

# Optimizing Ranking Systems Online as Bandits

**Chang Li**



# **Optimizing Ranking Systems Online as Bandits**

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor  
aan de Universiteit van Amsterdam  
op gezag van de Rector Magnificus  
prof. dr. ir. K.I.J. Maex

ten overstaan van een door het College voor Promoties ingestelde  
commissie, in het openbaar te verdedigen in de Agnietenkapel  
op donderdag 4 maart 2021, te 13.00 uur

door

Chang Li

geboren te Hebei

**Promotiecommissie**

Promotor: prof. dr. M. de Rijke Universiteit van Amsterdam

Copromotor: dr. M. Zoghi Google

Overige leden: prof. dr. T. Gevers Universiteit van Amsterdam  
prof. dr. ir. D. Hiemstra Radboud Universiteit  
dr. H.C. van Hoof Universiteit van Amsterdam  
prof. dr. E. Kanoulas Universiteit van Amsterdam  
dr. M. Lalmas Spotify

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The research was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 612.001.551.

Copyright © 2021 Chang Li, Amsterdam, The Netherlands  
Cover by Feng Li  
Printed by Off Page, Amsterdam

ISBN: 978-94-93197-46-6

## Acknowledgements

With the aim to experience a different type of life and continue working on machine learning, I submitted my application for a PhD position at the University of Amsterdam. After four years and 9 months, I notice that it has been one of the best decisions I have made ever, not only because I received the offer from my supervisor Maarten de Rijke, but also because doing a Ph.D. is way more interesting, meaningful and, of course, challenging than what I expected.

I used to think that doing a Ph.D. is nothing but producing some papers :P. After I joined the group, I realized that there are more things in Ph.D. life than papers. Thank you to all ILPSers for teaching me this. It has been a really interesting journey with all of you around. Special thanks to Maurits Bleeker for helping me translate the summary into a *samenvatting*.

Maarten has been always trying to tell me that a Ph.D. thesis is more meaningful than a stack of papers, in several ways. Unfortunately, I did not fully get his points and understand how important it is until the beginning of my third year. One day, I was asked by one of my co-authors, Branislav Kventon, what will your thesis topic be? All of a sudden, Maarten's words appeared, and I noticed that I should be thinking in a more structured way. Thank you Maarten, for the guidance and help during my study!

A frustrating part of my Ph.D. has been that my papers got rejected eleven times in a row. Thank you to all my co-authors for helping me get through this. However, in my mind, a bigger challenge beyond this is how to make my research meaningful. Fortunately, I had Maarten as my supervisor, who supported me and helped to connect me to other researchers in the community. This is how I got to know Masrour Zoghi (who became my co-promoter) and Brano, two awesome collaborators. Thank you both for helping me with my research. Last but not least, I want to thank Ilya Markov for helping me think scientifically in the early stages of my research.

I was fortunate enough to enjoy two internships. Thank you, Haoyun and Xinran at Bloomberg. That summer in New York was fantastic! Thank you, Hua at Apple. The internship in Cupertino was also amazingly special. I also want to thank Jeremy and Derek from the HR team at Apple for helping me relocate to the US during the COVID-19 pandemic.

Feng, I owe you a big thank you. Because of you, I chose Europe. Without you, it would have been hard for me to reach this point. I really feel lucky and full of joy to have you as my girlfriend. And, of course, thank you for creating the cover of my thesis.

最后，感谢来自父母的支持。读博这几年跟你们在一起的时间也就一个多月，感谢你们的理解，让我可以安心的在国外。

Although a bit sad, it is the time to conclude my student career. Thank you to all my classmates and teachers.

Chang Li  
Amsterdam  
December 2020



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Research Outline and Questions . . . . .	2
1.2	Main Contributions . . . . .	4
1.3	Thesis Overview . . . . .	5
1.4	Origins . . . . .	6
<b>2</b>	<b>Effective Large-Scale Online Ranker Evaluation</b>	<b>9</b>
2.1	Introduction . . . . .	9
2.2	Problem Setting . . . . .	11
2.3	Related Work . . . . .	12
2.4	Algorithm . . . . .	14
2.4.1	MergeDTS . . . . .	14
2.4.2	Theoretical guarantees . . . . .	18
2.4.3	Discussion . . . . .	20
2.5	Experimental Setup . . . . .	20
2.5.1	Research questions . . . . .	20
2.5.2	Datasets . . . . .	21
2.5.3	Evaluation methodology . . . . .	22
2.5.4	Click simulation . . . . .	23
2.5.5	Baselines . . . . .	23
2.5.6	Parameters . . . . .	24
2.5.7	Metrics . . . . .	25
2.6	Experimental Results . . . . .	26
2.6.1	Large-scale experiments . . . . .	27
2.6.2	Computational scalability . . . . .	27
2.6.3	Impact of noise . . . . .	28
2.6.4	Cycle experiment . . . . .	29
2.6.5	Beyond the Condorcet assumption . . . . .	31
2.6.6	Parameter sensitivity . . . . .	31
2.7	Conclusion . . . . .	32
<b>3</b>	<b>Safe Online Learning to Re-Rank</b>	<b>35</b>
3.1	Introduction . . . . .	35
3.2	Background . . . . .	36
3.2.1	Click models . . . . .	36
3.2.2	Stochastic click bandit . . . . .	37
3.3	Online Learning to Re-Rank . . . . .	38
3.3.1	Algorithm . . . . .	38
3.4	Theoretical Analysis . . . . .	40
3.4.1	Regret bound . . . . .	40
3.4.2	Safety . . . . .	41
3.4.3	Discussion . . . . .	41
3.4.4	Proof of Theorem 3.1 . . . . .	42
3.5	Experimental Results . . . . .	43

## CONTENTS

---

3.5.1	Experimental setup . . . . .	44
3.5.2	Results with regret . . . . .	46
3.5.3	Safety results . . . . .	46
3.5.4	Sanity check on regret bound . . . . .	47
3.5.5	Results with NDCG . . . . .	48
3.6	Related Work . . . . .	49
3.7	Lemmas . . . . .	50
3.8	Conclusions . . . . .	55
<b>4</b>	<b>Cascade Non-Stationary Bandits</b>	<b>57</b>
4.1	Introduction . . . . .	57
4.2	Background . . . . .	58
4.2.1	Cascade model . . . . .	59
4.2.2	Cascading bandits . . . . .	59
4.3	Cascading Non-Stationary Bandits . . . . .	60
4.3.1	Problem setup . . . . .	60
4.3.2	Algorithms . . . . .	61
4.4	Analysis . . . . .	63
4.4.1	Regret upper bound . . . . .	63
4.4.2	Regret lower bound . . . . .	64
4.4.3	Discussion . . . . .	66
4.5	Experimental Analysis . . . . .	66
4.6	Related Work . . . . .	68
4.7	Proofs . . . . .	68
4.7.1	Proof of Theorem 4.1 . . . . .	69
4.7.2	Proof of Theorem 4.2 . . . . .	72
4.8	Additional Experiments . . . . .	75
4.9	Conclusion . . . . .	75
<b>5</b>	<b>Online Learning to Rank for Relevance and Diversity</b>	<b>77</b>
5.1	Introduction . . . . .	77
5.2	Background . . . . .	79
5.2.1	Cascade model . . . . .	79
5.2.2	Cascading bandits . . . . .	80
5.2.3	Submodular coverage model . . . . .	81
5.3	Algorithm . . . . .	82
5.3.1	Problem formulation . . . . .	82
5.3.2	Competing with a greedy benchmark . . . . .	83
5.3.3	<i>CascadeHybrid</i> . . . . .	83
5.3.4	Computational complexity . . . . .	85
5.4	Experiments . . . . .	85
5.4.1	Experimental setup . . . . .	85
5.4.2	Experimental results . . . . .	88
5.5	Analysis . . . . .	90
5.5.1	Performance guarantee . . . . .	90
5.5.2	Proof of Theorem 5.1 . . . . .	91

5.6 Related Work . . . . .	94
5.7 Conclusion . . . . .	95
<b>6 Conclusions</b>	<b>97</b>
6.1 Results . . . . .	97
6.2 Future Work . . . . .	98
<b>Bibliography</b>	<b>101</b>
<b>Summary</b>	<b>109</b>
<b>Samenvatting</b>	<b>111</b>



# 1

## Introduction

Ranking is widely utilized in daily interactive systems, where a service provider responds to a user’s request by using a sorted list of items. A typical scenario is web search: a user issues a query to a search engine, and then the search engine returns a list of items. In this thesis, we focus on optimizing these interactive ranking systems. We simplify the ranking system as a ranking function, also called ranker, which, given a context, (e.g., query), scores candidate items, and then sorts them in decreasing order according to the scores. The optimization objective is to find or learn an optimal ranker that displays the most relevant candidate items at the top. The problem is challenging and one of the fundamental challenges is the notion of *relevance* [103].

About a decade ago, this problem was widely studied as an offline supervised learning problem [84]. Service providers, such as Microsoft and Yahoo, would generate big collections of queries and documents, and ask human workers to judge the relevance of each query-document pair [19, 96, 97]. The collected datasets would then be used to optimize rankers, and the optimal ranker was the one that had the highest offline ranking metric, e.g., NDCG [52]. This approach is still largely the industrial standard [86]. However, one disadvantage is that annotated relevance is not consistent with user preferences [40, 54, 121]. Thus, even an optimized ranker can be suboptimal in production [131].

As an alternative, optimizing rankers with interactive feedback has drawn great attention in the past decade [44, 126]. In these approaches, the interactive user feedback (e.g., clicks) is used as a label, and the optimal ranker is the one that attracts the highest number of clicks. Compared to human annotated feedback, there are three advantages in using interactive feedback [41, 54]: (1) Interactive feedback is abundant in interactive systems and easily obtained. (2) Interactive feedback is better aligned with the user’s information needs. (3) Interactive feedback is more timely and can capture shifts in user preferences. However, interactive feedback is noisy and highly affected by the presentation order in the system, i.e., position bias. To use this type of relatively low-quality feedback for ranker optimization, new learning algorithms need to be designed.

In this thesis, we focus on learning from this feedback. Even though the quality of interactive feedback is low, its advantages have the potential to assist in designing new and powerful ranking systems. For instance, non-stationarity is known to exist in user preference [49]. In traditional supervised learning, the annotated data may not be

enough to capture shifts in user preferences. But, with interactive feedback, which is more timely than human annotated data, we can design a ranker that changes its ranking policy according to the change in user preferences, capturing the non-stationarity. In this thesis, we study the problems of online ranker optimization, and translate our ideas into new algorithms that work for novel Online Learning to Rank (OLTR) problems. More specifically, we formulate the problem of learning from noisy and biased feedback in ranking systems as different bandit problems [71, 111], and propose new bandit algorithms to solve the proposed problems. Thus, the title of this thesis is *Optimizing Ranking Systems Online as Bandits*.

### 1.1 Research Outline and Questions

---

Through out the thesis, we want to answer the following question: how should we use implicit feedback to optimize ranking systems online? As implicit feedback is typically noisy, we face the challenge of *exploration versus exploitation*. Bandit algorithms are suitable for addressing this problem. In this thesis, we consider four different research directions in ranking system optimization and formulate each of them as a research question. The first one is about ranker evaluation and the other three are related to OLTR.

#### **RQ1** How to conduct effective large-scale online ranker evaluation?

Online ranker evaluation is one of the key challenges in information retrieval. While the user preferences for ranker over another can be inferred by using interleaving methods [130], the problem of how to effectively choose the ranker pair that generates the interleaved list without degrading the user experience too much is still challenging. On the one hand, if two rankers have not been compared enough, the inferred preference can be noisy and inaccurate. On the other hand, if two rankers are compared too many times, the interleaving process inevitably hurts user experience. This dilemma is known as the *exploration versus exploitation* dilemma. It is captured by the  $K$ -armed dueling bandit problem [126], which is a variant of the  $K$ -armed bandit problem, where feedback comes in the form of pairwise preferences. Today's deployed search systems evaluate large numbers of rankers concurrently, and scaling effectively in the presence of numerous rankers is a critical aspect of nline ranker evaluation.

We provide an answer to **RQ1** in Chapter 2 by introducing the MergeDTS algorithm. MergeDTS uses a divide-and-conquer strategy to localize the comparisons carried out by the algorithm to small batches of rankers, and then employs Thompson Sampling (TS) to reduce the comparisons between suboptimal rankers inside these small batches. We conduct extensive experiments on three large-scale datasets to assess the performance of MergeDTS. The experimental results demonstrate that MergeDTS outperforms the state-of-the-art baselines.

#### **RQ2** How to achieve safe online learning to re-rank?

Learning to rank has been studied in online and offline settings [35, 84]. In the offline setting, rankers are typically learned from relevance labels created by judges. This approach has become standard in industrial applications of ranking, such as web search

and recommender systems [83]. However, this approach lacks exploration and thus is limited by the information contained in the offline training data. In the online setting, an algorithm can experiment with ranked lists and learn from feedback on them in a sequential fashion. Bandit algorithms are well-suited for this setting but they tend to learn user preferences from scratch (i.e., the “cold-start” problem), which results in a high initial cost of exploration. This poses an additional challenge of *safe* exploration in ranked lists.

**RQ2** is set up to address the safe exploration problem in OLTR. We address this question by introducing the BubbleRank algorithm in Chapter 3. BubbleRank can be viewed as a combination of offline and online Learning to Rank (LTR) algorithms. It uses the offline trained ranker, e.g., the production ranker, to obtain the initial ranked list, and then conducts safe online pairwise exploration to improve this list. The safety comes from the fact that BubbleRank explores the ranked lists by randomly exchanging items with their neighbors. Thus, during exploration, an item never moves too far from its original position. We analyze the performance as well as the safety of BubbleRank theoretically, and then conduct experiments on a large-scale click dataset to evaluate BubbleRank empirically.

**RQ3** How to conduct online learning to rank when users change their preferences constantly?

Non-stationarity appears in many online applications such as web search and recommender systems [49]. User preferences are effected by different factors and may change constantly. An example is that user preferences are effected by abrupt incidents. Let us take the query “corona” as an example. Before January of 2020, the query was mainly related to a beer brand. But after the onset of the COVID-19 pandemic, the meaning of this word has shifted to a deadly virus. If the learning algorithm is not able to capture this change in user preferences, it would continue responding with beer-related items, which would hurt the user experience.

We provide one solution to **RQ3** in Chapter 4. Particularly, we consider *abruptly changing environments* where user preferences remain constant in certain time periods, named *epochs*, but change occurs abruptly at unknown moments called *breakpoints*. We introduce cascading non-stationary bandits, an online variant of the Cascade Model (CM) [27] with the goal of identifying the  $K$  most attractive items in a non-stationary environment, and propose two algorithms: CascadeDUCB and CascadeSWUCB to solve this bandit problem. The performance of the proposed algorithms are analyzed, and two gap-dependent upper bounds on their  $n$ -step regret are derived, respectively. We also establish a lower bound on the regret of the non-stationary OLTR problem, and show that both algorithms match the lower bound up to a logarithmic factor. At the end of the chapter, we evaluate the performance of the proposed algorithms on a real-world web search click dataset.

**RQ4** How to learn a ranker online considering both relevance and diversity?

Relevance ranking and result diversification are two important aspects in modern recommender systems. Relevance ranking aims at putting relevant items at the top of result lists, while result diversification focuses on generating ranked lists covering a broad range of topics. The former is achieved by displaying a list whose items are sorted

in decreasing order of relevance. However, this sorting process often makes the result list less diverse since items with the highest relevance generally come from the same topic. On the other hand, submodular functions are used for result diversification [38, 120], but this alone is not enough to guarantee items at the top of list to be relevant.

In Chapter 5, we provide a solution to **RQ4** by formulating the OLTR for relevance and diversity as the Cascade Hybrid Bandits (CHB). The CascadeHybrid algorithm is then proposed to solve CHB. We first provide a gap free bound on the  $n$ -step regret of CascadeHybrid, and then evaluate it on two real-world recommendation tasks: movie recommendation and music recommendation.

## 1.2 Main Contributions

---

We list the main contributions of the thesis in this section. The contributions of this thesis come in three groups: algorithmic contributions, theoretical contributions and empirical contributions.

**Algorithmic contributions** We proposed five novel bandit algorithms to solve four online optimization problems in ranking systems:

- A TS-based dueling bandit algorithm, MergeDTS, which solves the large-scale online ranker evaluation problem, Chapter 2.
- A safe online learning to re-rank algorithm, BubbleRank, which is inspired by bubble sort, Chapter 3. To the best of our knowledge, BubbleRank is the first safe online learning to re-rank algorithm.
- The CascadeDUCB and CascadeSWUCB algorithms for solving the non-stationary OLTR problem, Chapter 4. To the best of our knowledge, the proposed CascadeDUCB and CascadeSWUCB are the first bandit algorithms that solve the non-stationary OLTR problem.
- A hybrid algorithm, CascadeHybrid, for solving the OLTR problem, where both relevance ranking and result diversification are critical to users, Chapter 5. To the best of our knowledge, CascadeHybrid is the first bandit algorithm that considers both relevance and diversity in the ranking problem.

**Theoretical contributions** We theoretically analyze the performance of all proposed algorithms. Our theoretical analyses do not aim to show superior performance of the algorithms, but to provide worst-case guarantees on their performance when deployed online. We provide:

- A theoretical guarantee on the performance of MergeDTS, Chapter 2.
- A theoretical analysis of BubbleRank and the safety of BubbleRank, Chapter 3.
- The first theoretical analysis for the non-stationary OLTR problem, and an analysis of the proposed CascadeDUCB and CascadeSWUCB algorithms, Chapter 4.

- A theoretical analysis of CascadeHybrid for solving the OLTR for both relevance and diversity, Chapter 5.

**Evaluation contributions** To complement our theoretical findings, we conduct extensive experiments to evaluate the proposed algorithms for the considered tasks:

- An empirical evaluation of MergeDTS, and the comparison against the state of the art in large-scale online ranker evaluation, Chapter 2.
- An empirical comparison of BubbleRank against baselines and a sanity check on the proven theoretical results, Chapter 3.
- An empirical evaluation of CascadeDUCB and CascadeSWUCB for the non-stationary OLTR task, Chapter 4.
- An empirical comparison of CascadeHybrid against baselines in OLTR for relevance and diversity, Chapter 5.

### 1.3 Thesis Overview

---

In this section, we provide a brief overview of the remaining chapters. In Chapters 2 to 5, we study four online ranking optimization tasks, formulate each of them as a bandit problem, and provide corresponding algorithms for solving each problem. The proposed algorithms are theoretically analyzed and empirically compared against the state of the art.

More precisely, we organize them in the following order:

- Chapter 2 treats large-scale online ranker evaluation as a dueling bandits problem, and proposes the MergeDTS algorithm to solve it with theoretical guarantees on the performance. Experimental evaluation reveals that MergeDTS outperforms the state of the art baselines, e.g., MergeRUCB [130], DTS [115] and Self-Sparring [109].
- Chapter 3 studies the safe OLTR problem, and presents the BubbleRank algorithm. We analyze the performance of BubbleRank, and theoretically show that it is a safe algorithm. Then, we empirically compare BubbleRank with CascadeKL-UCB [65], BatchRank [132] and TopRank [72], and show that the performance of BubbleRank is comparable to those algorithms while only BubbleRank satisfies the theoretical safety constraint.
- Chapter 4 studies the non-stationary OLTR problem, where the user may change their preference over time. We formulate this problem as cascade non-stationary bandits and propose CascadeDUCB and CascadeSWUCB to solve the problem. Both algorithms have theoretical guarantees on their performance, and the experimental evaluation validates the theoretical findings.

- Chapter 5 presents the CascadeHybrid algorithm, which is designed to solve the OLTR problem for relevance and diversity. The proposed CascadeHybrid algorithm is theoretically sound and empirically outperforms CascadeLinUCB [133] and CascadeLSB [38].

In Chapter 6, we summarize all main findings of this thesis and provide several directions for future work.

We have tried to keep the research chapters as close as possible to their published version (see below for the publication details). This implies that some research chapters have some overlap, especially concerning the related work and background material that they present. But we believe that this disadvantage is outweighed by the advantage that the research chapters are self-contained and by the fact that sticking close to the published versions of the research chapters prevents us from creating alternative versions of published work.

Chapter 2 discusses online ranker evaluation. Chapters 3 to 5 consider the online learning to rank problem. It is recommended for the chapters to be read in order: Chapter 2, 3, 4 and 5.

## 1.4 Origins

---

The material in this thesis comes from the following publications:

- Chapter 2 is based on C. Li, I. Markov, M. de Rijke, and M. Zoghi. MergeDTS: A method for effective large-scale online ranker evaluation. *ACM Transactions on Information Systems*, 38(4):Article 40, August 2020 [79].

CL designed the algorithm, worked on the theory, and conducted the experiments. MZ helped with algorithm design and theory. All authors contributed to the writing.

- Chapter 3 is based on C. Li, B. Kveton, T. Lattimore, I. Markov, M. de Rijke, C. Szepesvári, and M. Zoghi. BubbleRank: Safe online learning to re-rank via implicit click feedback. In *UAI*, July 2019 [77].

CL designed the algorithm, and conducted the experiments. BK worked on the theory. BK and MZ helped with the algorithm design. CL, TL, CS and MZ helped with the theory. All authors contributed to the writing.

- Chapter 4 is based on C. Li and M. de Rijke. Cascading non-stationary bandits: Online learning to rank in the non-stationary cascade model. In *IJCAI*, pages 2859–2865, August 2019 [74].

CL designed the algorithm, worked on the theory, and conducted the experiments. CL and MdR contributed to the writing.

- Chapter 5 is based on C. Li, H. Feng, and M. de Rijke. Cascading hybrid bandits: Online learning to rank for relevance and diversity. In *RecSys*, pages 33–42. ACM, September 2020 [78].

CL designed the algorithm, worked on the theory and conducted the experiments. CL, HF and MdR contributed to the writing.

There are other publications that indirectly contributed to this thesis:

- C. Li and H. Ouyang. Federated unbiased learning to rank. In *SIGIR*. ACM, July 2021. Submitted [75].
- C. Li and M. de Rijke. Incremental sparse bayesian ordinal regression. *Neural Networks*, 106:294–302, October 2018 [73].
- B. Jiang, C. Li, M. de Rijke, X. Yao, and H. Chen. Probabilistic feature selection and classification vector machine. *ACM Transactions on Knowledge Discovery from Data*, 13(2):Article 21, April 2019 [53].
- C. Li, A. Grotov, I. Markov, and M. de Rijke. Online learning to rank with list-level feedback for image filtering. *arXiv preprint arXiv:1812.04910*, December 2018 [76].



# 2

## Effective Large-Scale Online Ranker Evaluation

This chapter is set up to address **RQ1**:

**RQ1** How to conduct effective large-scale online ranker evaluation?

Particularly, we focus on solving the large-scale online ranker evaluation problem under the so-called Condorcet assumption, where there exists an optimal ranker that is preferred to all other rankers. We propose the MergeDTS algorithm, which first utilizes a divide-and-conquer strategy that localizes the comparisons carried out by the algorithm to small batches of rankers, and then employs Thompson Sampling (TS) to reduce the comparisons between suboptimal rankers inside these small batches. The effectiveness (regret) and efficiency (time complexity) of MergeDTS are extensively evaluated using examples from the domain of online evaluation for web search. Our main finding is that for large-scale Condorcet ranker evaluation problems, MergeDTS outperforms the state-of-the-art dueling bandit algorithms.

### 2.1 Introduction

---

Online ranker evaluation concerns the task of determining the ranker with the best performance out of a finite set of rankers. It is an important challenge for information retrieval systems [48, 88]. In the absence of an oracle judge who can tell the preferences between all rankers, the best ranker is usually inferred from user feedback on the result lists produced by the rankers [45]. Since user feedback is known to be noisy [32, 55, 56, 90], how to infer ranker quality and when to stop evaluating a ranker are two important challenges in online ranker evaluation.

The former challenge, i.e., how to infer the quality of a ranker, is normally addressed by *interleaving* methods [21, 24, 41, 43]. Specifically, an interleaving method interleaves the result lists generated by two rankers for a given query and presents the interleaved list to the user. Then it infers the preferred ranker based on the user’s click feedback. As click feedback is noisy, the interleaved comparison of two rankers has to be repeated many times so as to arrive at a reliable outcome of the comparison.

---

This chapter was published as [79].

## 2. Effective Large-Scale Online Ranker Evaluation

---

Although interleaving methods address the first challenge of online ranker evaluation (how to infer the quality of a ranker), they give rise to another challenge, i.e., which rankers to compare and when to stop the comparisons. Without enough comparisons, we may mistakenly infer the wrong ranker preferences. But with too many comparisons we may degrade the user experience since we continue showing results from sub-optimal rankers. Based on previous work [14, 128, 130], the challenge of choosing and comparing rankers can be formalized as a  $K$ -armed dueling bandit problem [123], which is an important variant of the Multi-Armed Bandits (MAB) problem, where feedback is given in the form of pairwise preferences. In the  $K$ -armed dueling bandit problem, a ranker is defined as an arm and the best ranker is the arm that has the highest expectation to win the interleaving game against other candidates.

A number of dueling bandit algorithms have been proposed; cf. [16, 126] for an overview. However, the study of these algorithms has mostly been limited to small-scale dueling bandit problems, with the state-of-the-art being Double Thompson Sampling (DTS) [115]. By “small-scale” we mean that the number of arms being compared is small. But, in real-world online ranker evaluation problems, experiments involving hundreds or even thousands of rankers are commonplace [60]. Despite this fact, to the best of our knowledge, the only work that address this particular scalability issue is Merge Relative Upper Confidence Bound (MergeRUCB) [130]. As we demonstrate in this chapter, the performance of MergeRUCB can be improved upon substantially.

In this chapter, we propose and evaluate a novel algorithm, named Merge Double Thompson Sampling (MergeDTS). The main idea of MergeDTS is to combine the benefits of MergeRUCB, which is the state-of-the-art algorithm for large-scale dueling bandit problems, and the benefits of DTS, which is the state-of-the-art algorithm for small-scale problems, and attain improvements in terms of effectiveness (as measured in terms of regret) and efficiency (as measured in terms of time complexity). More specifically, what we borrow from MergeRUCB is the divide and conquer idea used to group rankers into small batches to avoid global comparisons. On the other hand, from DTS we import the idea of using Thompson Sampling (TS) [111], rather than using uniform randomness as in MergeRUCB, to choose the arms to be played.

We analyze the performance of MergeDTS, and demonstrate that the soundness of MergeDTS can be guaranteed if the time step  $T$  is known and the exploration parameter  $\alpha > 0.5$  (Theorem 2.1). Finally, we conduct extensive experiments to evaluate the performance of MergeDTS in the scenario of online ranker evaluation on three widely used real-world datasets: Microsoft, Yahoo! Learning to Rank, and ISTELLA [19, 85, 96]. We show that with tuned parameters MergeDTS outperforms MergeRUCB and DTS in large-scale online ranker evaluation under the Condorcet assumption, i.e., where there is a ranker preferred to all other rankers.<sup>1</sup> Moreover, we demonstrate the potential of using MergeDTS beyond the Condorcet assumption, i.e.,

---

<sup>1</sup>Our theoretical analysis is rather conservative and the regret bound only holds for the parameter values within a certain range. This is because our bound is proven using Chernoff-Hoeffding bound [39] together with the union bound [17], both of which, in our case, introduce gaps between theory and practice. In our experiments, we show that the parameter values outside of the theoretical regime can boost up the performance of MergeDTS as well as that of the baselines. Thus, our experimental results of MergeDTS are not restricted to the parameter values within the theoretical regime.

where there might be multiple best rankers.

In summary, the main contributions of this chapter are as follows:

- (1) We propose a novel  $K$ -armed dueling bandits algorithm for large-scale online ranker evaluation, called MergeDTS. We use the idea of divide and conquer together with Thompson sampling to reduce the number of comparisons of arms.
- (2) We analyze the performance of MergeDTS and theoretically demonstrate that the soundness of MergeDTS can be guaranteed in the case of known time horizon and parameter values in the theoretical regime.
- (3) We evaluate MergeDTS experimentally on the Microsoft, Yahoo! Learning to Rank and ISTELLA datasets, and show that, with the tuned parameters, MergeDTS outperforms baselines in most of the large-scale online ranker evaluation configurations.

The rest of the chapter is organized as follows. In Section 2.2, we detail the definition of the dueling bandit problem. We discuss prior work in Section 2.3. MergeDTS is proposed in Section 2.4. Our experimental setup is detailed in Section 2.5 and the results are presented in Section 2.6. We conclude the chapter in Section 2.7.

## 2.2 Problem Setting

---

In this section, we first describe in more precise terms the  $K$ -armed dueling bandit problem, which is a variation of the Multi-Armed Bandits (MAB) problem. The latter can be described as follows: given  $K$  choices, called “arms” and denoted by  $a_1, \dots, a_K$ , we are required to choose one arm at each step; choosing arm  $a_i$  generates a reward which is drawn i.i.d. from a random variable with mean, denoted by  $\mu_i$ , and our goal is to maximize the expected total reward accumulated by our choices of arms over time. This objective is more commonly formulated in terms of the *cumulative regret* of the MAB algorithm, where regret at step  $t$  is the difference between the reward of the chosen arm, e.g.,  $a_j$ , and the reward of the best arm, e.g.,  $a_k$ , in hindsight, and the average regret of arm  $a_j$  is defined to be  $\mu_k - \mu_j$ : cumulative regret is defined to be the sum of the instantaneous regret over time [9, 77].

The dueling bandit problem differs from the above setting in that at each step we can choose up to two arms,  $a_i$  and  $a_j$  ( $a_i$  and  $a_j$  can be the same); the feedback is either  $a_i$  or  $a_j$ , as the winner of the comparison between the two arms (rather than an absolute reward), where  $a_i$  is chosen as the winner with *preference probability*  $p_{ij}$  and  $a_j$  with probability  $p_{ji} = 1 - p_{ij}$ . These probabilities form the entries of a  $K \times K$  *preference matrix*  $\mathbf{P}$ , which defines the dueling bandit problem but is not revealed to the dueling bandit algorithm.

In a similar fashion to the MAB setting, we evaluate a dueling bandit algorithm based on its *cumulative regret*, which is the total regret incurred by choosing suboptimal arms comparing to the best arm over time [16, 110, 126]. However, the definition of regret is less clear-cut in the dueling bandit setting, due to the fact that our dueling bandit problem might not contain a clear winner that is preferred to all other arms, i.e., an arm  $a_C$ , called the *Condorcet winner*, such that  $p_{Cj} > 0.5$  for all  $j \neq C$ . There are

numerous proposals in the literature for alternative notions of winners in the absence of a Condorcet winner, e.g., Borda winner [51, 112], Copeland winner [62, 129], von Neumann winner [129], with each definition having its own disadvantages as well as practical settings where its use is appropriate.

MergeDTS, like most of the other dueling bandits algorithms [16, 112, 127, 128, 130], relies on the existence of a Condorcet winner, in which case the Condorcet winner is the clear choice for the best arm, since it is preferred to all other arms, and with respect to which regret can be defined. We pose, as an interesting direction for future work, the task of extending the method proposed in this chapter to each of the other notions of winner listed above.

In order to simplify the notation in the rest of the chapter, we re-label the arms such that  $a_1$  is the Condorcet winner, although this is not revealed to the algorithm. We define the *regret* incurred by comparing  $a_i$  and  $a_j$  at time  $t$  to be

$$r_t = (\Delta_{1i} + \Delta_{1j})/2, \quad (2.1)$$

where  $\Delta_{1k} := p_{1k} - 0.5$  for each  $k$ . Moreover, the *cumulative regret* after  $T$  steps is defined to be

$$\mathcal{R}(T) = \sum_{t=1}^T r_t, \quad (2.2)$$

where  $r_t$  is the regret incurred by our choice of arms at time  $t$ .

Let us translate the online ranker evaluation problem into the dueling bandit problem. The input, a finite set of arms, consists of a set of rankers, e.g., based on different ranking models or based on the same model but with different parameters [60]. The Condorcet winner is the ranker that is preferred, by the majority of users, over suboptimal rankers. More specifically, a result list from the Condorcet winner is expected to receive the highest number of clicks from users when compared to a list from a suboptimal ranker. The preference matrix  $\mathbf{P}$  records the users' relative preferences for all rankers. Regret measures the user frustration incurred by showing the interleaved list from suboptimal rankers instead of the Condorcet winner. In the rest of the chapter, we use the term *ranker* to indicate the term *arm* in  $K$ -armed dueling bandit problems since we focus on the online ranker evaluation task.

## 2.3 Related Work

---

There are two main existing approaches for solving dueling bandit problems: (1) reducing the problem to a MAB problem, e.g., Sparring [5], Self-Sparring [109] and Relative Exponential-weight algorithm for Exploration and Exploitation (REX3) [29]; (2) generalizing existing MAB algorithms to the dueling bandit setting, e.g., Relative Upper Confidence Bound (RUCB) [127], Relative Minimum Empirical Divergence (RMED1) [61] and DTS [115]. The advantage of the latter group of algorithms is that they come equipped with theoretical guarantees, proven for a broad class of problems. The first group, however, have guarantees that either only hold for a restricted class of problems, where the dueling bandit problem is obtained by comparing the arms of an underlying

MAB problem (a.k.a. utility-based dueling bandits), e.g., Self-Sparring, REX3 and Sparring T-INF [125], or have substantially suboptimal instance-dependent regret bounds as in the case of Sparring EXP3, which has a regret bound of the form  $O(\sqrt{KT})$ , as opposed to  $O(K \log T)$ . Indeed, as our experimental results below demonstrate, Sparring-type algorithms can perform poorly when the dueling bandit problem does not arise from a MAB problem.

Below, we describe some of these algorithms to provide context for our work. Sparring [5] uses two MAB algorithms, e.g., Upper Confidence Bound (UCB), to choose rankers. At each step, Sparring asks each MAB algorithm to output a ranker to be compared. The two rankers are then compared and the MAB algorithm that proposed the winning ranker gets a reward of 1 and the other a reward of 0.

Self-Sparring [109] improves upon Sparring by employing a single MAB algorithm, but at each step samples twice to choose rankers. More precisely, Sui et al. [109] use Thompson Sampling (TS) as the MAB algorithm. Self-Sparring assumes that the problem it solves arises from an MAB; it can perform poorly when there exists a cycle relation in rankers, i.e., if there are rankers  $a_i, a_j$  and  $a_k$  with  $p_{ij} > 0.5$ ,  $p_{jk} > 0.5$  and  $p_{ki} > 0.5$ . As Self-Sparring does not estimate confidence intervals of the comparison results, it does not eliminate rankers.

Another extension of Sparring is REX3 [29], which is designed for the adversarial setting. REX3 is inspired by the Exponential-weight algorithm for Exploration and Exploitation (EXP3) [10], an algorithm for adversarial bandits, and has a regret bound of the form  $O(\sqrt{K \ln(K)T})$ . Note that the regret bound grows as the square-root of time-steps, but sublinearly in the number of rankers, which shows the potential for improvement in the case of large-scale problems.

Relative Upper Confidence Bound (RUCB) [127] extends UCB to dueling bandits using a matrix of optimistic estimates of the relative preference probabilities. At each step, RUCB chooses the first ranker to be one that beats all other rankers based on the idea of *optimism in the face of uncertainty*. Then it chooses the second ranker to be the ranker that beats the first ranker with the same idea of optimism in the face of uncertainty, which translates to pessimism for the first ranker. The cumulative regret of RUCB after  $T$  steps is upper bounded by an expression of the form  $O(K^2 + K \log T)$ .

RMED1 [61] extends an asymptotically optimal MAB algorithm, called Deterministic Minimum Empirical Divergence (DMED) [46], by first proving an asymptotic lower bound on the cumulative regret of all dueling bandit algorithms, which has the order of  $\Omega(K \log T)$ , and pulling each pair of rankers the minimum number of times prescribed by the lower bound. RMED1 outperforms RUCB and Sparring.

Double Thompson Sampling (DTS) [115] improves upon RUCB by using TS to break ties when choosing the first ranker. Specifically, it uses one TS to choose the first ranker from a set of candidates that are pre-chosen by UCB. Then it uses another TS to choose the second ranker that performs the best compared to the first one. The cumulative regret of DTS is upper bounded by  $O(K \log T + K^2 \log \log T)$ . Note that the bound of DTS is higher than that of RUCB. We hypothesize that this is because the bound of DTS is rather loose. DTS outperforms other dueling bandits algorithms empirically and is the state-of-the-art in the case of small-scale dueling bandit problems [109, 115]. As discussed in Section 2.6, for computational reasons DTS is not suitable for large-scale problems.

The work that is the closest to ours is by Zoghi et al. [130]. They propose MergeRUCB, which is the state-of-the-art for large-scale dueling bandit problems. MergeRUCB partitions rankers into small batches and compares rankers within each batch. A ranker is eliminated from a batch once we realize that even according to the most optimistic estimate of the preference probabilities it loses to another ranker in the batch. Once enough rankers have been eliminated, MergeRUCB repartitions the remaining rankers and continues as before. Importantly, MergeRUCB does not require global pairwise comparisons between all pairs of rankers, and so it reduces the computational complexity and increases the time efficiency, as shown in Section 2.6.2. The cumulative regret of MergeRUCB can be upper bounded by  $O(K \log T)$  [130], i.e., with no quadratic dependence on the number of rankers. This upper bound has the same order as the lower bound proposed by Komiyama et al. [61] in terms of  $K \log T$ , but it is not optimal in the sense that it has large constant coefficients. As we demonstrate in our experiments, MergeRUCB can be improved by making use of TS to reduce the amount of randomness in the choice of rankers. More precisely, the cumulative regret of MergeRUCB is almost twice as large as that of MergeDTS in the large-scale setup shown in Section 2.6.

A recent extension of dueling bandits is called *multi-dueling bandits* [14, 102, 109], where more than two rankers can be compared at each step. Multi-Dueling Bandits (MDB) is the first proposed algorithm in this setting, which is specifically designed for online ranker evaluation. It maintains two UCB estimators for each pair of rankers, a looser confidence bound and a tighter one. At each step, if there is more than one ranker that is valid for the tighter UCB estimators, MDB compares all the rankers that are valid for the looser UCB estimators. MDB is outperformed by Self-Sparring, the state-of-the-art *multi-dueling bandit* algorithm, significantly [109]. In this chapter, we do not focus on the *multi-dueling bandit* setup. The reasons are two-fold. First, to the best of our knowledge, there are no theoretical results in the multi-dueling setting that allow for the presence of cyclical preference relationships among the rankers. Second, Saha and Gopalan [102] state that “(perhaps surprisingly) [...] the flexibility of playing size- $k$  subsets does not really help to gather information faster than the corresponding dueling case ( $k = 2$ ), at least for the current subset-wise feedback choice model.” This statement demonstrates that there is no clear advantage to using multi-dueling comparisons over pairwise dueling comparisons at this moments.

---

## 2.4 Algorithm

---

In this section, we propose the MergeDTS algorithm, and explain the intuition behind it. Then, we provide theoretical guarantees bounding the regret of MergeDTS.

### 2.4.1 MergeDTS

The MergeDTS algorithm combines the benefits of both the elimination-based divide and conquer strategy of MergeRUCB and the sampling strategy of DTS, producing an effective scalable dueling bandit algorithm.

The pseudo-code for MergeDTS is provided in Algorithms 1 to 3, with the notation

Table 2.1: Notation used in this chapter.

