collision indicator on arm $k \in [K]$ or for user $j \in [M]$, at time t
$C_{\boldsymbol{\mu}}$, $D_{\boldsymbol{\mu}}$, $G_{M,\boldsymbol{\mu}}$	In Chapter 6, constants depending on the problem parameters only and on M
D, D_k	In Chapter 5, total number of *dynamic* devices in the network or in channel k
$d^{(i)}$, $d^{(\ell)}$	Lower-bound on number of samples for accurate detection of change-points in Chapter 7
$g_k^j(t)$, $g^j(t)$	Index of (arm k) at time t of user j for an index policy in Chapter 6
i, j	Usually denotes the i-th or j-th player, $i, j \in [M]$ in Chapter 6
$I_k(t)$	Index of arm k at time t for an index policy
K	Number of arms for multi-armed bandit games
k	Usually denotes the k-th arm, $k \in [K]$, mainly used in subscripts
M	Number of player for multi-players bandit games in Chapter 6
m	Maximum length of the back-off interval after a collision, in Chapter 5
N	Usually denotes the number of independent repetitions of the same numerical experiments (*e.g.*, $N = 1000$). In Chapter 5, N denotes the number of IoT (dynamic) devices in the studied network

$N_k(t)$	Number of samples obtained for arm k at time t
O_t	Vector of observations until time t in Chapter 6
p	In Chapter 5, probability of transmission for the devices following a Bernoulli random emission pattern (*e.g.*, $p = 10^{-5}$)
$r(t)$, $r^j(t)$	Reward obtained at time t, for user j at time t
R_T, $R_T^{\mathcal{A}}$	Regret (of an algorithm $\mathcal{A}$) for horizon T
S, S_k	In Chapter 5, total number of *static* devices in the network or in channel k
T	Time horizon, the duration of the bandit game (always $T \geq 1$)
t, s, n, r	Time step, $t \in [T]$. Chapter 7 also uses s, n and r, *e.g.*, in the $\sup$ of the stopping times or some technical lemmas
T_0, T_1	Fixed durations of some algorithms based on different phases, *e.g.*, the Musical Chair algorithm from [RSS16]
$Y_{k,t}$	Random sample from the arm k at time t

Mathematical notations

$[K]$, T, $[N]$ etc	For an integer $N \in \mathbb{N}, N \geq 0$, $[N]$ denotes the set $\{1, \dots, N\} = \{n \in \mathbb{N} : 1 \leq n \leq N\}$. If the order is important, it is ordered from 1 to N.
$\mathcal{E}$, $\mathcal{E}_T$	Used in Chapter 7 to denote "good events" that happen most of the time
$\mathcal{F}$, $\mathcal{F}_t$	Filtration in a probabilistic model, after $t-1$ prior observations
$\mathcal{W}$	Lambert $\mathcal{W}$ function, the first branch of the inverse of $x \mapsto x\exp(x)$, cf. [CGH+96]
$\mathbb{E}$	Expectation under a probabilistic model
kl	Binary relative entropy, KullBack-Leibler divergence between two Bernoulli distribution: $\mathrm{kl}(x, y) = x\ln(x/y) + (1-x)\ln((1-x)/(1-y))$
$\lfloor\bullet\rfloor$, $\lceil\bullet\rceil$	Floor $\lfloor x \rfloor = \sup\{n \in \mathbb{Z}, n \leq x\}$ and ceil $\lceil x \rceil = \inf\{n \in \mathbb{Z}, x < n\}$ functions
$\mathbb{1}(E)$	Indicator function of an event E ($= 1$ if and only if the event E is true)
$\mathbb{P}$	Probability measure under a probabilistic model
$\widehat{X}$	Usually denotes an "empirical value", a mean or an estimate of a quantity X that depends on time, *e.g.*, $\widehat{\mu_k}$ the empirical mean of arm k or $\widehat{S}_t$ the set of selected arms in Section 6.3.3
$\widetilde{X}$	Usually denotes another "empirical value" or an estimate of a quantity X that depends on time, *e.g.*, $\widehat{S}_t$ the set of selected arms in Section 6.4.1. Also denote a surrogate for a function without a closed form, *e.g.*, $\widetilde{T}$ in Section 7.10.7
$d(x, y)$	Divergence function between two distributions of parameters x and y, in a one-dimensional exponential family, in Chapter 7
E^c	Complement of an event E
f, g, h	Real-valued functions, *e.g.*, exploration function used for kl-UCB indexes, $f(t) = \ln(t) + 3\ln(\ln(t))$
$o(\bullet)$, $\mathcal{O}(\bullet)$, $\Omega(\bullet)$	Landau notations for positive functions: $f(x) = o(g(x)$ means that $g(x) \neq 0$ and $f(x)/g(x) \to 0$ for $x \to \infty$, $f(x) = \mathcal{O}(g(x))$ means that there exists $x_0, K > 0$ such that $f(x) \leq Kg(x)$ from $x \geq x_0$, and $f(x) = \Omega(g(x))$ means $g(x) = \mathcal{O}(f(x))$
X'	Usually denotes a "wrong value" of a variable X, for instance M' denotes in Section 6.7.1. Also denotes the derivative of a function, *e.g.*, f'

List of Figures

List of Algorithms

List of Code Examples

List of Tables

List of References

[AB09] J-Y. Audibert and S. Bubeck. Minimax Policies for Adversarial and Stochastic Bandits. In *Conference on Learning Theory*, pages 217–226. PMLR, 2009.