Notation	Description
$K$	Number of rankers
$a_i$	The $i$ -th ranker
$p_{ij}$	Probability of $a_i$ beating $a_j$
$M$	Size of a batch
$\alpha$	Exploration parameter, $\alpha > 0.5$
$\epsilon$	Probability of failure
$\mathbf{W}$	The comparison matrix
$w_{ij}$	Number of times $a_i$ has beaten $a_j$
$s$	Stage of the algorithm
$\mathcal{B}_s$	Set of batches at the $s$ -th stage
$b_s$	Number of batches in $\mathcal{B}_s$
$\theta_{ij}$	Sampled probability of $a_i$ beating $a_j$
$a_c$	Ranker chosen in Phase I of MergeDTS
$\phi_i$	Sampled probability of $a_i$ beating $a_c$
$a_d$	Ranker chosen in Phase II of MergeDTS
$u_{ij}$	Upper confidence bound (UCB): $\frac{w_{ij}}{w_{ij}+w_{ji}} + \sqrt{\frac{\alpha \log(t+C(\epsilon))}{w_{ij}+w_{ji}}}$
$\Delta_{ij}$	$ p_{ij} - 0.5 $
$\Delta_{\min}$	$\min_{\Delta_{ij} > 0} \Delta_{ij}$
$\Delta_{B,\min}$	$\min_{a_i, a_j \in B \text{ and } i \neq j} \Delta_{ij}$
$C(\epsilon)$	$\left(\frac{(4\alpha-1)K^2}{(2\alpha-1)\epsilon}\right)^{\frac{1}{2\alpha-1}}$

summarized in Table 2.1 for the reader’s convenience. The input parameters are the exploration parameter  $\alpha$ , the size of a batch  $M$  and the failure probability  $\epsilon \in (0, 1)$ . The algorithm records the outcomes of the past comparisons in matrix  $\mathbf{W}$ , whose element  $w_{ij}$  is the number of times ranker  $a_i$  has beaten ranker  $a_j$  so far. MergeDTS stops when only one ranker remains and then returns that ranker, which it claims to be the Condorcet winner.<sup>2</sup>

MergeDTS begins by grouping rankers into small batches (line 4). At each time-step, MergeDTS checks whether there is more than one ranker remaining (line 7). If so, MergeDTS returns that ranker, the potential Condorcet winner. If not, MergeDTS considers one batch  $B_m$  and, using optimistic estimates of the preference probabilities (line 9), it purges any ranker that loses to another ranker even with an optimistic boost in favor of the former (line 10).

If, as a result of the above purge,  $B_m$  becomes a single-element batch, it is merged with the next batch  $B_{m+1}$  (line 12). Here,  $m + 1$  is interpreted as modulo  $b_s$ , where  $b_s$  is the number of batches in the current stage. This is done to avoid comparing a suboptimal ranker against itself, since if there is more than one batch, the best ranker

<sup>2</sup>In the online ranker evaluation application, we can stop MergeDTS once it finds the best ranker. However, in our experiments, we keep MergeDTS running by comparing the remaining ranker with itself. If the remaining ranker is the Condorcet winner, there will be no regret.

---

**Algorithm 1** MergeDTS (Merge Double Thompson Sampling)

**Input:**  $K$  rankers  $a_1, a_2, \dots, a_K$ ; partition size  $M$ ; exploration parameter  $\alpha > 0.5$ ; running time steps  $T$ ; probability of failure  $\epsilon = 1/T$ .

**Output:** The Condorcet winner.

```

1:  $\mathbf{W} \leftarrow \mathbf{0}_{K,K}$                                 {The comparison matrix}
2:  $C(\epsilon) = \left( \frac{(4\alpha-1)K^2}{(2\alpha-1)\epsilon} \right)^{\frac{1}{2\alpha-1}}$ 
3:  $s = 1$                                               {The stage of the algorithm}
4:  $\mathcal{B}_s = \left\{ \underbrace{[a_1, \dots, a_M]}_{B_1}, \dots, \underbrace{[a_{(b_1-1)M+1}, \dots, a_K]}_{B_{b_1}} \right\}$  {Disjoint batches of rankers,
   with  $b_1 = \lceil \frac{K}{M} \rceil \}$ 
5: for  $t = 1, 2, \dots, T$  do
6:    $m = t \bmod b_s$                                      {Index of the batches}
7:   if  $b_s = 1$  and  $|B_m| = 1$  {One ranker left} then
8:     Return the remaining ranker  $a \in B_m$ .
9:    $\mathbf{U} = \frac{\mathbf{W}}{\mathbf{W} + \mathbf{W}^T} + \sqrt{\left( \frac{\alpha \log(t+C(\epsilon))}{\mathbf{W} + \mathbf{W}^T} \right)}$  {UCB estimators: operations are
   element-wise and  $\frac{x}{0} := 1 \}$ 
10:  Remove  $a_i$  from  $B_m$  if  $u_{ij} < 0.5$  for any  $a_j \in B_m$ .
11:  if  $b_s > 1$  and  $|B_m| = 1$  then
12:    Merge  $B_m$  with the next batch and decrement  $b_s$ .
{Phase I: Choose the first candidate  $a_c$ }           {See Algorithm 2}
13:   $a_c = \text{SampleTournament}(\mathbf{W}, B_m)$ 
{Phase II: Choose the second candidate  $a_d$ }           {See Algorithm 3}
14:   $a_d = \text{RelativeTournament}(\mathbf{W}, B_m, a_c)$ 
{Phase III: Compare candidates and update batches}
15:  Compare pair  $(a_c, a_d)$  and increment  $w_{cd}$  if  $a_c$  wins otherwise increment  $w_{dc}$ .
{Phase IV: Update batch set}
16:  if  $\sum_m |B_m| \leq \frac{K}{2^s}$  then
17:    Pair the larger size batches with the smaller ones, making sure the size of every
    batch is in  $[0.5M, 1.5M]$ .
18:   $s = s + 1$ 
19:  Update  $\mathcal{B}_s, b_s = |\mathcal{B}_s|$ .

```

---

in any given batch is unlikely to be the Condorcet winner of the whole dueling bandit problem. As we will see again below, MergeDTS takes great care to avoid comparing suboptimal rankers against themselves because it results in added regret, but yields no extra information, since we know that each ranker is tied with itself.

After the above elimination step, the algorithm proceeds in four phases: choosing the first ranker (Phase I), choosing the second ranker based on the first ranker (Phase II), comparing the two rankers and updating the statistics (Phase III), and repartitioning the rankers at the end of each stage (Phase IV). Of the four phases, Phase I and Phase II are the major reasons that lead to a boost in effectiveness of MergeDTS when compared to MergeRUCB. We will elaborate both phases in the remainder of this section.

In Phase I, the method *SampleTournament* (Algorithm 2) chooses the first candidate ranker: MergeDTS samples preference probabilities  $\theta_{ij}$  from the posterior distributions to estimate the true preference probabilities  $p_{ij}$  for all pairs of rankers in the batch  $B_m$  (lines 1–3, the first TS). Based on these sampled probabilities, MergeDTS chooses the first candidate  $a_c$  so that it beats most of the other rankers according to the sampled preferences (line 5).

**Algorithm 2** SampleTournament

**Input:** The comparison matrix  $\mathbf{W}$  and the current batch  $B_m$ .

**Output:** The first candidate  $a_c$ .

- 1: **for**  $a_i, a_j \in B_m$  and  $i < j$  **do**
- 2:   Sample  $\theta_{ij} \sim Beta(w_{ij} + 1, w_{ji} + 1)$
- 3:    $\theta_{ji} = 1 - \theta_{ij}$
- 4:    $\kappa_i = \frac{1}{|B_m|-1} \sum_{a_j \in B_m, j \neq i} \mathbb{1}(\theta_{ij} > 0.5)$
- 5:    $a_c = \arg \max_{a_i \in B_m} \kappa_i$ ; *breaking ties randomly*                                      {First candidate}

In Phase II, the method *RelativeTournament* (Algorithm 3) chooses the second candidate ranker: MergeDTS samples another set of preference probabilities  $\phi_j$  from the posteriors of  $p_{jc}$  for all rankers  $a_j$  in  $B_m \setminus \{a_c\}$  (lines 1–2, the second TS). Moreover, we set  $\phi_c$  to be 1 (line 3). This is done to avoid self-comparisons between suboptimal rankers for the reasons that were described above.

**Algorithm 3** RelativeTournament

**Input:** The comparison matrix  $\mathbf{W}$ , the current batch  $B_m$  and the first candidate  $a_c$ .

**Output:** The second candidate  $a_d$ .

- 1: **for**  $a_j \in B_m$  and  $j \neq c$  **do**
- 2:   Sample  $\phi_j \sim Beta(w_{jc} + 1, w_{cj} + 1)$
- 3:    $\phi_c = 1$     {Avoid self-comparison}
- 4:    $a_d = \arg \min_{a_j \in B_m} \phi_j$ ; *breaking ties randomly*                                      {Second candidate}

Once the probabilities  $\phi_j$  have been sampled, we choose the ranker  $a_d$  that is going to be compared against  $a_c$ , using the following strategy. The worst ranker according to the sampled probabilities  $\phi_j$  is chosen as the second candidate  $a_d$  (line 4). The rationale for this discrepancy is that we would like to eliminate rankers as quickly as possible, so rather than using the upper confidence bounds to explore when choosing  $a_d$ , we use the lower confidence bounds to knock the weakest link out of the batch as quickly as possible.

In Phase III (line 15) of Algorithm 1, MergeDTS plays  $a_c$  and  $a_d$  and updates the comparison matrix  $\mathbf{W}$  based on the observed feedback.

Finally, in Phase IV (lines 16–19), if the number of remaining rankers in the current stage is half of the rankers of the previous stage (line 16), MergeDTS enters the next stage, before which it repartitions the rankers. Following the design of MergeRUCB, this is done by merging batches of rankers such that the smaller sized batches are

combined with the larger sized batches; we enforce that the number of rankers in the new batches is kept in the range of  $[0.5M, 1.5M]$ .

### 2.4.2 Theoretical guarantees

In this section, we state and prove a high probability upper bound on the regret accumulated by MergeDTS after  $T$  steps, under the assumption that the dueling bandit problem contains a Condorcet winner. Since the theoretical analysis of MergeDTS is based on that of MergeRUCB, we start by listing two assumptions that we borrow from MergeRUCB in [130, Section 7].

**Assumption 2.1.** *There is no repetition in rankers. All ranker pairs  $(a_i, a_j)$  with  $i \neq j$  are distinguishable, i.e.,  $p_{ij} \neq 0.5$ , unless both of them are “uninformative” rankers that provide random ranked lists and cannot beat any other rankers.*

**Assumption 2.2.** *The uninformative rankers are at most one third of the full set of rankers.*

These assumptions arise from the Yahoo! Learning to Rank Challenge dataset, where there are 181 out of 700 rankers that always provide random ranked lists. Assumption 2.1 ensures that each informative ranker is distinguishable from other rankers. Assumption 2.2 restricts the maximal percentage of uninformative rankers and thus ensures that the probability of triggering the merge condition (line 16 in Algorithm 1) is larger than 0.<sup>3</sup> Moreover, we emphasize that Assumptions 2.1 and 2.2 are milder than the assumptions made in Self-Sparring and DTS, where indistinguishability is simply not allowed.

We now state our main theoretical result:

**Theorem 2.1.** *With the known time step  $T$ , applying MergeDTS with  $\alpha > 0.5$ ,  $M \geq 4$  and  $\epsilon = 1/T$  to a  $K$ -armed Condorcet dueling bandit problem under Assumptions 2.1 and 2.2, with probability  $1 - \epsilon$  the cumulative regret  $\mathcal{R}(T)$  after  $T$  steps is bounded by:*

$$\mathcal{R}(T) < \frac{8\alpha MK \ln(T + C(\epsilon))}{\Delta_{\min}^2}, \quad (2.3)$$

where

$$\Delta_{\min} := \min_{\Delta_{ij} > 0} \Delta_{ij}, \quad (2.4)$$

is the minimal gap of two distinguishable rankers and  $C(\epsilon) = \left( \frac{(4\alpha-1)K^2}{(2\alpha-1)\epsilon} \right)^{\frac{1}{2\alpha-1}}$ .

The upper bound on the  $T$ -step cumulative regret of MergeDTS is  $O(K \ln(T)/\Delta_{\min}^2)$ . In other words, the cumulative regret grows linearly with the number of rankers,  $K$ . This is the most important advantage of MergeDTS, which states the potential of applying it to the large-scale online evaluation. We emphasize that for most of the  $K$ -armed

---

<sup>3</sup>In practice, MergeDTS works without Assumption 2.2 because the Condorcet winner eliminates all other arms eventually with  $O(K^2 \log T)$  comparisons. We keep Assumption 2.2 to ensure that MergeDTS also works in cases where we have the  $O(K \log(T))$  guarantee. We refer readers to [130] for a detailed discussion.

dueling bandit algorithms in the literature, the upper bounds contain a  $K^2$  term, which renders them unsuitable for large-scale online ranker evaluation. By the definition of  $\Delta_{\min}$  in Eq. (2.4) we have  $\Delta_{\min} > 0$ , and so our bound is well-defined. However, the performance of MergeDTS may degrade severely when  $\Delta_{\min}$  is small.  $\alpha$  is a common parameter in UCB-type algorithms, called the exploration parameter.  $\alpha$  controls the trade-off between exploitation and exploration: larger  $\alpha$  results in more exploration, whereas smaller  $\alpha$  makes the algorithm more exploitative. Theoretically,  $\alpha$  should be larger than 0.5. However, as shown in our experiments, using some values of  $\alpha$  that are outside the theoretical regime can lead to a boost in the effectiveness of MergeDTS.

Theorem 2.1 provides a finite-horizon high probability bound. From a practical point of view, this type of bound is of great utility. In practice, bandit algorithms are always deployed and evaluated within limited user iterations [80, 121]. Here, each time step is one user interaction. As the number of interactions is provided, we can choose a reasonable step  $T$  to make sure the high probability bound holds. We can also get an expected regret bound of MergeDTS at step  $T$  by setting  $\epsilon = 1/T$  and adding 1 to the right-hand side of Eq. (2.3): this is because  $\mathbb{E}[\mathcal{R}(T)]$  can be bounded by

$$\frac{1}{T} \cdot T + \frac{T-1}{T} \cdot \frac{8\alpha MK \ln(T+C(\epsilon))}{\Delta_{\min}^2} \leq 1 + \frac{8\alpha MK \ln(T+C(\epsilon))}{\Delta_{\min}^2}. \quad (2.5)$$

We note that the above expected bound holds only at time-step  $T$  and so the horizonless version of MergeDTS does not possess an expected regret bound.

The proof of Theorem 2.1 relies on the Lemma 3 in [130]. We repeat it here for the reader's convenience.

**Lemma 2.1** (Lemma 3 in [130]). *Given any pair of distinguishable rankers  $a_i, a_j \in B$  and  $\epsilon \in [0, 1]$ , with the probability of  $1 - \epsilon$ , the maximum number of comparisons that could have been carried out between these two rankers in the first  $T$  time-steps before a merger between  $B$  and another batch occurs, is bounded by*

$$\frac{4\alpha \ln(T+C(\epsilon))}{\Delta_{B,\min}^2}, \quad (2.6)$$

where  $\Delta_{B,\min} = \min_{a_i, a_j \in B \text{ and } i \neq j} \Delta_{ij}$  is the minimal gap of two distinguishable rankers in batch  $B$ .

*Proof of Theorem 2.1.* Lemma 2.1 states that with probability  $1 - \epsilon$  the number of comparisons between a pair of distinguishable rankers  $(i, j) \in B$  is bounded by

$$\frac{4\alpha \ln(T+C(\epsilon))}{\Delta_{B,\min}^2}, \quad (2.7)$$

regardless of the way the rankers are selected, as long as the same criterion as MergeRUCB is used for eliminating rankers. Since the elimination criterion for MergeDTS is the same as that of MergeRUCB, we can apply the same argument used to prove Theorem 1 in [130] to get a bound of

$$\frac{8\alpha MK \ln(T+C(\epsilon))}{\Delta_{\min}^2} \quad (2.8)$$

on the regret accumulated by MergeDTS. Here we use the fact that  $\Delta_{B,\min} \geq \Delta_{\min}$  and thus  $\frac{4\alpha \ln(T+C(\epsilon))}{\Delta_{B,\min}^2} \leq \frac{4\alpha \ln(T+C(\epsilon))}{\Delta_{\min}^2}$ .  $\square$

### 2.4.3 Discussion

The prefix “merge” in MergeDTS signifies the fact that it uses a similar divide-and-conquer strategy as merge sort. It partitions the  $K$ -arm set into small batches of size  $M$ . The comparisons only happen between rankers in the same batch, which, in turn, avoids global pairwise comparisons and gets rid of the  $O(K^2)$  dependence in the cumulative regret, which is the main limitation for using dueling bandits for large datasets.

In contrast to sorting, MergeDTS needs a large number of comparisons before declaring a difference between rankers since the feedback is stochastic. The harder two rankers are to distinguish or in other words the closer  $p_{ij}$  is to 0.5, the more comparisons are required. Moreover, if a batch only contains the uninformative rankers, the comparisons between those rankers will not stop, which incurs infinite regret. MergeDTS reduces the number of comparisons between hardly distinguishable rankers as follows:

- (1) MergeDTS compares the best ranker in the batch to the worst to avoid comparisons between hardly distinguishable rankers;
- (2) When half of the rankers of the previous stage are eliminated, MergeDTS pairs larger batches to smaller ones that contain at least one informative ranker and enters the next stage.

The second point is borrowed from the design of MergeRUCB.

MergeDTS and MergeRUCB follow the same “merge” strategy. The difference between these two algorithms is in their strategy of choosing rankers, i.e., Algorithms 2 and 3. MergeDTS employs a sampling strategy to choose the first ranker inside the batch and then uses another sampling strategy to choose the second ranker that is potentially beaten by the first one. As stated above, this design comes from the fact that MergeDTS is carefully designed to reduce the comparisons between barely distinguishable rankers. In contrast to MergeDTS, MergeRUCB randomly chooses the first ranker and chooses the second ranker to be the one that is the most likely to beat the first ranker, as discussed in Section 2.3. The uniformly random strategy inevitably increases the number of comparisons between those barely distinguishable rankers.

In summary, the double sampling strategy used by MergeDTS is the major factor that leads to the superior performance of MergeDTS as demonstrated by our experiments.

---

## 2.5 Experimental Setup

---

In this chapter, we investigate the application of dueling bandits to the large-scale online ranker evaluation setup. Our experiments are designed to answer **RQ1**, which we map into a set of six more refined research questions.

### 2.5.1 Research questions

**RQ1.1** In the large-scale online ranker evaluation task, does MergeDTS outperform the state-of-the-art large-scale dueling bandits algorithms in terms of cumulative regret, i.e., effectiveness?

In the bandit literature [16, 126], *regret* is a measure of the rate of convergence to the Condorcet winner in hindsight. Mapping this to the online ranker evaluation setting, **RQ1.1** asks whether MergeDTS hurts the user experience less than baselines while it is being used for large-scale online ranker evaluation.

**RQ1.2** How do MergeDTS and the baselines scale computationally?

What is the time complexity of MergeDTS? Does MergeDTS require less running time than the baselines?

**RQ1.3** How do different levels of noise in the feedback signal affect cumulative regret of MergeDTS and the baselines?

In particular, will we still observe the same results in **RQ1.1** after a (simulated) user changes its behavior? How sensitive are MergeDTS and the baselines to noise?

**RQ1.4** How do MergeDTS and the baselines perform when the Condorcet dueling bandit problem contains cycles?

Previous work has found that cyclical preference relations between rankers are abundant in online ranker comparisons [128, 130]. Can MergeDTS and the baselines find the Condorcet winner when the experimental setup features a large number of cyclical relations between rankers?

**RQ1.5** How does MergeDTS perform when the dueling bandit problem violates the Condorcet assumption?

We focus on the Condorcet dueling bandit task in this chapter. Can MergeDTS be applied to dueling bandit tasks without the existence of a Condorcet winner?

**RQ1.6** What is the parameter sensitivity of MergeDTS?

Can we improve the performance of MergeDTS by tuning its parameters, such as the exploration parameter  $\alpha$ , the size of a batch  $M$ , and the probability of failure  $\epsilon$ ?

## 2.5.2 Datasets

To answer our research questions, we use two types of dataset: three real-world datasets and a synthetic dataset. First, to answer **RQ1.1** to **RQ1.3**, we run experiments on three large-scale datasets: the Microsoft Learning to Rank (MSLR) WEB30K dataset [96], the Yahoo! Learning to Rank Challenge Set 1 (Yahoo) [19] and the ISTEELA dataset [85].<sup>4</sup> These datasets contain a large number of features based on unsupervised ranking functions, such as BM25, TF-IDF, etc. In our experiments, we take the widely used setup in which each individual feature is regarded as a ranker [41, 130]. This is different from a real-world setup, where a search system normally ranks documents using a well trained learning to rank algorithm that combines multiple features. However, the difficulty of a dueling bandit problem comes from the relative quality of pairs of rankers

---

<sup>4</sup>We omit the Yahoo Set 2 dataset because it contains far fewer queries than the Yahoo Set 1 dataset.

and not from their absolute quality. In other words, evaluating rankers with similar and possibly low performance is as hard as evaluating state-of-the-art rankers, e.g., LambdaMART [15]. Therefore, we stick to the standard setup of [41, 130], treating each feature as a ranker and each ranker as an arm in the  $K$ -armed dueling bandit problem. We leave experiments aimed at comparing different well trained learning to rank algorithms as future work. As a summary, the MSLR dataset contains 136 rankers, the Yahoo dataset contains 700 rankers and the ISTELLA dataset contains 220 rankers. Compared to the typical  $K$ -armed dueling bandit setups, where  $K$  is generally substantially smaller than 100 [5, 109, 115, 122], these are large numbers of rankers.

Second, to answer **RQ1.4**, we use a synthetic dataset, generated by Zoghi et al. [130], which contains cycles (called the *Cycle dataset* in the rest of the chapter). The Cycle dataset has 20 rankers with one Condorcet winner,  $a_1$ , and 19 suboptimal rankers,  $a_2, \dots, a_{20}$ . The Condorcet winner beats the other 19 suboptimal rankers. And those 19 rankers have a cyclical preference relationship between them. More precisely, following Zoghi et al. [130], the estimated probability  $p_{1j}$  of  $a_1$  beating  $a_j$  ( $j = 2, \dots, 20$ ) is set to  $p_{1j} = 0.51$ , and the preference relationships between the suboptimal rankers are described as follows: visualize the 19 rankers  $a_2, \dots, a_{20}$  sitting at a round table, then each ranker beats every ranker to its left with probability 1 and loses to every ranker to its right with probability 1. In this way we obtain the Cycle dataset.

Note that this is a difficult setup for Self-Sparring, because Self-Sparring chooses rankers based on their Borda scores, and the Borda scores ( $\sum_{j=1}^K p_{ij}$  for each ranker  $a_i$  [112]) are close to each other in the Cycle dataset. For example, the Borda score of the Condorcet winner is 10.19, while the Borda score of a suboptimal ranker is 9.99. This makes it hard for Self-Sparring to distinguish between rankers. In order to be able to conduct a fair comparison, we generate the *Cycle2 dataset*, where each suboptimal ranker beats every ranker to its left with a probability of 0.51 and the Condorcet winner beats all others with probability 0.6. Now, in the Cycle2 dataset, the Borda score of the Condorcet winner is 11.90, while the Borda score of a suboptimal ranker is 9.9. Thus, it is an easier setup for Self-Sparring.

Furthermore, to answer **RQ1.5**, we use the MSLR-non-Condorcet dataset from [115], which is a subset of the MSLR dataset that does not contain a Condorcet winner. This datasets has 32 rankers with two Copeland winners (instead of one), each of which beats the other 30 rankers. A *Copeland winner* is a ranker that beats the largest number of other rankers; every dueling bandit dataset contains at least one Copeland winner [129].

Finally, we use the MSLR dataset with the navigational configuration (described in Section 2.5.4) to assess the parameter sensitivity of MergeDTS **RQ1.6**.

### 2.5.3 Evaluation methodology

To evaluate dueling bandit algorithms, we follow the proxy approach from [130]. It first uses an interleaving algorithm to obtain a preference matrix, i.e., a matrix that for each pair of rankers contains the probability that one ranker beats the other. More precisely, for each pair of rankers  $a_i$  and  $a_j$ ,  $p_{ij}$  is the estimation that  $a_i$  beats  $a_j$  in the simulated interleaved comparisons. Then, this obtained preference matrix is used to evaluate dueling bandit algorithms: for two rankers  $a_i$  and  $a_j$  chosen by a dueling

bandit algorithm, we compare them by drawing a sample from a Bernoulli distribution with mean  $p_{ij}$ , i.e., 1 means that  $a_i$  beats  $a_j$  and vice versa. This is a standard approach to evaluating dueling bandit algorithms [115, 122, 128]. Moreover, the proxy approach has been shown to have the same quality as interleaving in terms of evaluating dueling bandit algorithms [130].

In this chapter, we adopt the procedure described by Zoghi et al. [130] and obtain preference matrices for MSLR, Yahoo and ISTELLA datasets. Specifically, we use Probabilistic Interleave [41] to obtain preference matrices.<sup>5</sup> The numbers of comparisons for every pair of rankers in MSLR, Yahoo and ISTELLA datasets are 400 000, 60 000 and 400 000. The reason for the discrepancy is pragmatic: the Yahoo dataset has roughly 27 times as many pairs of rankers to be compared.

### 2.5.4 Click simulation

As the interleaved comparisons mentioned above are carried out using click feedback, we follow Hofmann et al. [43] and simulate clicks using three configurations of a click model [23]: namely *perfect*, *navigational* and *informational*. The perfect configuration simulates a user who checks every document and clicks on a document with a probability proportional to the query-document relevance. This configuration is the easiest one for dueling bandit algorithms to find the best ranker, because it contains very little noise. The navigational configuration mimics a user who seeks specific information, i.e., who may be searching for the link of a website, and is likely to stop browsing results after finding a relevant document. The navigational configuration contains more noise than the perfect configuration and is harder for dueling bandit algorithms to find the best ranker. Finally, the informational configuration represents a user who wants to gather all available information for a query and may click on documents that are not relevant with high probability. In the informational configuration the feedback contains more noise than in the perfect and navigational configurations, which makes it the most difficult configuration for dueling bandit algorithms to determine the best ranker, which, in turn, may result in the highest cumulative regret among the three configurations.

To answer the research questions that concern large-scale dueling bandit problems, namely **RQ1.1**, **RQ1.2** and **RQ1.6**, we use the navigational configuration, which represents a reasonable middle ground between the perfect and informational configurations [41]. The corresponding experimental setups are called MSLR-Navigational, Yahoo-Navigational and ISTELLA-Navigational. To answer **RQ1.3** regarding the effect of feedback with different levels of noise, we use all three configurations on MSLR, Yahoo and ISTELLA datasets: namely MSLR-Perfect, MSLR-Navigational and MSLR-Informational; Yahoo-Perfect, Yahoo-Navigational and Yahoo-Informational; ISTELLA-Perfect, ISTELLA-Navigational and ISTELLA-Informational. Thus, we have nine large-scale setups in total.

### 2.5.5 Baselines

We compare MergeDTS to five state-of-the-art dueling bandit algorithms: MergeRUCB [130], DTS [115], RMED1 [61], Self-Sparring [109], and REX3 [29].

---

<sup>5</sup>We use the implementation of Probabilistic Interleave in the LEROT software package [104].

Among these algorithms, MergeRUCB is designed for large-scale online ranker evaluation and is the state-of-the-art large-scale dueling bandit algorithm. DTS is the state-of-the-art small-scale dueling bandit algorithm. RMED1 is motivated by the lower bound of the Condorcet dueling bandit problem and matches the lower bound up to a factor of  $O(K^2)$ , which indicates that RMED1 has low regret in small-scale problems but may have large regret when the number of rankers is large. Self-Sparring is a more recently proposed dueling bandit algorithm that is the state-of-the-art algorithm in the multi-dueling setup, with which multiple rankers can be compared in each step. REX3 is proposed for the adversarial dueling bandit problem but also performs well for the large-scale stochastic dueling bandit problem [29]. We do not include RUCB [127] and Sparring [5] in our experiments since they have been outperformed by more than one of our baselines [29, 109, 130].

### 2.5.6 Parameters

Recall that Theorem 2.1 is based on Lemma 3 in [130]. The latter provides a high probability guarantee that the confidence intervals will not mislead the algorithm into eliminating the Condorcet winner by mistake. However, this result is proven using the Chernoff-Hoeffding [39] bound together with an application of the union bound [17], both of which introduce certain gaps between theory and practice. That is, the analysis of regret mainly considers the worst-case scenario rather than the average-case scenario, which makes regret bounds much looser than they could have been. We conjecture that the expression for  $C(\epsilon)$ , which derives its form from Lemma 3 in [130], is excessively conservative. Put differently, Theorem 2.1 specifies a sufficient condition for the proper functioning of MergeDTS, not a necessary one. So, a natural question that arises is the following: to what extent can restrictions imposed by our theoretical results be violated without the algorithm failing in practice? In short, what is the gap between theory and practice and what is the parameter sensitivity of MergeDTS?

To address these questions and answer **RQ1.6**, we conduct extensive parameter sensitivity analyses in the MSLR-Navigational setup with the following parameters:  $\alpha \in \{0.8^0, 0.8^1, \dots, 0.8^9\}$ ,  $C \in \{4 \times 10^2, 4 \times 10^3, \dots, 4 \times 10^6, 4\,726\,908\}$ , and  $M \in \{2, 4, 8, 16\}$ .  $C$  is short for  $C(\epsilon)$ , where  $C(\epsilon)$  is the exploration bonus added to the confidence intervals. According to Table 2.1,  $C(\epsilon)$  is a function of  $\alpha$  and  $\epsilon$ . However, to simplify our experimental setup, we consider  $C$  as an individual parameter rather than a function parameterized by  $\alpha$  and  $\epsilon$ , and study the impact of  $C$  directly. The details about the choice of the values are explained in the following paragraph.

When choosing candidate values for  $\alpha$ , we want them to cover the optimal theoretical value  $\alpha > 1$ , the lowest theoretically legal value  $\alpha > 0.5$ , and for smaller values of  $\alpha$  we want to decrease the differences between two consecutive  $\alpha$ 's. This last condition is imposed because smaller values of  $\alpha$  may mislead MergeDTS to eliminate the Condorcet winner. So we shrink the search space for smaller values of  $\alpha$ . The powers of 0.8 from 0 to 9 seem to satisfy the above conditions, particularly  $0.8^3 \approx 0.5$  and obviously  $0.8^0 = 1$  with the difference between  $0.8^n$  and  $0.8^{n+1}$  becoming smaller with larger  $n$ . The value  $C = 4\,726\,908$  is calculated from the definition of  $C(\epsilon)$  with the default  $\alpha = 1.01$  and  $M = 4$  (see Table 2.1), noting that the MSLR-Navigational setup contains 136 rankers, i.e.,  $K = 136$ . As discussed before, the design of  $C(\epsilon)$  may

be too conservative. So, we only choose candidate values smaller than 4 726 908. We use the log-scale of  $C(\epsilon)$  because the upper bound is logarithmic with  $C(\epsilon)$ .

The sensitivity of parameters is analyzed by following the order of their importance to Theorem 2.1, i.e.,  $\alpha$  and  $M$  have a linear relation to the cumulative regret and  $C$  has a logarithmic relation to the cumulative regret. We first evaluate the sensitivity of  $\alpha$  with the default values of  $M$  and  $C$ . Then we use the best value of  $\alpha$  to test a range of values of  $M$  (with default  $C$ ). Finally, we analyze the impact of  $C$  using the best values of  $\alpha$  and  $M$ .

We discover the practically optimal parameters for MergeDTS to be  $\alpha = 0.8^6$ ,  $M = 16$  and  $C = 4\,000\,000$ , in Section 2.6.6. We repeat the procedure for MergeRUCB and DTS, and use their optimal parameter values in our experiments, which are  $\alpha = 0.8^6$ ,  $M = 8$ ,  $C = 400\,000$  for MergeRUCB and  $\alpha = 0.8^7$  for DTS. Then, we use these values to answer **RQ1.1** to **RQ1.5**. Self-Sparring does not have any parameters, so further analysis and tuning are not needed here.

The shrewd readers may notice that the parameters are somewhat overtuned in the MSLR-Navigational setup, and MergeDTS with the tuned parameters does not enjoy the theoretical guarantees in Theorem 2.1. However, because of the existence of the gap between theory and practice, we want to answer the question whether we can improve the performance of MergeDTS as well as that of the baselines by tuning the parameters outside of their theoretical limits. We also want to emphasize that the parameters of MergeDTS and baselines are only tuned in the MSLR-Navigational setup, but MergeDTS is compared to baselines in nine setups. If, with the tuned parameters, MergeDTS outperforms baselines in other eight setups, we can also show the potential of improving MergeDTS in an empirical way.

### 2.5.7 Metrics

In our experiments, we assess the efficiency (time complexity) and effectiveness (cumulative regret) of MergeDTS and baselines. The metric for efficiency is the running time in days. We compute the running time from the start of the first step to the end of the  $T$ -th step, where  $T = 10^8$  in our experiments. A commercial search engine may serve more than 1 billion search requests per day [34], and each search request can be used to conduct one dueling bandit comparison. The total number of time steps, i.e.  $T$ , considered in our experiments is about 1% of the one-week traffic of a commercial search engine.

We use cumulative regret in  $T$  steps to measure the effectiveness of algorithms, which is computed as follows:

$$\mathcal{R}(T) = \sum_{t=1}^T r(t) = \sum_{t=1}^T \frac{1}{2} (\Delta_{1,c_t} + \Delta_{1,d_t}), \quad (2.9)$$

where  $r(t)$  is the regret at step  $t$ ,  $c_t$  and  $d_t$  are the indices of rankers chosen at step  $t$ , and without loss of generality, we assume  $a_1$  to be the Condorcet winner. The regret  $r(t)$  arises from the comparisons between the two suboptimal rankers at  $t$  step. It is the average sub-optimality of comparing two rankers  $a_i$  and  $a_j$  with respect to the Condorcet winner  $a_1$ , i.e.,  $\frac{p_{1i}+p_{1j}}{2} - 0.5$ . In a real-world scenario, we have a fixed time

## 2. Effective Large-Scale Online Ranker Evaluation

---

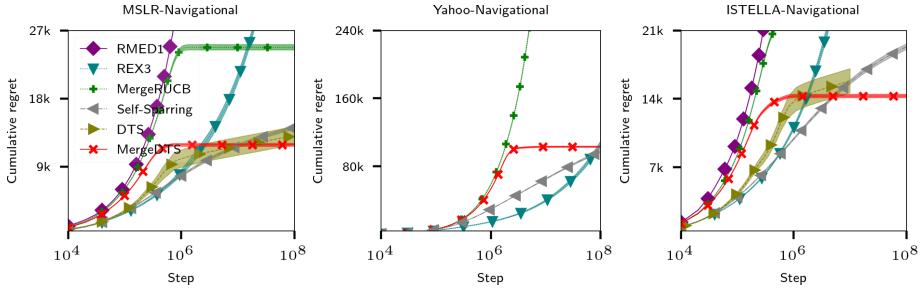


Figure 2.1: Cumulative regret on large-scale online ranker evaluation: lower is better. Note that the scales of the y-axes are different. The shaded areas are  $\pm$  standard error. The results are averaged over 50 independent runs.

period to conduct our online ranker evaluation, and thus, the number of steps  $T$  can be estimated beforehand. In our online ranker evaluation task, the  $T$ -step cumulative regret is related to the drop in user satisfaction during the evaluation process, i.e. higher regret means larger degradation of user satisfaction, because the preference  $p_{1i}$  can be interpreted as the probability of the Condorcet winner being preferred to ranker  $i$ .

Unless stated differently, the results in all experiments are averaged over 50 and 100 independent runs on large- and small-scale datasets respectively, where both numbers are either equal to or larger than the choices in previous studies [109, 115, 130]. In the effectiveness experiments, we also report the standard error of the average cumulative regret, which measures the differences between the average of samples and the expectation of the sample population.

All experiments on the MSLR and Yahoo datasets are conducted on a server with Intel(R) Xeon(R) CPU E5-2650 2.00GHz (32 Cores) and 64 Gigabyte. All experiments on the ISTELLA dataset are conducted on servers with Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz (48 Cores) and 256 Gigabyte. To be precise, an individual run of each algorithm is conducted on a single core with 1 Gigabyte.

## 2.6 Experimental Results

---

In this section, we analyze the results of our experiments. In Section 2.6.1, we compare the effectiveness (cumulative regret) of MergeDTS and the baselines in three large-scale online evaluation setups. In Section 2.6.2, we compare and analyze the efficiency (time complexity) of MergeDTS and the baselines. In Section 2.6.3, we study the impact of different levels of noise in the click feedback on the algorithms. In Section 2.6.4 and Section 2.6.5, we evaluate MergeDTS and the baselines in two alternative setups: the cyclic case and the non-Condorcet case, respectively. In Section 2.6.6, we analyze the parameter sensitivity of MergeDTS.

### 2.6.1 Large-scale experiments

To answer **RQ1.1**, we compare MergeDTS to the large-scale state-of-the-art baseline, MergeRUCB, as well as the more recently proposed Self-Sparring, in three large-scale online evaluation setups, namely MSLR-Navigational, Yahoo-Navigational and ISTELLA-Navigational. The results are reported in Fig. 2.1, which depicts the cumulative regret of each algorithm, averaged over 50 independent runs. As mentioned in Section 2.5, cumulative regret measures the rate of convergence to the Condorcet winner in hindsight and thus lower regret curves are better. DTS and RMED1 are not considered in the Yahoo-Navigational setup, and the cumulative regret of DTS is reported with in  $10^7$  steps on the ISTELLA dataset, because of the computational issues, which are further discussed in Section 2.6.2. Fig. 2.1 shows that MergeDTS outperforms the large-scale state-of-the-art MergeRUCB with large gaps. Regarding the comparison with Self-Sparring and DTS, we would like to point out the following facts: (1) MergeDTS outperforms DTS and Self-Sparring in MSLR-Navigational and ISTELLA-Navigational setups; (2) In the Yahoo-Navigational setup, MergeDTS has slightly higher cumulative regret than Self-Sparring, but MergeDTS converges to the Condorcet winner after three million steps while the cumulative regret of Self-Sparring is still growing after 100 million steps. Translating these facts into real-world scenarios, MergeDTS has higher regret compared to DTS and Self-Sparring in the early steps, but MergeDTS eventually outperforms DTS and Self-Sparring with longer experiments. As for REX3, we see that it has a higher order of regret than other algorithms since REX3 is designed for the adversarial dueling bandits and the regret of REX3 is  $O(\sqrt{T})$ . In this chapter, we consider the stochastic dueling bandits, and the regret of the other algorithms is  $\log(T)$ . MergeDTS outperforms the baselines in most setups, but we need to emphasize that the performance of MergeDTS here cannot be guaranteed by Theorem 2.1. This is because we use the parameter setup outside of the theoretical regime, as discussed in Section 2.5.6.

### 2.6.2 Computational scalability

To address **RQ1.2**, we report in Table 2.2 the average running time (in days) of each algorithm in three large-scale dueling bandit setups, namely MSLR-Navigational, ISTELLA and Yahoo-Navigational. As before, each algorithm is run for  $10^8$  steps. An individual run of DTS and RMED1 in the Yahoo-Navigational setup takes around 100.27 and 18.39 days, respectively, which is simply impractical for our experiments; therefore, the running time of DTS and RMED1 in this setup is estimated by multiplying the average running time at  $10^5$  steps by  $10^3$ . For a similar reason, we estimate the running time of DTS on the ISTELLA-Navigational setup by multiplying the average running time at  $10^7$  by 10.

Table 2.2 shows that MergeDTS and MergeRUCB have very low running times. This is due to the fact that they perform computations inside batches and their computational complexity is  $O(TM^2)$ , where  $T$  is the number of steps and  $M$  is the size of batches. Moreover, MergeDTS is considerably faster in the MSLR-Navigational setup, because there it finds the best ranker with fewer steps, as can be seen in Fig. 2.1. After finding this best ranker, the size of batches  $M$  becomes 1 and, from that moment

Table 2.2: Average running time in days of each algorithm on large-scale problems for  $10^8$  steps averaged over 50 independent runs. The running time of DTS in the ISTELLA-Navigational setup is estimated based on the running time with  $10^7$  steps multiplied by 10. The running time of DTS and RMED1 in the Yahoo-Navigational setup is estimated by multiplying the average running time at  $10^5$  steps by  $10^3$ . The experiments on the ISTELLA dataset are conducted on different computer clusters from those on the MSLR and Yahoo datasets. The speed of the former ones is about one time faster than the latter ones. Therefore the numbers may not be directly compared.

	MSLR	ISTELLA	Yahoo
# Rankers	136	220	700
MergeDTS	0.08	0.03	0.11
MergeRUCB	0.08	0.03	0.11
Self-Sparring	0.18	0.22	0.90
DTS	5.23	9.88	<i>100.27</i>
RMED1	0.36	0.19	<i>18.39</i>
REX3	0.25	0.11	0.27

on, MergeDTS does not perform any extra computations. The running time of Self-Sparring is also low, but grows with the number of rankers, roughly linearly. This is because at each step Self-Sparring draws a sample from the posterior distribution of each ranker and its running time is  $\Omega(TK)$ , where  $K$  is the number of rankers. DTS is orders of magnitude slower than other algorithms and its running time grows roughly quadratically, because DTS requires a sample for each pair of rankers at each step and its running time is  $\Omega(TK^2)$ .

Large-scale commercial search systems process over a billion queries a day [34], and run hundreds of different experiments [60] concurrently, in each of which two rankers are compared. The running time for DTS and RMED1 that appears in Table 2.2 is far beyond the realm of what might be considered reasonable to process on even 20% of one day’s traffic: note that one query in the search setting corresponds to one step for a dueling bandit algorithm, since each query could be used to compare two rankers by performing an interleaved comparison. Given the estimated running times listed in Table 2.2, we exclude DTS and RMED1 from our experiments on the large-scale datasets for practical reasons.

### 2.6.3 Impact of noise

To address **RQ1.3**, we run MergeDTS in the perfect, navigational and informational configurations (see Section 2.5.4). As discussed in Section 2.5.4, the perfect configuration is the easiest one for dueling bandit algorithms to reveal the best ranker, while the informational configuration is the hardest. We report the results of the perfect and informational configurations in Fig. 2.2. For comparison, we refer readers to plots in Fig. 2.1 for the results of navigational configuration.

On the MSLR and ISTELLA datasets, in all three configurations, MergeDTS with

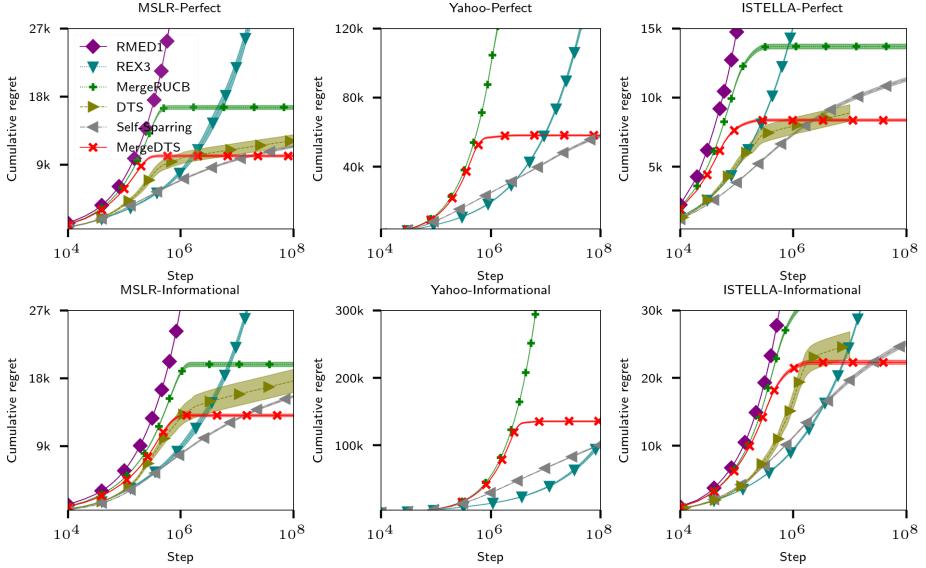


Figure 2.2: Effect of the level of noise on cumulative regret in click feedback (top row: perfect configuration and bottom row: informational configuration). The results are averaged over 50 independent runs. (Contrast with figures in Fig. 2.1.)

the chosen parameters outperforms the baselines, and the gaps get larger as click feedback gets noisier. The results also show that Self-Sparring is severely affected by the level of noise. This is because Self-Sparring estimates the Borda score [112] of a ranker and, in our experiments, the noisier click feedback is, the closer the Borda scores are to each other, making it harder for Self-Sparring to identify the winner.

Results on the Yahoo dataset disagree with results on the MSLR dataset. On the Yahoo dataset, MergeDTS is affected more severely by the level of noise than Self-Sparring. This is because of the existence of uninformative rankers as stated in Assumption 2.1. In noisier configurations, the gaps between uninformative and informative rankers are smaller, which results in the long time of comparisons for MergeDTS to eliminate the uninformative rankers. Comparing those uninformative rankers leads to high regret.

In summary, the performance of MergeDTS is largely affected when the gaps between rankers are small, which is consistent with our theoretical findings.

#### 2.6.4 Cycle experiment

We address **RQ1.4** by running the algorithms that we consider on the Cycle and Cycle2 datasets introduced in Section 2.5.2. Particularly, we have already observed that Self-Sparring performs well in some cases (see the above experiments and results), but we argue that Self-Sparring may perform poorly when a dueling bandit problem contains cyclic preference relationships. This has been identified as a point of grave concern

## 2. Effective Large-Scale Online Ranker Evaluation

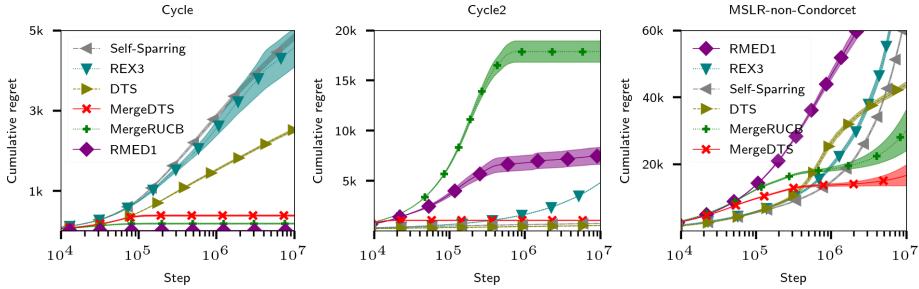


Figure 2.3: Cumulative regret in cyclic and non-Condorcet setups. The results are averaged over 100 independent runs.

in online evaluation [128]. Therefore, in this section we assess how dueling bandit algorithms behave when a dueling bandit problem contains cycles.

In this section we conduct experiments for 10 million steps and repeat 100 times since Merge-style algorithms converge to the Condorcet ranker within less than 1 million steps and running longer only increases the gaps between MergeDTS and baselines.

For the Cycle dataset (the first plot in Fig. 2.3), the cumulative regret of Self-Sparring is an order of magnitude higher than that of MergeDTS, although it performs well in some cases (see the above experiments). As we discussed in Section 2.5.2, Self-Sparring chooses rankers based on their Borda scores and when the Borda scores of different arms become close to each other as in the Cycle dataset, Self-Sparring may perform poorly. Also, we notice that when the gaps in Borda scores of the Condorcet winner and other rankers are large, Self-Sparring performs well, as shown in the middle plot in Fig. 2.3.

Other than Self-Sparring, we also notice that the other baselines performs quite differently on the two cyclic configurations. In the harder configuration of the two, the Cycle dataset, only RMED1 and MergeRUCB outperform MergeDTS. RMED1 excludes rankers from consideration based on relative preferences between two rankers. And, in the Cycle dataset, the preferences between suboptimal rankers are large. Thus, RMED1 can easily exclude a ranker based on its relative comparison to another suboptimal ranker. For the Cycle2 dataset, where the relative preferences between two rankers are small, RMED1 performs worse than MergeDTS.

MergeRUCB also slightly outperforms MergeDTS on the Cycle dataset. This can be explained as follows. In the Cycle dataset, the preference gap between the Condorcet winner and suboptimal rankers is small (i.e., 0.01), while the gaps between suboptimal rankers are relatively large (i.e., 1.0). Under this setup, MergeDTS tends to use the Condorcet winner to eliminate suboptimal rankers in the final stage. On the other hand, MergeRUCB eliminates a ranker by another ranker who beats it with the largest probability. So, MergeDTS requires more comparisons to eliminate suboptimal rankers than MergeRUCB. However, the gap between MergeRUCB and MergeDTS is small.

### 2.6.5 Beyond the Condorcet assumption

To answer **RQ1.5**, we evaluate MergeDTS on the MSLR-non-Condorcet dataset that does not contain a Condorcet winner. Instead, the dataset contains two Copeland winners and this dueling bandit setup is called the Copeland dueling bandit [115, 129]. The Copeland winner is selected by the Copeland score  $\zeta_i = \frac{1}{K-1} \sum_{k \neq i} \mathbb{1}(p_{ik} > 1/2)$  that measures the number of rankers beaten by ranker  $a_i$ . The Copeland winner is defined as  $\zeta^* = \max_{1 \leq i \leq K} \zeta_i$ . In the MSLR-non-Condorcet dataset, each Copeland winner beats 30 other rankers. Specifically, one of the Copeland winners beats the other one but is beaten by a suboptimal ranker. In the Copeland dueling bandit setup, regret is computed differently from the Condorcet dueling bandit setup. Given a pair of rankers  $(a_i, a_j)$ , regret at step  $t$  is computed as:

$$r_t = \zeta^* - 0.5(\zeta_i + \zeta_j). \quad (2.10)$$

Among the considered algorithms, only DTS can solve the Copeland dueling bandit problem and is the state-of-the-art Copeland dueling bandit algorithm. We conduct the experiment for 10 million steps with which DTS converges to the Copeland winners. And we run each algorithm 100 times independently. The results are shown in the last plot in Fig. 2.3.

MergeDTS has the lowest cumulative regret. However, in our experiments, we find that MergeDTS eliminates the two Copeland winners one time out of 100 individual repeats. In the other 99 repeats, we find that MergeDTS eliminates one of the two existing winners, which may not be ideal in practice. Note that we evaluate MergeDTS in a relatively easy setup, where only two Copeland winners are considered. For more complicated setups, where more than two Copeland winners are considered or the Copeland winners are beaten by several suboptimal rankers, we speculate that MergeDTS can fail more frequently. In our experiments, we do not evaluate MergeDTS in the more complicated setups, because MergeDTS is designed for the Condorcet dueling bandits and is only guaranteed to work under the Condorcet assumption. The answer to **RQ1.5** is that MergeDTS may perform well for some easy setups that go beyond the Condorcet assumption without any guarantees.

### 2.6.6 Parameter sensitivity

We answer **RQ1.6** and analyze the parameter sensitivity of MergeDTS using the setup described in Section 2.5.6. Since MergeDTS converges to the Condorcet winner within 10 million steps, we conduct the experiments with 10 million steps and repeat 100 times. Recall that we conduct the experiments in the MSLR-Navigational setup. The results are reported in Fig. 2.4. We also report the standard errors in the plots.

The left plot in Fig. 2.4 shows the effect of the exploration parameter  $\alpha$  on the performance of MergeDTS. First, lowering  $\alpha$  can significantly increase the performance, e.g., the cumulative regret for  $\alpha = 0.8^4$  is about one third of the reward for  $\alpha = 1.0$  (which is close to the theoretically optimal value  $\alpha = 1.01$ ). Second, as we decrease  $\alpha$ , the number of failures increases, where a failure is an event that MergeDTS eliminates the Condorcet winner: with  $\alpha = \{0.8^9, 0.8^8, 0.8^7\}$  we observe 10, 4, 1 failures, respectively, and, thus, the cumulative regret increases linearly w.r.t.  $T$ . Since

## 2. Effective Large-Scale Online Ranker Evaluation

---

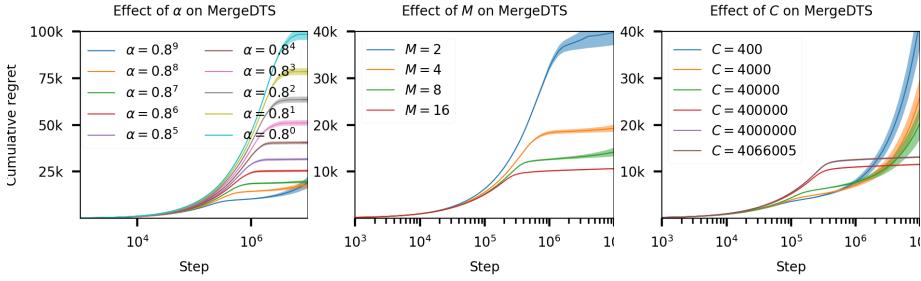


Figure 2.4: Effect of the parameters  $\alpha$ ,  $M$ , and  $C$  on the performance of MergeDTS in the MSLR-Navigational setup. The results are averaged over 100 independent runs. The shaded areas are  $\pm$  standard error.

in practice we do not want to eliminate the best ranker, we choose  $\alpha = 0.8^6 \approx 0.2621$  in our experiments.

The middle plot in Fig. 2.4 shows the effect of the batch size  $M$ . The larger the batch size, the lower the regret. This can be explained as follows. The DTS-based strategy uses the full local knowledge in a batch to choose the best ranker. A larger batch size  $M$  provides more knowledge to MergeDTS to make decisions, which leads to a better choice of rankers. But the time complexity of MergeDTS is  $O(TM^2)$ , i.e., quadratic in the batch size. Thus, for realistic scenarios we cannot increase  $M$  indefinitely. We choose  $M = 16$  as a tradeoff between effectiveness (cumulative regret) and efficiency (running time).

The right plot in Fig. 2.4 shows the dependency of MergeDTS on  $C$ . Similarly to the effect of  $\alpha$ , lower values of  $C$  lead to lower regret, but also to a larger number of failures.  $C = 4\,000\,000$  is the lowest value that does not lead to any failures, so we choose it in our experiments.

In summary, the theoretical constraints on the parameters of MergeDTS are rather conservative. There is a range of values for the key parameters  $\alpha$ ,  $M$  and  $C$ , where the theoretical guarantees fail to hold, but where MergeDTS performs better than it would if we were to constrain ourselves only to values permitted by theory.

## 2.7 Conclusion

---

In this chapter, we have studied the large-scale online ranker evaluation problem under the Condorcet assumption, which can be formalized as a  $K$ -armed dueling bandit problem. We have answered **RQ1** by introducing a scalable version of the state-of-the-art Double Thompson Sampling algorithm, which we call MergeDTS. Our experiments have shown that, by choosing the parameter values outside of the theoretical regime, MergeDTS is considerably more efficient than DTS in terms of computational complexity, and that it significantly outperforms the large-scale state-of-the-art algorithm MergeRUCB. Furthermore, we have demonstrated the robustness of MergeDTS when dealing with difficult dueling bandit problems containing cycles among the arms. Lastly, we have shown that the performance of MergeDTS is guaranteed if the parameter values

fall within the theoretical regime.

Several interesting directions for future work arise from this chapter: (1) In our experiments, we have shown that there is a large gap between theory and practice. It will be interesting to study this gap and provide a tighter theoretical bound. (2) We only study dueling bandits in this chapter. We believe that it is interesting to study a generalization of MergeDTS, as well as the theoretical analysis presented here, to the case of online ranker evaluation tasks with a multi-dueling setup. (3) Since multi-dueling bandits also compare multiple rankers at each step based on relative feedback, it is an interesting direction to compare dueling bandits to multi-dueling bandits in the large-scale setup. (4) We suspect that the UCB-based elimination utilized in MergeDTS is too conservative, it might be that more recent minimum empirical divergence based techniques [62] may be leveraged to speed up the elimination of the rankers. (5) The feature rankers are chosen as arms in our experiments. A more interesting and realistic way of choosing arms is to use well trained learning to rank algorithms.



# 3

## Safe Online Learning to Re-Rank

In this chapter, we answer the following research question:

**RQ2** How to achieve safe online learning to re-rank?

Particularly, we study the problem of safe online learning to re-rank, where user feedback is used to improve the quality of displayed lists. We propose BubbleRank, a bandit algorithm for safe re-ranking that combines the strengths of both the offline and online settings. The algorithm starts with an initial base list and improves it online by gradually exchanging higher-ranked less attractive items for lower-ranked more attractive items. We prove an upper bound on the  $n$ -step regret of BubbleRank that degrades gracefully with the quality of the initial base list. Our theoretical findings are supported by extensive experiments on a large-scale real-world click dataset.

### 3.1 Introduction

---

Learning to rank (LTR) is an important problem in many application domains, such as information retrieval, ad placement, and recommender systems [84]. More generally, LTR arises in any situation where multiple items, such as web pages, are presented to users. It is particularly relevant when the diversity of users makes it hard to decide which item should be presented to a specific user [99, 120].

A traditional approach to LTR is offline learning of rankers from either relevance labels created by judges [97] or user interactions [54, 87]. Recent experimental results [131] shows that such rankers, even in a highly-optimized search engine, can be improved by online LTR with exploration. Exploration is the key component in multi-armed bandit algorithms [9]. Many such algorithms have been proposed recently for online LTR in specific user-behavior models [58, 65, 70], the so-called *click models* [23]. Compared to earlier online LTR algorithms [99], these click model-based algorithms gain in statistical efficiency while giving up on generality. Empirical results indicate that click model-based algorithms are likely to be beneficial in practice.

Yet, existing algorithms for online LTR in click models are impractical for at least three reasons. First, an actual model of user behavior is typically unknown. This problem was initially addressed by Zoghi et al. [132]. They showed that the list of

---

This chapter was published as [77].

items in the descending order of relevance is optimal in several click models and proposed BatchRank for learning it. Then Lattimore et al. [72] built upon this work and proposed TopRank, which is the state-of-the-art online LTR algorithm. Second, these algorithms lack safety constraints and explore aggressively by placing potentially irrelevant items at high positions, which may significantly degrade user experience [114]. A third and related problem is that the algorithms are not well suited for so-called *warm start* scenarios [113], where the offline-trained production ranker already generates a good list, which only needs to be safely improved. Warm-starting an online LTR algorithm is challenging since existing posterior sampling algorithms, such as Thompson sampling [111], require item-level priors while only list-level priors are available practically.

We make the following contributions. First, motivated by the exploration scheme of Radlinski and Joachims [98], we propose a bandit algorithm for online LTR that addresses all three issues mentioned above. The proposed algorithm gradually improves upon an initial base list by exchanging higher-ranked less attractive items for lower-ranked more attractive items. The algorithm resembles bubble sort [26], and therefore we call it BubbleRank. Second, we prove an upper bound on the  $n$ -step regret of BubbleRank. The bound reflects the behavior of BubbleRank: worse initial base lists lead to a higher regret. Third, we define our safety constraint, which is based on incorrectly-ordered item pairs in the ranked list, and prove that BubbleRank never violates this constraint with a high probability. Finally, we evaluate BubbleRank extensively on a large-scale real-world click dataset.

## 3.2 Background

---

This section introduces our online learning problem. We first review click models [23] and then introduce a stochastic click bandit [132], a learning to rank framework for multiple click models.

The following notation is used in the rest of the chapter. We denote  $\{1, \dots, n\}$  by  $[n]$ . For any sets  $A$  and  $B$ , we denote by  $A^B$  the set of all vectors whose entries are indexed by  $B$  and take values from  $A$ . We use boldface letters to denote random variables.

### 3.2.1 Click models

A *click model* is a model of how a user clicks on a list of documents. We refer to the documents as *items* and denote the universe of all items by  $\mathcal{D} = [L]$ . The user is presented a *ranked list*, an ordered list of  $K$  documents out of  $L$ . We denote this list by  $\mathcal{R} \in \Pi_K(\mathcal{D})$ , where  $\Pi_K(\mathcal{D})$  is the set of all  $K$ -tuples with distinct items from  $\mathcal{D}$ . We denote by  $\mathcal{R}(k)$  the item at position  $k$  in  $\mathcal{R}$ ; and by  $\mathcal{R}^{-1}(i)$  the position of item  $i$  in  $\mathcal{R}$ , if item  $i$  is in  $\mathcal{R}$ .

Many click models are parameterized by *item-dependent attraction probabilities*  $\alpha \in [0, 1]^L$ , where  $\alpha(i)$  is the *attraction probability* of item  $i$ . We discuss the two most fundamental click models below.

In the *cascade model* (CM) [27], the user scans list  $\mathcal{R}$  from the first item  $\mathcal{R}(1)$  to the last  $\mathcal{R}(K)$ . If item  $\mathcal{R}(k)$  is *attractive*, the user *clicks* on it and does not examine the remaining items. If item  $\mathcal{R}(k)$  is not attractive, the user *examines* item  $\mathcal{R}(k+1)$ . The first item  $\mathcal{R}(1)$  is examined with probability one. Therefore, the expected number of clicks is equal to the probability of clicking on any item, and is  $r(\mathcal{R}) = \sum_{k=1}^K \chi(\mathcal{R}, k) \alpha(\mathcal{R}(k))$ , where  $\chi(\mathcal{R}, k) = \prod_{i=1}^{k-1} (1 - \alpha(\mathcal{R}(i)))$  is the examination probability of position  $k$  in list  $\mathcal{R}$ .

In the *position-based model* (PBM) [100], the probability of clicking on an item depends on both its identity and position. Therefore, in addition to item-dependent attraction probabilities  $\alpha$ , the PBM is parameterized by  $K$  *position-dependent examination probabilities*  $\chi \in [0, 1]^K$ , where  $\chi(k)$  is the examination probability of position  $k$ . The user interacts with list  $\mathcal{R}$  as follows. The user *examines* position  $k \in [K]$  with probability  $\chi(k)$  and then *clicks* on item  $\mathcal{R}(k)$  at that position with probability  $\alpha(\mathcal{R}(k))$ . Therefore, the expected number of clicks on list  $\mathcal{R}$  is  $r(\mathcal{R}) = \sum_{k=1}^K \chi(k) \alpha(\mathcal{R}(k))$ .

CM and PBM are similar models, because the probability of clicking factors into item and position dependent factors. Therefore, both in the CM and PBM, under the assumption that  $\chi(1) \geq \dots \geq \chi(K)$ , the expected number of clicks is maximized by listing the  $K$  most attractive items in descending order of their attraction. More precisely, the most clicked list is

$$\mathcal{R}^* = (1, \dots, K) \quad (3.1)$$

when  $\alpha(1) \geq \dots \geq \alpha(L)$ . Therefore, perhaps not surprisingly, the problem of learning the optimal list in both models can be viewed as the same problem, a stochastic click bandit [132].

### 3.2.2 Stochastic click bandit

An instance of a *stochastic click bandit* [132] is a tuple  $(K, L, P_\alpha, P_\chi)$ , where  $K \leq L$  is the number of positions,  $L$  is the number of items,  $P_\alpha$  is a distribution over binary attraction vectors  $\{0, 1\}^L$ , and  $P_\chi$  is a distribution over binary examination matrices  $\{0, 1\}^{\Pi_K(\mathcal{D}) \times K}$ .

The learning agent interacts with the stochastic click bandit as follows. At time  $t$ , it chooses a list  $\mathcal{R}_t \in \Pi_K(\mathcal{D})$ , which depends on its history up to time  $t$ , and then observes *clicks*  $\mathbf{c}_t \in \{0, 1\}^K$  on all positions in  $\mathcal{R}_t$ . A position is clicked if and only if it is examined and the item at that position is attractive. More specifically, for any  $k \in [K]$ ,

$$\mathbf{c}_t(k) = \mathbf{X}_t(\mathcal{R}_t, k) \mathbf{A}_t(\mathcal{R}_t(k)), \quad (3.2)$$

where  $\mathbf{X}_t \in \{0, 1\}^{\Pi_K(\mathcal{D}) \times K}$  and  $\mathbf{X}_t(\mathcal{R}, k)$  is the *examination indicator* of position  $k$  in list  $\mathcal{R} \in \Pi_K(\mathcal{D})$  at time  $t$ ; and  $\mathbf{A}_t \in \{0, 1\}^L$  and  $\mathbf{A}_t(i)$  is the *attraction indicator* of item  $i$  at time  $t$ . Both  $\mathbf{A}_t$  and  $\mathbf{X}_t$  are stochastic and drawn i.i.d. from  $P_\alpha \otimes P_\chi$ .

The key assumption that allows learning in this model is that the attraction of any item is independent of the examination of its position. In particular, for any list

### 3. Safe Online Learning to Re-Rank

---

$\mathcal{R} \in \Pi_K(\mathcal{D})$  and position  $k \in [K]$ ,

$$\mathbb{E} [\mathbf{c}_t(k) | \mathcal{R}_t = \mathcal{R}] = \chi(\mathcal{R}, k) \alpha(\mathcal{R}(k)), \quad (3.3)$$

where  $\alpha = \mathbb{E} [\mathbf{A}_t]$  and  $\alpha(i)$  is the *attraction probability* of item  $i$ ; and  $\chi = \mathbb{E} [\mathbf{X}_t]$  and  $\chi(\mathcal{R}, k)$  is the *examination probability* of position  $k$  in  $\mathcal{R}$ . Note that the above independence assumption is in expectation only. We do not require that the clicks are independent of the position or other displayed items.

The *expected reward* at time  $t$  is the expected number of clicks at time  $t$ . Based on our independence assumption,  $\sum_{k=1}^K \mathbb{E} [\mathbf{c}_t(k)] = r(\mathcal{R}_t, \alpha, \chi)$ , where  $r(\mathcal{R}, A, X) = \sum_{k=1}^K X(\mathcal{R}, k) A(\mathcal{R}(k))$  for any  $\mathcal{R} \in \Pi_K(\mathcal{D})$ ,  $A \in [0, 1]^L$ , and  $X \in [0, 1]^{\Pi_K(\mathcal{D}) \times K}$ . The learning agent maximizes the expected number of clicks in  $n$  steps. This problem can be equivalently viewed as minimizing the *expected cumulative regret* in  $n$  steps, which we define as

$$R(n) = \sum_{t=1}^n \mathbb{E} \left[ \max_{\mathcal{R} \in \Pi_K(\mathcal{D})} r(\mathcal{R}, \alpha, \chi) - r(\mathcal{R}_t, \alpha, \chi) \right]. \quad (3.4)$$

## 3.3 Online Learning to Re-Rank

---

Multi-stage ranking is widely used in production ranking systems [22, 57, 83], with the re-ranking stage at the very end [22]. In the re-ranking stage, a relatively small number of items, typically 10–20, are re-ranked. One reason for re-ranking is that offline rankers are typically trained to minimize the average loss across a large number of queries. Therefore, they perform well on very frequent queries and poorly on infrequent queries. On moderately frequent queries, the so-called *torso queries*, their performance varies. As torso queries are sufficiently frequent, an online algorithm can be used to re-rank so as to optimize their value, such as the number of clicks [131].

We propose an online algorithm that addresses the above problem and adaptively re-ranks a list of items generated by a production ranker with the goal of placing more attractive items at higher positions. We study a non-contextual variant of the problem, where we re-rank a small number of items in a single query. Generalization across queries and items is an interesting direction for future work. We follow the setting in Section 3.2.2, except that  $\mathcal{D} = [K]$ . Despite these simplifying assumptions, our learning problem remains a challenge. In particular, the attraction of items is only observed through clicks in Eq. (3.2), which are affected by other items in the list.

### 3.3.1 Algorithm

Our algorithm is presented in Algorithm 4. The algorithm gradually improves upon an *initial base list*  $\mathcal{R}_0$  by “bubbling up” more attractive items. Therefore, we refer to it as BubbleRank. BubbleRank determines more attractive items by randomly exchanging neighboring items. If the lower-ranked item is found to be more attractive, the items are permanently exchanged and never randomly exchanged again. If the lower-ranked item is found to be less attractive, the items are never randomly exchanged again. We describe BubbleRank in detail below.

**Algorithm 4** BubbleRank

---

**Input:** Initial list  $\mathcal{R}_0$  over  $[K]$ 

- 1:  $\forall i, j \in [K] : s_0(i, j) \leftarrow 0, n_0(i, j) \leftarrow 0$
  - 2:  $\mathcal{R}_1 \leftarrow \mathcal{R}_0$
  - 3: **for**  $t = 1, \dots, n$  **do**
  - 4:    $h \leftarrow t \bmod 2$
  - 5:    $\mathcal{R}_t \leftarrow \bar{\mathcal{R}}_t$
  - 6:   **for**  $k = 1, \dots, \lfloor (K - h)/2 \rfloor$  **do**
  - 7:      $i \leftarrow \mathcal{R}_t(2k - 1 + h), j \leftarrow \mathcal{R}_t(2k + h)$
  - 8:     **if**  $s_{t-1}(i, j) \leq 2\sqrt{n_{t-1}(i, j) \log(1/\delta)}$  **then**
  - 9:       Randomly exchange items  $\mathcal{R}_t(2k - 1 + h)$  and  $\mathcal{R}_t(2k + h)$  in list  $\mathcal{R}_t$
  - 10:   Display list  $\mathcal{R}_t$  and observe clicks  $c_t \in \{0, 1\}^K$
  - 11:    $s_t \leftarrow s_{t-1}, n_t \leftarrow n_{t-1}$
  - 12:   **for**  $k = 1, \dots, \lfloor (K - h)/2 \rfloor$  **do**
  - 13:      $i \leftarrow \mathcal{R}_t(2k - 1 + h), j \leftarrow \mathcal{R}_t(2k + h)$
  - 14:     **if**  $|c_t(2k - 1 + h) - c_t(2k + h)| = 1$  **then**
  - 15:        $s_t(i, j) \leftarrow s_t(i, j) + c_t(2k - 1 + h) - c_t(2k + h)$
  - 16:        $n_t(i, j) \leftarrow n_t(i, j) + 1$
  - 17:        $s_t(j, i) \leftarrow s_t(j, i) + c_t(2k + h) - c_t(2k - 1 + h)$
  - 18:        $n_t(j, i) \leftarrow n_t(j, i) + 1$
  - 19:    $\mathcal{R}_{t+1} \leftarrow \bar{\mathcal{R}}_t$
  - 20:   **for**  $k = 1, \dots, K - 1$  **do**
  - 21:      $i \leftarrow \bar{\mathcal{R}}_{t+1}(k), j \leftarrow \bar{\mathcal{R}}_{t+1}(k + 1)$
  - 22:     **if**  $s_t(j, i) > 2\sqrt{n_t(j, i) \log(1/\delta)}$  **then**
  - 23:       Exchange items  $\bar{\mathcal{R}}_{t+1}(k)$  and  $\bar{\mathcal{R}}_{t+1}(k + 1)$  in list  $\bar{\mathcal{R}}_{t+1}$
- 

BubbleRank maintains a *base list*  $\bar{\mathcal{R}}_t$  at each time  $t$ . From the viewpoint of BubbleRank, this is the best list at time  $t$ . The list is initialized by the initial base list  $\mathcal{R}_0$  (line 2). At time  $t$ , BubbleRank permutes  $\bar{\mathcal{R}}_t$  into a *displayed list*  $\mathcal{R}_t$  (lines 5–9). Two kinds of permutations are employed. If  $t$  is odd and so  $h = 0$ , the items at positions 1 and 2, 3 and 4, and so on, are randomly exchanged. If  $t$  is even and so  $h = 1$ , the items at positions 2 and 3, 4 and 5, and so on are randomly exchanged. The items are exchanged only if BubbleRank is uncertain regarding which item is more attractive (line 8).

The list  $\mathcal{R}_t$  is displayed and BubbleRank gets feedback (line 10). Then it updates its statistics (lines 11–18). For any exchanged items  $i$  and  $j$ , if item  $i$  is clicked and item  $j$  is not, the belief that  $i$  is more attractive than  $j$ ,  $s_t(i, j)$ , increases; and the belief that  $j$  is more attractive than  $i$ ,  $s_t(j, i)$ , decreases. The number of observations,  $n_t(i, j)$  and  $n_t(j, i)$ , increases. These statistics are updated only if one of the items is clicked (line 14), not both.

At the end of time  $t$ , the base list  $\bar{\mathcal{R}}_t$  is improved (lines 19–23). More specifically, if any lower-ranked item  $j$  is found to be more attractive than its higher-ranked neighbor

$i$  (line 22), the items are permanently exchanged in the next base list  $\bar{\mathcal{R}}_{t+1}$ .

A notable property of BubbleRank is that it explores safely, since any item in the displayed list  $\mathcal{R}_t$  is at most one position away from its position in the base list  $\bar{\mathcal{R}}_t$ . Moreover, any base list improves upon the initial base list  $\mathcal{R}_0$ , because it is obtained by bubbling up more attractive items with a high confidence. We make this notion of safety more precise in Section 3.4.2.

## 3.4 Theoretical Analysis

---

In this section, we provide theoretical guarantees on the performance of BubbleRank, by bounding the  $n$ -step regret in Eq. (3.4).

The content is organized as follows. In Section 3.4.1, we present our upper bound on the  $n$ -step regret of BubbleRank, together with our assumptions. In Section 3.4.2, we prove that BubbleRank is safe. In Section 3.4.3, we discuss our theoretical results. The regret bound is proved in Section 3.4.4. Our technical lemmas are stated and proved in Section 3.7.

### 3.4.1 Regret bound

Before we present our result, we introduce our assumptions<sup>1</sup> and complexity metrics.

**Assumption 3.1.** *For any lists  $\mathcal{R}, \mathcal{R}' \in \Pi_K(\mathcal{D})$  and positions  $k, \ell \in [K]$  such that  $k < \ell$ :*

- A1.  $r(\mathcal{R}, \alpha, \chi) \leq r(\mathcal{R}^*, \alpha, \chi)$ , where  $\mathcal{R}^*$  is defined in Eq. (3.1);
- A2.  $\{\mathcal{R}(1), \dots, \mathcal{R}(k-1)\} = \{\mathcal{R}'(1), \dots, \mathcal{R}'(k-1)\} \implies \chi(\mathcal{R}, k) = \chi(\mathcal{R}', k)$ ;
- A3.  $\chi(\mathcal{R}, k) \geq \chi(\mathcal{R}, \ell)$ ;
- A4. *If  $\mathcal{R}$  and  $\mathcal{R}'$  differ only in that the items at positions  $k$  and  $\ell$  are exchanged, then  $\alpha(\mathcal{R}(k)) \leq \alpha(\mathcal{R}(\ell)) \iff \chi(\mathcal{R}, \ell) \geq \chi(\mathcal{R}', \ell)$ ; and*
- A5.  $\chi(\mathcal{R}, k) \geq \chi(\mathcal{R}^*, k)$ .

The above assumptions hold in the CM. In the PBM, they hold when the examination probability decreases with the position.

Our assumptions can be interpreted as follows. Assumption A1 says that the list of items in the descending order of attraction probabilities is optimal. Assumption A2 says that the examination probability of any position depends only on the identities of higher-ranked items. Assumption A3 says that a higher position is at least as examined as a lower position. Assumption A4 says that a higher-ranked item is less attractive if and only if it increases the examination of a lower position. Assumption A5 says that any position is examined the least in the optimal list.

---

<sup>1</sup>Our assumptions are slightly weaker than those of Zoghi et al. [132]. For instance, Assumption A2 is on the probability of examination. Zoghi et al. [132] make this assumption on the realization of examination.

To simplify our exposition, we assume that  $\alpha(1) > \dots > \alpha(K) > 0$ . Let  $\chi_{\max} = \chi(\mathcal{R}^*, 1)$  denote the *maximum examination probability*,  $\chi_{\min} = \chi(\mathcal{R}^*, K)$  denote the *minimum examination probability*, and

$$\Delta_{\min} = \min_{k \in [K-1]} \alpha(k) - \alpha(k+1)$$

be the *minimum gap*. Then the  $n$ -step regret of BubbleRank can be bounded as follows.

**Theorem 3.1.** *In any stochastic click bandit that satisfies Assumption 3.1, and for any  $\delta \in (0, 1)$ , the expected  $n$ -step regret of BubbleRank is bounded as*

$$R(n) \leq 180K \frac{\chi_{\max}}{\chi_{\min}} \frac{K-1+2|\mathcal{V}_0|}{\Delta_{\min}} \log(1/\delta) + \delta^{\frac{1}{2}} K^3 n^2.$$

### 3.4.2 Safety

Let

$$\mathcal{V}(\mathcal{R}) = \{(i, j) \in [K]^2 : i < j, \mathcal{R}^{-1}(i) > \mathcal{R}^{-1}(j)\} \quad (3.5)$$

be the set of *incorrectly-ordered item pairs* in list  $\mathcal{R}$ . Then our algorithm is safe in the following sense.

**Lemma 3.1.** *Let*

$$\mathcal{V}_0 = \mathcal{V}(\mathcal{R}_0) \quad (3.6)$$

*be the incorrectly-ordered item pairs in the initial base list  $\mathcal{R}_0$ . Then the number of incorrectly-ordered item pairs in any displayed list  $\mathcal{R}_t$  is at most  $|\mathcal{V}_0| + K/2$ , that is  $|\mathcal{V}(\mathcal{R}_t)| \leq |\mathcal{V}_0| + K/2$  holds uniformly over time with probability of at least  $1 - \delta^{\frac{1}{2}} K^2 n$ .*

*Proof.* Our claim follows from two observations. First, by the design of BubbleRank, any displayed list  $\mathcal{R}_t$  contains at most  $K/2$  item pairs that are ordered differently from its base list  $\bar{\mathcal{R}}_t$ . Second, no base list  $\bar{\mathcal{R}}_t$  contains more incorrectly-ordered item pairs than  $\mathcal{R}_0$  with a high probability. In particular, under event  $\mathcal{E}$  in Lemma 3.8, any change in the base list (line 23 of BubbleRank) reduces the number of incorrectly-ordered item pairs by one. In Lemma 3.8, we prove that  $\mathbb{P}(\mathcal{E}) \geq 1 - \delta^{\frac{1}{2}} K^2 n$ .  $\square$

### 3.4.3 Discussion

Our upper bound on the  $n$ -step regret of BubbleRank (Theorem 3.1) is  $O(\Delta_{\min}^{-1} \log n)$  for  $\delta = n^{-4}$ . This dependence is considered to be optimal in gap-dependent bounds. Our gap  $\Delta_{\min}$  is the minimum difference in the attraction probabilities of items, and reflects the hardness of sorting the items by their attraction probabilities. This sorting problem is equivalent to the problem of learning  $\mathcal{R}^*$ . So, a gap like  $\Delta_{\min}$  is expected, and is the same as that in [132].

Our regret bound is notable because it reflects two key characteristics of BubbleRank. First, the bound is linear in the number of incorrectly-ordered item pairs in the initial

base list  $\mathcal{R}_0$ . This suggests that BubbleRank should have lower regret when initialized with a better list of items. We validate this dependence empirically in Section 3.5. In many domains, such lists exist and are produced by existing ranking policies. They only need to be safely improved.

Second, the bound is  $O(\chi_{\max}\chi_{\min}^{-1})$ , where  $\chi_{\max}$  and  $\chi_{\min}$  are the maximum and minimum examination probabilities, respectively. In Section 3.5.4, we show that this dependence can be observed in problems where most attractive items are placed at infrequently examined positions. This limitation is intrinsic to BubbleRank, because attractive lower-ranked items cannot be placed at higher positions unless they are observed to be attractive at lower, potentially infrequently examined, positions.

The safety constraint of BubbleRank is stated in Lemma 3.1. For  $\delta = n^{-4}$ , as discussed above, BubbleRank becomes a rather safe algorithm, and is unlikely to display any list with more than  $K/2$  incorrectly-ordered item pairs than the initial base list  $\mathcal{R}_0$ . More precisely,  $|\mathcal{V}(\mathcal{R}_t)| \leq |\mathcal{V}_0| + K/2$  holds uniformly over time with probability of at least  $1 - K^2/n$ . This safety feature of BubbleRank is confirmed by our experiments in Section 3.5.3.

The above discussion assumes that the time horizon  $n$  is known. However, in practice, this is not always possible. We can extend BubbleRank to the setting of an unknown time horizon by using the so-called *doubling trick* [18, Section 2.3]. Let  $n$  be the estimated horizon. Then at time  $n + 1$ ,  $\bar{\mathcal{R}}_{n+1}$  is set to  $\mathcal{R}_0$  and  $n$  is doubled. The statistics do not need to be reset.

BubbleRank is computationally efficient. The time complexity of BubbleRank is linear in the number of time steps and in each step  $O(K)$  operations are required.