[AB10] J.-Y. Audibert and S. Bubeck. Regret Bounds And Minimax Policies Under Partial Monitoring. *Journal of Machine Learning Research*, 11:2785–2836, 2010.

[ABM10] J-Y. Audibert, S. Bubeck, and R. Munos. Best Arm Identification in Multi-Armed Bandits. In *Conference on Learning Theory*, page 13. PMLR, 2010.

[Abr70] N. Abramson. The ALOHA System: Another Alternative for Computer Communications. In *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference*, AFIPS '70 (Fall), pages 281–285, New York, NY, USA, 1970. ACM.

[AC18] A. Azari and C. Cavdar. Self-organized Low-power IoT Networks: A Distributed Learning Approach. In *Global Communications Conference*, Abu Dhabi, UAE, December 2018. IEEE.

[ACBF02] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time Analysis of the Multi-armed Bandit Problem. *Machine Learning*, 47(2):235–256, 2002.

[ACBFS95] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a Rigged Casino: The Adversarial Multi-Armed Bandit Problem. In *Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE, 1995.

[ACBFS02] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The Non-Stochastic Multi-Armed Bandit Problem. *SIAM journal on computing*, 32(1):48–77, 2002.

[AF15] R. Allesiardo and R. Féraud. Exp3 with Drift Detection for the Switching Bandit Problem. In *International Conference on Data Science and Advanced Analytics*, pages 1–7. IEEE, 2015.

[AF17] R. Alami and O.-A. Maillard R. Féraud. Memory Bandits: Towards the Switching Bandit Problem Best Resolution. In *Conference on Neural Information Processing Systems*, 2017.

[AFM17] R. Allesiardo, R. Féraud, and O.-A. Maillard. The Non-Stationary Stochastic Multi-Armed Bandit Problem. *International Journal of Data Science and Analytics*, 3(4):267–283, 2017.

[AG12] S. Agrawal and N. Goyal. Analysis of Thompson sampling for the Multi-Armed Bandit problem. In *Conference On Learning Theory*, pages 36–65. PMLR, 2012.

[AGO18] P. Auer, P. Gajane, and R. Ortner. Adaptively Tracking the Best Arm with an Unknown Number of Distribution Changes. *European Workshop on Reinforcement Learning*, 2018.

[Agr95] R. Agrawal. Sample mean based index policies by $\mathcal{O}(\ln n)$ regret for the Multi-Armed Bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

[AHK12] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[AHK17] S. Adish, H. Hassani, and A. Krause. Learning to Use Learners' Advice. *arXiv preprint arXiv:1702.04825*, 2017.

[ALK19] P. Alatur, K. Y. Levy, and A. Krause. Multi-Player Bandits: The Adversarial Case. *arXiv preprint arXiv:1902.08036*, 2019.

[ALNS17] A. Agarwal, H. Luo, B. Neyshabur, and R. E. Schapire. Corralling a Band of Bandit Algorithms. In *Conference on Learning Theory*, pages 12–38. PMLR, 2017.

[AM15] O. Avner and S. Mannor. Learning to Coordinate Without Communication in Multi-User Multi-Armed Bandit Problems. *arXiv preprint arXiv:1504.08167*, 2015.

[AM16] O. Avner and S. Mannor. Multi-User Lax Communications: a Multi-Armed Bandit approach. In *International Conference on Computer Communications*. IEEE, 2016.

[AM18] O. Avner and S. Mannor. Multi-user Communication Networks: A Coordinated Multi-armed Bandit Approach. *arXiv preprint arXiv:1808.04875*, 2018.

[AMT10] A. Anandkumar, N. Michael, and A. K. Tang. Opportunistic Spectrum Access with multiple users: Learning under competition. In *International Conference on Computer Communications*. IEEE, 2010.

[AMTA11] A. Anandkumar, N. Michael, A. K. Tang, and S. Agrawal. Distributed Algorithms for Learning and Cognitive Medium Access with Logarithmic Regret. *Journal on Selected Areas in Communications*, 29(4):731–745, 2011.

[AO10] P. Auer and R. Ortner. UCB Revisited: Improved Regret Bounds For The Stochastic Multi-Armed Bandit Problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

[AVW87a] V. Anantharam, P. Varaiya, and J. Walrand. Asymptotically efficient allocation rules for the Multi-Armed Bandit problem with multiple plays - Part I: IID rewards. *Transactions on Automatic Control*, 32(11):968–976, 1987.

[AVW87b] V. Anantharam, P. Varaiya, and J. Walrand. Asymptotically efficient allocation rules for the Multi-Armed Bandit problem with multiple plays - Part II: Markovian rewards. *Transactions on Automatic Control*, 32(11):977–982, 1987.

[B$^+$18] G. Brandl et al. Sphinx: Python documentation generator. Online at: `sphinx-doc.org`, 2018.

[Bar59] G.A. Barnard. Control charts and stochastic processes. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 239–271, 1959.

[BBM$^+$17] R. Bonnefoi, L. Besson, C. Moy, E. Kaufmann, and J. Palicot. Multi-Armed Bandit Learning in IoT Networks: Learning helps even in non-stationary settings. In *12th EAI Conference on Cognitive Radio Oriented Wireless Network and Communication*, Lisboa, Portugal, 2017.

[BBM18] L. Besson, R. Bonnefoi, and C. Moy. Multi-Arm Bandit Algorithms for Internet of Things Networks: A TestBed Implementation and Demonstration that Learning Helps. Demonstration presented at International Conference on Telecommunications, June 2018.

[BBM19] L. Besson, R. Bonnefoi, and C. Moy. GNU Radio Implementation of MALIN: "Multi-Armed bandits Learning for Internet-of-things Networks". In *Wireless Communications and Networking Conference*, Marrakech, Morocco, April 2019. IEEE. Following a Demonstration presented at International Conference on Telecommunications (ICT) 2018.

[BBMVM19] R. Bonnefoi, L. Besson, J. C. Manco-Vasquez, and C. Moy. Upper-Confidence Bound for Channel Selection in LPWA Networks with Retransmissions. In *MOTIoN Workshop*, Marrakech, Morocco, April 2019. IEEE.

[BBS$^+$19] S. Behnel, R. Bradshaw, Dag S. Seljebotn, G. Ewing, W. Stein, G. Gellner, et al. Cython: C-extensions for python. Online at: `cython.org`, 2019.

[BCB12] S. Bubeck and N. Cesa-Bianchi. Regret Analysis of Stochastic and Non-Stochastic Multi-Armed Bandit Problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[Bes18] L. Besson. SMPyBandits: an Experimental Framework for Single and Multi-Players Multi-Arms Bandits Algorithms in Python. Preprint, submitted to JMLR MLOSS, `hal.archives-ouvertes.fr/hal-01840022`, 2018.

[Bes19] L. Besson. SMPyBandits: an Open-Source Research Framework for Single and Multi-Players Multi-Arms Bandits (MAB) Algorithms in Python, 2016–2019. Code at `GitHub.com/SMPyBandits/SMPyBandits/`, documentation at `SMPyBandits.GitHub.io/`.

[BGZ14] O. Besbes, Y. Gur, and A. Zeevi. Stochastic Multi-Armed Bandit Problem with Non-Stationary Rewards. In *Advances in Neural Information Processing Systems*, pages 199–207, 2014.

[BK18a] L. Besson and E. Kaufmann. Multi-Player Bandits Revisited. In *Algorithmic Learning Theory*, Lanzarote, Spain, 2018. Mehryar Mohri and Karthik Sridharan.

[BK18b] L. Besson and E. Kaufmann. What Doubling Trick Can and Can't Do for Multi-Armed Bandits. Preprint, `hal.archives-ouvertes.fr/hal-01736357`, February 2018.

[BK19a] L. Besson and E. Kaufmann. Analyse non asymptotique d'un test séquentiel de détection de ruptures et application aux bandits non stationnaires. *GRETSI*, August 2019. `hal.archives-ouvertes.fr/hal-XXX`.

[BK19b] L. Besson and E. Kaufmann. Combining the Generalized Likelihood Ratio Test and kl-UCB for Non-Stationary Bandits. Preprint, `hal.archives-ouvertes.fr/hal-02006471`, February 2019.

[BKM18] L. Besson, E. Kaufmann, and C. Moy. Aggregation of Multi-Armed Bandits Learning Algorithms for Opportunistic Spectrum Access. In *Wireless Communications and Networking Conference*, Barcelona, Spain, 2018. IEEE.

[BL18] I. Bistritz and A. Leshem. Distributed Multi-Player Bandits: a Game Of Thrones Approach. In *Advances in Neural Information Processing Systems*, pages 7222–7232, 2018.

[BLM13] S. Boucheron, G. Lugosi, and P. Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford university press, 2013.

[BMM14] A. Baransi, O.-A. Maillard, and S. Mannor. Sub-sampling for Multi-armed Bandits. *Proceedings of the European Conference on Machine Learning*, 2014.

[BN93] M. Basseville and I. Nikiforov. *Detection of Abrupt Changes: Theory And Application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[Bod17] Q. Bodinier. *Coexistence of Communication Systems Based on Enhanced Multi-Carrier Waveforms with Legacy OFDM Networks*. PhD thesis, CentraleSupélec, 2017.

[BP$^+$16] I. Bicking, PyPA, et al. Virtualenv: a tool to create isolated Python environments. Online at: `virtualenv.pypa.io`, November 2016.

[BP18] E. Boursier and V. Perchet. SIC-MMAB: Synchronisation Involves Communication in Multiplayer Multi-Armed Bandits. *arXiv preprint arXiv:1809.08151*, 2018.

[BR19] D. Bouneffouf and I. Rish. A Survey on Practical Applications of Multi-Armed and Contextual Bandits. *arXiv preprint arXiv:1904.10040*, 2019.

[BS12] S. Bubeck and A. Slivkins. The Best Of Both Worlds Stochastic And Adversarial Bandits. In *Conference on Learning Theory*, pages 42–1. PMLR, 2012.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Univ. Press, 2004.

[BV19] M. Bande and V. V. Veeravalli. Adversarial Multi-user Bandits for Uncoordinated Spectrum Access. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4514–4518. IEEE, 2019.

[C$^+$18] A. Collette et al. h5py: HDF5 for Python. Online at: `www.h5py.org`, 2018.

[CBL06] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[CGH$^+$96] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth. On the Lambert W Function. In *Advances in Computational Mathematics*, pages 329–359, 1996.

[CGM$^+$13] O. Cappé, A. Garivier, O-A. Maillard, R. Munos, and G. Stoltz. Kullback-Leibler Upper Confidence Bounds For Optimal Sequential Allocation. *Annals of Statistics*, 41(3):1516–1541, 2013.