In this chapter, we focus on re-ranking. But BubbleRank can be extended to the full ranking problem as follows. Define  $s_t(i, j)$  and  $n_t(i, j)$  for all item pairs  $(i, j)$ . For even (odd)  $K$  at odd (even) time steps, select a random item below position  $K$  that has not been shown to be worse than the item at position  $K$ , and swap these items with probability 0.5. The item that is not displayed gets feedback 0. The rest of BubbleRank remains the same. This algorithm can be analyzed in the same way as BubbleRank.

#### 3.4.4 Proof of Theorem 3.1

In Lemma 3.8, we establish that there exists a favorable event  $\mathcal{E}$  that holds with probability  $1 - \delta^{\frac{1}{2}}K^2n$ , when all beliefs  $s_t(i, j)$  are at most  $2\sqrt{n_t(i, j)\log(1/\delta)}$  from their respective means, uniformly for  $i < j$  and  $t \in [n]$ . Since the maximum  $n$ -step regret is  $Kn$ , we get that

$$R(n) \leq \mathbb{E} \left[ \hat{R}(n)I\mathcal{E} \right] + \delta^{\frac{1}{2}}K^3n^2,$$

where  $\hat{R}(n) = \sum_{t=1}^n r(\mathcal{R}^*, \alpha, \chi) - r(\mathcal{R}_t, \alpha, \chi)$ . We bound  $\hat{R}(n)$  next. For this, let

$$\begin{aligned} \mathcal{P}_t &= \{(i, j) \in [K]^2 : i < j, |\bar{\mathcal{R}}_t^{-1}(i) - \bar{\mathcal{R}}_t^{-1}(j)| = 1 \\ &\quad s_{t-1}(i, j) \leq 2\sqrt{n_{t-1}(i, j)\log(1/\delta)}\} \end{aligned}$$

be the set of potentially randomized item pairs at time  $t$ . Then, by Lemma 3.4 on event  $\mathcal{E}$ , which bounds the regret of list  $\mathcal{R}_t$  with the difference in the attraction probabilities

of items  $(i, j) \in \mathcal{P}_t$ , we have that

$$\hat{R}(n) \leq 3K\chi_{\max} \sum_{i=1}^K \sum_{j=i+1}^K \sum_{t=1}^n (\alpha(i) - \alpha(j)) I(i, j) \in \mathcal{P}_t.$$

Now note that for any randomized  $(i, j) \in \mathcal{P}_t$  at time  $t$ ,

$$\begin{aligned} \chi_{\min}(\alpha(i) - \alpha(j)) &\leq \mathbb{E}_{t-1} [\mathbf{c}_t(\mathcal{R}_t^{-1}(i)) - \mathbf{c}_t(\mathcal{R}_t^{-1}(j))] \\ &= \mathbb{E}_{t-1} [\mathbf{s}_t(i, j) - \mathbf{s}_{t-1}(i, j)], \end{aligned}$$

where  $\mathbb{E}_{t-1}[\cdot]$  is the expectation conditioned on the history up to time  $t$ ,  $\mathcal{R}_1, \mathbf{c}_1, \dots, \mathcal{R}_{t-1}, \mathbf{c}_{t-1}$ . The inequality is from  $\alpha(i) \geq \alpha(j)$ , and Assumptions Item A2 and Item A4. The above two inequalities yield

$$\begin{aligned} \hat{R}(n) &\leq 6K \frac{\chi_{\max}}{\chi_{\min}} \sum_{i=1}^K \sum_{j=i+1}^K \sum_{t=1}^n \mathbb{E}_{t-1} [\mathbf{s}_t(i, j) - \mathbf{s}_{t-1}(i, j)] I(i, j) \in \mathcal{P}_t \\ &\leq 6K \frac{\chi_{\max}}{\chi_{\min}} \sum_{i=1}^K \sum_{j=i+1}^K I \exists t \in [n] : (i, j) \in \mathcal{P}_t \times \\ &\quad \sum_{t=1}^n \mathbb{E}_{t-1} [\mathbf{s}_t(i, j) - \mathbf{s}_{t-1}(i, j)], \end{aligned}$$

where the extra factor of two is because BubbleRank randomizes any pair of items  $(i, j) \in \mathcal{P}_t$  at least once in any two consecutive steps. Moreover, for any  $i < j$  on event  $\mathcal{E}$ ,

$$\sum_{t=1}^n (\mathbf{s}_t(i, j) - \mathbf{s}_{t-1}(i, j)) = \mathbf{s}_n(i, j) \leq 15 \frac{\alpha(i) + \alpha(j)}{\alpha(i) - \alpha(j)} \log(1/\delta) \leq \frac{30}{\Delta_{\min}} \log(1/\delta).$$

The first inequality is by Lemma 3.6, which establishes that the maximum difference in clicks of any randomized pair of items is bounded. After that, the better item is found and the pair of items is never randomized again. The last inequality is by  $\alpha(i) + \alpha(j) \leq 2$  and  $\alpha(i) - \alpha(j) \geq \Delta_{\min}$ . Now we chain the above two inequalities and get that

$$\hat{R}(n) \leq 180K \frac{\chi_{\max}}{\chi_{\min}} \frac{1}{\Delta_{\min}} \log(1/\delta) \times \sum_{i=1}^K \sum_{j=i+1}^K I \exists t \in [n] : (i, j) \in \mathcal{P}_t.$$

Finally, let  $\mathcal{P} = \bigcup_{t \in [n]} \mathcal{P}_t$ . Then, on event  $\mathcal{E}$ ,  $|\mathcal{P}| \leq K - 1 + 2|\mathcal{V}_0|$ . This follows from the design of BubbleRank (Lemma 3.5) and completes the proof.  $\square$

## 3.5 Experimental Results

We conduct four experiments to evaluate BubbleRank. In Section 3.5.1, we describe our experimental setup. In Section 3.5.2, we report the regret of compared algorithms,

### 3. Safe Online Learning to Re-Rank

which measures the rate of convergence to the optimal list in hindsight. In Section 3.5.3, we validate the safety of BubbleRank. In Section 3.5.4, we validate the tightness of the regret bound in Theorem 3.1. We report the *Normalized Discounted Cumulative Gain* (NDCG) of compared algorithms, which measures the quality of displayed lists, in Section 3.5.5.

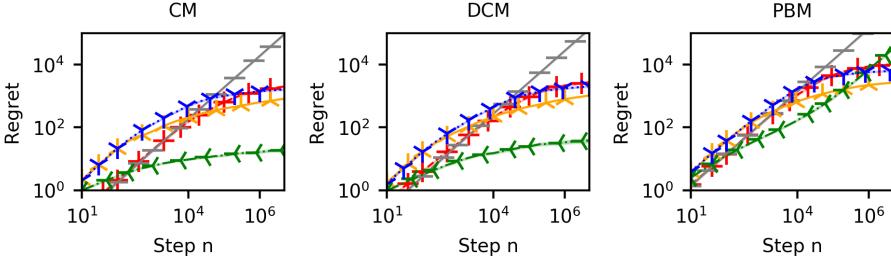


Figure 3.1: The  $n$ -step regret of BubbleRank (red), CascadeKL-UCB (green), BatchRank (blue), TopRank (orange), and Baseline (grey) in the CM, DCM, and PBM in up to 5 million steps. Lower is better. The results are averaged over all 100 queries and 10 runs per query. The shaded regions represent standard errors of our estimates.

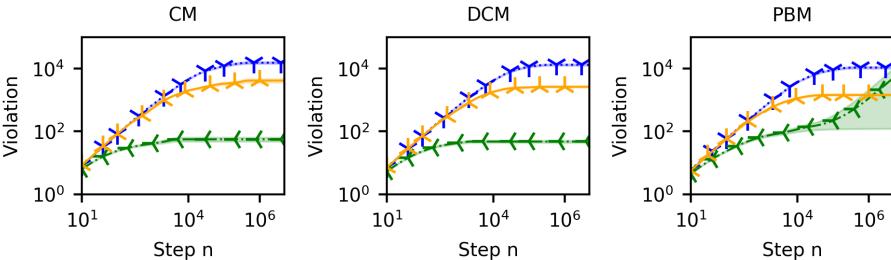


Figure 3.2: The  $n$ -step violation of the safety constraint of BubbleRank by CascadeKL-UCB (green), BatchRank (blue), and TopRank (orange) in the CM, DCM, and PBM in up to 5 million steps. Lower is better. The shaded regions represent standard errors of our estimates.

#### 3.5.1 Experimental setup

We evaluate BubbleRank on the *Yandex* click dataset.<sup>2</sup> The dataset contains user search sessions from the log of the *Yandex* search engine. It is the largest publicly available dataset containing user clicks, with more than 30 million search sessions. Each session contains at least one search query together with 10 ranked items.

<sup>2</sup>[https://academy.yandex.ru/events/data\\_analysis/relpred2011](https://academy.yandex.ru/events/data_analysis/relpred2011)

We preprocess the dataset as in [132]. In particular, we randomly select 100 frequent search queries, and then learn the parameters of three click models using the PyClick<sup>3</sup> package: CM and PBM, described in Section 3.2.1, as well as the *dependent click model* (DCM) [36].

The DCM is an extension of the CM [27] where each position  $k$  is associated with an abandonment probability  $v(k)$ . When the user clicks on an item at position  $k$ , the user stops scanning the list with probability  $v(k)$ . Therefore, the DCM can model multiple clicks. Following the work in [58], we incorporate abandonment into our definition of reward for DCM and define it as the number of abandonment clicks. The *abandonment click* is a click after which a user stops browsing the list, and each time step contains at most one abandonment click. So, the expected reward for DCM equals the probability of abandonment clicks, which is computed as follows:

$$r(\mathcal{R}, \alpha, \chi) = \sum_{k=1}^K \chi(\mathcal{R}, k) v(k) \alpha(\mathcal{R}(k)),$$

$$\chi(\mathcal{R}, k) = \prod_{i=1}^{k-1} (1 - v(i)) \alpha(\mathcal{R}(i))$$

is the examination probability of position  $k$  in list  $\mathcal{R}$ . A high reward means a user stops the search session because of clicking on an item with high attraction probability.

The learned CM, DCM, and PBM are used to simulate user click feedback. We experiment with multiple click models to show the robustness of BubbleRank to multiple models of user feedback.

For each query, we choose 10 items. The number of positions is equal to the number of items,  $K = L = 10$ . The objective of our re-ranking problem is to place 5 most attractive items in descending order of attractiveness at the 5 highest positions, as in [132]. The performance of BubbleRank and our baselines is also measured only at the top 5 positions.

BubbleRank is compared to three baselines Cascade- KL-UCB [65], BatchRank [132], and TopRank [72]. The former is near optimal in the CM [65], but can have linear regret in other click models. Note that linear regret arises when CascadeKL-UCB erroneously converges to a suboptimal ranked list. BatchRank and TopRank can learn the optimal list  $\mathcal{R}^*$  in a wide range of click models, including the CM, DCM, and PBM. However, they can perform poorly in early stages of learning because they randomly shuffles displayed lists to average out the position bias. All experiments are run for 5 million steps, after which at least two algorithms converge to the optimal ranked list.

In the *Yandex* dataset, each query is associated with many different ranked lists, due to the presence of various personalization features of the production ranker. We take the most frequent ranked list for each query as the initial base list  $\mathcal{R}_0$  in BubbleRank, since we assume that the most frequent ranked list is what the production ranker would produce in the absence of any personalization. We also compare BubbleRank to a production baseline, called **Baseline**, where the initial list  $\mathcal{R}_0$  is applied for  $n$  steps.

---

<sup>3</sup><https://github.com/markovi/PyClick>

### 3.5.2 Results with regret

In the first experiment, we compare BubbleRank to CascadeKL-UCB, BatchRank, and TopRank in the CM, DCM, and PBM of all 100 queries. Among them, TopRank is the state-of-the-art online LTR algorithm in multiple click models. We evaluate these algorithms by their cumulative regret, which is defined in Eq. (3.4), at the top 5 positions. The regret, a measure of convergence, is a widely-used metric in the bandit literature [9, 58, 65, 132]. In the CM and PBM, the regret is the cumulative loss in clicks when a sequence of learned lists is compared to the optimal list in hindsight. In the DCM, the regret is the cumulative loss in abandonment clicks. We also report the regret of Baseline.

Our results are reported in Fig. 3.1. We observe that the regret of Baseline grows linearly with time  $n$ , which means that it is not optimal on average. CascadeKL-UCB learns  $\mathcal{R}^*$  quickly in both the CM and DCM, but has linear regret in the PBM. This is expected since CascadeKL-UCB is designed for the CM, and the DCM is an extension of the CM. As for the PBM, which is beyond the modeling assumptions of CascadeKL-UCB, there is no guarantee on the performance of CascadeKL-UCB. BubbleRank, BatchRank, and TopRank can learn in all three click models. Compared to BatchRank and TopRank, BubbleRank has a higher regret in 5 million steps. However, in earlier steps, BubbleRank has a lower regret than BatchRank and TopRank, as it takes advantage of the initial base list  $\mathcal{R}_0$ . In general, these results show that BubbleRank converges to the optimal list slower than BatchRank and TopRank. This is expected because BubbleRank is designed to be a safe algorithm, and only learns better lists by exchanging neighboring items in the base list.

### 3.5.3 Safety results

In the previous experiment, BubbleRank does not learn as fast as CascadeKL-UCB, BatchRank, and TopRank, because of the additional safety constraint in Lemma 3.1. The constraint is that BubbleRank is unlikely to display any list with more than  $K/2$  incorrectly-ordered item pairs than the initial base list  $\mathcal{R}_0$ . More precisely,  $|\mathcal{V}(\mathcal{R}_t)| \leq |\mathcal{V}(\mathcal{R}_0)| + K/2$  holds uniformly over time with probability of at least  $1 - K^2/n$  for  $\delta = n^{-4}$  (Section 3.4.3), where  $\mathcal{V}$  is defined in Eq. (3.5). In the second experiment, we answer the question how often do BubbleRank, CascadeKL-UCB, BatchRank, and TopRank violate this constraint empirically. We define the *safety constraint violation in  $n$  steps* as

$$V(n) = \sum_{t=1}^n I|\mathcal{V}(\mathcal{R}_t)| > |\mathcal{V}(\mathcal{R}_0)| + K/2, \quad (3.7)$$

where  $\mathcal{R}_t$  is the displayed list at time  $t$ .

We report the  $n$ -step safety constraint violation of CascadeKL-UCB, BatchRank, and TopRank in Fig. 3.2. We do not include results of BubbleRank since BubbleRank never violates the constraint in our experiments. We observe that the safety constraint violations of CascadeKL-UCB in the first 100 steps are  $24.12 \pm 0.76$ ,  $23.33 \pm 0.90$ , and  $23.63 \pm 0.96$  in the CM, DCM, and PBM, respectively. Translating this to a search scenario, CascadeKL-UCB may show unsafe results, which are significantly worse than

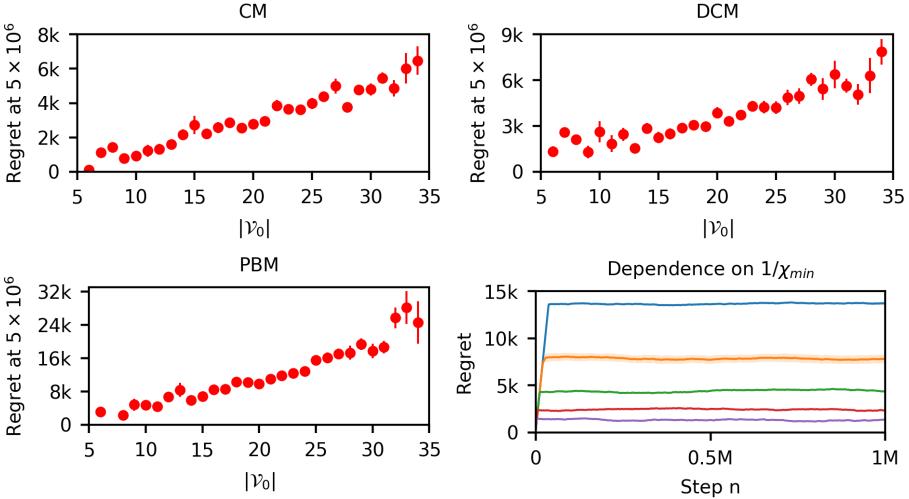


Figure 3.3: Regret of BubbleRank as a function of the number of incorrectly-ordered item pairs  $|\mathcal{V}_0|$ , and the minimal examination probability  $\chi_{\min}$ . In the bottom-right plot, the purple, red, green, orange, and blue colors represent  $\chi_{\min}$  equals  $0.5, 0.5^2, 0.5^3, 0.5^4$ , and  $0.5^5$ , respectively. The shaded regions represent standard errors of our estimates.

the initial base list  $\mathcal{R}_0$ , and may hurt user experience, more than 20% of search sessions in the first 100 steps. Even worse, the violations of CascadeKL-UCB grow linearly with time in the PBM. The safety issues of BatchRank, and TopRank are more severe than that of CascadeKL-UCB. More precisely, the violations of BatchRank in the first 100 steps are  $83.01 \pm 0.56$ ,  $71.89 \pm 0.92$ , and  $59.63 \pm 0.98$  in the CM, DCM, and PBM, respectively. And the violations of TopRank are  $83.56 \pm 0.70$ ,  $71.47 \pm 1.07$ , and  $57.00 \pm 1.24$  in the CM, DCM, and PBM, respectively. Note that the performance of TopRank is close to that of BatchRank in the first 100 steps since they both require the ranked lists to be randomly shuffled during the initial stages. Thus, BatchRank, and TopRank would frequently hurt the user experience during the early stages of learning.

To conclude, BubbleRank learns without violating its safety constraint, while CascadeKL-UCB, BatchRank, and TopRank violate the constraint frequently. Together with results in Section 3.5.2, BubbleRank is a safe algorithm but, to satisfy the safety constraint, it compromises the performance and learns slower than BatchRank and TopRank. In Section 3.5.5, we compare BubbleRank to baselines in NDCG and show that BubbleRank converges to the optimal lists in hindsight.

### 3.5.4 Sanity check on regret bound

We prove an upper bound on the  $n$ -step regret of BubbleRank in Theorem 3.1. In comparison to the upper bounds of CascadeKL-UCB [65] and BatchRank [132], we have two new problem-specific constants:  $|\mathcal{V}_0|$  and  $1/\chi_{\min}$ . In this section, we show that these constants are intrinsic to the behavior of BubbleRank.

### 3. Safe Online Learning to Re-Rank

---

We first study how the number of incorrectly-ordered item pairs in the initial base list  $\mathcal{R}_0$ ,  $|\mathcal{V}_0|$ , impacts the regret of BubbleRank. We choose 10 random initial base lists  $\mathcal{V}_0$  in each of our 100 queries and plot the regret of BubbleRank as a function of  $|\mathcal{V}_0|$ . Our results are shown in Fig. 3.3. We observe that the regret of BubbleRank is linear in  $|\mathcal{V}_0|$  in the CM, DCM, and PBM. This is the same dependence as in our regret bound (Theorem 3.1).

We then study the impact of the minimum examination probability  $\chi_{\min}$  on the regret of BubbleRank. We experiment with a synthetic PBM with 10 items, which is parameterized by  $\alpha = (0.9, 0.5, \dots, 0.5)$  and  $\chi = (0.9, \dots, 0.9, 0.5^i, 0.5^i)$  for  $i \geq 1$ . The most attractive item is placed at the last position in  $\mathcal{R}_0$ ,  $\mathcal{R}_0 = (2, \dots, K-1, 1)$ . Since this position is examined with probability  $0.5^i$ , we expect the regret to double when  $i$  increases by one. We experiment with  $i \in [5]$  in Fig. 3.3 and observe this trend in 1 million steps. This confirms that the dependence on  $1/\chi_{\min}$  in Theorem 3.1 is generally unavoidable.

#### 3.5.5 Results with NDCG

In this section, we report the NDCG of compared algorithms, which measures the quality of displayed lists. Since CascadeKL-UCB fails in the PBM and we focus on learning from all types of click feedback, we leave out CascadeKL-UCB from this section.

In the first two experiments, we evaluate algorithms by their regret in Eq. (3.4) and safety constraint violation in Eq. (3.7). Neither of these metrics measure the quality of ranked lists directly. In this experiment, we report the per-step NDCG@5 of BubbleRank, BatchRank, TopRank, and Baseline (Fig. 3.4), which directly measures the quality of ranked lists and is widely used in the LTR literature [6, 52]. Since the Yandex dataset does not contain relevance scores for all query-item pairs, we take the attraction probability of the item in its learned click model as a proxy to its relevance score. This substitution is natural since our goal is to rank items in descending order of their attraction probabilities [23]. We compute the NDCG@5 of a ranked list  $\mathcal{R}$  as

$$NDCG@5(\mathcal{R}) = \frac{DCG@5(\mathcal{R})}{DCG@5(\mathcal{R}^*)}, \quad DCG@5(\mathcal{R}) = \sum_{k=1}^5 \frac{\alpha(\mathcal{R}(k))}{\log_2(k+1)},$$

where  $\mathcal{R}^*$  is the optimal list and  $\alpha(\mathcal{R}(k))$  is the attraction probability of the  $k$ -th item in list  $\mathcal{R}$ . This is a standard evaluation metric, and is used in TREC evaluation benchmarks [6], for instance. It measures the discounted gain over the attraction probabilities of the 5 highest ranked items in list  $\mathcal{R}$ , which is normalized by the DCG@5 of  $\mathcal{R}^*$ .

In Fig. 3.4, we observe that Baseline has good NDCG@5 scores in all click models. Yet there is still room for improvement. BubbleRank, BatchRank, and TopRank have similar NDCG@5 scores after 5 million steps. But BubbleRank starts with NDCG@5 close to that of Baseline, while BatchRank and TopRank start with lists with very low NDCG@5.

These results validate our earlier findings. As in Section 3.5.2, we observe that BubbleRank converges to the optimal list in hindsight, since its NDCG@5 approaches

- As in Section 3.5.3, we observe that BubbleRank is safe, since its NDCG@5 is never much worse than that of Baseline.

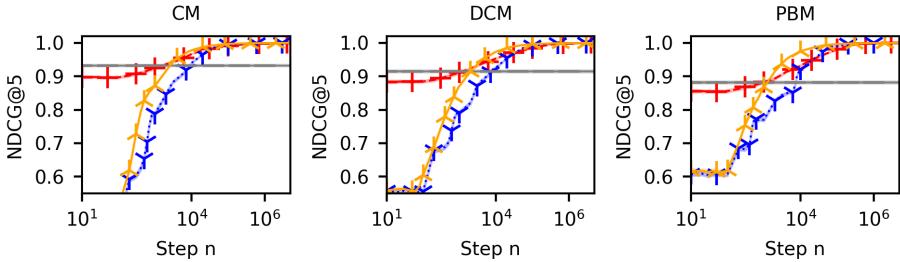


Figure 3.4: The per-step NDCG@5 of BubbleRank (red), BatchRank (blue), TopRank (orange), and Baseline (grey) in the CM, DCM, and PBM in up to 5 million steps. Higher is better. The shaded regions represent standard errors of our estimates.

## 3.6 Related Work

Online LTR via click feedback has been mainly studied in two approaches: under specific click models [25, 58, 65, 66, 70, 133]; or without a particular assumption on click models [72, 99, 107, 132]. Algorithms from the first group efficiently learn optimal rankings in the their considered click models but do not have guarantees beyond their specific click models. Algorithms from the second group, on the other hand, learn the optimal rankings in a broader class of click models. TopRank [72] is the state-of-the-art of the second group, which has the regret of  $O(K^2 \log(n))$  in our re-ranking setup, that is  $L = K$ . BubbleRank also belongs to the second group and the regret of BubbleRank is comparable to that of TopRank given a good initial list, when  $|\mathcal{V}_0| = O(K)$ . However, unlike BubbleRank, TopRank and all the previous algorithms do not consider safety. They explore aggressively in the initial steps and may display irrelevant items at high positions, which may then hurt user experiences [114].

Our safety problem is related to the warm start problem [108]. Contextual bandits [2, 80, 89] deal with a broader class of models than we do and are used to address the warm start problem. But they are limited to small action sets, and thus unsuitable for the ranking setup that we consider in this chapter.

The warm start LTR has been studied in multiple papers [42, 113, 118], where the goal is to use an online algorithm to fine tune the results generated by an offline-trained ranker. In these papers, different methods for learning prior distributions of Thompson sampling based online LTR algorithms from offline datasets have been proposed. However, these methods have the following drawbacks. First, the offline data may not well align with user preferences [131], which may result in a biased prior assumption. Second, grid search with online A/B tests may alleviate this and find a proper prior assumption [113], but the online A/B test requires additional costs in terms of user experience. Third, there is no safety constraint in these methods. Even with carefully picked priors, they may recommend irrelevant items to users, e.g., new

items with little prior knowledge. In contrast, BubbleRank starts from the production ranked list and learns under the safety constraint. Thus, BubbleRank gets rid of these drawbacks.

Another related line of work are conservative bandits [59, 117]. In conservative bandits, the learned policy is safe in the sense that its expected *cumulative* reward is at least  $1 - \alpha$  fraction of that of the baseline policy with high probability. This notion of safety is less stringent than that in our work (Section 3.4.2). In particular, our notion of safety is *per-step*, in the sense that any displayed list is only slightly worse than the initial base list with a high probability. We do not compare to conservative bandits in our experiments because existing algorithms for conservative bandits require the action space to be small. The actions in our problem are ranked lists, and their number is exponential in  $K$ .

## 3.7 Lemmas

---

**Lemma 3.2.** *Let  $\mathcal{R}$  be any list over  $[K]$ . Let*

$$\Delta(\mathcal{R}) = \sum_{k=1}^{K-1} I\alpha(\mathcal{R}(k+1)) - \alpha(\mathcal{R}(k)) > 0 \times (\alpha(\mathcal{R}(k+1)) - \alpha(\mathcal{R}(k))) \quad (3.8)$$

*be the attraction gap of list  $\mathcal{R}$ . Then the expected regret of  $\mathcal{R}$  is bounded as*

$$\sum_{k=1}^K (\chi(\mathcal{R}^*, k)\alpha(k) - \chi(\mathcal{R}, k)\alpha(\mathcal{R}(k))) \leq K\chi_{\max}\Delta(\mathcal{R}).$$

*Proof.* Fix position  $k \in [K]$ . Then

$$\begin{aligned} \chi(\mathcal{R}^*, k)\alpha(k) - \chi(\mathcal{R}, k)\alpha(\mathcal{R}(k)) &\leq \chi(\mathcal{R}^*, k)(\alpha(k) - \alpha(\mathcal{R}(k))) \\ &\leq \chi_{\max}(\alpha(k) - \alpha(\mathcal{R}(k))), \end{aligned}$$

where the first inequality follows from the fact that the examination probability of any position is the lowest in the optimal list (Assumption A5) and the second inequality follows from the definition of  $\chi_{\max}$ . In the rest of the proof, we bound  $\alpha(k) - \alpha(\mathcal{R}(k))$ . We consider three cases. First, let  $\alpha(\mathcal{R}(k)) \geq \alpha(k)$ . Then  $\alpha(k) - \alpha(\mathcal{R}(k)) \leq 0$  and can be trivially bounded by  $\Delta(\mathcal{R})$ . Second, let  $\alpha(\mathcal{R}(k)) < \alpha(k)$  and  $\pi(k) > k$ , where  $\pi(k)$  is the position of item  $k$  in list  $\mathcal{R}$ . Then

$$\begin{aligned} \alpha(k) - \alpha(\mathcal{R}(k)) &= \alpha(\mathcal{R}(\pi(k))) - \alpha(\mathcal{R}(k)) \\ &\leq \sum_{i=k}^{\pi(k)-1} I\alpha(\mathcal{R}(i+1)) - \alpha(\mathcal{R}(i)) > 0\alpha(\mathcal{R}(i+1)) - \alpha(\mathcal{R}(i)). \end{aligned}$$

From the definition of  $\Delta(\mathcal{R})$ , this quantity is bounded from above by  $\Delta(\mathcal{R})$ . Finally, let  $\alpha(\mathcal{R}(k)) < \alpha(k)$  and  $\pi(k) < k$ . This implies that there exists an item at a lower

position than  $k$ ,  $j > k$ , such that  $\alpha(\mathcal{R}(j)) \geq \alpha(k)$ . Then

$$\begin{aligned} \alpha(k) - \alpha(\mathcal{R}(k)) &\leq \alpha(\mathcal{R}(j)) - \alpha(\mathcal{R}(k)) \\ &\leq \sum_{i=k}^{j-1} I\alpha(\mathcal{R}(i+1)) - \alpha(\mathcal{R}(i)) > 0(\alpha(\mathcal{R}(i+1)) - \alpha(\mathcal{R}(i))). \end{aligned}$$

From the definition of  $\Delta(\mathcal{R})$ , this quantity is bounded from above by  $\Delta(\mathcal{R})$ . This concludes the proof.  $\square$

**Lemma 3.3.** *Let*

$$\mathcal{P}_t =$$

$$\{(i, j) \in [K]^2 : i < j, |\bar{\mathcal{R}}_t^{-1}(i) - \bar{\mathcal{R}}_t^{-1}(j)| = 1, s_{t-1}(i, j) \leq 2\sqrt{n_{t-1}(i, j) \log(1/\delta)}\}$$

be the set of potentially randomized item pairs at time  $t$  and  $\Delta_t = \max_{\mathcal{R}_t} \Delta(\mathcal{R}_t)$  be the maximum attraction gap of any list  $\mathcal{R}_t$ , where  $\Delta(\mathcal{R}_t)$  is defined in Eq. (3.8). Then on event  $\mathcal{E}$  in Lemma 3.8,

$$\Delta_t \leq 3 \sum_{i=1}^K \sum_{j=i+1}^K I(i, j) \in \mathcal{P}_t(\alpha(i) - \alpha(j))$$

holds at any time  $t \in [n]$ .

*Proof.* Fix list  $\mathcal{R}_t$  and position  $k \in [K-1]$ . Let  $i', i, j, j'$  be items at positions  $k-1, k, k+1, k+2$  in  $\bar{\mathcal{R}}_t$ . If  $k=1$ , let  $i'=i$ ; and if  $k=K-1$ , let  $j'=j$ . We consider two cases.

First, suppose that the permutation at time  $t$  is such that  $i$  and  $j$  could be exchanged. Then

$$\begin{aligned} &\alpha(\mathcal{R}_t^{-1}(k+1)) - \alpha(\mathcal{R}_t^{-1}(k)) \\ &\leq I(\min\{i, j\}, \max\{i, j\}) \in \mathcal{P}_t(\alpha(\min\{i, j\}) - \alpha(\max\{i, j\})) \end{aligned}$$

holds on event  $\mathcal{E}$  by the design of BubbleRank. More specifically,  $(\min\{i, j\}, \max\{i, j\}) \notin \mathcal{P}_t$  implies that  $\alpha(\mathcal{R}_t^{-1}(k+1)) - \alpha(\mathcal{R}_t^{-1}(k)) \leq 0$ .

Second, suppose that the permutation at time  $t$  is such that  $i$  and  $i'$  could be exchanged,  $j$  and  $j'$  could be exchanged, or both. Then

$$\begin{aligned} &\alpha(\mathcal{R}_t^{-1}(k+1)) - \alpha(\mathcal{R}_t^{-1}(k)) \leq \\ &I(\min\{i, i'\}, \max\{i, i'\}) \in \mathcal{P}_t(\alpha(\min\{i, i'\}) - \alpha(\max\{i, i'\})) + \\ &\alpha(j) - \alpha(i) + \\ &I(\min\{j, j'\}, \max\{j, j'\}) \in \mathcal{P}_t(\alpha(\min\{j, j'\}) - \alpha(\max\{j, j'\})) \end{aligned}$$

holds by the same argument as in the first case. Also note that

$$\alpha(j) - \alpha(i) \leq I(\min\{i, j\}, \max\{i, j\}) \in \mathcal{P}_t(\alpha(\min\{i, j\}) - \alpha(\max\{i, j\}))$$

### 3. Safe Online Learning to Re-Rank

---

holds on event  $\mathcal{E}$  by the design of BubbleRank. Therefore, for any position  $k \in [K - 1]$  in both above cases,

$$\begin{aligned} & \alpha(\mathcal{R}_t^{-1}(k + 1)) - \alpha(\mathcal{R}_t^{-1}(k)) \leq \\ & \sum_{\ell=k-1}^{k+1} I\left(\min\left\{\bar{\mathcal{R}}_t^{-1}(\ell), \bar{\mathcal{R}}_t^{-1}(\ell + 1)\right\}, \max\left\{\bar{\mathcal{R}}_t^{-1}(\ell), \bar{\mathcal{R}}_t^{-1}(\ell + 1)\right\}\right) \in \mathcal{P}_t \times \\ & \left(\alpha\left(\min\left\{\bar{\mathcal{R}}_t^{-1}(\ell), \bar{\mathcal{R}}_t^{-1}(\ell + 1)\right\}\right) - \alpha\left(\max\left\{\bar{\mathcal{R}}_t^{-1}(\ell), \bar{\mathcal{R}}_t^{-1}(\ell + 1)\right\}\right)\right). \end{aligned}$$

Now we sum over all positions and note that each pair of  $\bar{\mathcal{R}}_t^{-1}(\ell)$  and  $\bar{\mathcal{R}}_t^{-1}(\ell + 1)$  appears on the right-hand side at most three times, in any list  $\mathcal{R}_t$ . This concludes our proof.  $\square$

**Lemma 3.4.** *Let  $\mathcal{P}_t$  be defined as in Lemma 3.3. Then on event  $\mathcal{E}$  in Lemma 3.8,*

$$\begin{aligned} & \sum_{k=1}^K (\chi(\mathcal{R}^*, k) \alpha(k) - \chi(\mathcal{R}_t, k) \alpha(\mathcal{R}_t(k))) \\ & \leq 3K \chi_{\max} \sum_{i=1}^K \sum_{j=i+1}^K I(i, j) \in \mathcal{P}_t(\alpha(i) - \alpha(j)) \end{aligned}$$

holds at any time  $t \in [n]$ .

*Proof.* A direct consequence of Lemmas 3.2 and 3.3.  $\square$

**Lemma 3.5.** *Let  $\mathcal{P}_t$  be defined as in Lemma 3.3,  $\mathcal{P} = \bigcup_{t=1}^n \mathcal{P}_t$ , and  $\mathcal{V}_0$  be defined as in Eq. (3.6). Then on event  $\mathcal{E}$  in Lemma 3.8,*

$$|\mathcal{P}| \leq K - 1 + 2 |\mathcal{V}_0| .$$

*Proof.* From the design of BubbleRank,  $|\mathcal{P}_1| = K - 1$ . The set of randomized item pairs grows only if the base list in BubbleRank changes. When this happens, the number of incorrectly-ordered item pairs decreases by one, on event  $\mathcal{E}$ , and the set of randomized item pairs increases by at most two pairs. This event occurs at most  $|\mathcal{V}_0|$  times. This concludes our proof.  $\square$

**Lemma 3.6.** *For any items  $i$  and  $j$  such that  $i < j$ ,*

$$s_n(i, j) \leq 15 \frac{\alpha(i) + \alpha(j)}{\alpha(i) - \alpha(j)} \log(1/\delta)$$

on event  $\mathcal{E}$  in Lemma 3.8.

*Proof.* To simplify notation, let  $s_t = s_t(i, j)$  and  $n_t = n_t(i, j)$ . The proof has two parts. First, suppose that  $s_t \leq 2\sqrt{n_t \log(1/\delta)}$  holds at all times  $t \in [n]$ . Then from this assumption and on event  $\mathcal{E}$  in Lemma 3.8,

$$\frac{\alpha(i) - \alpha(j)}{\alpha(i) + \alpha(j)} n_t - 2\sqrt{n_t \log(1/\delta)} \leq s_t \leq 2\sqrt{n_t \log(1/\delta)} .$$

This implies that

$$\mathbf{n}_t \leq \left[ 4 \frac{\alpha(i) + \alpha(j)}{\alpha(i) - \alpha(j)} \right]^2 \log(1/\delta)$$

at any time  $t$ , and in turn that

$$s_t \leq 2\sqrt{\mathbf{n}_t \log(1/\delta)} \leq 8 \frac{\alpha(i) + \alpha(j)}{\alpha(i) - \alpha(j)} \log(1/\delta)$$

at any time  $t$ . Our claim follows from setting  $t = n$ .

Now suppose that  $s_t \leq 2\sqrt{\mathbf{n}_t \log(1/\delta)}$  does not hold at all times  $t \in [n]$ . Let  $\tau$  be the first time when  $s_\tau > 2\sqrt{\mathbf{n}_\tau \log(1/\delta)}$ . Then from the definition of  $\tau$  and on event  $\mathcal{E}$  in Lemma 3.8,

$$\begin{aligned} \frac{\alpha(i) - \alpha(j)}{\alpha(i) + \alpha(j)} \mathbf{n}_\tau - 2\sqrt{\mathbf{n}_\tau \log(1/\delta)} &\leq s_\tau \leq s_{\tau-1} + 1 \\ &\leq 2\sqrt{\mathbf{n}_\tau \log(1/\delta)} + 1 \\ &\leq 3\sqrt{\mathbf{n}_\tau \log(1/\delta)}, \end{aligned}$$

where the last inequality holds for any  $\delta \leq 1/e$ . This implies that

$$\mathbf{n}_\tau \leq \left[ 5 \frac{\alpha(i) + \alpha(j)}{\alpha(i) - \alpha(j)} \right]^2 \log(1/\delta),$$

and in turn that

$$s_\tau \leq 3\sqrt{\mathbf{n}_\tau \log(1/\delta)} \leq 15 \frac{\alpha(i) + \alpha(j)}{\alpha(i) - \alpha(j)} \log(1/\delta).$$

Now note that  $s_t = s_\tau$  for any  $t > \tau$ , from the design of BubbleRank. This concludes our proof.  $\square$

For some  $\mathcal{F}_t = \sigma(\mathcal{R}_1, \mathbf{c}_1, \dots, \mathcal{R}_t, \mathbf{c}_t)$ -measurable event  $A$ , let  $\mathbb{P}_t(A) = \mathbb{P}(A \mid \mathcal{F}_t)$  be the conditional probability of  $A$  given history  $\mathcal{R}_1, \mathbf{c}_1, \dots, \mathcal{R}_t, \mathbf{c}_t$ . Let the corresponding conditional expectation operator be  $\mathbb{E}_t[\cdot]$ . Note that  $\mathcal{R}_t$  is  $\mathcal{F}_{t-1}$ -measurable.

**Lemma 3.7.** *Let  $i, j \in [K]$  be any items at consecutive positions in  $\mathcal{R}_t$  and*

$$\mathbf{z} = \mathbf{c}_t(\mathcal{R}_t^{-1}(i)) - \mathbf{c}_t(\mathcal{R}_t^{-1}(j)).$$

*Then, on the event that  $i$  and  $j$  are subject to randomization at time  $t$ ,*

$$\mathbb{E}_{t-1}[\mathbf{z} \mid \mathbf{z} \neq 0] \geq \frac{\alpha(i) - \alpha(j)}{\alpha(i) + \alpha(j)}$$

*when  $\alpha(i) > \alpha(j)$ , and  $\mathbb{E}_{t-1}[-\mathbf{z} \mid \mathbf{z} \neq 0] \leq 0$  when  $\alpha(i) < \alpha(j)$ .*

### 3. Safe Online Learning to Re-Rank

---

*Proof.* The first claim is proved as follows. From the definition of expectation and  $\mathbf{z} \in \{-1, 0, 1\}$ ,

$$\begin{aligned}\mathbb{E}_{t-1}[\mathbf{z} \mid \mathbf{z} \neq 0] &= \frac{\mathbb{P}_{t-1}(\mathbf{z} = 1, \mathbf{z} \neq 0) - \mathbb{P}_{t-1}(\mathbf{z} = -1, \mathbf{z} \neq 0)}{\mathbb{P}_{t-1}(\mathbf{z} \neq 0)} \\ &= \frac{\mathbb{P}_{t-1}(\mathbf{z} = 1) - \mathbb{P}_{t-1}(\mathbf{z} = -1)}{\mathbb{P}_{t-1}(\mathbf{z} \neq 0)} \\ &= \frac{\mathbb{E}_{t-1}[\mathbf{z}]}{\mathbb{P}_{t-1}(\mathbf{z} \neq 0)},\end{aligned}$$

where the last equality is a consequence of  $\mathbf{z} = 1 \implies \mathbf{z} \neq 0$  and that  $\mathbf{z} = -1 \implies \mathbf{z} \neq 0$ .

Let  $\chi_i = \mathbb{E}_{t-1}[\chi(\mathcal{R}_t, \mathcal{R}_t^{-1}(i))]$  and  $\chi_j = \mathbb{E}_{t-1}[\chi(\mathcal{R}_t, \mathcal{R}_t^{-1}(j))]$  denote the average examination probabilities of the positions with items  $i$  and  $j$ , respectively, in  $\mathcal{R}_t$ ; and consider the event that  $i$  and  $j$  are subject to randomization at time  $t$ . By Assumption A2, the values of  $\chi_i$  and  $\chi_j$  do not depend on the randomization of other parts of  $\bar{\mathcal{R}}_t$ , only on the positions of  $i$  and  $j$ . Then  $\chi_i \geq \chi_j$ ; from  $\alpha(i) > \alpha(j)$  and Assumption A4. Based on this fact,  $\mathbb{E}_{t-1}[\mathbf{z}]$  is bounded from below as

$$\mathbb{E}_{t-1}[\mathbf{z}] = \chi_i \alpha(i) - \chi_j \alpha(j) \geq \chi_i (\alpha(i) - \alpha(j)),$$

where the inequality is from  $\chi_i \geq \chi_j$ . Moreover,  $\mathbb{P}_{t-1}(\mathbf{z} \neq 0)$  is bounded from above as

$$\begin{aligned}\mathbb{P}_{t-1}(\mathbf{z} \neq 0) &= \mathbb{P}_{t-1}(\mathbf{z} = 1) + \mathbb{P}_{t-1}(\mathbf{z} = -1) \\ &\leq \chi_i \alpha(i) + \chi_j \alpha(j) \\ &\leq \chi_i (\alpha(i) + \alpha(j)),\end{aligned}$$

where the first inequality is from inequalities  $\mathbb{P}_{t-1}(\mathbf{z} = 1) \leq \chi_i \alpha(i)$  and  $\mathbb{P}_{t-1}(\mathbf{z} = -1) \leq \chi_j \alpha(j)$ , and the last inequality is from  $\chi_i \geq \chi_j$ .

Finally, we chain all above inequalities and get our first claim. The second claim follows from the observation that  $\mathbb{E}_{t-1}[-\mathbf{z} \mid \mathbf{z} \neq 0] = -\mathbb{E}_{t-1}[\mathbf{z} \mid \mathbf{z} \neq 0]$ .  $\square$

**Lemma 3.8.** Let  $S_1 = \{(i, j) \in [K]^2 : i < j\}$  and  $S_2 = \{(i, j) \in [K]^2 : i > j\}$ . Let

$$\begin{aligned}\mathcal{E}_{t,1} &= \left\{ \forall (i, j) \in S_1 : \frac{\alpha(i) - \alpha(j)}{\alpha(i) + \alpha(j)} \mathbf{n}_t(i, j) - 2\sqrt{\mathbf{n}_t(i, j) \log(1/\delta)} \leq \mathbf{s}_t(i, j) \right\}, \\ \mathcal{E}_{t,2} &= \left\{ \forall (i, j) \in S_2 : \mathbf{s}_t(i, j) \leq 2\sqrt{\mathbf{n}_t(i, j) \log(1/\delta)} \right\}.\end{aligned}$$

Let  $\mathcal{E} = \bigcap_{t \in [n]} (\mathcal{E}_{t,1} \cap \mathcal{E}_{t,2})$  and  $\bar{\mathcal{E}}$  be the complement of  $\mathcal{E}$ . Then  $\mathbb{P}(\bar{\mathcal{E}}) \leq \delta^{1/2} K^2 n$ .

*Proof.* First, we bound  $\mathbb{P}(\bar{\mathcal{E}}_{t,1})$ . Fix  $(i, j) \in S_1$ ,  $t \in [n]$ , and  $(\mathbf{n}_\ell(i, j))_{\ell=1}^t$ . Let  $\tau(m)$  be the time of observing item pair  $(i, j)$  for the  $m$ -th time,  $\tau(m) = \min\{\ell \in [t] : \mathbf{n}_\ell(i, j) = m\}$  for  $m \in [\mathbf{n}_t(i, j)]$ . Let  $\mathbf{z}_\ell = \mathbf{c}_\ell(\mathcal{R}_\ell^{-1}(i)) - \mathbf{c}_\ell(\mathcal{R}_\ell^{-1}(j))$ . Since  $(\mathbf{n}_\ell(i, j))_{\ell=1}^t$  is fixed, note that  $\mathbf{z}_\ell \neq 0$  if  $\ell = \tau(m)$  for some  $m \in [\mathbf{n}_t(i, j)]$ . Let  $\mathbf{X}_0 = 0$  and

$$\mathbf{X}_\ell = \sum_{\ell'=1}^{\ell} \mathbb{E}_{\tau(\ell')-1}[\mathbf{z}_{\tau(\ell')} \mid \mathbf{z}_{\tau(\ell')} \neq 0] - \mathbf{s}_{\tau(\ell)}(i, j)$$

for  $\ell \in [\mathbf{n}_t(i, j)]$ . Then  $(\mathbf{X}_\ell)_{\ell=1}^{\mathbf{n}_t(i, j)}$  is a martingale, because

$$\begin{aligned}\mathbf{X}_\ell - \mathbf{X}_{\ell-1} &= \mathbb{E}_{\tau(\ell)-1}[\mathbf{z}_{\tau(\ell)} \mid \mathbf{z}_{\tau(\ell)} \neq 0] - (\mathbf{s}_{\tau(\ell)}(i, j) - \mathbf{s}_{\tau(\ell-1)}(i, j)) \\ &= \mathbb{E}_{\tau(\ell)-1}[\mathbf{z}_{\tau(\ell)} \mid \mathbf{z}_{\tau(\ell)} \neq 0] - \mathbf{z}_{\tau(\ell)},\end{aligned}$$

where the last equality follows from the definition of  $\mathbf{s}_{\tau(\ell)}(i, j) - \mathbf{s}_{\tau(\ell-1)}(i, j)$ . Now we apply the Azuma-Hoeffding inequality and get that

$$P\left(\mathbf{X}_{\mathbf{n}_t(i, j)} - \mathbf{X}_0 \geq 2\sqrt{\mathbf{n}_t(i, j) \log(1/\delta)}\right) \leq \delta^{\frac{1}{2}}.$$

Moreover, from the definitions of  $\mathbf{X}_0$  and  $\mathbf{X}_{\mathbf{n}_t(i, j)}$ , and by Lemma 3.7, we have that

$$\begin{aligned}\delta^{\frac{1}{2}} &\geq P\left(\mathbf{X}_{\mathbf{n}_t(i, j)} - \mathbf{X}_0 \geq 2\sqrt{\mathbf{n}_t(i, j) \log(1/\delta)}\right) \\ &= P\left(\sum_{\ell'=1}^{\mathbf{n}_t(i, j)} \mathbb{E}_{\tau(\ell')-1}[\mathbf{z}_{\tau(\ell')} \mid \mathbf{z}_{\tau(\ell')} \neq 0] - \mathbf{s}_t(i, j) \geq 2\sqrt{\mathbf{n}_t(i, j) \log(1/\delta)}\right) \\ &\geq P\left(\frac{\alpha(i) - \alpha(j)}{\alpha(i) + \alpha(j)} \mathbf{n}_t(i, j) - \mathbf{s}_t(i, j) \geq 2\sqrt{\mathbf{n}_t(i, j) \log(1/\delta)}\right) \\ &= P\left(\frac{\alpha(i) - \alpha(j)}{\alpha(i) + \alpha(j)} \mathbf{n}_t(i, j) - 2\sqrt{\mathbf{n}_t(i, j) \log(1/\delta)} \geq \mathbf{s}_t(i, j)\right).\end{aligned}$$

The above inequality holds for any  $(\mathbf{n}_\ell(i, j))_{\ell=1}^t$ , and therefore also in expectation over  $(\mathbf{n}_\ell(i, j))_{\ell=1}^t$ . From the definition of  $\mathcal{E}_{t,1}$  and the union bound, we have  $\mathbb{P}(\overline{\mathcal{E}_{t,1}}) \leq \frac{1}{2}\delta^{\frac{1}{2}}K(K-1)$ .

The claim that  $\mathbb{P}(\overline{\mathcal{E}_{t,2}}) \leq \frac{1}{2}\delta^{\frac{1}{2}}K(K-1)$  is proved similarly, except that we use  $\mathbb{E}_{\tau(\ell)-1}[\mathbf{z}_{\tau(\ell)} \mid \mathbf{z}_{\tau(\ell)} \neq 0] \leq 0$ . From the definition of  $\overline{\mathcal{E}}$  and the union bound,

$$\mathbb{P}(\overline{\mathcal{E}}) \leq \sum_{t=1}^n \mathbb{P}(\overline{\mathcal{E}_{t,1}}) + \sum_{t=1}^n \mathbb{P}(\overline{\mathcal{E}_{t,2}}) \leq \delta^{\frac{1}{2}}K^2n.$$

This completes our proof.  $\square$

## 3.8 Conclusions

In this chapter, we have provided an answer to **RQ2** by combining the advantages of both offline and online Learning to Rank (LTR) algorithms. In particular, we fill a gap in the LTR literature by proposing BubbleRank, a re-ranking algorithm that gradually improves an initial base list, which we assume to be provided by an offline LTR approach. The improvements are learned from small perturbations of base lists, which are unlikely to degrade the user experience greatly. We prove a gap-dependent upper bound on the regret of BubbleRank and evaluate it on a large-scale click dataset from a commercial search engine.

We leave open several questions of interest. For instance, our chapter studies BubbleRank in the setting of re-ranking. Although we explain an approach of extending

### 3. Safe Online Learning to Re-Rank

---

BubbleRank to the general ranking setup in Section 3.4.3, we expect further experiments to validate this approach. Our general topic of interest are exploration schemes that are more conservative than those of existing online LTR methods. Existing methods are not very practical because they can explore highly irrelevant items at frequently examined positions.

# 4

## Cascade Non-Stationary Bandits

This chapter provides an answer to **RQ3**:

**RQ3** How to conduct online learning to rank when users change their preferences constantly?

We study the online learning to rank problem in a *non-stationary* environment where user preferences change abruptly at an unknown moment in time. We consider the problem of identifying the  $K$  most attractive items and propose *cascading non-stationary bandits*, an online learning variant of the *cascading model*, where a user browses a ranked list from top to bottom and clicks on the first attractive item. We propose two algorithms for solving this non-stationary problem: CascadeDUCB and CascadeSWUCB. We analyze their performance and derive gap-dependent upper bounds on the  $n$ -step regret of these algorithms. We also establish a lower bound on the regret for cascading non-stationary bandits and show that both algorithms match the lower bound up to a logarithmic factor. Finally, we evaluate their performance on a real-world web search click dataset.

### 4.1 Introduction

---

Learning to rank LTR [84] is a combination of machine learning and information retrieval. It is a core problem in many applications, such as web search and recommendation [84, 127]. The goal of LTR is to rank items, e.g., documents, and show the top  $K$  items to a user. Traditional LTR algorithms are supervised, offline algorithms; they learn rankers from human annotated data [97] and/or users' historical interactions [54]. Every day billions of users interact with modern search engines and leave a trail of interactions. It is feasible and important to design online algorithms that directly learn from such user clicks to help improve users' online experience. Indeed, recent studies show that even well-trained production rankers can be optimized by using users' online interactions, such as clicks [131].

Generally, interaction data is noisy [54], which gives rise to the well-known exploration vs. exploitation dilemma. Multi-armed bandit (MAB) [9] algorithms have been designed to balance exploration and exploitation. Based on MABs, many online LTR algorithms have been published [58, 65, 70, 77, 99, 132]. These algorithms address

---

This chapter was published as [74].

## 4. Cascade Non-Stationary Bandits

---

the exploration vs. exploitation dilemma in an elegant way and aim to maximize user satisfaction in a stationary environment where users do not change their preferences over time. Moreover, they often come with regret bounds.

Despite the success of the algorithms mentioned above in the stationary case, they may have linear regret in a non-stationary environment where users may change their preferences abruptly at any unknown moment in time. Non-stationarity widely exists in real-world application domains, such as search engines and recommender systems [49, 92, 116, 119]. Particularly, we consider *abruptly changing environments* where user preferences remain constant in certain time periods, named *epochs*, but change occurs abruptly at unknown moments called *breakpoints*. The abrupt changes in user preferences give rise to a new challenge of balancing “remembering” and “forgetting” [11]: the more past observations an algorithm retains the higher the risk of making a biased estimator, while the fewer observations retained the higher stochastic error it has on the estimates of the user preferences.

In this chapter, we propose *cascading non-stationary bandits*, an online variant of the Cascade Model (CM) [27] with the goal of identifying the  $K$  most attractive items in a non-stationary environment. CM is a widely-used model of user click behavior [23, 132]. In CM, a user browses the ranked list from top to bottom and clicks the first attractive item. The items ranked above the first clicked item are browsed but not attractive since they are not clicked. The items ranked below the first clicked item are not browsed since the user stops browsing the ranked list after a click. Although CM is a simple model, it effectively explains user behavior [65].

Our key technical contributions in this chapter are: (1) We formalize a non-stationary Online Learning to Rank (OLTR) problem as cascading non-stationary bandits. (2) We propose two algorithms, CascadeDUCB and CascadeSWUCB, for solving it. They are motivated by *discounted UCB* (DUCB) and *sliding window UCB* (SWUCB), respectively [31]. CascadeDUCB balances “remembering” and “forgetting” by using a discounting factor of past observations, and CascadeSWUCB balances the two by using statistics inside a fixed-size sliding window. (3) We derive gap-dependent upper bounds on the regret of the proposed algorithms. (4) We derive a lower bound on the regret of cascading non-stationary bandits. We show that the upper bounds match this lower bound up to a logarithmic factor. (5) We evaluate the performance of CascadeSWUCB and CascadeDUCB empirically on a real-world web search click dataset.

## 4.2 Background

---

We define the learning problem at the core of this chapter in terms of cascading non-stationary bandits. Their definition builds on the CM and its online variant *cascading bandits*, which we review in this section.

We write  $[n]$  for  $\{1, \dots, n\}$ . For sets  $A$  and  $B$ , we write  $A^B$  for the set of all vectors whose entries are indexed by  $B$  and take values from  $A$ . We use boldface letters to denote random variables. We denote a set of candidate items by  $\mathcal{D} = [L]$ , e.g., a set of preselected documents. The presented ranked list is denoted as  $\mathcal{R} \in \Pi_K(\mathcal{D})$ , where  $\Pi_K(\mathcal{D})$  denotes the set of all possible combinations of  $K$  distinct items from  $\mathcal{D}$ . The item at position  $k$  in  $\mathcal{R}$  is denoted as  $\mathcal{R}(k)$ , and the position of item  $a$  in  $\mathcal{R}$  is denoted

as  $\mathcal{R}^{-1}(a)$

### 4.2.1 Cascade model

We refer readers to [23] for an introduction to click models. Briefly, a click model models a user’s interaction behavior with the search system. The user is presented with a  $K$ -item ranked list  $\mathcal{R}$ . Then the user browses the list  $\mathcal{R}$  and clicks items that potentially attract him or her. Many click models have been proposed and each models a certain aspect of interaction behavior. We can parameterize a click model by attraction probabilities  $\alpha \in [0, 1]^L$  and a click model assumes:

**Assumption 4.1.** *The attraction probability  $\alpha(a)$  only depends on item  $a$  and is independent of other items.*

CM is a widely-used click model [27, 132]. In the CM, a user browses the ranked list  $\mathcal{R}$  from the first item  $\mathcal{R}(1)$  to the last item  $\mathcal{R}(K)$ , which is called the *cascading assumption*. After the user browses an item  $\mathcal{R}(i)$ , he or she clicks on  $\mathcal{R}(i)$  with attraction probability  $\alpha(\mathcal{R}(i))$ , and then stops browsing the remaining items. Thus, the examination probability of item  $\mathcal{R}(j)$  equals the probability of no click on the higher ranked items:  $\prod_{i=1}^{j-1} (1 - \alpha(\mathcal{R}(i)))$ . The expected number of clicks equals the probability of clicking any item in the list:  $1 - \prod_{i=1}^K (1 - \alpha(\mathcal{R}(i)))$ . Note that the reward does not depend on the order in  $\mathcal{R}$ , and thus, in the CM, the goal of ranking is to find the  $K$  most attractive items.

The CM accepts at most one click in each search session. It cannot explain scenarios where a user may click multiple items. The CM has been extended in different ways to capture multi-click cases [20, 36]. Nevertheless, CM is still the fundamental click model and fits historical click data reasonably well. Thus, in this chapter, we focus on the CM and in the next section we introduce an online variant of CM, called *cascading bandits*.

### 4.2.2 Cascading bandits

*Cascading bandits* (CB) is defined by a tuple  $B = (\mathcal{D}, P, K)$ , where  $\mathcal{D} = [L]$  is the set of candidate items,  $K \leq L$  is the number of positions,  $P \in \{0, 1\}^L$  is a distribution over binary attractions.

In CB, at time  $t$ , a learning agent builds a ranked list  $\mathbf{R}_t \in \Pi_K(\mathcal{D})$  that depends on the historical observations up to  $t$  and shows it to the user.  $\mathbf{A}_t \in \{0, 1\}^L$  is defined as the *attraction indicator*, which is drawn from  $P$  and  $\mathbf{A}_t(\mathbf{R}_t(i))$  is the attraction indicator of item  $\mathbf{R}_t(i)$ . The user examines  $\mathbf{R}_t$  from  $\mathbf{R}_t(1)$  to  $\mathbf{R}_t(K)$  and clicks the first attractive item. Since a CM allows at most one click each time, a random variable  $\mathbf{c}_t$  is used to indicate the position of the clicked item, i.e.,  $\mathbf{c}_t = \arg \min_{i \in [K]} \mathbb{1}\{\mathbf{A}_t(\mathbf{R}_t(i))\}$ . If there is no attractive item, the user will not click, and we set  $\mathbf{c}_t = K + 1$  to indicate this case. Specifically, if  $\mathbf{c}_t \leq K$ , the user clicks an item, otherwise, the user does not click anything. After the click or browsing the last item in  $\mathbf{R}_t$ , the user leaves the search session. The click feedback  $\mathbf{c}_t$  is then observed by the learning agent. Because of the cascading assumption, the agent knows that items ranked above position  $\mathbf{c}_t$  are

## 4. Cascade Non-Stationary Bandits

---

observed. The reward at time  $t$  is defined by the number of clicks:

$$r(\mathbf{R}_t, \mathbf{A}_t) = 1 - \prod_{i=1}^K (1 - \mathbf{A}_t(\mathbf{R}_t(i))). \quad (4.1)$$

Under Assumption 4.1, the attraction indicators of each item in  $\mathcal{D}$  are independently distributed. Moreover, cascading bandits make another assumption.

**Assumption 4.2.** *The attraction indicators are distributed as:*

$$P(\mathbf{A}) = \prod_{a \in \mathcal{D}} P_a(\mathbf{A}(a)), \quad (4.2)$$

where  $P_a$  is a Bernoulli distribution with a mean of  $\alpha(a)$ .

Under Assumption 4.1 and Assumption 4.2, the attraction indicator of item  $a$  at time  $t$   $\mathbf{A}_t(a)$  is drawn independently from other items. Thus, the expectation of reward of the ranked list at time  $t$  can be computed as  $\mathbb{E}[r(\mathbf{R}_t, \mathbf{A}_t)] = r(\mathbf{R}_t, \alpha)$ . And the goal of the agent is to maximize the expected number of clicks in  $n$  steps, which is equivalent to minimizing the  $n$ -step cumulative regret:

$$R(n) = \sum_{t=1}^n \mathbb{E} \left[ \max_{R \in \Pi_K(\mathcal{D})} r(R, \alpha) - r(\mathbf{R}_t, \alpha) \right]. \quad (4.3)$$

Cascading bandits are designed for a stationary environment, where the attraction probability  $P$  remains constant. However, in real-world applications, users change their preferences constantly [49], which is called a *non-stationary environment*, and learning algorithms proposed for cascading bandits, e.g., CascadeKL-UCB and CascadeUCB1 [65], may have linear regret in this setting. In the next section, we propose cascading non-stationary bandits, the first non-stationary variant of cascading bandits, and then propose two algorithms for solving this problem.

## 4.3 Cascading Non-Stationary Bandits

---

We first define our non-stationary online learning setup, and then we propose two algorithms learning in this setup.

### 4.3.1 Problem setup

The learning problem we study is called *cascading non-stationary bandits*, a variant of CB. We define it by a tuple  $B = (\mathcal{D}, P, K, \Upsilon_n)$ , where  $\mathcal{D} = [L]$  and  $K \leq L$  are the same as in CB bandits,  $P \in \{0, 1\}^{n \times L}$  is a distribution over binary attractions and  $\Upsilon_n$  is the number of abrupt changes in  $P$  up to step  $n$ . We use  $P_t(\mathbf{R}_t(i))$  to indicate the attraction probability distribution of item  $\mathbf{R}_t(i)$  at time  $t$ . If  $\Upsilon_n = 0$ , this setup is same as CB. The difference is that we consider a non-stationary learning setup in which  $\Upsilon_n > 0$  and the non-stationarity in attraction probabilities characterizes our learning problem.

---

**Algorithm 5** UCB-type algorithm for Cascading non-stationary bandits.

---

```

1: Input: discounted factor  $\gamma$  or sliding window size  $\tau$ 
2: // Initialization
3:  $\forall a \in \mathcal{D} : \mathbf{N}_0(a) = 0$ 
4:  $\forall a \in \mathcal{D} : \mathbf{X}_0(a) = 0$ 
5: for  $t = 1, 2, \dots, n$  do
6:   for  $a \in \mathcal{D}$  do
7:     // Compute UCBs
8:      $\mathbf{U}_t(a) \leftarrow \begin{cases} \text{Eq. (4.6)} & (\text{CascadeDUCB}) \\ \text{Eq. (4.8)} & (\text{CascadeSWUCB}) \end{cases}$ 
9:   // Recommend top  $K$  items and receive clicks
10:   $\mathbf{R}_t \leftarrow \arg \max_{\mathbf{R} \in \Pi_K(\mathcal{D})} r(\mathbf{R}, \mathbf{U}_t)$ 
11:  Show  $\mathbf{R}_t$  and receive clicks  $\mathbf{c}_t \in \{1, \dots, K + 1\}$ 
12:  // Update statistics
13:  if CascadeDUCB then
14:    // for CascadeDUCB
15:     $\forall a \in \mathcal{D} : \mathbf{N}_t(a) = \gamma \mathbf{N}_{t-1}(a)$ 
16:     $\forall a \in \mathcal{D} : \mathbf{X}_t(a) = \gamma \mathbf{X}_{t-1}(a)$ 
17:  else
18:    // for CascadeSWUCB
19:     $\forall a \in \mathcal{D} : \mathbf{N}_t(a) = \sum_{s=t-\tau+1}^{t-1} \mathbb{1}\{a \in \mathbf{R}_s\}$ 
20:     $\forall a \in \mathcal{D} : \mathbf{X}_t(a) = \sum_{s=t-\tau+1}^{t-1} \mathbb{1}\{\mathbf{R}_s^{-1}(a) = \mathbf{c}_s\}$ 
21:    for  $i = 1, \dots, \min\{\mathbf{c}_t, K\}$  do
22:       $a \leftarrow \mathbf{R}_t(i)$ 
23:       $\mathbf{N}_t(a) = \mathbf{N}_t(a) + 1$ 
24:       $\mathbf{X}_t(a) = \mathbf{X}_t(a) + \mathbb{1}\{i = \mathbf{c}_t\}$ 

```

---

In this chapter, we consider an abruptly changing environment, where the attraction probability  $P$  remains constant within an epoch but can change at any unknown moment in time and the number of abrupt changes up to  $n$  steps is  $\Upsilon_n$ . The learning agent interacts with cascading non-stationary bandits in the same way as with CB. Since the agent is in a non-stationary environment, we write  $\alpha_t$  for the mean of the attraction probabilities at time  $t$  and we evaluate the agent by the expected cumulated regret expressed as:

$$R(n) = \sum_{t=1}^n \mathbb{E} \left[ \max_{R \in \Pi_K(\mathcal{D})} r(R, \alpha_t) - r(\mathbf{R}_t, \mathbf{A}_t) \right]. \quad (4.4)$$

The goal of the agent is to minimize the  $n$ -step regret.

### 4.3.2 Algorithms

We propose two algorithms, CascadeDUCB and CascadeSWUCB, for solving cascading non-stationary bandits. The former one is inspired by DUCB and the latter

## 4. Cascade Non-Stationary Bandits

---

one is inspired by SWUCB [31]. We summarize the pseudocode of both algorithms in Algorithm 5.

CascadeDUCB and CascadeSWUCB learn in a similar pattern. They differ in the way they estimate the Upper Confidence Bound (UCB)  $\mathbf{U}_t(\mathbf{R}_t(i))$  of the attraction probability of item  $\mathbf{R}_t(i)$  as time  $t$ , as discussed later in this section. After estimating the UCBs (line 8), both algorithms construct  $\mathbf{R}_t$  by including the top  $K$  most relevant items by UCB. Since the order of top  $K$  items only affects the observation but does not affect the payoff of  $\mathbf{R}_t$ , we construct  $\mathbf{R}_t$  as follows:

$$\mathbf{R}_t = \arg \max_{\mathcal{R} \in \Pi_K(\mathcal{D})} r(\mathcal{R}, \mathbf{U}_t). \quad (4.5)$$

After receiving the user's click feedback  $\mathbf{c}_t$ , both algorithms update their statistics (lines 13–24). We use  $\mathbf{N}_t(i)$  and  $\mathbf{X}_t(i)$  to indicate the number of items  $i$  that have been observed and clicked up to  $t$  step, respectively.

To tackle the challenge of non-stationarity, CascadeDUCB penalizes old observations with a discount factor  $\gamma \in (0, 1)$ . Specifically, each of the previous statistics is discounted by  $\gamma$  (lines 15–16). The UCB of item  $a$  is estimated as:

$$\mathbf{U}_t(a) = \bar{\alpha}_t(\gamma, a) + c_t(\gamma, a), \quad (4.6)$$

where  $\bar{\alpha}_t(\gamma, a) = \frac{\mathbf{X}_t(a)}{\mathbf{N}_t(a)}$  is the average of discounted attraction indicators of item  $i$  and

$$c_t(\gamma, a) = 2 \sqrt{\frac{\epsilon \ln N_t(\gamma)}{\mathbf{N}_t(a)}} \quad (4.7)$$

is the confidence interval around  $\bar{\alpha}_t(i)$  at time  $t$ . Here, we compute  $N_t(\gamma) = \frac{1-\gamma^t}{1-\gamma}$  as the discounted time horizon. As shown in [31],  $\alpha_t(a) \in [\bar{\alpha}_t(\gamma, a) - c_t(\gamma, a), \bar{\alpha}_t(\gamma, a) + c_t(\gamma, a)]$  holds with high probability.

As to CascadeSWUCB, it estimates UCBs by observations inside a sliding window with size  $\tau$ . Specifically, it only considers the observations in the previous  $\tau$  steps (lines 19–20). The UCB of item  $i$  is estimated as

$$\mathbf{U}_t(a) = \bar{\alpha}_t(\tau, a) + c_t(\tau, a), \quad (4.8)$$

where  $\bar{\alpha}_t(\tau, a) = \frac{\mathbf{X}_t(a)}{\mathbf{N}_t(a)}$  is the average of observed attraction indicators of item  $a$  inside the sliding window and

$$c_t(\tau, a) = \sqrt{\frac{\epsilon \ln(t \wedge \tau)}{\mathbf{N}_t(a)}} \quad (4.9)$$

is the confidence interval, and  $t \wedge \tau = \min(t, \tau)$ .

**Initialization.** In the initialization phase, we set all the statistics to 0 and define  $\frac{x}{0} := 1$  for any  $x$  (lines 3–4). Mapping back this to UCB, at the beginning, each item has the optimal assumption on the attraction probability with an optimal bonus on uncertainty. This is a common initialization strategy for UCB-type bandit algorithms [79].

## 4.4 Analysis

In this section, we analyze the  $n$ -step regret of CascadeDUCB and CascadeSWUCB. We first derive regret upper bounds on CascadeDUCB and CascadeSWUCB, respectively. Then we derive a regret lower bound on cascading non-stationary bandits. Finally, we discuss our theoretical results.

### 4.4.1 Regret upper bound

We refer to  $\mathcal{D}_t^* \subseteq [L]$  as the set of the  $K$  most attractive items in set  $\mathcal{D}$  at time  $t$  and  $\bar{\mathcal{D}}_t$  as the complement of  $\mathcal{D}_t^*$ , i.e.,  $\forall a \in \mathcal{D}_t^*, \forall a^* \in \bar{\mathcal{D}}_t : \alpha_t(a) \geq \alpha_t(a^*)$  and  $\mathcal{D}_t^* \cup \bar{\mathcal{D}}_t = \mathcal{D}, \mathcal{D}_t^* \cap \bar{\mathcal{D}}_t = \emptyset$ . At time  $t$ , we say an item  $a^*$  is optimal if  $a^* \in \mathcal{D}_t^*$  and an item  $a$  is suboptimal if  $a \in \bar{\mathcal{D}}_t$ . The regret at time  $t$  is caused by the case that  $\mathbf{R}_t$  includes at least one suboptimal and examined items. Let  $\Delta_{a,a^*}^t$  be the gap of attraction probability between a suboptimal item  $a$  and an optimal  $a^*$  at time  $t$ :  $\Delta_{a,a^*}^t = \alpha_t(a^*) - \alpha_t(a)$ . Then we refer to  $\Delta_{a,K}$  as the smallest gap of between item  $a$  and the  $K$ -th most attractive item in all  $n$  steps when  $a$  is not the optimal items:  $\Delta_{a,K} = \min_{t \in [n], a^* \in \mathcal{D}_t^*} \alpha_t(a^*) - \alpha_t(a)$ .

**Theorem 4.1.** *Let  $\epsilon \in (1/2, 1)$  and  $\gamma \in (1/2, 1)$ , the expected  $n$ -step regret of CascadeDUCB is bounded as:*

$$R(n) \leq L\Upsilon_n \frac{\ln[(1-\gamma)\epsilon]}{\ln \gamma} + \sum_{a \in \mathcal{D}} C(\gamma, a) \lceil n(1-\gamma) \rceil \ln \frac{1}{1-\gamma}, \quad (4.10)$$

where

$$C(\gamma, a) = \frac{4}{1-1/e} \ln(1+4\sqrt{1-1/2\epsilon}) + \frac{32\epsilon}{\Delta_{a,K}\gamma^{1/(1-\gamma)}}. \quad (4.11)$$

We outline the proof in 4 steps below; the full version is in Section 4.7.1.<sup>1</sup>

*Proof.* Our proof is adapted from the analysis in [65]. The novelty of the proof comes from the fact that, in a non-stationary environment, the discounted estimator  $\bar{\alpha}_t(\gamma, a)$  is now a biased estimator of  $\alpha_t(a)$  (*Step 1, 2 and 4*).

*Step 1.* We bound the regret of the event that estimators of the attraction probabilities are biased by  $L\Upsilon_n \frac{\ln[(1-\gamma)\epsilon]}{\ln \gamma}$ . This event happens during the steps following a breakpoint.

*Step 2.* We bound the regret of the event that  $\alpha_t(a)$  falls outside of the confidence interval around  $\bar{\alpha}_t(\gamma, a)$  by  $\frac{4}{1-1/e} \ln(1+4\sqrt{1-1/2\epsilon})n(1-\gamma) \ln \frac{1}{1-\gamma}$ .

*Step 3.* We decompose the regret at time  $t$  based on [65, Theorem 1].

*Step 4.* For each item  $a$ , we bound the number of times that item  $a$  is chosen when  $a \in \bar{\mathcal{D}}_t$  in  $n$  steps and get the term  $\frac{32\epsilon \lceil n(1-\gamma) \rceil \ln \frac{1}{1-\gamma}}{\Delta_{a,K}\gamma^{1/(1-\gamma)}}$ . Finally, we sum up all the regret.  $\square$

<sup>1</sup><https://arxiv.org/abs/1905.12370>

## 4. Cascade Non-Stationary Bandits

---

The bound depends on step  $n$  and the number of breakpoints  $\Upsilon_n$ . If they are known beforehand, we can choose  $\gamma$  by minimizing the right hand side of Eq. (4.10). Choosing  $\gamma = 1 - 1/4\sqrt{(\Upsilon_n/n)}$  leads to  $R(n) = O(\sqrt{n\Upsilon_n} \ln n)$ . When  $\Upsilon_n$  is independent of  $n$ , we have  $R(n) = O(\sqrt{n\Upsilon} \ln n)$ .

**Theorem 4.2.** *Let  $\epsilon \in (1/2, 1)$ . For any integer  $\tau$ , the expected  $n$ -step regret of CascadeSWUCB is bounded as:*

$$R(n) \leq L\Upsilon_n\tau + \frac{L\ln^2\tau}{\ln(1+4\sqrt{(1-1/2\epsilon)})} + \sum_{a \in \mathcal{D}} C(\tau, a) \frac{n\ln\tau}{\tau}, \quad (4.12)$$

where

$$C(\tau, a) = \frac{2}{\ln\tau} \left\lceil \frac{\ln\tau}{\ln(1+4\sqrt{(1-1/2\epsilon)})} \right\rceil + \frac{8\epsilon}{\Delta_{a,K}} \frac{\lceil n/\tau \rceil}{n/\tau}. \quad (4.13)$$

When  $\tau$  goes to infinity and  $n/\tau$  goes to 0,

$$C(\tau, a) = \frac{2}{\ln(1+4\sqrt{(1-1/2\epsilon)})} + \frac{8\epsilon}{\Delta_{a,K}}. \quad (4.14)$$

We outline the proof in 4 steps below and the full version is in Section 4.7.2.

*Proof.* The proof follows the same lines as the proof of Theorem 4.1.

*Step 1.* We bound the regret of the event that estimators of the attraction probabilities are biased by  $L\Upsilon_n\tau$ .

*Step 2.* We bound the regret of the event that  $\alpha_t(a)$  falls outside of the confidence interval around  $\bar{\alpha}_t(\tau, a)$  by

$$\ln^2\tau + 2n \left\lceil \frac{\ln\tau}{\ln(1+4\sqrt{(1-1/2\epsilon)})} \right\rceil. \quad (4.15)$$

*Step 3.* We decompose the regret at time  $t$  based on [65, Theorem 1].

*Step 4.* For each item  $a$ , we bound the number of times that item  $a$  is chosen when  $a \in \bar{\mathcal{D}}_t$  in  $n$  steps and get the term  $\frac{8\epsilon}{\Delta_{a,K}} \lceil \frac{n}{\tau} \rceil$ . Finally, we sum up all the regret.  $\square$

If we know  $\Upsilon_n$  and  $n$  beforehand, we can choose the window size  $\tau$  by minimizing the right hand side of Eq. (4.12). Choosing  $\tau = 2\sqrt{n\ln(n)/\Upsilon_n}$  leads to  $R(n) = O(\sqrt{n\Upsilon_n} \ln n)$ . When  $\Upsilon_n$  is independent of  $n$ , we have  $R(n) = O(\sqrt{n\Upsilon} \ln n)$ .

### 4.4.2 Regret lower bound

We consider a particular cascading non-stationary bandit and refer to it as  $B_L = (L, K, \Delta, p, \Upsilon)$ . We have a set of  $L$  items  $\mathcal{D} = [L]$  and  $K = \frac{1}{2}L$  positions. At any time  $t$ , the distribution of attraction probability of each item  $a \in \mathcal{D}$  is parameterized by:

$$\alpha_t(a) = \begin{cases} p & \text{if } a \in \mathcal{D}_t^* \\ p - \Delta & \text{if } a \in \bar{\mathcal{D}}_t, \end{cases} \quad (4.16)$$

where  $\mathcal{D}_t^*$  is the set of optimal items at time  $t$ ,  $\bar{\mathcal{D}}_t$  is the set suboptimal items at time  $t$ , and  $\Delta \in (0, p]$  is the gap between optimal items and suboptimal items. Thus, the attraction probabilities only take two values:  $p$  for optimal items and  $p - \Delta$  for suboptimal items up to  $n$ -step.  $\Upsilon$  is the number of breakpoints when the attraction probability of an item changes from  $p$  to  $p - \Delta$  or other way around. Particularly, we consider a simple variant that the distribution of attraction probability of each item is piecewise constant and has two breakpoints. And we assume another constraint on the number of optimal items that  $|\mathcal{D}_t^*| = K$  for all time steps  $t \in [n]$ . Then, the regret that any learning policy can achieve when interacting with  $B_L$  is lower bounded by Theorem 4.3.

**Theorem 4.3.** *The  $n$ -step regret of any learning algorithm interacting with cascading non-stationary bandit  $B_L$  is lower bounded as follows:*

$$\liminf_{n \rightarrow \infty} R(n) \geq L\Delta(1-p)^{K-1} \sqrt{\frac{2n}{3D_{KL}(p - \Delta || p)}}, \quad (4.17)$$

where  $D_{KL}(p - \Delta || p)$  is the Kullback-Leibler (KL) divergence between two Bernoulli distributions with means  $p - \Delta$  and  $p$ .

*Proof.* The proof is based on the analysis in [65]. We first refer to  $\mathcal{R}_t^*$  as the optimal list at time  $t$  that includes  $K$  items. For any time step  $t$ , any item  $a \in \bar{\mathcal{D}}_t$  and any item  $a^* \in \mathcal{D}_t^*$ , we define the event that item  $a$  is included in  $\mathbf{R}_t$  instead of item  $a^*$  and item  $a$  is examined but not clicked at time step  $t$  by:

$$G_{t,a,a^*} = \{\exists 1 \leq k < \mathbf{c}_t \sim s.t. \sim \mathbf{R}_t(k) = a, \mathcal{R}_t(k) = a^*\}. \quad (4.18)$$

By [65, Theorem 1], the regret at time  $t$  is decomposed as:

$$\mathbb{E}[r(\mathbf{R}_t, \alpha_t)] \geq \Delta(1-p)^{K-1} \sum_{a \in \bar{\mathcal{D}}_t} \sum_{a^* \in \mathcal{D}_t^*} \mathbb{1}\{G_{t,a,a^*}\}. \quad (4.19)$$

Then, we bound the  $n$ -step regret as follows:

$$\begin{aligned} R(n) &\geq \Delta(1-p)^{K-1} \sum_{t=1}^n \sum_{a \in \bar{\mathcal{D}}_t} \sum_{a^* \in \mathcal{D}_t^*} \mathbb{1}\{G_{t,a,a^*}\} \\ &\geq \Delta(1-p)^{K-1} \sum_{a \in \bar{\mathcal{D}}} \sum_{t=1}^n \mathbb{1}\{a \in \bar{\mathcal{D}}_t, a \in \mathbf{R}_t\} \\ &= \Delta(1-p)^{K-1} \sum_{a \in \bar{\mathcal{D}}} \mathbf{T}_n(a), \end{aligned} \quad (4.20)$$

where  $\mathbf{T}_n(a) = \sum_{t=1}^n \mathbb{1}\{a \in \bar{\mathcal{D}}_t, a \in \mathbf{R}_t, \mathbf{R}_t^{-1}(a) \leq \mathbf{c}_t\}$ . The second inequality is based on the fact that, at time  $t$ , the event  $G_{t,a,a^*}$  happens if and only if item  $a$  is suboptimal and examined. By the results of [31, Theorem 3], if a suboptimal item  $a$  has not been examined enough times, the learning policy may play this item for a long

## 4. Cascade Non-Stationary Bandits

---

period after a breakpoint. And we get:

$$\liminf_{n \rightarrow \infty} \mathbf{T}(n) \geq \sqrt{\frac{2n}{3D_{KL}(p - \Delta || p)}}. \quad (4.21)$$

We sum up all the inequalities and obtain:

$$\liminf_{n \rightarrow \infty} R(n) \geq L\Delta(1-p)^{K-1} \sqrt{\frac{2n}{3D_{KL}(p - \Delta || p)}}. \quad \square$$

### 4.4.3 Discussion

We have shown that the  $n$ -step regret upper bounds of CascadeDUCB and CascadeSWUCB have the order of  $O(\sqrt{n} \ln n)$  and  $O(\sqrt{n} \ln n)$ , respectively. They match the lower bound proposed in Theorem 4.3 up to a logarithmic factor. Specifically, the upper bound of CascadeDUCB matches the lower bound up to  $\ln n$ . The upper bound of CascadeSWUCB matches the lower bound up to  $\sqrt{\ln n}$ , an improvement over CascadeDUCB, as confirmed by experiments in Section 4.5.