[CLLW19] Y. Chen, C. Lee, H. Luo, and C. Wei. A New Algorithm for Non-stationary Contextual Bandits: Efficient, Optimal, and Parameter-free. *arXiv preprint arXiv:1902.00980*, 2019.

[CMP17] R. Combes, S. Magureanu, and A. Proutiere. Minimal Exploration in Structured Stochastic Bandits. In *Advances in Neural Information Processing Systems*, pages 1761–1769, 2017.

[CVZZ16] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi. Long-range communications in unlicensed bands: the rising stars in the IoT and smart city scenarios. *Wireless Communications*, 23(5):60–67, 2016.

[CZKX19] Y. Cao, W. Zheng, B. Kveton, and Y. Xie. Nearly Optimal Adaptive Procedure for Piecewise-Stationary Bandit: a Change-Point Detection Approach. In *International Conference on Artificial Intelligence and Statistics*, Okinawa, Japan, 2019.

[DH18] S. J. Darak and M. K. Hanawal. Distributed Learning and Stable Orthogonalization in Ad-Hoc Networks with Heterogeneous Channels. *arXiv preprint arXiv:1812.11651*, 2018.

[DMNM16] S. J. Darak, N. Modi, A. Nafkha, and C. Moy. Spectrum Utilization and Reconfiguration Cost Comparison of Various Decision Making Policies for Opportunistic Spectrum Access Using Real Radio Signals. In *11th EAI Conference on Cognitive Radio Oriented Wireless Network and Communication*, Grenoble, France, 2016.

[DMP16] S. J. Darak, C. Moy, and J. Palicot. Proof-of-Concept System for Opportunistic Spectrum Access in Multi-user Decentralized Networks. *EAI Endorsed Transactions on Cognitive Communications*, 2:1–10, 2016.

[DNMP16] S. J. Darak, A. Nafkha, C. Moy, and J. Palicot. Is Bayesian Multi Armed Bandit Algorithm Superior? Proof of Concept for Opportunistic Spectrum Access in Decentralized Networks. In *11th EAI Conference on Cognitive Radio Oriented Wireless Network and Communication*, Grenoble, France, 2016.

[DP16] R. Degenne and V. Perchet. Anytime Optimal Algorithms In Stochastic Multi Armed Bandits. In *International Conference on Machine Learning*, pages 1587–1595, 2016.

[Ett] Ettus. USRP Hardware Driver and USRP Manual. files.ettus.com/manual/page_usrp2.html. Accessed: 2018-09-25.

[Fou17] Python Software Foundation. Python language reference, version 3.6. Online at: `www.python.org`, October 2017.

[GBV18] G. Gautier, R. Bardenet, and M. Valko. DPPy: Sampling Determinantal Point Processes with Python. *arXiv preprint arXiv:1809.07258*, 2018. Code at `github.com/guilgautier/DPPy`. Documentation at `dppy.readthedocs.io`.

[GC11] A. Garivier and O. Cappé. The KL-UCB Algorithm for Bounded Stochastic Bandits and Beyond. In *Conference on Learning Theory*, pages 359–376. PMLR, 2011.

[GGCA11] N. Gupta, O. Granmo-Christoffer, and A. Agrawala. Thompson Sampling for Dynamic Multi Armed Bandits. In *International Conference on Machine Learning and Applications Workshops*, pages 484–489. IEEE, 2011.

[GHMS18] A. Garivier, H. Hadiji, P. Menard, and G. Stoltz. KL-UCB-switch: optimal regret bounds for stochastic bandits from both a distribution-dependent and a distribution-free viewpoints. *arXiv preprint arXiv:1805.05071*, 2018.

[GK16] A. Garivier and E. Kaufmann. Optimal Best Arm Identification with Fixed Confidence. In *PMLR*, volume 49 of *Conference on Learning Theory*, 2016.

[GKL16] A. Garivier, E. Kaufmann, and T. Lattimore. On Explore-Then-Commit Strategies. In *PMLR*, volume 29 of *Advances in Neural Information Processing Systems*, 2016.

[GM11] A. Garivier and E. Moulines. On Upper-Confidence Bound Policies For Switching Bandit Problems. In *Algorithmic Learning Theory*, pages 174–188. PMLR, 2011.

[GNUa] GNU Radio Companion Documentation and Website. wiki.gnuradio.org/index.php/GNURadioCompanion. Accessed: 2018-09-25.

[GNUb] GNU Radio Documentation and Website. www.gnuradio.org/about/. Accessed: 2018-09-25.

[H. 18] H. Joshi and R. Kumar and A. Yadav and S. J. Darak. Distributed Algorithm for Dynamic Spectrum Access in Infrastructure-Less Cognitive Radio Network. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2018.

[H$^+$16] E. Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[Hay05] S. Haykin. Cognitive Radio: Brain-Empowered Wireless Communications. *Journal on Selected Areas in Communications*, 23(2):201–220, 2005.

[HGB$^+$06] C. Hartland, S. Gelly, N. Baskiotis, O. Teytaud, and M. Sebag. Multi-Armed Bandit, Dynamic Environments and Meta-Bandits. In *NeurIPS 2006 Workshop, Online Trading Between Exploration And Exploitation*, 2006.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[Hon19] J. Honda. A Note on KL-UCB+ Policy for the Stochastic Bandit. *arXiv preprint arXiv:1903.07839*, 2019.

[Hun07] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[I$^+$17] Anaconda Inc. et al. Numba, NumPy aware dynamic Python compiler using LLVM. Online at: numba.pydata.org, 2017.