We have assumed that step  $n$  is known beforehand. This may not always be the case. We can extend CascadeDUCB and CascadeSWUCB to the case where  $n$  is unknown by using the *doubling trick* [31]. Namely, for  $t > n$  and any  $k$ , such that  $2^k \leq t < 2^{k+1}$ , we reset  $\gamma = 1 - \frac{1}{4\sqrt{2^k}}$  and  $\tau = 2\sqrt{2^k \ln(2^k)}$ .

CascadeDUCB and CascadeSWUCB can be computed efficiently. Their complexity is linear in the number of time steps. However, CascadeSWUCB requires extra memory to remember past ranked lists and rewards to update  $\mathbf{X}_t$  and  $\mathbf{N}_t$ .

## 4.5 Experimental Analysis

---

We evaluate CascadeDUCB and CascadeSWUCB on the *Yandex* click dataset,<sup>2</sup> which is the largest public click collection. It contains more than 30 million search sessions, each of which contains at least one search query. We process the queries in the same manner as in [77, 132]. Namely, we randomly select 100 frequent search queries with the 10 most attractive items in each query, and then learn a CM for each query using PyClick.<sup>3</sup> These CMs are used to generate click feedback. In this setup, the size of candidate items is  $L = 10$  and we choose  $K = 3$  as the number of positions. The objective of the learning task is to identify 3 most attractive items and then maximize the expected number of clicks at the 3 highest positions.

We consider a simulated non-stationary environment setup, where we take the learned attraction probabilities as the default and change the attraction probabilities periodically. Our simulation can be described in 4 steps: (1) For each query, the attraction probabilities of the top 3 items remain constant over time. (2) We randomly choose three suboptimal items and set their attraction probabilities to 0.9 for the next

<sup>2</sup>[https://academy.yandex.ru/events/data\\_analysis/relpred2011](https://academy.yandex.ru/events/data_analysis/relpred2011)

<sup>3</sup><https://github.com/markovi/PyClick>

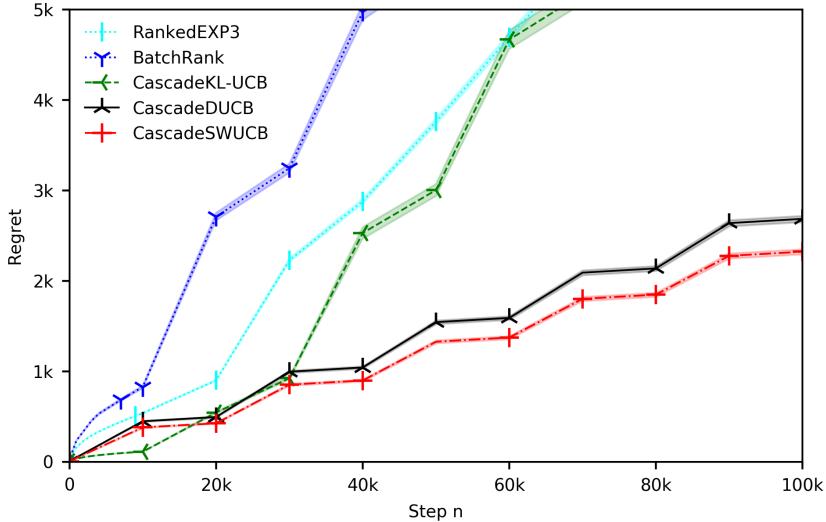


Figure 4.1: The  $n$ -step regret in up to 100k steps. Lower is better. The results are averaged over all 100 queries and 10 runs per query. The shaded regions represent standard errors of our estimates.

$m_1$  steps. (3) Then we reset the attraction probabilities and keep them constant for the next  $m_2$  steps. (4) We repeat step (2) and step (3) iteratively. This simulation mimics abrupt changes in user preferences and is widely used in previous work on non-stationarity [31, 49]. In our experiment, we set  $m_1 = m_2 = 10k$  and choose 10 breakpoints. In total, we run experiments for 100k steps. Although the non-stationary aspects in our setup are simulated, the other parameters of a CM are learned from the Yandex click dataset.

We compare CascadeDUCB and CascadeSWUCB, to RankedEXP3 [99], CascadeKL-UCB [65] and BatchRank [132]. We describe the baseline algorithms in slightly more details in Section 4.6. Briefly, RankedEXP3, a variant of ranked bandits, is based on an adversarial bandit algorithm EXP3 [8]; it is the earliest bandit-based ranking algorithm and is popular in practice. CascadeKL-UCB [65] is a near optimal algorithm in CM. BatchRank [132] can learn in a wide range of click models. However, these algorithms only learn in a stationary environment. We choose them as baselines to show the superiority of our algorithms in a non-stationary environment. In experiments, we set  $\epsilon = 0.5$ ,  $\gamma = 1 - 1/(4\sqrt{n})$  and  $\tau = 2\sqrt{n \ln(n)}$ , the values that roughly minimize the upper bounds.

We report the  $n$ -step regret averaged over 100 queries and 10 runs per query in Fig. 4.1. All baselines have linear regret in time step. They fail in capturing the breakpoints. Non-stationarity makes the baselines perform even worse during epochs where the attraction probability are set as the default. E.g., CascadeKL-UCB has  $111.50 \pm 1.12$  regret in the first 10k steps but has  $447.82 \pm 137.16$  regret between step 80k and 90k. Importantly, the attraction probabilities equal the default and remain constant inside these two epochs. This is caused by the fact that the baseline algorithms

## 4. Cascade Non-Stationary Bandits

---

rank items based on all historical observations, i.e., they do not balance “remembering” and “forgetting.” Because of the use of a discounting factor and/or a sliding window, CascadeDUCB and CascadeSWUCB can detect breakpoints and show convergence. CascadeSWUCB outperforms CascadeDUCB with a small gap; this is consistent with our theoretical finding that CascadeSWUCB outperforms CascadeDUCB by a  $\sqrt{\ln n}$  factor.

## 4.6 Related Work

---

The idea of directly learning to rank from user feedback has been widely studied in a stationary environment. *Ranked bandits* [99] are among the earliest OLTR approaches. In ranked bandits, each position in the list is modeled as an individual underlying MABs. The ranking task is then solved by asking each individual MAB to recommend an item to the attached position. Since the reward, e.g., click, of a lower position is affected by higher positions, the underlying MAB is typically adversarial, e.g., EXP3 [8]. BatchRank is a recently proposed OLTR method [132]; it is an elimination-based algorithm: once an item is found to be inferior to  $K$  items, it will be removed from future consideration. BatchRank outperforms ranked bandits in the stationary environment. In our experiments, we include BatchRank and RankedEXP3, the EXP3-based ranked bandit algorithm, as baselines.

Several OLTR algorithms have been proposed in specific click models [58, 65, 70]. They can efficiently learn an optimal ranking given the click model they consider. Our work is related to cascading bandits and we compare our algorithms to CascadeKL-UCB, an algorithm proposed for solving cascading bandits [65], in Section 4.5. Our work differs from cascading bandits in that we consider learning in a non-stationary environment.

Non-stationary bandit problems have been widely studied [11, 31, 82, 105, 119]. However, previous work requires a small action space. In our setup, actions are (exponentially many) ranked lists. Thus, we do not consider them as baselines in our experiments.

In *adversarial bandits* the reward realizations, in our case attraction indicators, are selected by an *adversary*. Adversarial bandits originate from game theory [12] and have been widely studied, cf. [8, 18] for an overview. Within adversarial bandits, the performance of a policy is often measured by comparing to a static oracle which always chooses a single best arm that is obtained after seeing all the reward realizations up to step  $n$ . This static oracle can perform poorly in a non-stationary case when the single best arm is suboptimal for a long time between two breakpoints. Thus, even if a policy performs closely to the static oracle, it can still perform sub-optimally in a non-stationary environment. Our work differs from adversarial bandits in that we compare to a dynamic oracle that can balance the dilemma of “remembering” and “forgetting” and chooses the per-step best action.

## 4.7 Proofs

---

In this proofs, we refer to  $\mathcal{R}_t^*$  as the optimal list at time  $t$  that includes  $K$  items sorted by the decreasing order of their attraction probabilities. We refer to  $\mathcal{D}_t^* \subseteq [L]$  as the

set of the  $K$  most attractive items in set  $\mathcal{D}$  at time  $t$  and  $\bar{\mathcal{D}}_t$  as the complement of  $\mathcal{D}_t^*$ , i.e.  $\forall a^* \in \mathcal{D}_t^*, \forall a \in \bar{\mathcal{D}}_t : \alpha_t(a^*) \geq \alpha_t(a)$  and  $\mathcal{D}_t^* \cup \bar{\mathcal{D}}_t = \mathcal{D}, \mathcal{D}_t^* \cap \bar{\mathcal{D}}_t = \emptyset$ . At time  $t$ , we say an item  $a^*$  is optimal if  $a^* \in \mathcal{D}_t^*$  and an item  $a$  is suboptimal if  $a \in \bar{\mathcal{D}}_t$ . We denote  $\mathbf{r}_t = \max_{R \in \Pi_K(\mathcal{D})} r(R, \alpha_t) - r(\mathbf{R}_t, \mathbf{A}_t)$  to be the regret at time  $t$  of the learning algorithm. Let  $\Delta_{a,a^*}^t$  be the gap of attraction probability between a suboptimal item  $a$  and an optimal  $a^*$  at time  $t$ :  $\Delta_{a,a^*}^t = \alpha_t(a^*) - \alpha_t(a)$ . Then we refer to  $\Delta_{a,K}$  as the smallest gap between item  $a$  and the  $K$ -th most attractive item in all  $n$  time steps when  $a$  is not the optimal item:  $\Delta_{a,K} = \min_{t \in [n], a \notin \mathcal{D}_t^*} \alpha_t(K) - \alpha_t(a)$ .

### 4.7.1 Proof of Theorem 4.1

Let  $M_t = \{\exists a \in \mathcal{D} \sim s.t. \sim |\alpha_t(a) - \bar{\alpha}_t(\gamma, a)| > c_t(\gamma, t)\}$  be the event that  $\alpha_t(a)$  falls out of the confidence interval around  $\bar{\alpha}_t(\gamma, a)$  at time  $t$ , and  $\bar{M}_t$  be the complement of  $M_t$ . We re-write the  $n$ -step regret of CascadeDUCB as follows:

$$R(n) = \mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{M_t\} \mathbf{r}_t \right] + \mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{\bar{M}_t\} \mathbf{r}_t \right]. \quad (4.22)$$

We then bound both terms above separately.

We refer to  $\mathcal{T}$  as the set of all time steps such that for  $t \in \mathcal{T}, s \in [t - B(\gamma), t]$  and any item  $a \in \mathcal{D}$  we have  $\alpha_s(a) = \alpha_t(a)$ , where  $B(\gamma) = \lceil \log_\gamma(\epsilon(1 - \gamma)) \rceil$ . In other words,  $\mathcal{T}$  is the set of time steps that do not follow too close after breakpoints. Since for any time step  $t \notin \mathcal{T}$  the estimators of attraction probabilities are biased, CascadeDUCB may recommend suboptimal items constantly. Thus, we get the following bound:

$$\mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{M_t\} \mathbf{r}_t \right] \leq L \Upsilon_n B(\gamma) + \mathbb{E} \left[ \sum_{t \in \mathcal{T}} \mathbb{1}\{M_t\} \mathbf{r}_t \right]. \quad (4.23)$$

Then, we show that for steps  $t \in \mathcal{T}$ , the attraction probabilities are well estimated for all items with high probability. For an item  $a$ , we consider the bias and variance of  $\bar{\alpha}_t(\gamma, a)$  separately. We denote:

$$\mathbf{X}_t(\gamma, a) = \sum_{s=1}^t \gamma^{(t-s)} \mathbb{1}\{a \in \mathbf{R}_s, \mathbf{R}_s(\mathbf{c}_s) = a\}, \quad \mathbf{N}_t(\gamma, a) = \sum_{s=1}^t \gamma^{(t-s)} \mathbb{1}\{a \in \mathbf{R}_s\},$$

as the discounted number of being clicked, and the discounted number of being examined.

First, we consider the bias. The ‘‘bias’’ at time  $t$  can be written as  $x_t(\gamma, a)/\mathbf{N}_t(\gamma, a) - \alpha_t$ , where  $x_t(\gamma, a) = \sum_{s=1}^t \gamma^{(t-s)} \mathbb{1}\{a \in \mathbf{R}_s\} \alpha_s(a)$ . For  $t \in \mathcal{T}$ :

$$\begin{aligned} |x_t(\gamma, a) - \alpha_t(a) \mathbf{N}_t(\gamma, a)| &= \left| \sum_{s=1}^{t-B(\gamma)} \gamma^{(t-s)} (\alpha_s(a) - \alpha_t(a)) \mathbb{1}\{a \in \mathbf{R}_t\} \right| \\ &\leq \sum_{s=1}^{t-B(\gamma)} \gamma^{(t-s)} |(\alpha_s(a) - \alpha_t(a)) \mathbb{1}\{a \in \mathbf{R}_t\}| \\ &\leq \gamma^{B(\gamma)} \mathbf{N}_{t-B(\gamma)}(\gamma, a) \leq \gamma^{B(\gamma)} \frac{1}{1-\gamma}, \end{aligned}$$

#### 4. Cascade Non-Stationary Bandits

---

where the last inequality is due to the fact that  $\mathbf{N}_{t-B(\gamma)}(\gamma, a) \leq 1/(1-\gamma)$ . Thus, we get:

$$\begin{aligned} \left| \frac{x_t(\gamma, a)}{\mathbf{N}_t(\gamma, a)} - \alpha_t(a) \right| &\leq \frac{\gamma^{B(\gamma)}}{(1-\gamma)\mathbf{N}_t(\gamma, a)} \leq \left( 1 \wedge \frac{\gamma^{B(\gamma)}}{(1-\gamma)\mathbf{N}_t(\gamma, a)} \right) \\ &\leq \sqrt{\frac{\gamma^{B(\gamma)}}{(1-\gamma)\mathbf{N}_t(\gamma, a)}} \leq \sqrt{\frac{\epsilon \ln N_t(\gamma)}{\mathbf{N}_t(\gamma, a)}} \leq \frac{1}{2} c_t(\gamma, a), \end{aligned}$$

where the third inequality is due to the fact that  $1 \wedge x \leq \sqrt{x}$  and the last inequality is due to the definition of  $B(\gamma)$ . So,  $B(\gamma)$  time steps after a breakpoint, the ‘‘bias’’ is at most half as large as the confidence bonus; and the second half of the confidence interval is used for the variance.

Second, we consider the variance. For  $a \in \mathcal{D}$  and  $t \in \mathcal{T}$ , let  $M_{t,a} = \{|\alpha_t(a) - \bar{\alpha}_t(\gamma, a)| > c_t(\gamma, t)\}$  be the event that  $\alpha_t(a)$  falls out of the confidence interval around  $\bar{\alpha}_t(\gamma, a)$  at time  $t$ . By using a Hoeffding-type inequality [31, Theorem 4], for an item  $a \in \mathcal{D}$ ,  $t \in \mathcal{T}$ , and any  $\eta > 0$ , we get:

$$\begin{aligned} P(M_{t,a}) &\leq P\left(\frac{\mathbf{X}_t(\gamma, a) - x_t(\gamma, a)}{\sqrt{\mathbf{N}_t(\gamma^2, a)}} > \sqrt{\frac{\epsilon \ln N_t(\gamma)}{\mathbf{N}_t(\gamma^2, a)}}\right) \\ &\leq P\left(\frac{\mathbf{X}_t(\gamma, a) - x_t(\gamma, a)}{\sqrt{\mathbf{N}_t(\gamma^2, a)}} > \sqrt{\epsilon \ln N_t(\gamma)}\right) \\ &\leq \left\lceil \frac{\ln N_t(\gamma)}{\ln(1+\eta)} \right\rceil N_t(\gamma)^{-2\epsilon(1-\frac{\eta^2}{16})}. \end{aligned}$$

Thus, we get the following bound:

$$\mathbb{E}\left[\sum_{t \in \mathcal{T}} \mathbb{1}\{M_t\} \mathbf{r}_t\right] \leq 2L \sum_{t \in \mathcal{T}} \left\lceil \frac{\ln N_t(\gamma)}{\ln(1+\eta)} \right\rceil N_t(\gamma)^{-2\epsilon(1-\frac{\eta^2}{16})}.$$

By taking  $\eta = 4\sqrt{1-1/2\epsilon}$  such that  $1 - \frac{\eta^2}{16} = 1$ , and with  $t_0 = (1-\gamma)^{(-1)}$  we get:

$$\begin{aligned} &\sum_{t \in \mathcal{T}} \left\lceil \frac{\ln N_t(\gamma)}{\ln(1+\eta)} \right\rceil N_t(\gamma)^{-2\epsilon(1-\frac{\eta^2}{16})} \\ &\leq t_0 + \sum_{t \in \mathcal{T}, t \geq t_0} \left\lceil \frac{\ln N_{t_0}(\gamma)}{\ln(1+\eta)} \right\rceil N_{t_0}(\gamma)^{-1} \\ &\leq t_0 + \left\lceil \frac{\ln N_{t_0}(\gamma)}{\ln(1+\eta)} \right\rceil \frac{n}{N_{t_0}(\gamma)} \\ &\leq \frac{1}{1-\gamma} + \left\lceil \frac{\ln N_{t_0}(\gamma)}{\ln(1+\eta)} \right\rceil \frac{n(1-\gamma)}{1-\gamma^{1/(1-\gamma)}}. \end{aligned}$$

We sum up and get the upper bound:

$$\mathbb{E}\left[\sum_{t=1}^n \mathbb{1}\{\bar{M}_t\} \mathbf{r}_t\right] \leq L\Upsilon_n B(\gamma) + 2L \frac{1}{1-\gamma} + 2L \left\lceil \frac{\ln N_{t_0}(\gamma)}{\ln(1+\eta)} \right\rceil \frac{n(1-\gamma)}{1-\gamma^{1/(1-\gamma)}}. \quad (4.24)$$

Third, we upper bound the second term in Eq. (4.22). The regret is caused by recommending a suboptimal item to the user and the user examines but does not click the item. Since there are  $\Upsilon_n$  breakpoints, we refer to  $[t_1, \dots, t_{\Upsilon_n}]$  as the time step of a breakpoint that occurs. We consider the time step in the individual epoch that does not contain a breakpoint. For any epoch and any time  $t \in \{t_e, t_e + 1, \dots, t_{e+1} - 1\}$ , any item  $a \in \bar{\mathcal{D}}_e$  and any item  $a^* \in \mathcal{D}_e^*$ , we define the event that item  $a$  is included in  $\mathbf{R}_t$  instead of item  $a^*$ , and item  $a$  is examined but not clicked at time  $t$  by:

$$G_{t,a,a^*} = \{\exists 1 \leq k < c_t \sim s.t. \sim \mathbf{R}_t(k) = a, \mathcal{R}_t(k) = a^*\}.$$

Since the attraction probability remains constant in the epoch, we refer to  $\mathcal{D}_e^*$  as the optimal items and  $\bar{\mathcal{D}}_e$  as the suboptimal items in epoch  $e$ . By [65, Theorem 1], the regret at time  $t$  is decomposed as:

$$\mathbb{E}[\mathbf{r}_t] \leq \Delta_{a,a^*}^t \sum_{a \in \bar{\mathcal{D}}_e} \sum_{a^* \in \mathcal{D}_e^*} \mathbb{1}\{G_{a,a^*,t}\}. \quad (4.25)$$

Then we have:

$$\mathbb{E} \left[ \sum_{t=t_i}^{t_{i+1}-1} \mathbb{1}\{\bar{M}_t\} \mathbf{r}_t \right] = \sum_{t=t_i}^{t_{i+1}-1} \mathbb{1}\{\bar{M}_t\} \mathbb{E}[\mathbf{r}_t] \leq \sum_{a \in \bar{\mathcal{D}}_e} \mathbb{E} \left[ \sum_{a^* \in \mathcal{D}_e^*} \sum_{t=t_i}^{t_{i+1}-1} \Delta_{a,a^*}^t \mathbb{1}\{G_{a,a^*,t}\} \right], \quad (4.26)$$

where the first equality is due to the tower rule, and the inequality is due to Eq. (4.25).

Now, for any suboptimal item  $a$  in epoch  $e$ , we upper bound  $\mathbb{E} \left[ \sum_{a^* \in \mathcal{D}_e^*} \sum_{t=t_i}^{t_{i+1}-1} \Delta_{a,a^*}^t \mathbb{1}\{G_{a,a^*,t}\} \right]$ . At time  $t$ , event  $\mathbb{1}\{\bar{M}_t\}$  and event  $\mathbb{1}\{a \in \mathbf{R}_t, a \in \bar{\mathcal{D}}_t\}$  happen when there exists an optimal item  $a^* \in \mathcal{D}_e^*$  such that:

$$\alpha_t(a) + 2c_t(\gamma, a) \geq \mathbf{U}_t(a) \geq \mathbf{U}(a^*) \geq \alpha_t(a^*),$$

which implies that  $2c_t(\gamma, a) \geq \alpha_t(a^*) - \alpha_t(a)$ . Taking the definition of the confidence interval, we get:

$$\mathbf{N}_t(\gamma, a) \leq \frac{16\epsilon \ln N_t(\gamma)}{\Delta_{t,a,a^*}^2},$$

where we set  $\Delta_{t,a,a^*} = \Delta_{a,a^*}^t$ .

Together with Eq. (4.26), we get:

$$\begin{aligned} & \mathbb{E} \left[ \sum_{t=t_i}^{t_{i+1}-1} \mathbb{1}\{\bar{M}_t\} \mathbf{r}_t \right] \\ & \leq \sum_{a \in \bar{\mathcal{D}}_e} \mathbb{E} \left[ \sum_{a^* \in \mathcal{D}_e^*} \frac{16\epsilon \ln N_t(\gamma)}{\Delta_{t,a,a^*}} \right] \\ & \leq 16\epsilon \ln N_t(\gamma) \left[ \Delta_{t,a,1} \frac{1}{\Delta_{t,a,1}^2} + \sum_{a^*=2}^K \Delta_{t,a,a^*} \left( \frac{1}{\Delta_{t,a,a^*}^2} - \frac{1}{\Delta_{t,a,a^*-1}^2} \right) \right] \\ & \leq \frac{32\epsilon \ln N_t(\gamma)}{\Delta_{t,a,K}}, \end{aligned} \quad (4.27)$$

## 4. Cascade Non-Stationary Bandits

---

where the last inequality is due to [64, Lemma 3]. Let  $\Delta_{a,K} = \min_{t \in [n]} \Delta_{t,a,K}$  be the smallest gap between the suboptimal item  $a$  and an optimal item in all time steps. When  $\mathbf{N}_t(\gamma, a) > \frac{32\epsilon \ln N_t(\gamma)}{\Delta_{a,K}^2}$ , CascadeDUCB will not select item  $a$  at time  $t$ . Thus we get:

$$\begin{aligned} & \sum_{a \in \mathcal{D}} \mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{\bar{M}_t\} \mathbb{1}\{a \in \mathbf{R}_t, a \in \bar{\mathcal{D}}_t\} \right] \\ & \leq \sum_{e \in [\Upsilon_n]} \sum_{a \in \bar{\mathcal{D}}_e} \frac{32\epsilon \ln N_t(\gamma)}{\Delta_{t,a,K}} \\ & \leq \sum_{a \in \mathcal{D}} \lceil n(1-\gamma) \rceil \frac{32\epsilon \ln N_n(\gamma)}{\Delta_{a,K}} \gamma^{1/(1-\gamma)}, \end{aligned} \quad (4.28)$$

where the last inequality is based on [31, Lemma 1].

Finally, together with Eqs. (4.22) to (4.26) and (4.28), we get Theorem 4.1.

### 4.7.2 Proof of Theorem 4.2

Let  $M_t = \{\exists a \in \mathcal{D} : |\alpha_t(a) - \bar{\alpha}_t(\tau, a)| > c_t(\tau, t)\}$  be the event that  $\alpha_t(a)$  falls out of the confidence interval around  $\bar{\alpha}_t(\tau, a)$  at time  $t$ , and let  $\bar{M}_t$  be the complement of  $M_t$ . We re-write the  $n$ -step regret of CascadeSWUCB as follows:

$$R(n) = \mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{M_t\} \mathbf{r}_t \right] + \mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{\bar{M}_t\} \mathbf{r}_t \right]. \quad (4.29)$$

We then bound both terms in Eq. (4.29) separately.

First, we refer to  $\mathcal{T}$  as the set of all time steps such that for  $t \in \mathcal{T}$ ,  $s \in [t-\tau, t]$  and any item  $a \in \mathcal{D}$  we have  $\alpha_s(a) = \alpha_t(a)$ . In other words,  $\mathcal{T}$  is the set of time steps that do not follow too close after breakpoints. Obviously, for any time step  $t \notin \mathcal{T}$  the estimators of attraction probabilities are biased and CascadeSWUCB may recommend suboptimal items constantly. Thus, we get the following bound:

$$\mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{M_t\} \mathbf{r}_t \right] \leq L \Upsilon_n \tau + \mathbb{E} \left[ \sum_{t \in \mathcal{T}} \mathbb{1}\{M_t\} \mathbf{r}_t \right]. \quad (4.30)$$

$\tau$  time steps after a breakpoint, the estimators of the attraction probabilities are not biased.

Then, we consider the variance. By using a Hoeffding-type inequality [30, Corollary

21], for an item  $a \in \mathcal{D}$ ,  $t \in \mathcal{T}$ , and any  $\eta > 0$ , we get:

$$\begin{aligned} P(|\bar{\alpha}_t(\tau, a) - \alpha_t(a)| > c_t(\tau, t)) &\leq P\left(\bar{\alpha}_t(\tau, a) > \alpha_t(a) + \sqrt{\frac{\epsilon \ln(t \wedge \tau)}{\mathbf{N}_t(\tau, a)}}\right) \\ &\quad + P\left(\bar{\alpha}_t(\tau, a) < \alpha_t(a) - \sqrt{\frac{\epsilon \ln(t \wedge \tau)}{\mathbf{N}_t(\tau, a)}}\right) \\ &\leq 2 \left[ \frac{\ln(t \wedge \tau)}{\ln(1 + \eta)} \right] \exp\left(-2\epsilon \ln(t \wedge \tau)(1 - \frac{\eta}{16})\right) \\ &= 2 \left[ \frac{\ln(t \wedge \tau)}{\ln(1 + \eta)} \right] (t \wedge \tau)^{-2\epsilon(1 - \eta^2/16)}. \end{aligned}$$

Taking  $\eta = 4\sqrt{1 - \frac{1}{2\epsilon}}$ , we have:  $P(|\bar{\alpha}_t(\tau, a) - \alpha_t(a)|) \leq 2^{\left[\frac{\ln(t \wedge \tau)}{\ln(1 + \eta)}\right]}$ . Thus, we get the following bound:

$$\begin{aligned} \mathbb{E}\left[\sum_{t \in \mathcal{T}} \mathbf{1}\{M_t\} \mathbf{r}_t\right] &\leq 2L \sum_{t \in \mathcal{T}} \frac{\left[\frac{\ln(t \wedge \tau)}{\ln(1 + \eta)}\right]}{t \wedge \tau} \\ &\leq \frac{L \ln^2(\tau)}{\ln(1 + 4\sqrt{1 - 1/2\epsilon})} + \frac{2Ln \ln \tau}{\tau \ln(1 + 4\sqrt{1 - 1/2\epsilon})}. \end{aligned}$$

We sum up and get the upper bound:

$$\begin{aligned} \mathbb{E}\left[\sum_{t=1}^n \mathbf{1}\{\bar{M}_t\} \mathbf{R}_t\right] &\leq L\Upsilon_n \tau + \frac{L \ln^2(\tau)}{\ln(1 + 4\sqrt{1 - 1/2\epsilon})} \\ &\quad + \frac{2Ln \ln \tau}{\tau \ln(1 + 4\sqrt{1 - 1/2\epsilon})}. \end{aligned} \tag{4.31}$$

Third, we upper bound the second term in Eq. (4.29). The regret is caused by recommending a suboptimal item to the user and the user examines but does not click the item. Since there are  $\Upsilon_n$  breakpoints, we refer to  $[t_1, \dots, t_{\Upsilon_n}]$  as the time steps of a breakpoint. We consider the time step in the individual epoch that does not contain a breakpoint. For any epoch and any time  $t \in \{t_e, t_e + 1, \dots, t_{e+1} - 1\}$ , any item  $a \in \bar{\mathcal{D}}_e$  and any item  $a^* \in \mathcal{D}_e^*$ , we define the event that item  $a$  is included in  $\mathbf{R}_t$  instead of item  $a^*$  and item  $a$  is examined but not clicked at time  $t$  by:

$$G_{t,a,a^*} = \{\exists 1 \leq k < \mathbf{c}_t \sim s.t. \sim \mathbf{R}_t(k) = a, \mathcal{R}_t(k) = a^*\}.$$

Since the attraction probabilities remain constant in the epoch, we refer to  $\mathcal{D}_e^*$  as the optimal items and  $\bar{\mathcal{D}}_e$  as the suboptimal items in epoch  $e$ . By [65, Theorem 1], the regret at time  $t$  is decomposed as:

$$\mathbb{E}[\mathbf{r}_t] \leq \Delta_{a,a^*}^t \sum_{a \in \bar{\mathcal{D}}_e} \sum_{a^* \in \mathcal{D}_e^*} \mathbf{1}\{G_{a,a^*,t}\}. \tag{4.32}$$

## 4. Cascade Non-Stationary Bandits

---

Then we have:

$$\begin{aligned} \mathbb{E} \left[ \sum_{t=t_i}^{t_{i+1}-1} \mathbb{1}\{\bar{M}_t\} \mathbf{r}_t \right] &= \sum_{t=t_i}^{t_{i+1}-1} \mathbb{1}\{\bar{M}_t\} \mathbb{E}[\mathbf{r}_t] \\ &\leq \sum_{a \in \bar{\mathcal{D}}_e} \mathbb{E} \left[ \sum_{a^* \in \mathcal{D}_e^*} \sum_{t=t_i}^{t_{i+1}-1} \Delta_{a,a^*}^t \mathbb{1}\{G_{a,a^*,t}\} \right], \end{aligned} \quad (4.33)$$

where the first equality is due to the tower rule, and the inequality if due to Eq. (4.32).

Now, for any suboptimal item  $a$  in epoch  $e$ , we upper bound

$$\mathbb{E} \left[ \sum_{a^* \in \mathcal{D}_e^*} \sum_{t=t_i}^{t_{i+1}-1} \Delta_{a,a^*}^t \mathbb{1}\{G_{a,a^*,t}\} \right].$$

At time  $t$ , event  $\mathbb{1}\{\bar{M}_t\}$  and event  $\mathbb{1}\{a \in \mathbf{R}_t, a \in \bar{\mathcal{D}}_t\}$  happen when there exists an optimal item  $a^* \in \mathcal{D}_e^*$  such that:

$$\alpha_t(a) + 2c_t(\tau, a) \geq \mathbf{U}_t(a) \geq \mathbf{U}(a^*) \geq \alpha_t(a^*),$$

which implies that  $2c_t(\tau, a) \geq \alpha_t(a^*) - \alpha_t(a)$ . Taking the definition of the confidence interval, we get:

$$\mathbf{N}_t(\tau, a) \leq \frac{4\epsilon \ln N_t(\tau)}{\Delta_{t,a,a^*}^2},$$

where we set  $\Delta_{t,a,a^*} = \Delta_{a,a^*}^t$ .

Together with Eq. (4.33), we get:

$$\begin{aligned} &\mathbb{E} \left[ \sum_{t=t_i}^{t_{i+1}-1} \mathbb{1}\{\bar{M}_t\} \mathbf{r}_t \right] \\ &\leq \sum_{a \in \bar{\mathcal{D}}_e} \mathbb{E} \left[ \sum_{a^* \in \mathcal{D}_e^*} \frac{4\epsilon \ln N_t(\gamma)}{\Delta_{t,a,a^*}} \right] \\ &\leq 4\epsilon \ln N_t(\gamma) \left[ \Delta_{t,a,1} \frac{1}{\Delta_{t,a,1}^2} + \sum_{a^*=2}^K \Delta_{t,a,a^*} \left( \frac{1}{\Delta_{t,a,a^*}^2} - \frac{1}{\Delta_{t,a,a^*-1}^2} \right) \right] \\ &\leq \frac{8\epsilon \ln N_t(\gamma)}{\Delta_{t,a,K}}, \end{aligned} \quad (4.34)$$

where the last inequality is due to [64, Lemma 3]. Let  $\Delta_{a,K} = \min_{t \in [n]} \Delta_{t,a,K}$  be the smallest gap between the suboptimal item  $a$  and an optimal item in all time steps. When

$N_t(\tau, a) > \frac{8\epsilon \ln N_t(\tau)}{\Delta_{a,K}^2}$ , CascadeDUCB will not select item  $a$  at time  $t$ . Thus we get:

$$\begin{aligned} & \sum_{a \in \mathcal{D}} \mathbb{E} \left[ \sum_{t=1}^n \mathbb{1}\{\bar{M}_t\} \mathbb{1}\{a \in \mathbf{R}_t, a \in \bar{\mathcal{D}}_t\} \right] \\ & \leq \sum_{e \in [\Upsilon_n]} \sum_{a \in \bar{\mathcal{D}}_e} \frac{8\epsilon \ln N_t(\tau)}{\Delta_{t,a,K}} \\ & \leq \sum_{a \in \mathcal{D}} \left\lceil \frac{n}{\tau} \right\rceil \frac{8\epsilon \ln(n \wedge \tau)}{\Delta_{a,K}}, \end{aligned}$$

where the last inequality is based on [30, Lemma 25].

Finally, together with Eqs. (4.29) to (4.31), (4.33) and (4.35), we get Theorem 4.2.

## 4.8 Additional Experiments

In this section, we compare CascadeDUCB, CascadeSWUCB and baselines on single queries. We pick 20 queries and report the results in Fig. 4.2. The results exemplify that CascadeDUCB and CascadeSWUCB have sub-linear regret while other baselines have linear regret.

## 4.9 Conclusion

In this chapter, we have answered **RQ3** by studying the Online Learning to Rank (OLTR) problem in a non-stationary environment where user preferences change abruptly. We focus on a widely-used user click behavior model Cascade Model (CM) and have proposed an online learning variant of it called *cascading non-stationary bandits*. Two algorithms, CascadeDUCB and CascadeSWUCB, have been proposed for solving it. Our theoretical have shown that they have sub-linear regret. These theoretical findings have been confirmed by our experiments on the Yandex click dataset. We open several future directions for non-stationary OLTR.

First, we have only considered the CM setup. Although a CM is powerful in explaining user behavior, it can only learn up to the first click, which may ignore part of user feedback. Other click models that can handle multiple clicks such as DCM [36] and DBN [20] may be considered as part of future work. We believe that our analysis can be adapted to those cases easily. Second, we focus on UCB1-based policy in building rankings. Another possibility is to use the family of softmax policies, which has originally been designed for adversarial bandits [10, 11]. Along this line, one may obtain upper bounds independent from the number of breakpoints.

#### 4. Cascade Non-Stationary Bandits

---

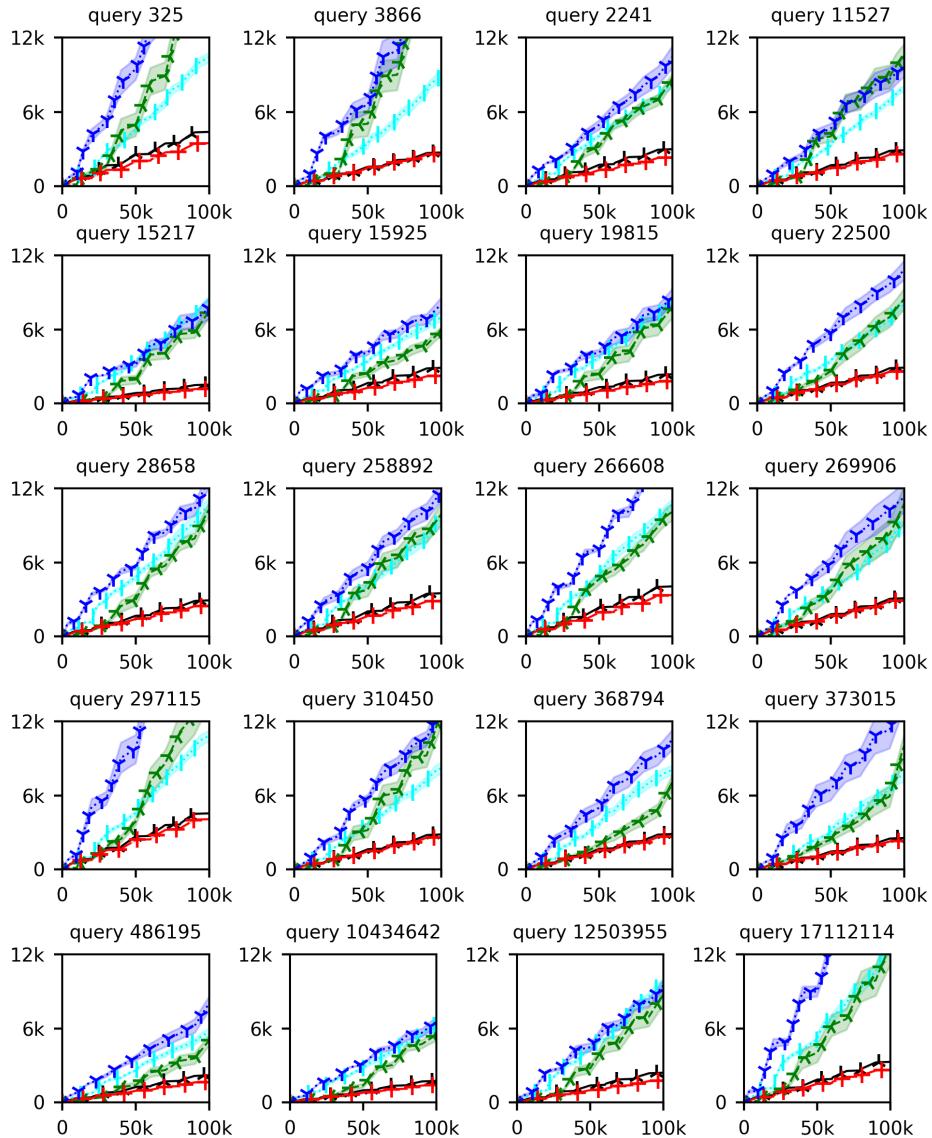


Figure 4.2: The n-step regret of CascadeDUCB (black), CascadeSWUCB (red), RankedEXP3 (cyan), CascadeKL-UCB (green) and BubbleRank (blue) on single queries in up to 100k steps. Lower is better. The x-axis is step  $n$ . The results are averaged over 10 runs per query. The shaded regions represent standard errors of our estimates.

# 5

## Online Learning to Rank for Relevance and Diversity

This chapter is devoted to answering the following question:

**RQ4** How to learn a ranker online considering both relevance and diversity?

We study an online learning setting that aims to recommend a ranked list with  $K$  items that maximizes the ranking utility, i.e., a list whose items are relevant and whose topics are diverse. We formulate it as the *cascade hybrid bandits* (CHB) problem. CHB assumes a cascading user behavior, where a user browses the displayed list from top to bottom, clicks the first attractive item, and stops browsing the rest. We propose a hybrid contextual bandit approach, called *CascadeHybrid*, for solving this problem. *CascadeHybrid* models item relevance and topical diversity using two independent functions and simultaneously learns those functions from user click feedback. We conduct experiments to evaluate *CascadeHybrid* on two real-world recommendation datasets: MovieLens and Yahoo music datasets. Our experimental results show that *CascadeHybrid* outperforms the baselines. In addition, we prove theoretical guarantees on the  $n$ -step performance demonstrating the soundness of *CascadeHybrid*.

### 5.1 Introduction

---

Ranking is at the heart of modern interactive systems, such as recommender and search systems. Learning to rank (LTR) addresses the ranking problem in such systems by using machine learning approaches [84]. Traditionally, LTR has been studied in an offline fashion, in which human labeled data is required [84]. Human labeled data is expensive to obtain, cannot capture future changes in user preferences, and may not well align with user needs [40]. To circumvent these limitations, recent work has shifted to learning directly from users' interaction feedback, e.g., clicks [42, 50, 131].

User feedback is abundantly available in interactive systems and is a valuable source for training online LTR algorithms [35]. When designing an algorithm to learn from this source, three challenges need to be addressed: (1) The learning algorithm should address position bias (the phenomenon that higher ranked items are more likely be observed than

---

This chapter was published as [78].

lower ranked items); (2) The learning algorithm should infer item relevance from user feedback and recommend lists containing relevant items (relevance ranking); (3) The recommended list should contain no redundant items and cover a broad range of topics (result diversification).

To address the position bias, a common approach is to make assumptions on the user’s click behavior and model the behavior using a click model [23]. The cascade model (CM) [27] is a simple but effective click model to explain user behavior. It makes the so-called *cascade assumption*, which assumes that a user browses the list from the first ranked item to the last one and clicks on the first attractive item and then stops browsing. The clicked item is considered to be positive, items before the click are treated as negative and items after the click will be ignored. Previous work has shown that the cascade assumption can explain the position bias effectively and several algorithms have been proposed under this assumption [38, 65, 74, 133].

In online LTR, the implicit signal that is inferred from user interactions is noisy [40]. If the learning algorithm only learns from these signals, it may reach a suboptimal solution where the optimal ranking is ignored simply because it is never exposed to users. This problem can be tackled by exploring new solutions, where the learning algorithm displays some potentially “good” rankings to users and obtains more signals. This behavior is called *exploration*. However, exploration may hurt the user experience. Thus, learning algorithms face an exploration vs. exploitation dilemma. Multi-armed bandit (MAB) [9, 71] algorithms are commonly used to address this dilemma. Along this line, multiple algorithms have been proposed [42, 77, 80]. They all address the dilemma in elegant ways and aim at recommending the top- $K$  most relevant items to users. However, only recommending the most relevant items may result in a list with redundant items, which diminishes the utility of the list and decreases user satisfaction [4, 120].

The submodular coverage model [91] can capture the pattern of diminishing utility and has been used in online LTR for diversified ranking. One assumption in this line of work is that items can be represented by a set of topics.<sup>1</sup> The task, then, is to recommend a list of items that ensures a maximal coverage of topics. Yue and Guestrin [120] develop an online feature-based diverse LTR algorithm by optimizing submodular utility models [120]. Hiranandani et al. [38] improve online diverse LTR by bringing the cascading assumption into the objective function. However, we argue that not all features that are used in a LTR setting can be represented by topics [84]. Previous online diverse LTR algorithms tend to ignore the relevance of individual items and may recommend a diversified list with less relevant items.

In this chapter, we address the aforementioned challenges and make four contributions:

- (1) We focus on a novel online LTR setting that targets both relevance ranking and result diversification. We formulate it as a Cascade Hybrid Bandits (CHB) problem, where the goal is to select  $K$  items from a large candidate set that maximize the utility of the ranked list (Section 5.3.1).

---

<sup>1</sup>In general, each topic may only capture a tiny aspect of the information of an item, e.g., a single phrase of a news title or a singer of a song [4, 120].

- (2) We propose CascadeHybrid, which utilizes a hybrid model, to solve this problem (Section 5.3.3).
- (3) We evaluate CascadeHybrid on two real-world recommendation datasets: MovieLens and Yahoo and show that CascadeHybrid outperforms state-of-the-art baselines (Section 5.4).
- (4) We theoretically analyze the performance of CascadeHybrid and provide guarantees on its proper behavior; moreover, we are the first to show that the regret bounds on feature-based ranking algorithms with the cascade assumption are linear in  $\sqrt{K}$ .

The rest of the chapter is organized as follows. We recapitulate the background knowledge in Section 5.2. In Section 5.3, we formulate the learning problem and propose our CascadeHybrid algorithm that optimizes both item relevance and list diversity. Section 5.4 contains our empirical evaluations of CascadeHybrid, comparing it with several state-of-the-art baselines. An analysis of the upper bound on the  $n$ -step performance of CascadeHybrid is presented in Section 5.5. In Section 5.6, we review related work. Conclusions are formulated in Section 5.7.

## 5.2 Background

---

In this section, we recapitulate the Cascade Model (CM), Cascading Bandits (CB), and the submodular coverage model. Throughout the chapter, we consider the ranking problem of  $L$  candidate items and  $K$  positions with  $K \leq L$ . We denote  $\{1, \dots, n\}$  by  $[n]$  and for the collection of items we write  $\mathcal{D} = [L]$ . A ranked list contains  $K \leq L$  items and is denoted by  $\mathcal{R} \in \Pi_K(\mathcal{D})$ , where  $\Pi_K(\mathcal{D})$  is the set of all permutations of  $K$  distinct items from the collection  $\mathcal{D}$ . The item at the  $k$ -th position of the list is denoted by  $\mathcal{R}(k)$  and, if  $\mathcal{R}$  contains an item  $i$ , the position of this item in  $\mathcal{R}$  is denoted by  $\mathcal{R}^{-1}(i)$ . All vectors are column vectors. We use bold font to indicate a vector and bold font with a capital letter to indicate a matrix. We write  $\mathbf{I}_d$  to denote the  $d \times d$  identity matrix and  $\mathbf{0}_{d \times m}$  the  $d \times m$  zero matrix.

### 5.2.1 Cascade model

Click models have been widely used to interpret user's interactive click behavior; cf. [23]. Briefly, a user is shown a ranked list  $\mathcal{R}$ , and then browses the list and leaves click feedback. Every click model makes unique assumptions and models a type of user interaction behavior. In this chapter, we consider a simple but widely used click model, the *cascade model* [27, 65, 74], which makes the cascade assumption about user behavior. Under the cascade assumption, a user browses a ranked list  $\mathcal{R}$  from the first item to the last one by one and clicks the first attractive item. After the click, the user stops browsing the remaining items. A click on an examined item  $\mathcal{R}(i)$  can be modeled as a Bernoulli random variable with a probability of  $\alpha(\mathcal{R}(i))$ , which is also called the *attraction probability*. Here, the Cascade Model (CM) assumes that each item attracts the user independent of other items in  $\mathcal{R}$ . Thus, the CM is parametrized by a set of

attraction probabilities  $\alpha \in [0, 1]^L$ . The examination probability of item  $\mathcal{R}(i)$  is 1 if  $i = 1$ , otherwise  $1 - \prod_{j=1}^{i-1} (1 - \alpha(\mathcal{R}(j)))$ .

With the CM, we translate the implicit feedback to training labels as follows: Given a ranked list, items ranked below the clicked item are ignored since none of them are browsed. Items ranked above the clicked item are negative samples and the clicked item is the positive sample. If no item is clicked, we know that all items are browsed but not clicked. Thus, all of them are negative samples.

The vanilla CM is only able to capture the first click in a session, and there are various extensions of CM to model multi-click scenarios; cf. [23]. However, we still focus on the CM, because it has been shown in multiple publications that the CM achieves good performance in both online and offline setups [23, 65, 77].

### 5.2.2 Cascading bandits

Cascading bandits (CB) are a type of online variant of the CM [65]. A CB is represented by a tuple  $(\mathcal{D}, K, P)$ , where  $P$  is a binary distribution over  $\{0, 1\}^L$ . The learning agent interacts with the CB and learns from the feedback. At each step  $t$ , the agent generates a ranked list  $\mathcal{R}_t \in \Pi_K(\mathcal{D})$  depending on observations in the previous  $t - 1$  steps and shows it to the user. The user browses the list with cascading behavior and leaves click feedback. Since the CM accepts at most one click, we write  $c_t \in [K + 1]$  as the click indicator, where  $c_t$  indicates the position of the click and  $c_t = K + 1$  indicates no click. Let  $A_t \in \{0, 1\}^L$  be the attraction indicator, where  $A_t$  is drawn from  $P$  and  $A_t(\mathcal{R}_t(i)) = 1$  indicates that item  $\mathcal{R}_t(i)$  attracts the user at step  $t$ . The number of clicks at step  $t$  is considered as the reward and computed as follows:

$$r(\mathcal{R}_t, A_t) = 1 - \prod_{i=1}^K (1 - A_t(\mathcal{R}_t(i))). \quad (5.1)$$

Then, we assume that the attraction indicators of items are distributed independently as Bernoulli variables:

$$P(A) = \prod_{i \in \mathcal{D}} P_{\alpha(i)}(A(i)), \quad (5.2)$$

where  $P_{\alpha(i)}(\cdot)$  is the Bernoulli distribution with mean  $\alpha(i)$ . The expected number of clicks at step  $t$  is computed as  $\mathbb{E}[r(\mathcal{R}_t, A_t)] = r(\mathcal{R}_t, \alpha)$ . The goal of the agent is to maximize the expected number of clicks in  $n$  steps or minimize the expected  $n$ -step regret:

$$R(n) = \sum_{t=1}^n \mathbb{E} \left[ \max_{\mathcal{R} \in \Pi_K(\mathcal{D})} r(\mathcal{R}, \alpha) - r(\mathcal{R}_t, A_t) \right]. \quad (5.3)$$

CB has several variants depending on assumptions on the attraction probability  $\alpha$ . Briefly, cascade linear bandits [133] assume that an item  $a$  is represented by a feature vector  $\mathbf{z}_a \in \mathbb{R}^m$  and that the attraction probability of an item  $a$  to a user is a linear combination of features:  $\alpha(a) \approx \mathbf{z}_a^T \beta^*$ , where  $\beta^* \in \mathbb{R}^m$  is an unknown parameter. With this assumption, the attraction probability of an item is independent of other items in the list, and this assumption is used in relevance ranking problems. CascadeLinUCB has been proposed to solve this problem. For other problems, Hiranandani et al. [38]

assume the attraction probability to be submodular, and propose CascadeLSB to solve result diversification.

### 5.2.3 Submodular coverage model

Before we recapitulate the submodular function, we introduce two properties of a diversified ranking. Different from the relevance ranking, in a diversified ranking, the utility of an item depends on other items in the list. Suppose we focus on news recommendation. Items that we want to rank are news items, and each news item covers a set of topics, e.g., weather, sports, politics, a celebrity, etc. We want to recommend a list that covers a broad range of topics. Intuitively, adding a news item to a list does not decrease the number of topics that are covered by the list, but adding a news item to a list that covers highly overlapping topics might not bring much extra benefit to the list. The first property can be thought of as a monotonicity property, and the second one is the notion of diminishing gain in the utility. They can be captured by the submodular function [120].

We introduce two properties of submodular functions. Let  $g(\cdot)$  be a set function, which maps a set to a real value. We say that  $g(\cdot)$  is monotone and submodular if given two item sets  $\mathcal{A}$  and  $\mathcal{B}$ , where  $\mathcal{B} \subseteq \mathcal{A}$ , and an item  $a$ ,  $g(\cdot)$  has the following two properties:

$$\begin{aligned} \text{monotonicity} : g(\mathcal{A} \cup \{a\}) &\geq g(\mathcal{A}); \\ \text{submodularity} : g(\mathcal{B} \cup \{a\}) - g(\mathcal{B}) &\geq g(\mathcal{A} \cup \{a\}) - g(\mathcal{A}). \end{aligned}$$

In other words, the gain in utility of adding an item  $a$  to a subset of  $\mathcal{A}$  is larger than or equal to that of adding an item to  $\mathcal{A}$ , and adding an item  $a$  to  $\mathcal{A}$  does not decrease the utility. Monotonicity and submodularity together provide a natural framework to capture the properties of a diversified ranking. The shrewd reader may notice that a linear function is a special case of submodular functions, where only the inequalities in *monotonicity* and *submodularity* hold. However, as discussed above, the linear model assumes that the attraction probability of an item is independent of other items: it cannot capture the diminishing gain in the result diversification. In the rest of this section, we introduce the *probabilistic coverage model*, which is a widely used submodular function for result diversification [4, 7, 38, 94, 120].

Suppose that an item  $a \in \mathcal{D}$  is represented by a  $d$ -dimensional vector  $\mathbf{x}_a \in [0, 1]^d$ . Each entry of the vector  $\mathbf{x}_a(j)$  describes the probability of item  $a$  covering topic  $j$ . Given a list  $\mathcal{A}$ , the probability of  $\mathcal{A}$  covering topic  $j$  is

$$g_j(\mathcal{A}) = 1 - \prod_{a \in \mathcal{A}} (1 - \mathbf{x}_a(j)). \quad (5.4)$$

The gain in topic coverage of adding an item  $a$  to  $\mathcal{A}$  is:

$$\Delta(a \mid \mathcal{A}) = (\Delta_1(a \mid \mathcal{A}), \dots, \Delta_d(a \mid \mathcal{A})), \quad (5.5)$$

where  $\Delta_j(a \mid \mathcal{A}) = g_j(\mathcal{A} \cup \{a\}) - g_j(\mathcal{A})$ . With this model, the attraction probability of the  $i$ -th item in a ranked list  $\mathcal{R}$  is defined as:

$$\alpha(\mathcal{R}(i)) = \boldsymbol{\omega}_{\mathcal{R}(i)}^T \boldsymbol{\theta}^*, \quad (5.6)$$

where  $\omega_{\mathcal{R}(i)} = \Delta(\mathcal{R}(i) | (\mathcal{R}(1), \dots, \mathcal{R}(i-1)))$  and  $\theta^*$  is the unknown user preference to different topics [38]. In Eq. (5.6), the attraction probability of an item depends on the items ranked above it;  $\alpha(\mathcal{R}(i))$  is small if  $\mathcal{R}(i)$  covers similar topics as higher ranked items. CascadeLSB [38] has been proposed to solve cascading bandits with this type of attraction probability and aims at building diverse ranked lists.

## 5.3 Algorithm

In this section, we first formulate our online learning to rank problem, and then propose CascadeHybrid to solve it.

### 5.3.1 Problem formulation

We study a variant of cascading bandits, where the attraction probability of an item in a ranked list depends on two aspects: item relevance and item novelty. Item relevance is independent of other items in the list. Novelty of an item depends on the topics covered by higher ranked items; a novel item brings a large gain in the topic coverage of the list, i.e., a large value in Eq. (5.6). Thus, given a ranked list  $\mathcal{R}$ , the attraction probability of item  $\mathcal{R}(i)$  is defined as follows:

$$\alpha(\mathcal{R}(i)) = \mathbf{z}_{\mathcal{R}(i)}^T \beta^* + \omega_{\mathcal{R}(i)}^T \theta^*, \quad (5.7)$$

where  $\omega_{\mathcal{R}(i)} = \Delta(\mathcal{R}(i) | (\mathcal{R}(1), \dots, \mathcal{R}(i-1)))$  is the topic coverage gain discussed in Section 5.2.3, and  $\mathbf{z}_{\mathcal{R}(i)} \in \mathbb{R}^m$  is the relevance feature,  $\theta^* \in \mathbb{R}^d$  and  $\beta^* \in \mathbb{R}^m$  are two unknown parameters that characterize the user preference. In other words, the attraction probability is a hybrid of a modular (linear) function parameterized by  $\beta^*$  and a submodular function parameterized by  $\theta^*$ .

Now, we define our learning problem, Cascade Hybrid Bandits (CHB), as a tuple  $(\mathcal{D}, \theta^*, \beta^*, K)$ . Here,  $\mathcal{D} = [L]$  is the item candidate set and each item  $a$  can be represented by a feature vector  $[\mathbf{x}_a^T, \mathbf{z}_a^T]^T$ , where  $\mathbf{x}_a \in [0, 1]^d$  is the topic coverage of item  $a$  discussed in Section 5.2.3.  $K$  is the number of positions. The action space for the problem are all permutations of  $K$  individual items from  $\mathcal{D}$ ,  $\Pi_K(\mathcal{D})$ . The reward of an action at step  $t$  is the number of clicks, defined in Eq. (5.1). Together with Eqs. (5.1), (5.2) and (5.7), the expectation of reward at step  $t$  is computed as follows:

$$\mathbb{E}[r(\mathcal{R}_t, A_t)] = 1 - \prod_{a \in \mathcal{R}_t} (1 - \mathbf{z}_a^T \beta^* - \omega_a^T \theta^*). \quad (5.8)$$

In the rest of the chapter, we write  $r(\mathcal{R}_t) = \mathbb{E}[r(\mathcal{R}_t, A_t)]$  for short. And the goal of the learning agent is to maximize the reward or, equivalently, to minimize the  $n$ -step regret defined as follow:

$$R(n) = \sum_{t=1}^n \left[ \max_{\mathcal{R} \in \Pi_K(\mathcal{D})} r(\mathcal{R}) - r(\mathcal{R}_t) \right]. \quad (5.9)$$

The previously proposed CascadeLinUCB [133] cannot solve CHB since it only handles the linear part of the attraction probability. CascadeLSB [38] cannot solve CHB, either,

because it only uses one submodular function and handles the submodular part of the attraction probability. Thus, we need to extend the previous models or, in other words, propose a new hybrid model that can handle both linear and submodular properties in the attraction probability.

### 5.3.2 Competing with a greedy benchmark

Finding the optimal set that maximizes the utility of a submodular function is an NP-hard problem [91]. In our setup, the attraction probability of each item also depends on the order in the list. To the best of our knowledge, we cannot find the optimal ranking

$$\mathcal{R}^* = \arg \max_{\mathcal{R} \in \Pi_K(\mathcal{D})} r(\mathcal{R}) \quad (5.10)$$

efficiently. Thus, we compete with a greedy benchmark that approximates the optimal ranking  $\mathcal{R}^*$ . The greedy benchmark chooses the items that have the highest attraction probability given the higher ranked items: for any positions  $k \in [K]$ ,

$$\tilde{\mathcal{R}}(k) = \arg \max_{a \in \mathcal{D} \setminus \{\tilde{\mathcal{R}}(1), \dots, \tilde{\mathcal{R}}(k-1)\}} \mathbf{z}_a^T \boldsymbol{\beta}^* + \omega_a^T \boldsymbol{\theta}^*, \quad (5.11)$$

where  $\tilde{\mathcal{R}}(k)$  is the ranked list generated by the benchmark.

This greedy benchmark has been used in previous literature [38, 120]. As shown by Hiranandani et al. [38], in the CM, the greedy benchmark is at least a  $\eta$ -approximation of  $\mathcal{R}^*$ . That is,

$$r(\tilde{\mathcal{R}}) \geq \eta r(\mathcal{R}^*), \quad (5.12)$$

where  $\eta = (1 - \frac{1}{e}) \max\{\frac{1}{K}, 1 - \frac{K-1}{2} \alpha_{max}\}$  with  $\alpha_{max} = \max_{a \in \mathcal{D}} \mathbf{z}_a^T \boldsymbol{\beta}^* + \mathbf{x}_a^T \boldsymbol{\theta}^*$ . In the rest of the chapter, we focus on competing with this greedy benchmark.

### 5.3.3 CascadeHybrid

We propose CascadeHybrid to solve the CHB. As the name suggests, the algorithm is a hybrid of a linear function and a submodular function. The linear function is used to capture item relevance and the submodular function to capture diversity in topics. CascadeHybrid has access to item features,  $[\mathbf{x}_a^T, \mathbf{z}_a^T]^T$ , and uses the probabilistic coverage model to compute the gains in topic coverage. The user preferences  $\boldsymbol{\theta}^*$  and  $\boldsymbol{\beta}^*$  are unknown to CascadeHybrid. They are estimated from interactions with users. The only tunable hyperparameter for CascadeHybrid is  $\gamma \in \mathbb{R}_+$ , which controls exploration: a larger value of  $\gamma$  means more exploration.

The details of CascadeHybrid are provided in Algorithm 6. At the beginning of each step  $t$  (line 5), CascadeHybrid estimates the user preference as  $\hat{\boldsymbol{\theta}}_t$  and  $\hat{\boldsymbol{\beta}}_t$  based on the previous  $t - 1$  step observations.  $\hat{\boldsymbol{\theta}}_t$  and  $\hat{\boldsymbol{\beta}}_t$  can be viewed as maximum likelihood estimators on the rewards,<sup>2</sup> where  $\mathbf{M}_t$ ,  $\mathbf{H}_t$ ,  $\mathbf{B}_t$  and  $\mathbf{y}_t$ ,  $\mathbf{u}_t$  summarize the features and click feedback of all observed items in the previous  $t - 1$  steps. Then, CascadeHybrid builds the ranked list  $\mathcal{R}_t$ , sequentially (lines 7–14). In particular, for each position  $k$ ,

<sup>2</sup>The derivation is based on matrix block-wise inversion. We omit the derivation since it is not a major contribution of this chapter.

---

**Algorithm 6** CascadeHybrid

---

**Input:**  $\gamma$

- 1: {Initialization}
- 2:  $\mathbf{H}_1 \leftarrow \mathbf{I}_d, \mathbf{u}_1 \leftarrow \mathbf{0}_d, \mathbf{M}_1 \leftarrow \mathbf{I}_m, \mathbf{y}_1 \leftarrow \mathbf{0}_m, \mathbf{B}_1 = \mathbf{0}_{d \times m}$
- 3: **for**  $t = 1, 2, \dots, n$  **do**
- 4:   {Estimate parameters}
- 5:    $\hat{\boldsymbol{\theta}}_t \leftarrow \mathbf{H}_t^{-1} \mathbf{u}_t, \hat{\boldsymbol{\beta}}_t \leftarrow \mathbf{M}_t^{-1} (\mathbf{y}_t - \mathbf{B}_t^T \mathbf{H}_t^{-1} \mathbf{u}_t)$
- 6:   {Build ranked list}
- 7:    $\mathcal{S}_0 \leftarrow \emptyset$
- 8:   **for**  $k = 1, 2, \dots, K$  **do**
- 9:     **for**  $a \in \mathcal{D} \setminus \mathcal{S}_{k-1}$  **do**
- 10:        $\omega_a \leftarrow \Delta(\mathbf{x}_a | \mathcal{S}_{k-1})$  {Recalculate the topic coverage gain. }
- 11:        $\mu_a \leftarrow \text{Eq. (5.13)}$  {Compute UCBs.}
- 12:        $a_k^t \leftarrow \arg \max_{a \in \mathcal{D} \setminus \mathcal{S}_{k-1}} \mu_a$
- 13:        $\mathcal{S}_k \leftarrow \mathcal{S}_{k-1} + a_k^t$
- 14:      $\mathcal{R}_t = (a_1^t, \dots, a_K^t)$  {Ranked list}
- 15:     Display  $\mathcal{R}_t$  and observe click feedback  $c_t \in [K+1]$
- 16:      $k_t \leftarrow \min(K, c_t)$
- 17:     {Update statistics}
- 18:      $\mathbf{H}_t \leftarrow \mathbf{H}_t + \mathbf{B}_t \mathbf{M}_t^{-1} \mathbf{B}_t^T, \mathbf{u}_t \leftarrow \mathbf{u}_t + \mathbf{B}_t \mathbf{M}_t^{-1} \mathbf{y}_t$
- 19:     **for**  $a \in \mathcal{R}_t (1 : k_t)$  **do**
- 20:        $\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t + \mathbf{z}_a \mathbf{z}_a^T, \mathbf{B}_{t+1} \leftarrow \mathbf{B}_t + \omega_a \mathbf{z}_a^T, \mathbf{H}_t \leftarrow \mathbf{H}_t + \omega_a \omega_a^T$
- 21:       **if**  $c_t \leq K$  **then**
- 22:          $\mathbf{y}_{t+1} \leftarrow \mathbf{y}_t + \mathbf{z}_{\mathcal{R}_t(c_t)}, \mathbf{u}_t \leftarrow \mathbf{u}_t + \omega_{\mathcal{R}_t(c_t)}$
- 23:      $\mathbf{H}_{t+1} \leftarrow \mathbf{H}_t - \mathbf{B}_{t+1} \mathbf{M}_{t+1}^{-1} \mathbf{B}_{t+1}^T, \mathbf{u}_{t+1} \leftarrow \mathbf{u}_t - \mathbf{B}_{t+1} \mathbf{M}_{t+1}^{-1} \mathbf{y}_{t+1}$

---

we recalculate the topic coverage gain of each item (line 7). The new gains are used to estimate the attraction probability of items. CascadeHybrid makes an optimistic estimate of the attraction probability of each item (lines 9–11) and chooses the one with the highest estimated attraction probability (line 12). This is known as the principle of optimism in the face of uncertainty [9], and the estimator for an item  $a$  is called the Upper Confidence Bound (UCB):

$$\mu_a = \omega_a^T \hat{\boldsymbol{\theta}}_t + \mathbf{z}_a^T \hat{\boldsymbol{\beta}}_t + \gamma \sqrt{s_a}, \quad (5.13)$$

with

$$s_a = \omega_a^T \mathbf{H}_t^{-1} \omega_a - 2 \omega_a^T \mathbf{H}_t^{-1} \mathbf{B}_t \mathbf{M}_t^{-1} \mathbf{z}_a + \mathbf{z}_a \mathbf{M}_t^{-1} \mathbf{z}_a + \mathbf{z}_a \mathbf{M}_t^{-1} \mathbf{B}_t^T \mathbf{H}_t^{-1} \mathbf{B}_t \mathbf{M}_t^{-1} \mathbf{z}_a. \quad (5.14)$$

Finally, CascadeHybrid displays the ranked list  $\mathcal{R}_t$  to the user and collects click feedback (lines 15–23). Since CascadeHybrid only accepts one click, we use  $c_t \in [K+1]$  to indicate the position of the click;<sup>3</sup>  $c_t = K+1$  indicates that no item in  $\mathcal{R}_t$  is clicked.

---

<sup>3</sup>For multiple-click cases, we only consider the first click and keep the rest of CascadeHybrid the same.

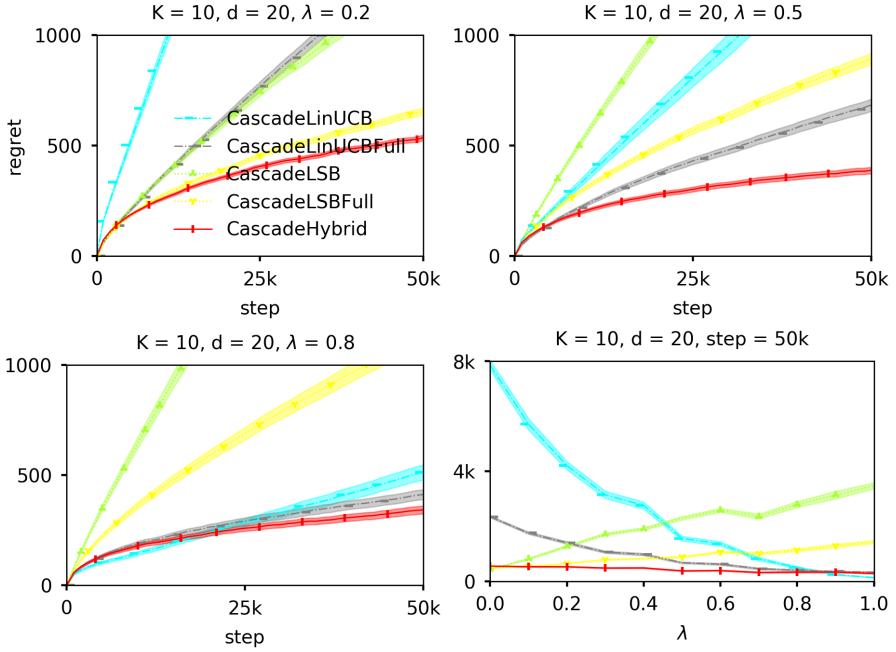


Figure 5.1:  $n$ -step regret on the MovieLens dataset with different  $\lambda$ . Results are averaged over 500 users with 2 repeats per user. Lower regret means more clicks received by the algorithm during the online learning. Shaded areas are the standard errors of estimates.

### 5.3.4 Computational complexity

The main computational cost of Algorithm 6 is incurred by computing matrix inverses, which is cubic in the dimensions of the matrix. However, in practice, we can use the Woodbury matrix identity [33] to update  $\mathbf{H}_t^{-1}$  and  $\mathbf{M}_t^{-1}$  instead of  $\mathbf{H}_t$  and  $\mathbf{M}_t$ , which is square in the dimensions of the matrix. Thus, computing the UCB of each item is  $O(m^2 + d^2)$ . As CascadeHybrid greedily chooses  $K$  items out of  $L$ , the per-step computational complexity of CascadeHybrid is  $O(LK(m^2 + d^2))$ .

## 5.4 Experiments

This section starts with the experimental setup, where we first introduce the datasets, click simulator and baselines. After that we report our experimental results.

### 5.4.1 Experimental setup

Off-policy evaluation [80] is an approach to evaluate interaction algorithms without live experiments. However, in our problem, the action space is exponential in  $K$ , which is too large for commonly used off-policy evaluation methods. As an alternative, we

evaluate the CascadeHybrid in a simulated interaction environment, where the simulator is built based on offline datasets. This is a commonly used evaluation setup in the literature [38, 65, 133].

**Datasets.** We evaluate CascadeHybrid on two datasets: MovieLens 20M [37] and Yahoo.<sup>4</sup> The MovieLens dataset contains 20M ratings on 27k movies by 138k users, with 20 genres.<sup>5</sup> Each movie belongs to at least one genre. The Yahoo dataset contains over 700M ratings of 136k songs given by 1.8M users and genre attributes of each song; we consider the top level attribute, which has 20 different genres; each song belongs to a single genre. All the ratings in the two datasets are on a 5-point scale. All movies and songs are considered as items and genres are considered as topics.

**Data preprocessing.** We follow the data preprocessing approach in [38, 81, 133]. First, we extract the 1k most active users and the 1k most rated items. Let  $\mathcal{U} = [1000]$  be the user set, and  $\mathcal{D} = [1000]$  be the item set. Then, the ratings are mapped onto a binary scale: rating 5 is converted to 1 and others to 0. After this mapping, in the MovieLens dataset, about 7% of the user-item pairs get rating 1, and, in the Yahoo dataset, about 11% of user-item pairs get rating 1. Then, we use the matrix  $\mathbf{F} \in \{0, 1\}^{|\mathcal{U}| \times |\mathcal{D}|}$  to capture the converted ratings and  $\mathbf{G} \in \{0, 1\}^{|\mathcal{D}| \times d}$  to record the items and topics, where  $d$  is the number of topics and each entry  $\mathbf{G}_{jk} = 1$  indicates that item  $j$  belongs to topic  $k$ .

**Click simulator.** In our experiments, the click simulator follows the cascade assumption, and considers both item relevance and diversity of the list. To design such a simulator, we combine the simulators used in [81] and [38]. Because of the cascading assumption, we only need to define the way of computing attraction probabilities of items in a list.

We first divide the users into training and test groups evenly, i.e.,  $\mathbf{F}_{train}$  and  $\mathbf{F}_{test}$ . The training group is used to estimate features of items used by online algorithms, while the test group are used to define the click simulator. This is to mimic the real-world scenarios that online algorithms estimate user preferences without knowing the perfect topic coverage of items. Then, we follow [81] to obtain the relevance part of the attraction probability, i.e.,  $\mathbf{z}$  and  $\beta^*$ , and the process in [38] to get the topic coverage of items,  $\mathbf{x}$ , and the user preferences on topics,  $\theta^*$ .

In particular, the relevance features  $\mathbf{z}$  are obtained by conducting singular-value decomposition on  $\mathbf{F}_{train}$ . We pick the 10 largest singular values and thus the dimension of relevance features is  $m = 10$ . Then, we normalize each relevance feature by the transformation:  $\mathbf{z}_a \leftarrow \frac{\mathbf{z}_a}{\|\mathbf{z}_a\|_2}$ , where  $\|\mathbf{z}_a\|_2$  is the L2 norm of  $\mathbf{z}_a$ . The user preference  $\beta^*$  is computed by solving the least square on  $\mathbf{F}_{test}$  and then  $\beta^*$  is normalized by the same transformation. Note that  $\forall a \in \mathcal{D} : \mathbf{z}_a^T \beta^* \in [0, 1]$ , since  $\|\mathbf{z}_a\|_2 = 1$  and  $\|\beta^*\|_2 = 1$ .

Then, we follow the process in [38]. If item  $a$  belongs to topic  $j$ , we compute the topic coverage of item  $a$  to topic  $j$  as the quotient of the number of users rating item  $a$  to

---

<sup>4</sup>R2 - Yahoo! Music User Ratings of Songs with Artist, Album, and Genre Meta Information, v. 1.0 <https://webscope.sandbox.yahoo.com/catalog.php?datatype=r>

<sup>5</sup>In both datasets, one of the 20 genres is called *unknown*.

be attractive to the number of users who rate at least one item in topic  $j$  to be attractive:

$$\mathbf{x}_{a,j} = \frac{\sum_{u \in \mathcal{U}} F_{u,a} G_{a,j}}{\sum_{u \in \mathcal{U}} \mathbb{1}\{\exists a' \in \mathcal{D} : F_{u,a'} G_{a',j} > 0\}}. \quad (5.15)$$

Given user  $u$ , the preference for topic  $j$  is computed as the number of items rated to be attractive in topic  $j$  over the number of items in all topics rated by  $u$  to be attractive:

$$\boldsymbol{\theta}_j^* = \frac{\sum_{a \in \mathcal{D}} F_{u,a} G_{a,j}}{\sum_{j' \in [d]} \sum_{a' \in \mathcal{D}} F_{u,a'} G_{a',j'}}. \quad (5.16)$$

For some cases, we may have  $\exists a : \sum_{j \in [d]} \mathbf{x}_{a,j} > 1$  and thus  $\mathbf{x}_a^T \boldsymbol{\theta}^* > 1$ . However, given the high sparsity in our datasets, we have  $\mathbf{x}_a^T \boldsymbol{\theta}^* \in [0, 1]$  for all items during our experiments.

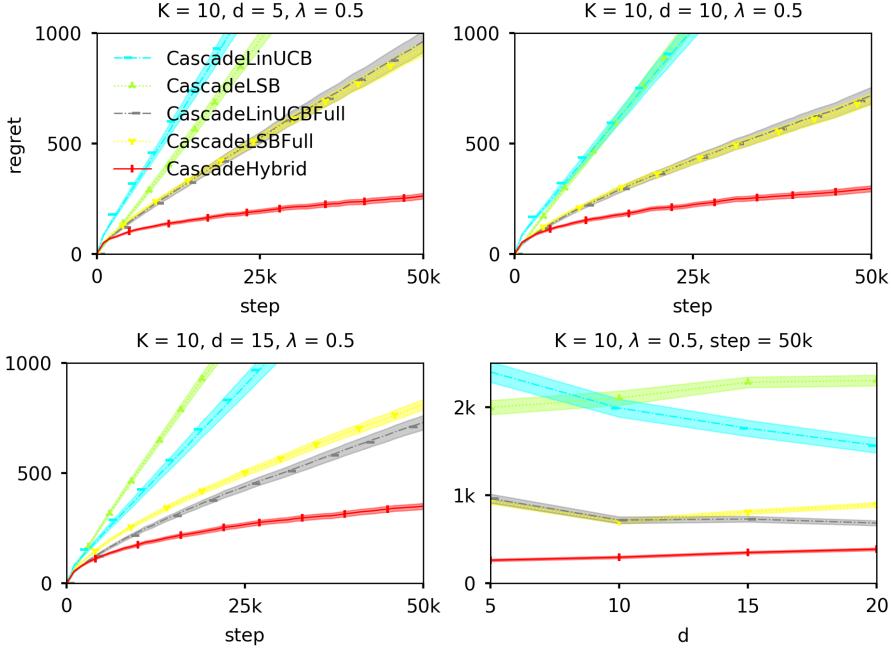
Finally, we combine the two parts and obtain the attraction probability used in our click simulator. To simulate different types of user preferences, we introduce a trade-off parameter  $\lambda \in [0, 1]$ , which is unknown to online algorithms, and compute the attraction probability of the  $i$ th item in  $\mathcal{R}$  as follows:

$$\alpha(\mathcal{R}(i)) = \lambda \mathbf{z}_{\mathcal{R}(i)}^T \boldsymbol{\beta}^* + (1 - \lambda) \mathbf{w}_{\mathcal{R}(i)}^T \boldsymbol{\theta}^*. \quad (5.17)$$

By changing the value of  $\lambda$ , we simulate different types of user preference: a larger value of  $\lambda$  means that the user prefers items to be relevant; a smaller value of  $\lambda$  means that the user prefers the topics in the ranked list to be diverse.

**Baselines.** We compare CascadeHybrid to two online algorithms, each of which has two configurations. In total, we have four baselines, namely CascadeLinUCB and CascadeLinUCBFull [133], and CascadeLSB and CascadeLSBFull [38]. The first two only consider relevance ranking. The differences are that CascadeLinUCB takes  $\mathbf{z}$  as the features, while CascadeLinUCBFull takes  $\{\mathbf{x}, \mathbf{z}\}$  as the features. CascadeLSB and CascadeLSBFull only consider the result diversification, where CascadeLSB takes  $\mathbf{x}$  as features, while CascadeLSBFull takes  $\{\mathbf{x}, \mathbf{z}\}$  as features. CascadeLinUCBFull and CascadeLinUCB are expected to perform well when  $\lambda \rightarrow 1$ , and that CascadeLSB and CascadeLSBFull perform well when  $\lambda \rightarrow 0$ . For all baselines, we set the exploration parameter  $\gamma = 1$  and the learning rate to 1. This parameter setup is used in [120], which leads to better empirical performance. We also set  $\gamma = 1$  for CascadeHybrid.

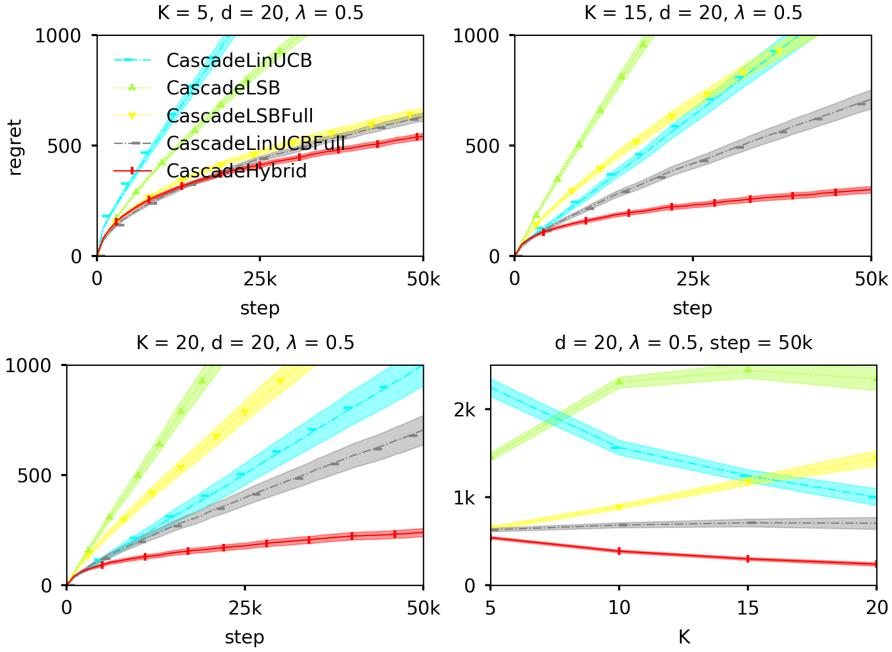
We report the cumulative regret, Eq. (5.9), within 50k steps, called *n-step regret*. The *n*-step regret is commonly used to evaluate bandit algorithms [38, 74, 120, 133]. In our setup, it measures the difference in number of received clicks between the oracle that knows the ideal  $\boldsymbol{\beta}^*$  and  $\boldsymbol{\theta}^*$  and the online algorithm, e.g., CascadeHybrid, in  $n$  steps. The lower regret means the more clicks received by the algorithm. We conduct our experiments with 500 users from the test group and 2 repeats per user. In total, the results are averaged over 1k repeats. We also include the standard errors of our estimates. To show the impact of different factors on the performance of online LTR algorithms, we choose  $\lambda \in \{0.0, 0.1, \dots, 1.0\}$ , the number of positions  $K \in \{5, 10, 15, 20\}$ , and the number of topics  $d \in \{5, 10, 15, 20\}$ . For the number of topics  $d$ , we choose the topics with the maximum number of items.