[JEMP09] W. Jouini, D. Ernst, C. Moy, and J. Palicot. Multi-Armed Bandit Based Policies for Cognitive Radio's Decision Making Issues. In *International Conference Signals, Circuits and Systems*. IEEE, 2009.

[JEMP10] W. Jouini, D. Ernst, C. Moy, and J. Palicot. Upper Confidence Bound Based Decision Making Strategies and Dynamic Spectrum Access. In *International Conference on Communications*, pages 1–5. IEEE, 2010.

[JMP12] W. Jouini, C. Moy, and J. Palicot. Decision Making for Cognitive Radio Equipment: Analysis of the First 10 Years of Exploration. *EURASIP Journal on Wireless Communications and Networking*, 2012(1), 2012.

[JOP$^+$01] E. Jones, T. E. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python. Online at: `www.scipy.org`, 2001.

[Jor10] M. Jordan. Stat 260/CS 294. `people.eecs.berkeley.edu/~jordan/courses/260-spring10/`, 2010. Chapter 8 covers the exponential family.

[K$^+$16] T. Kluyver et al. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

[KAF$^+$18] R. Kerkouche, R. Alami, R. Féraud, N. Varsier, and P. Maillé. Node-based optimization of LoRa transmissions with Multi-Armed Bandit algorithms. In *International Conference on Telecommunications*, Saint-Malo, France, 2018. J. Palicot and R. Pyndiah.

[KCG12] E. Kaufmann, O. Cappé, and A. Garivier. On Bayesian Upper Confidence Bounds for Bandit Problems. In *International Conference on Artificial Intelligence and Statistics*, pages 592–600, 2012.

[KDH$^+$19] R. Kumar, S. J. Darak, M. K. Hanawal, A. K. Sharma, and R. K. Tripathi. Distributed Algorithm for Learning to Coordinate in Infrastructure-Less Network. *IEEE Communications Letters*, 23(2):362–365, 2019.

[KDY$^+$16] R. Kumar, S. J. Darak, A. Yadav, A. K. Sharma, and R. K. Tripathi. Two-stage Decision Making Policy for Opportunistic Spectrum Access and Validation on USRP Testbed. *Wireless Networks*, pages 1–15, 2016.

[KDY$^+$17] R. Kumar, S. J. Darak, A. Yadav, A. K. Sharma, and R. K. Tripathi. Channel Selection for Secondary Users in Decentralized Network of Unknown Size. *Communications Letters*, 21(10):2186–2189, 2017.

[KHN15] J. Komiyama, J. Honda, and H. Nakagawa. Optimal Regret Analysis of Thompson Sampling in Stochastic Multi-Armed Bandit Problem with Multiple Plays. In *International Conference on Machine Learning*, volume 37, pages 1152–1161. PMLR, 2015.

[KK18] E. Kaufmann and W.M. Koolen. Mixture Martingales Revisited with Applications to Sequential Tests and Confidence Intervals. *arXiv preprint arXiv:1811.11419*, 2018.

[KKM12] E. Kaufmann, N. Korda, and R. Munos. Thompson Sampling: an Asymptotically Optimal Finite-Time Analysis. In *Algorithmic Learning Theory*, pages 199–213. PMLR, 2012.

[KL51] S. Kullback and R.A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[KM19] E. Kaufmann and A. Mehrabian. New Algorithms for Multiplayer Bandits when Arm Means Vary Among Players. *arXiv preprint arXiv:1902.01239*, 2019.

[KNJ12] D. Kalathil, N. Nayyar, and R. Jain. Decentralized Learning for Multi-Player Multi-Armed Bandits. In *Conference on Decision and Control*. IEEE, 2012.

[KS06] L. Kocsis and C. Szepesvári. Discounted UCB. In *2nd PASCAL Challenges Workshop*, 2006.

[KSGB19] B. Kveton, C. Szepesvari, M. Ghavamzadeh, and C. Boutilier. Perturbed-History Exploration in Stochastic Multi-Armed Bandits. *arXiv preprint arXiv:1902.10089*, 2019.

[KT19] B. Kim and A. Tewari. On the Optimality of Perturbations in Stochastic and Adversarial Multi-armed Bandit Problems. *arXiv preprint arXiv:1902.00610*, 2019.

[KYDH18] R. Kumar, A. Yadav, S. J. Darak, and M. K. Hanawal. Trekking Based Distributed Algorithm for Opportunistic Spectrum Access in Infrastructure-Less Network. In *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad-Hoc, and Wireless Networks (WiOpt)*, pages 1–8, 2018.

[Lat16a] T. Lattimore. Library for Multi-Armed Bandit Algorithms. Online at: `github.com/tor/libbandit`, 2016.

[Lat16b] T. Lattimore. Regret Analysis Of The Finite Horizon Gittins Index Strategy For Multi Armed Bandits. In *Conference on Learning Theory*, pages 1214–1245. PMLR, 2016.

[Lat18] T. Lattimore. Refining the confidence level for optimistic bandit strategies. *The Journal of Machine Learning Research*, 19(1):765–796, 2018.

[LCLS10] L. Li, W. Chu, J. Langford, and R. E. Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *International Conference on World Wide Web*, pages 661–670. ACM, 2010.

[LKC16] A. Luedtke, E. Kaufmann, and A. Chambaz. Asymptotically Optimal Algorithms for Multiple Play Bandits with Partial Feedback. *arXiv preprint arXiv:1606.09388*, 2016.

[LLL19] H. Li, J. Luo, and C. Liu. Selfish Bandit based Cognitive Anti-jamming Strategy for Aeronautic Swarm Network in Presence of Multiple Jammert. *IEEE Access*, 2019.

[LLS18] F. Liu, J. Lee, and N. Shroff. A Change-Detection based Framework for Piecewise-stationary Multi-Armed Bandit Problem. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 2018.

[LM18] G. Lugosi and A. Mehrabian. Multiplayer bandits without observing collision information. *arXiv preprint, arXiv:1808.08416*, 2018.

[LR85] T. L. Lai and H. Robbins. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[LRC$^+$16] J. Louëdec, L. Rossi, M. Chevalier, A. Garivier, and J. Mothe. Algorithme de bandit et obsolescence : un modèle pour la recommandation. In *18ème Conférence francophone sur l'Apprentissage Automatique, 2016 (Marseille, France)*, 2016.

[LS19] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2019. Draft of Wednesday 1st of May, 2019.

[Lue68] D. G. Luenberger. Quasi-Convex Programming. *SIAM Journal on Applied Mathematics*, 16(5):1090–1095, 1968.

[LWAL18] H. Luo, C. Wei, A. Agarwal, and J. Langford. Efficient Contextual Bandits in Non-stationary Worlds. In S. Bubeck, V. Perchet, and P. Rigollet, editors, *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 1739–1776. PMLR, 2018.

[LX10] T. L. Lai and H. Xing. Sequential change-point detection when the pre-and post-change parameters are unknown. *Sequential Analysis*, 29(2):162–175, 2010.

[LZ08] K. Liu and Q. Zhao. A Restless Bandit Formulation of Opportunistic Access: Indexablity and Index Policy. In *Annual Communications Society Conference on Sensor, Mesh and Ad-Hoc Communications and Networks Workshops*. IEEE, 2008.

[LZ10] K. Liu and Q. Zhao. Distributed Learning in Multi-Armed Bandit with Multiple Players. *Transaction on Signal Processing*, 58(11):5667–5681, 2010.

[Mai19] O.-A. Maillard. Sequential change-point detection: Laplace concentration of scan statistics and non-asymptotic delay bounds. In *Algorithmic Learning Theory*, 2019.

[MB19] C. Moy and L. Besson. Decentralized Spectrum Learning for IoT Wireless Networks Collision Mitigation. In *ISIoT workshop*, Santorin, Greece, May 2019.

[MG17] P. Ménard and A. Garivier. A Minimax and Asymptotically Optimal Algorithm for Stochastic Bandits. In *Algorithmic Learning Theory*, volume 76, pages 223–237. PMLR, 2017.

[MGMM+15] L. Melián-Gutiérrez, N. Modi, C. Moy, F. Bader, I. Pérez-Álvarez, and S. Zazo. Hybrid UCB-HMM: A Machine Learning Strategy for Cognitive Radio in HF Band. *IEEE Transactions on Cognitive Communications and Networking*, 1(3):347–358, 2015.

[MM99] J. Mitola and G. Q. Maguire. Cognitive Radio: making software radios more personal. *Personal Communications*, 6(4):13–18, 1999.

[MM11] O.-A. Maillard and R. Munos. Adaptive Bandits: Towards the best history-dependent strategy. In *International Conference on Artificial Intelligence and Statistics*, pages 570–578, 2011.

[MM17] J. Mourtada and O.-A. Maillard. Efficient Tracking of a Growing Number of Experts. In *Algorithmic Learning Theory*, volume 76 of *Proceedings of Algorithmic Learning Theory*, pages 1–23, Tokyo, Japan, 2017.

[Mod17] N. Modi. *Machine Learning and Statistical Decision Making for Green Radio*. PhD thesis, CentraleSupélec, IETR, Rennes, 2017.

[Moy14] C. Moy. Reinforcement Learning Real Experiments for Opportunistic Spectrum Access. In *WSR'14*, page 10, Karlsruhe, Germany, 2014.

[MS13] J. Mellor and J. Shapiro. Thompson Sampling in Switching Environments with Bayesian Online Change Detection. In *Artificial Intelligence and Statistics*, pages 442–450, 2013.

[MTC+16] A. Maskooki, V. Toldov, L. Clavier, V. Loscrí, and N. Mitton. Competition: Channel Exploration/Exploitation Based on a Thompson Sampling Approach in a Radio Cognitive Environment. In *International Conference on Embedded Wireless Systems and Networks (dependability competition)*, Graz, Austria, February 2016.

[NC17] O. Naparstek and K. Cohen. Deep Multi-User Reinforcement Learning for Dynamic Spectrum Access in Multichannel Wireless Networks. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–7, 2017.

[Nor98] J. R. Norris. *Markov Chains*, volume 2 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 1998.

[Oct] OctoClock Clock Distribution Module with GPSDO - Ettus Research. `www.ettus.com/product/details/OctoClock-G`. Accessed: 2018-09-25.

[PG07] F. Pérez and B. E. Granger. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[PGNN19] V. Patil, G. Ghalme, V. Nair, and Y. Narahari. Stochastic Multi-armed Bandits with Arm-specific Fairness Guarantees. *arXiv preprint arXiv:1905.11260*, 2019.

[PPS11] K. Patil, R. Prasad, and K. Skouby. A Survey of Worldwide Spectrum Occupancy Measurement Campaigns for Cognitive Radio. In *2011 International Conference on Devices and Communications (ICDeCom)*, pages 1–5. IEEE, 2011.

[Raj17] V. Raj. A Julia Package for providing Multi Armed Bandit Experiments. Online at: `github.com/v-i-s-h/MAB.jl`, 2017.

[RK17] V. Raj and S. Kalyani. Taming Non-Stationary Bandits: a Bayesian Approach. arXiv preprint arXiv:1707.09727, 2017.

[RKS17] U. Raza, P. Kulkarni, and M. Sooriyabandara. Low power wide area networks (lpwan): An overview. *Communications Surveys Tutorials*, 19(2):855–873, 2017.

[RMZ14] C. Robert, C. Moy, and H. Zhang. Opportunistic Spectrum Access Learning Proof of Concept. In *SDR-WinnComm'14*, page 8, Schaumburg, United States, 2014.

[Rob52] H. Robbins. Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[Rob75] L. G. Roberts. ALOHA packet system with and without slots and capture. *SIGCOMM Computer Communication Review*, 5(2):28–42, 1975.

[RSS16] J. Rosenski, O. Shamir, and L. Szlak. Multi-Player Bandits – A Musical Chairs Approach. In *International Conference on Machine Learning*, pages 155–163. PMLR, 2016.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An introduction*. MIT press, 2018.

[SKHD18] S. Sawant, R. Kumar, M. K. Hanawal, and S. J. Darak. Learning to Coordinate in a Decentralized Cognitive Radio Network in Presence of Jammers. *arXiv preprint arXiv:1803.06810*, 2018.

[SL17] Y. Seldin and G. Lugosi. An Improved Parametrization and Analysis of the EXP3++ Algorithm for Stochastic and Adversarial Bandits. In *Conference on Learning Theory*, volume 65, pages 1–17. PMLR, 2017.

[SLC$^+$19] J. Seznec, A. Locatelli, A. Carpentier, A. Lazaric, and M. Valko. Rotting bandits are no harder than stochastic ones. *International Conference on Artificial Intelligence and Statistics*, 2019.

[Sli19] A. Slivkins. Introduction to Multi-Armed Bandits. *arXiv preprint arXiv:1904.07272*, 2019.

[SV95] D. Siegmund and E.S. Venkatraman. Using the Generalized Likelihood Ratio Statistic for Sequential Detection of a Change Point. *The Annals of Statistics*, pages 255–271, 1995.

[TCLM16] V. Toldov, L. Clavier, V. Loscrí, and N. Mitton. A Thompson Sampling Approach to Channel Exploration Exploitation Problem in Multihop Cognitive Radio Networks. In *PIMRC*, pages 1–6, Valencia, Spain, September 2016.

[TdSCC13] F. S. Truzzi, V. F. da Silva, A. H. Reali Costa, and F. Gagliardi Cozman. AdBandit: A New Algorithm For Multi-Armed Bandits. *ENIAC*, 2013(1), 2013.

[Tho33] W. R. Thompson. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25, 1933.

[TL12] C. Tekin and M. Liu. Online Learning in Decentralized Multi-User Spectrum Access with Synchronized Explorations. In *Military Communications Conference*. IEEE, 2012.

[Tol14a] J. Toledano. Executive summary: Dynamic spectrum management for innovation and growth. June 2014.

[Tol14b] J. Toledano. Une gestion dynamique du spectre pour l'innovation et la croissance. June 2014.

[TPHD19] H. Tibrewal, S. Patchala, M. K. Hanawal, and S. J. Darak. Distributed Learning and Optimal Assignment in Multiplayer Heterogeneous Networks. *arXiv preprint arXiv:1901.03868*, 2019.

[TRY17] K. Tomer, L. Roi, and M. Yishay. Bandits with Movement Costs and Adaptive Pricing. In *Conference on Learning Theory*, volume 65, pages 1242–1268. PMLR, 2017.

[TZZ19] C. Tao, Q. Zhang, and Y. Zhou. Collaborative Learning with Limited Interaction: Tight Bounds for Distributed Exploration in Multi-Armed Bandits. *arXiv preprint arXiv:1904.03293*, 2019.

[Val16] M. Valko. *Bandits on Graphs and Structures*. Habilitation thesis to supervise research, École normale supérieure de Cachan, 2016.

[Var17] G. Varoquaux. Joblib: running Python functions as pipeline jobs. Online at: `joblib.readthedocs.io`, March 2017.

[vdWCV11] S. van der Walt, C. S. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2):22–30, March 2011.

[VMB$^+$10] V. Valenta, R. Maršálek, G. Baudoin, M. Villegas, M. Suarez, and F. Robert. Survey on spectrum utilization in Europe: Measurements, analyses and observations. In *5th EAI Conference on Cognitive Radio Oriented Wireless Network and Communication*, pages 1–5. IEEE, 2010.

[W$^+$17] M. Waskom et al. Seaborn: statistical data visualization. Online at: `seaborn.pydata.org`, September 2017.

[Wal45] A. Wald. Some Generalizations of the Theory of Cumulative Sums of Random Variables. *The Annals of Mathematical Statistics*, 16(3):287–293, 1945.

[WBMB$^+$19] F. Wilhelmi, S. Barrachina-Muñoz, B. Bellalta, C. Cano, A. Jonsson, and G. Neu. Potential and pitfalls of multi-armed bandits for decentralized spatial reuse in wlans. *Journal of Network and Computer Applications*, 127:26–42, 2019.

[WCN$^+$19] F. Wilhelmi, C. Cano, G. Neu, B. Bellalta, A. Jonsson, and S. Barrachina-Muñoz. Collaborative spatial reuse in wireless networks via selfish multi-armed bandits. *Ad Hoc Networks*, 2019.

[WHCW19] Y. Wang, J. Hu, X. Chen, and L. Wang. Distributed Bandit Learning: How Much Communication is Needed to Achieve (Near) Optimal Regret. *arXiv preprint arXiv:1904.06309*, 2019.

[Whi88] P. Whittle. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25(A):287–298, 1988.

[Wil38] S. S. Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, 9(1):60–62, 1938.

[WS18a] L. Wei and V. Srivastava. On Distributed Multi-player Multi-Armed Bandit Problems in Abruptly-Changing Environment. In *Conference on Decision and Control*, pages 5783–5788. IEEE, 2018.

[WS18b] L. Wei and V. Srivatsva. On Abruptly-Changing And Slowly-Varying Multi-Armed Bandit Problems. In *American Control Conference*, pages 6291–6296. IEEE, 2018.

[Yaa77] M. E. Yaari. A Note on Separability and Quasiconcavity. *Econometrica*, 45(5):1183–1186, 1977.

[YFE12] X. Yang, A. Fapojuwo, and E. Egbogah. Performance Analysis and Parameter Optimization of Random Access Backoff Algorithm in LTE. In *Vehicular Technology Conference*, pages 1–5. IEEE, September 2012.

[YM09] J. Y. Yu and S. Mannor. Piecewise-Stationary Bandit Problems with Side Observations. In *International Conference on Machine Learning*, pages 1177–1184. ACM, 2009.

[ZBLN19] S. M. Zafaruddin, I. Bistritz, A. Leshem, and D. Niyato. Distributed Learning for Channel Allocation Over a Shared Spectrum. *arXiv preprint arXiv:1902.06353*, 2019.

[ZS07] Q. Zhao and B. M. Sadler. A Survey of Dynamic Spectrum Access. *Signal Processing magazine*, 24(3):79–89, 2007.

[ZS18] J. Zimmert and Y. Seldin. An Optimal Algorithm for Stochastic and Adversarial Bandits. *arXiv preprint arXiv:1807.07623*, 2018.

Titre : **Algorithmes de Bandits Multi-Joueurs pour les Réseaux de l'Internet des Objets**

Mots clés : Internet des Objets (IoT), Radio Intelligente, Théorie de l'apprentissage, Apprentissage séquentiel, Apprentissage par renforcement, Bandits multi-bras (MAB), Apprentissage décentralisé, Bandits multi-bras multi-joueurs, Détection des points de changement, Bandits multi-bras non stationnaires

Résumé : Dans cette thèse de doctorat, nous étudions les réseaux sans fil et les appareils reconfigurables qui peuvent accéder à des réseaux de radio intelligente, dans des bandes non licenciées et sans supervision centrale. Nous considérons des réseaux de l'Internet des Objets (IoT), avec l'objectif d'augmenter la durée de vie de la batterie des appareils, en les équipant d'algorithmes d'apprentissage machine peu coûteux mais efficaces, qui leur permettent d'améliorer automatiquement l'efficacité de leurs communications sans fil. Nous proposons différents modèles de réseaux IoT, et nous montrons empiriquement, par des simulations numériques et une validation expérimentale réaliste, le gain que peuvent apporter nos méthodes, qui utilisent l'apprentissage par renforcement. Les différents problèmes d'accès au réseau sont modélisés avec des Bandits Multi-Bras (MAB), mais leur analyse est difficile à réaliser, car il est délicat de prouver la convergence de nombreux appareils jouant à un jeu collaboratif sans communication ni aucune coordination, lorsque les appareils suivent tous un modèle d'activation aléatoire. Le reste de ce manuscrit étudie donc deux modèles restreints, d'abord des bandits multi-joueurs dans des problèmes stationnaires, puis des bandits mono-joueur non stationnaires. Nous détaillons également une autre contribution, la bibliothèque Python open-source SMPyBandits pour des simulations numériques de problèmes MAB, qui couvre les modèles étudiés et d'autres.

Title : **Multi-Players Bandit Algorithms for Internet of Things Networks**

Keywords: Internet of Things (IoT), Cognitive Radio, Learning Theory, Sequential Learning, Reinforcement Learning, Multi-Armed Bandits (MAB), Decentralized Learning, Multi-Player Multi-Armed Bandits, Change Point Detection, Non-Stationary Multi-Armed Bandits

Abstract: In this PhD thesis, we study wireless networks and reconfigurable end-devices that can access large-scale Cognitive Radio networks, in unlicensed bands and without central control. We focus on Internet of Things networks (IoT), with the objective of extending the devices' battery life, by equipping them with low-cost but efficient machine learning algorithms, to let them automatically improve the efficiency of their wireless communications. We propose different models of IoT networks, and we show empirically on both numerical simulations and real-world validation the possible gain of our methods, that use Reinforcement Learning. The different network access problems are modeled as Multi-Armed Bandits (MAB), but we found that analyzing the realistic models was intractable, because proving the convergence of many end-devices playing a collaborative game without communication nor coordination is hard, when end-devices all follow random active pattern. The rest of this manuscript thus studies two restricted models, first multi-players bandits in stationary problems, then non-stationary single-player bandits. We also detail another contribution, SMPyBandits, an open-source Python library for numerical MAB simulations, covering all the studied models and more.