 Figure 5.2:  $n$ -step regret on the MovieLens dataset with different topics  $d$ .

### 5.4.2 Experimental results

We first study the movie recommendation task on the MovieLens dataset and show the results in Figs. 5.1 to 5.3. Fig. 5.1 shows the impact of  $\lambda$ , where we fix  $K = 10$  and  $d = 20$ .  $\lambda$  is a trade-off parameter in our simulation. It balances the relevance and diversity, and is unknown to online algorithms. Choosing a small  $\lambda$ , the simulated user prefers recommended movies in the list to be relevant, while choosing a large  $\lambda$ , the simulated user prefers the recommended movies to be diverse. As shown in the top row, CascadeHybrid outperforms all baselines when  $\lambda \in \{0.1, 0.2, \dots, 0.8\}$ , and only loses to the particularly designed baselines (CascadeLSB and CascadeLinUCB) with small gaps in some extreme cases where they benefit most. This is reasonable since they have fewer parameters to be estimated than CascadeHybrid. In all cases, CascadeHybrid has lower regret than CascadeLinUCBFull and CascadeLSBFULL that work with the same features as CascadeHybrid. This result indicates that including more features in CascadeLSB and CascadeLinUCB is not sufficient to capture both item relevance and result diversification.

Fig. 5.2 shows the impact of different numbers of topics, where we fix  $\lambda = 0.5$  and  $K = 10$ . We see that CascadeHybrid outperforms all baselines with large gaps. In the last plot of the middle row, we see that the gap of regret between CascadeHybrid and CascadeLinUCBFull decreases with larger values of  $d$ . This is because, on the MovieLens dataset, when  $d$  is small, a user tends to prefer a diverse ranked list: when  $d$  is small, an item is more likely to belong to only one topic, and each entry of  $\theta_j^*$

Figure 5.3:  $n$ -step regret on the MovieLens dataset with positions  $K$ .

becomes relatively larger since  $\sum_{j \in [d]} \theta_j^* = 1$ . And given an item  $a$  and a set  $\mathcal{S}$ , the difference between  $\Delta(a|\mathcal{S})^T \theta^*$  and  $\Delta(a|\emptyset)^T \theta^*$  is large. This behavior is also confirmed by the fact that CascadeLinUCBFull outperforms CascadeLSBFull for large  $d$  while they perform similarly for small  $d$ . Finally, we study the impact of the number of positions on the regret. The results are displayed in Fig. 5.3, where we choose  $\lambda = 0.5$  and  $d = 20$ . Again, we see that CascadeHybrid outperforms baselines with large gaps.

Next, we report the results on the Yahoo dataset in Figs. 5.4 to 5.6. We follow the same setup as for the MovieLens dataset and observe a similar behavior. CascadeHybrid has slightly higher regret than the best performing baselines in three cases: CascadeLSB when  $\lambda = 0$  and CascadeLinUCB when  $\lambda \in \{0.9, 1\}$ . Note that these are relatively extreme cases, where the particularly designed baselines can benefit most. Meanwhile, CascadeLSB and CascadeLinUCB do not generalize well with different  $\lambda$ s. In all setups, CascadeHybrid has lower regret than CascadeLSBFull and CascadeLinUCBFull, which confirms our hypothesis that the hybrid model has benefit in capturing both relevance and diversity.

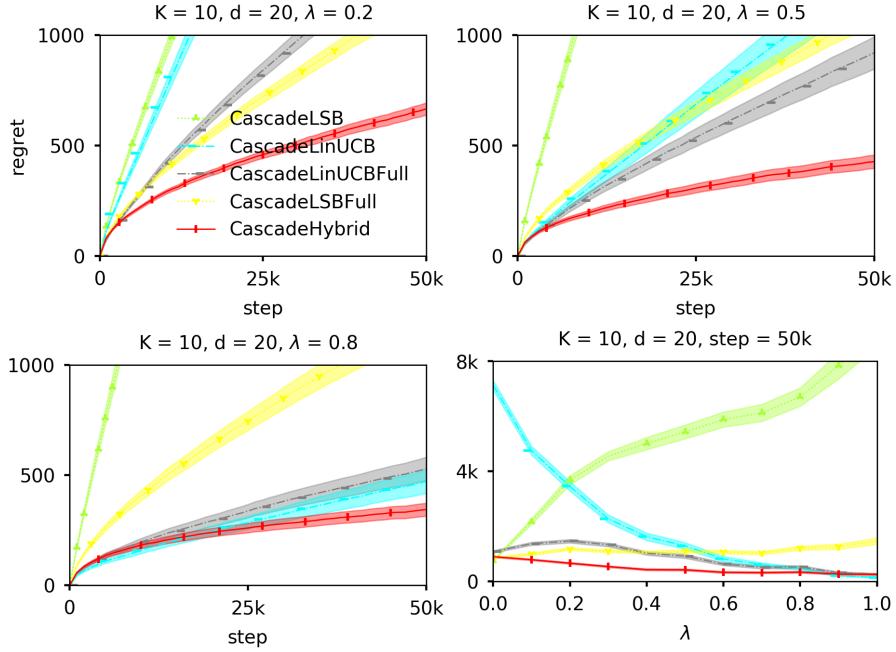


Figure 5.4:  $n$ -step regret on the Yahoo dataset with different  $\lambda$ . Results are averaged over 500 users with 2 repeats per user.

## 5.5 Analysis

### 5.5.1 Performance guarantee

Since CascadeHybrid competes with the greedy benchmark, we focus on the  $\eta$ -scaled expected  $n$ -step regret which is defined as:

$$R_\eta(n) = \sum_{t=1}^n \mathbb{E} [\eta r(\mathcal{R}^*, \alpha) - r(\mathcal{R}_t, A_t)], \quad (5.18)$$

where  $\eta = (1 - \frac{1}{e}) \max\{\frac{1}{K}, 1 - \frac{K-1}{2} \alpha_{\max}\}$ . This is a reasonable metric, since computing the optimal  $\mathcal{R}^*$  is computationally inefficient. A similar scaled regret has previously been used in diversity problems [38, 95, 120]. For simplicity, we write  $\mathbf{w}^* = [\boldsymbol{\theta}^{*T}, \boldsymbol{\beta}^{*T}]^T$ . Then, we bound the  $\eta$ -scaled regret of CascadeHybrid as follows:

**Theorem 5.1.** For  $\|\mathbf{w}^*\|_2 \leq 1$  and any

$$\gamma \geq \sqrt{(m+d) \log \left( 1 + \frac{nK}{m+d} \right) + 2 \log(n)} + \|\mathbf{w}^*\|_2, \quad (5.19)$$

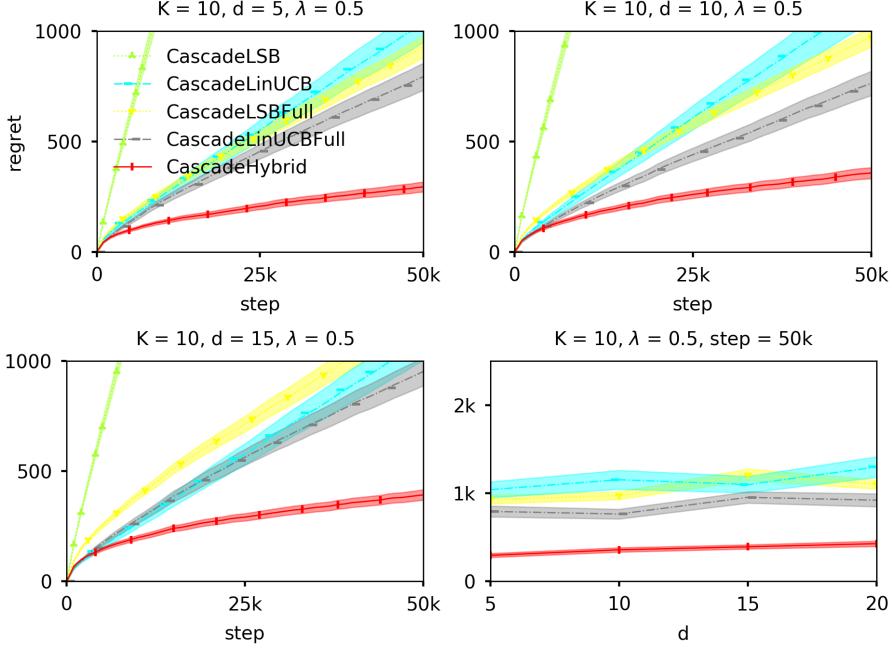


Figure 5.5:  $n$ -step regret on the Yahoo dataset with different topics  $d$ .

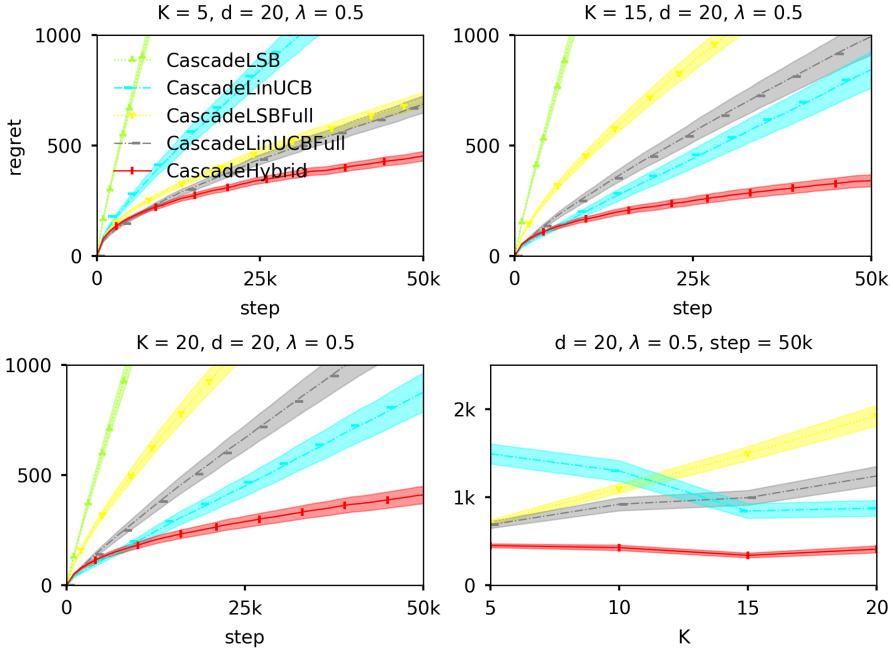
we have

$$R_\eta(n) \leq 2\gamma \sqrt{2nK(m+d) \log \left(1 + \frac{nK}{m+d}\right)} + 1. \quad (5.20)$$

Combining Eqs. (5.19) and (5.20), we have  $R_\eta(n) = \tilde{O}((m+d)\sqrt{Kn})$ , where the  $\tilde{O}$  notation ignores logarithmic factors. Our bound has three characteristics: (i) Theorem 5.1 states a gap-free bound, where the factor  $\sqrt{n}$  is considered near optimal; (ii) This bound is linear in the number of features, which is a common dependence in learning bandit algorithms [1]; and (iii) Our bound is  $\tilde{O}(\sqrt{K})$  lower than other bounds for linear bandit algorithms in CB [38, 133]. We include a proof of Theorem 5.1 in Section 5.5.2. We use a similar strategy to decompose the regret as in [38, 133], but we have a better analysis on how to sum up the regret of individual items. Thus, our bound depends on  $\tilde{O}(\sqrt{K})$  rather than  $\tilde{O}(K)$ . We believe that our analysis can be applied to both CascadeLinUCB and CascadeLSB, and then show that their regret is actually bounded by  $\tilde{O}(\sqrt{K})$  rather than  $\tilde{O}(K)$ .

### 5.5.2 Proof of Theorem 5.1

We first define some additional notation. We write  $\mathbf{w}^* = [\boldsymbol{\theta}^{*T}, \boldsymbol{\beta}^{*T}]^T$ . Given a ranked list  $\mathcal{R}$  and  $a = \mathcal{R}(i)$ , we write  $\phi_a = [\omega_a^T, \mathbf{z}_a^T]^T$ . With the  $\phi_a$  and  $\mathbf{w}^*$  notation, CascadeHybrid can be viewed as an extension of CascadeLSB, where two


 Figure 5.6:  $n$ -step regret on the Yahoo dataset with different positions  $K$ .

submodular functions instead of one are used in a single model. We write  $\mathbf{O}_t = \mathbf{I}_{m+d} + \sum_{i=1}^{t-1} \sum_{a \in \mathcal{O}_i} \phi_a \phi_a^T$  as the collected features in  $t$  steps,  $\mathcal{H}_t$  as the collected features and clicks up to step  $t$ , and  $\mathcal{R}^i = (\mathcal{R}(1), \dots, \mathcal{R}(i))$ . Then, the confidence bound in Eq. (5.14) on the  $i$ -th item in  $\mathcal{R}$  can be re-written as:

$$s(\mathcal{R}^i) = \phi_{\mathcal{R}(i)}^T \mathbf{O}_t^{-1} \phi_{\mathcal{R}(i)}. \quad (5.21)$$

Let  $\Pi(\mathcal{D}) = \bigcup_{i=1}^L \Pi_i(\mathcal{D})$  be the set of all ranked lists of  $\mathcal{D}$  with length  $[L]$ , and  $\kappa : \Pi(\mathcal{D}) \rightarrow [0, 1]$  be an arbitrary list function. For any  $\mathcal{R} \in \Pi(\mathcal{D})$  and any  $\kappa$ , we define

$$f(\mathcal{R}, \kappa) = 1 - \prod_{i=1}^{|\mathcal{R}|} (1 - \kappa(\mathcal{R}^i)). \quad (5.22)$$

We define upper and lower confidence bounds, and  $\kappa$  as:

$$\begin{aligned} u_t(\mathcal{R}) &= F_{[0,1]}[\phi_{\mathcal{R}(l)}^T \hat{\mathbf{w}}_t + s(\mathcal{R}^l)] \\ l_t(\mathcal{R}) &= F_{[0,1]}[\phi_{\mathcal{R}(l)}^T \hat{\mathbf{w}}_t - s(\mathcal{R}^l)] \\ \kappa(\mathcal{R}) &= \phi_{\mathcal{R}(l)}^T \mathbf{w}^*, \end{aligned} \quad (5.23)$$

where  $l = |\mathcal{R}|$  and  $F_{[0,1]}[\cdot] = \max(0, \min(1, \cdot))$ . With the definitions in Eq. (5.23),  $f(\mathcal{R}, \kappa) = r(\mathcal{R}, \alpha)$  is the reward of list  $\mathcal{R}$ .

*Proof.* Let  $g_t = \{l_t(\mathcal{R}) \leq \kappa(\mathcal{R}) \leq u_t(\mathcal{R}), \forall \mathcal{R} \in \Pi(\mathcal{D})\}$  be the event that the attraction probabilities are bounded by the lower and upper confidence bound, and  $\bar{g}_t$  be the complement of  $g_t$ . We have

$$\begin{aligned} \mathbb{E} [\eta r(\mathcal{R}^*, \boldsymbol{\alpha}) - r(\mathcal{R}_t, \mathbf{A}_t)] &= \mathbb{E} [\eta f(\mathcal{R}^*, \kappa) - f(\mathcal{R}_t, \kappa)] \\ &\stackrel{(a)}{\leq} P(g_t) \mathbb{E} [\eta f(\mathcal{R}^*, \kappa) - f(\mathcal{R}_t, \kappa)] + P(\bar{g}_t) \\ &\stackrel{(b)}{\leq} P(g_t) \mathbb{E} [\eta f(\mathcal{R}^*, u_t) - f(\mathcal{R}_t, \kappa)] + P(\bar{g}_t) \\ &\stackrel{(c)}{\leq} P(g_t) \mathbb{E} [f(\mathcal{R}_t, u_t) - f(\mathcal{R}_t, \kappa)] + P(\bar{g}_t), \end{aligned} \quad (5.24)$$

where (a) holds because  $\mathbb{E} [\eta f(\mathcal{R}^*, \kappa) - f(\mathcal{R}_t, \kappa)] \leq 1$ , (b) holds because under event  $g_t$  we have  $f(\mathcal{R}, l_t) \leq f(\mathcal{R}, \kappa) \leq f(\mathcal{R}, u_t)$ ,  $\forall \mathcal{R} \in \Pi(\mathcal{D})$ , and (c) holds by the definition of the  $\eta$ -approximation, where we have

$$\eta f(\mathcal{R}^*, u_t) \leq \max_{\mathcal{R} \in \Pi_K(\mathcal{D})} \eta f(\mathcal{R}, u_t) \leq f(\mathcal{R}_t, u_t). \quad (5.25)$$

By the definition of the list function  $f(\cdot, \cdot)$  in Eq. (5.22), we have

$$\begin{aligned} &f(\mathcal{R}_t, u_t) - f(\mathcal{R}_t, \kappa) \\ &= \prod_{k=1}^K (1 - \kappa(\mathcal{R}_t^k)) - \prod_{k=1}^K (1 - u_t(\mathcal{R}_t^k)) \\ &\stackrel{(a)}{=} \sum_{k=1}^K \left[ \prod_{i=1}^{k-1} (1 - \kappa(\mathcal{R}_t^i)) \right] (u_t(\mathcal{R}_t^k) - \kappa(\mathcal{R}_t^k)) \left[ \prod_{j=k+1}^K (1 - u(\mathcal{R}_t^j)) \right] \\ &\stackrel{(b)}{\leq} \sum_{k=1}^K \left[ \prod_{i=1}^{k-1} (1 - \kappa(\mathcal{R}_t^i)) \right] (u_t(\mathcal{R}_t^k) - \kappa(\mathcal{R}_t^k)), \end{aligned} \quad (5.26)$$

where (a) follows from Lemma 1 in [133] and (b) is because of the fact that  $0 \leq \kappa(\mathcal{R}_t) \leq u_t(\mathcal{R}_t) \leq 1$ . We then define the event  $h_{ti} = \{\text{item } \mathcal{R}_t(i) \text{ is observed}\}$ , where we have  $\mathbb{E} \left[ \stackrel{\text{ind}}{\sim} h_{ti} \right] = \prod_{k=1}^{i-1} (1 - \kappa(\mathcal{R}_t^k))$ . For any  $\mathcal{H}_t$  such that  $g_t$  holds, we have

$$\begin{aligned} &\mathbb{E} [f(\mathcal{R}_t, u_t) - f(\mathcal{R}_t, \kappa) \mid \mathcal{H}_t] \\ &\leq \sum_{i=1}^K \mathbb{E} \left[ \stackrel{\text{ind}}{\sim} h_{ti} \mid \mathcal{H}_t \right] (u_t(\mathcal{R}_t^i) - l_t(\mathcal{R}_t^i)) \\ &\stackrel{(a)}{\leq} 2\gamma \mathbb{E} \left[ \stackrel{\text{ind}}{\sim} h_{ti} \sum_{i=1}^K \sqrt{s(\mathcal{R}_t^i)} \mid \mathcal{H}_t \right] \\ &\stackrel{(b)}{\leq} 2\gamma \mathbb{E} \left[ \sum_{i=1}^{\min(K, c_t)} \sqrt{s(\mathcal{R}_t^i)} \mid \mathcal{H}_t \right], \end{aligned} \quad (5.27)$$

where inequality (a) follows from the definition of  $u_t$  and  $l_t$  in Eq. (5.23), and inequality (b) follows from the definition of  $h_{ti}$ . Now, together with Eqs. (5.18) and (5.27) and Section 5.5.2, we have

$$\begin{aligned} R_\eta(n) &= \sum_{t=1}^n \mathbb{E} [\eta r(\mathcal{R}^*, \boldsymbol{\alpha}) - r(\mathcal{R}_t, \mathbf{A}_t)] \\ &\leq \sum_{t=1}^n \left[ 2\gamma \mathbb{E} \left[ \sum_{i=1}^{\min(K, c_t)} \sqrt{s(\mathcal{R}_t^i)} \mid g_t \right] P(g_t) + P(\bar{g}_t) \right] \\ &\leq 2\gamma \mathbb{E} \left[ \sum_{t=1}^n \sum_{i=1}^K \sqrt{s(\mathcal{R}_t^i)} \right] + \sum_{t=1}^n p(\bar{g}_t). \end{aligned} \quad (5.28)$$

For the first term in Eq. (5.28), we have

$$\sum_{t=1}^n \sum_{i=1}^K \sqrt{s(\mathcal{R}_t^i)} \stackrel{(a)}{\leq} \sqrt{nK \sum_{t=1}^n \sum_{i=1}^K s(\mathcal{R}_t^i)} \stackrel{(b)}{\leq} \sqrt{nK 2 \log \det(\mathbf{O}_t)}, \quad (5.29)$$

where inequality (a) follows from the Cauchy-Schwarz inequality and (b) follows from Lemma 5 in [120]. Note that  $\log \det(\mathbf{O}_t) \leq (m+d) \log(K(1+n/(m+d)))$ , which can be obtained by the determinant and trace inequality, and together with Eq. (5.29):

$$\sum_{t=1}^n \sum_{i=1}^K \sqrt{s(\mathcal{R}_t^i)} \leq \sqrt{2nK(m+d) \log(K(1 + \frac{n}{m+d}))}. \quad (5.30)$$

For the second term in Eq. (5.28), by Lemma 3 in [38], we have  $P(\bar{g}_t) \leq 1/n$  for any  $\gamma$  that satisfies Eq. (5.19). Thus, together with Eqs. (5.28) to (5.30), we have

$$R_\eta(n) \leq 2\gamma \sqrt{2nK(m+d) \log(1 + \frac{nK}{m+d})} + 1.$$

This concludes the proof of Theorem 5.1.  $\square$

## 5.6 Related Work

The literature on offline Learning to Rank (LTR) methods that account for position bias and diversity is too broad to review in detail. We refer readers to [3] for an overview. In this section, we mainly review online LTR papers that are closely related to our work, i.e., stochastic click bandit models.

Online LTR in a stochastic click models has been well-studied [58, 65, 74, 77, 81, 99, 106, 120, 132, 133]. Previous work can be categorized into two groups: feature-free models and feature-rich models. Algorithms from the former group use a tabular representation on items and maintain an estimator for each item. They learn inefficiently and are limited to the problem with a small number of item candidates. In this chapter, we focus on the ranking problem with a large number of items. Thus, we do not consider feature-free model in the experiments.

Feature-rich models learn efficiently in terms of the number of items. They are suitable for large-scale ranking problems. Among them, ranked bandits [99, 106] are early approaches to online LTR. In ranked bandits, each position is model as a MAB and diversity of results is addressed in the sense that items ranked at lower positions are less likely to be clicked than those at higher positions, which is different from the topical diversity as we study. Also, ranked bandits do not consider the position bias and are suboptimal in the problem where a user browse different possition unevenly, e.g., CM [65]. LSBGreedy [120] and C<sup>2</sup>UCB [95] use submodular functions to solve the online diverse LTR problem. They assume that the user browses all displayed items and, thus, do not consider the position bias either.

Our work is closely related to CascadeLinUCB [133] and CascadeLSB [38], the baselines, and can be viewed as a combination of both. CascadeLinUCB solves the relevance ranking in the CM and assumes the attraction probability is a linear combination of features. CascadeLSB is designed for result diversification and assumes that the attraction probability is computed as a submodular function; see Eq. (5.6). In our CascadeHybrid, the attraction probability is a hybrid of both; see Eq. (5.7). Thus, CascadeHybrid handles both relevance ranking and result diversification. RecurRank [81] is a recently proposed algorithm that aims at learning the optimal list in term of item relevance in most click models. However, to achieve this task, RecurRank requires a lot of randomly shuffled lists and is outperformed by CascadeLinUCB in the CM [81].

The hybrid of a linear function and a submodular function has been used in solving combinatorial semi-bandits. Perrault et al. [93] use a linear set function to model the expected reward of arm, and use the submodular function to compute the *exploration bonus*. This is different from our hybrid model, where both the linear and submodular functions are used to model the attraction probability and the confident bound is used as the exploration bonus.

## 5.7 Conclusion

---

In real world interactive systems, both relevance of individual items and topical diversity of result lists are critical factors in user satisfaction. This chapter has provide one solution to this problem, i.e., **RQ4**. In order to better meet users' information needs, we propose a novel online LTR algorithm that optimizes both factors in a hybrid fashion. We formulate the problem as Cascade Hybrid Bandits (CHB), where the attraction probability is a hybrid function that combines a function of relevance features and a submodular function of topic features. CascadeHybrid utilizes a hybrid model as a scoring function and the UCB policy for exploration. We provide a gap-free bound on the  $\eta$ -scaled  $n$ -step regret of CascadeHybrid, and conduct experiments on two real-world datasets. Our empirical study shows that CascadeHybrid outperforms two existing online LTR algorithms that exclusively consider either relevance ranking or result diversification.

In future work, we intend to conduct experiments on live systems, where feedback is obtained from multiple users so as to test whether CascadeHybrid can learn across users. Another direction is to adapt Thompson Sampling (TS) [111] to our hybrid model, since TS generally outperforms UCB-based algorithms [79, 133].



# 6

## Conclusions

We set up this thesis to study the online optimization of ranking systems as bandit problems. We consider two typical ranking tasks: online ranker evaluation and online learning to rank. A typical challenge in these tasks is that the feedback is implicit and partially-observed. Specifically, in the online evaluation task, the feedback is the relative comparison of two rankers; and in Online Learning to Rank (OLTR), the feedback is the click but due to the position bias, the examination of each item is unobserved. In this thesis, we have considered four research directions in the online ranker optimization and formulated each of them as a bandit problem and proposed the corresponding algorithm. In the rest of this chapter, we first summarize the main results of this thesis and then outline some potential directions for future research.

### 6.1 Results

---

We first revisit the research questions that are asked in Section 1.1.

**RQ1** How to conduct effective large-scale online ranker evaluation?

The question has been answered in Chapter 2, where we introduce the Thompson Sampling (TS) based algorithm, called MergeDTS. Although in theory, the regret of MergeDTS has the same order as some baselines, i.e., MergeRUCB and Self-Sparring, MergeDTS outperforms those baselines with large gaps in our large-scale online ranker evaluation experiments, as shown in Section 2.6. Specifically, we have conducted experiments on three large-scale datasets, each of which has been combined with 3 different types of click behavior. MergeDTS has lower regret than the baselines in 7 out of 9 configurations, and only has slightly higher regret than Self-Sparring in 2 configurations. Meanwhile, the time efficiency of MergeDTS is better than that of Self-Sparring.

To answer **RQ1**, we conduct an effective large-scale online ranker evaluation by combining Thompson sampling and the divide-and-conquer idea in MergeRUCB.

**RQ2** How to achieve safe online learning to re-rank?

Chapter 3 has provided one answer to this question, where the BubbleRank algorithm is introduced. BubbleRank is inspired by the bubble sort algorithm and the pairwise

## 6. Conclusions

---

comparison in dueling bandits. BubbleRank starts from the ranked list produced by a production ranker, and conducts pairwise comparisons between an item with its neighbors. To get rid of the position bias, the position of an item is randomly exchanged with its neighbors, during the comparisons. Thus, during the exploration, an item will not displayed too far from its original position. On the other hand, if one item is considered better than its upper neighbor with a high confidence, the positions of two items are exchanged permanently. The safety feature of BubbleRank has been theoretically proved in Lemma 3.1. The  $n$ -step regret of BubbleRank is  $O(K^3 \log(n))$ , where  $K$  is the number of ranking positions and  $n$  is the time horizon. This regret bound is only  $O(K)$  higher than the regret of TopRank, the state-of-the-art but unsafe baseline. We have conducted an empirical evaluation on a large-scale click dataset, the *Yandex* click dataset, and the empirical results have confirmed the theoretical findings.

**RQ3** How to conduct online learning to rank when users change their preference constantly?

Our answer to this question has been provided in Chapter 4. Firstly, the considered non-stationary OLTR problem has been formulated as a cascading non-stationary bandit problem, where we assume that users follow the cascading browsing behavior. Then, two algorithms, CascadeDUCB and CascadeSWUCB, have been proposed to solve this problem. To deal with the non-stationarity, the former algorithm uses the discount factor, and the latter uses a sliding window. The  $n$ -step regret of CascadeDUCB and CascadeSWUCB is  $O(\sqrt{n} \ln(n))$  and  $O(\sqrt{n \ln(n)})$ , respectively, where  $n$  is the time horizon. They both match the lower bound up to logarithmic factors. Finally, we have evaluated the proposed algorithms in a semi-synthetic experiment on the *Yandex* click dataset. The experimental results are consistent with our theoretical findings.

**RQ4** How to learn a ranker online considering both relevance and diversity?

This question is answered in Chapter 5. As a follow up to Chapter 4, the cascade click model has also been used to interpret the click feedback. We have first formulated the OLTR problem for relevance and diversity as a cascade hybrid bandit problem, and then introduced the CascadeHybrid algorithm to solve the proposed problem. The term *hybrid* comes from the fact that we use a linear function for relevance and a submodular function for diversity. The  $n$ -step regret of the proposed CascadeHybrid is with the form of  $\tilde{O}(\sqrt{Kn})$ , where  $\tilde{O}(\cdot)$  ignores logarithmic terms,  $K$  is the number of positions, and  $n$  is the time horizon. This bound is comparable to other linear bandit approaches for OLTR. In our experiments, we have evaluated CascadeHybrid on two tasks: movie recommendation on the MovieLens dataset and music recommendation on the *Yahoo* music dataset. The experimental results indicate that with the underlying hybrid model, CascadeHybrid learns a ranker that considers both item relevance and result diversification.

## 6.2 Future Work

---

As with any research, this thesis has some limitations, and we believe that more questions have been generated than answered by our results. in addition to the suggestions for

future work provided at the end of each of the research chapters, we believe there are two important future directions.

**Beyond the linear model** The proposed algorithms in this thesis are based on linear or tabular models. Although they learn efficiently, the relatively low learning capacity limits the application scenarios of these algorithms. To model more complicated problems, it is appealing to consider non-linear models, e.g., tree-based models and deep models, which have a more powerful learning capacity. However, training non-linear models online with implicit feedback is not without difficulty. One of the challenges is exploration with non-linear models.

Let us step back and take a look at the exploration strategies in this thesis, i.e., UCB and TS. They both require closed-form solutions of the underlying models to estimate the reward of each arm. This requirement can easily be satisfied when linear or tabular models are used. However, there are generally no closed-form solutions for non-linear models. To use UCB and TS for non-linear models, additional approximation methods, e.g., the Laplacian approximation, are required [101, 124], which can be computationally expensive. On the other hand,  $\epsilon$ -greedy is one of the basic exploration strategies, and can be used for non-linear models. Although  $\epsilon$ -greedy is far from optimal in Multi-Armed Bandits (the tabular case) and linear bandits (the linear case), for some non-linear bandit setups,  $\epsilon$ -greedy is shown to be rather competitive [69, 101].

A more sophisticated policy is bootstrapped sampling or *bootstrapping* [67–69, 101]. Bootstrapping is a statistical technique that estimates the distribution of random variables by conducting sampling with replacement. Thus, it can be used to estimate the variance of the predictions of non-linear models. However, there is no conclusive study to indicate which policy is the optimal one for the ranking task. We believe that it is an interesting direction to study how to conduct exploration for non-linear OLTR.

**Privacy** This direction comes from the recently regulations on data protection, e.g., the General Data Protection Regulation (GDPR)<sup>1</sup> by European Union and consumer data protection in the White House report [47]. The algorithms we have proposed in this thesis depend on the collection of user interaction data, which inevitably brings privacy risks. One way to provide protection w.r.t. user privacy is *federated learning* (FL) [13, 63, 75], where the learning happens on the local devices under the coordination of a central server, called *federator*.

Importantly, federated learning is easily combined with randomization techniques from differential privacy [28], and provides theoretically sound approaches to protect user privacy. In federated learning, the private user data is not collected by any central server but the learned model gradients are. Briefly, the federator maintains a global model that is trained by the participating clients, e.g., local mobile devices. The training goes in rounds. At the start of each round, the federator broadcasts the global model to local clients. Each client trains the model based on the local dataset, and then sends back the local gradient or the updated model to the federator. The federator updates the global model according to all local updates. This process continues until a certain termination criterion has been met, e.g., the loss stops dropping.

---

<sup>1</sup><https://gdpr-info.eu/>

## 6. Conclusions

Federated learning also works in an online fashion. That is, after each round, the new global model is updated based on user requests. However, in FL, the update at each round consists of several local updates, while in online learning, there is only one update per round. To combine the two learning paradigms, we have to deal with batched feedback or updates, which again brings new challenges to exploration. As federated learning to rank is a rather new but important topic, we expect to see more algorithms published in the future.

# Bibliography

- [1] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári. Improved algorithms for linear stochastic bandits. In *NIPS*, pages 2312–2320, 2011. (Cited on page 91.)
- [2] A. Agarwal, D. Hsu, S. Kale, J. Langford, L. Li, and R. Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *ICML*, pages II–1638–II–1646, 2014. (Cited on page 49.)
- [3] C. C. Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016. (Cited on page 94.)
- [4] R. Agrawal, S. Gollapudi, A. Halverson, and S. Jeong. Diversifying search results. In *WSDM*, pages 5–14, 2009. (Cited on pages 78 and 81.)
- [5] N. Ailon, Z. Karnin, and T. Joachims. Reducing dueling bandits to cardinal bandits. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML’14, pages II–856–II–864. JMLR.org, 2014. URL <http://dl.acm.org/citation.cfm?id=3044805.3044988>. (Cited on pages 12, 13, 22, and 24.)
- [6] J. Allan, D. Harman, E. Kanoulas, D. Li, C. Van Gysel, and E. Voorhees. TREC 2017 common core track overview. TREC, 2017. (Cited on page 48.)
- [7] A. Ashkan, B. Kveton, S. Berkovsky, and Z. Wen. Optimal greedy diversity for recommendation. In *IJCAI*, pages 1742–1748, 2015. (Cited on page 81.)
- [8] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *FOCS*, pages 322–331, 1995. (Cited on pages 67 and 68.)
- [9] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352. URL <https://doi.org/10.1023/A:1013689704352>. (Cited on pages 11, 35, 46, 57, 78, and 84.)
- [10] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, Jan. 2003. ISSN 0097-5397. doi: 10.1137/S0097539701398375. (Cited on pages 13 and 75.)
- [11] O. Besbes, Y. Gur, and A. Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. In *NIPS*, 2014. (Cited on pages 58, 68, and 75.)
- [12] D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, November 1956. (Cited on page 68.)
- [13] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingberman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019. (Cited on page 99.)
- [14] B. Brost, Y. Seldin, I. J. Cox, and C. Lioma. Multi-dueling bandits and their application to online ranker evaluation. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM’16, pages 2161–2166, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983659. URL <http://doi.acm.org/10.1145/2983323.2983659>. (Cited on pages 10 and 14.)
- [15] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, June 2010. URL <https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/>. (Cited on page 22.)
- [16] R. Busa-Fekete and E. Hüllermeier. A survey of preference-based online learning with bandit algorithms. In *Proceedings of the 25th International Conference on Algorithmic Learning Theory*, ALT’14, pages 18–39, Heidelberg, Germany, 2014. Springer. doi: 10.1007/978-3-319-11662-4\_3. URL [https://doi.org/10.1007/978-3-319-11662-4\\_3](https://doi.org/10.1007/978-3-319-11662-4_3). (Cited on pages 10, 11, 12, and 21.)
- [17] G. Casella and R. L. Berger. *Statistical Inference*. Duxbury Pacific Grove, CA, 2002. (Cited on pages 10 and 24.)
- [18] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006. (Cited on pages 42 and 68.)
- [19] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge - Volume 14*, YLRC’10, pages 1–24. JMLR.org, 2010. URL <http://dl.acm.org/citation.cfm?id=3045754.3045756>. (Cited on pages 1, 10, and 21.)
- [20] O. Chapelle and Y. Zhang. A dynamic Bayesian network click model for web search ranking. In *WWW*, pages 1–10, 2009. (Cited on pages 59 and 75.)
- [21] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale validation and analysis of interleaved

## 6. Bibliography

---

- search evaluation. *ACM Trans. Inf. Syst.*, 30(1):6:1–6:41, Mar. 2012. ISSN 1046-8188. doi: 10.1145/2094072.2094078. URL <http://doi.acm.org/10.1145/2094072.2094078>. (Cited on page 9.)
- [22] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. Efficient cost-aware cascade ranking in multi-stage retrieval. In *SIGIR*, pages 445–454, New York, NY, USA, 2017. ACM. (Cited on page 38.)
  - [23] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Morgan & Claypool, 2015. doi: 10.2200/S00654ED1V01Y201507ICR043. URL <https://doi.org/10.2200/S00654ED1V01Y201507ICR043>. (Cited on pages 23, 35, 36, 48, 58, 59, 78, 79, and 80.)
  - [24] A. Chuklin, A. Schuth, K. Zhou, and M. de Rijke. A comparative analysis of interleaving methods for aggregated search. *ACM Trans. Inf. Syst.*, 33(2):5:1–5:38, Feb. 2015. ISSN 1046-8188. doi: 10.1145/2668120. URL <http://doi.acm.org/10.1145/2668120>. (Cited on page 9.)
  - [25] R. Combes, S. Magureanu, A. Proutiere, and C. Laroché. Learning to rank: Regret lower bounds and efficient algorithms. In *ACM SIGMETRICS*, pages 231–244, 2015. (Cited on page 49.)
  - [26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. (Cited on page 36.)
  - [27] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94, 2008. (Cited on pages 3, 37, 45, 58, 59, 78, and 79.)
  - [28] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014. (Cited on page 99.)
  - [29] P. Gajane, T. Urvoy, and F. Clérot. A relative exponential weighing algorithm for adversarial utility-based dueling bandits. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15, pages 218–227. JMLR.org, 2015. URL <http://dl.acm.org/citation.cfm?id=3045118.3045143>. (Cited on pages 12, 13, 23, and 24.)
  - [30] A. Garivier and E. Moulines. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*, 2008. (Cited on pages 72 and 75.)
  - [31] A. Garivier and E. Moulines. On upper-confidence bound policies for switching bandit problems. In *ALT*, pages 174–188, 2011. (Cited on pages 58, 62, 65, 66, 67, 68, 70, and 72.)
  - [32] D. Goldberg, A. Trotman, X. Wang, W. Min, and Z. Wan. Further insights on drawing sound conclusions from noisy judgments. *ACM Trans. Inf. Syst.*, 36(4):36:1–36:31, Apr. 2018. ISSN 1046-8188. doi: 10.1145/3186195. URL <http://doi.acm.org/10.1145/3186195>. (Cited on page 9.)
  - [33] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins, Baltimore, MD, 3 edition, 1996. (Cited on page 85.)
  - [34] B. Gomes. Google Instant, Behind the Scenes, 2010. <https://googleblog.blogspot.no/2010/09/google-instant-behind-scenes.html>. (Cited on pages 25 and 28.)
  - [35] A. Grotov and M. de Rijke. Online learning to rank for information retrieval: Sigir 2016 tutorial. In *SIGIR*, pages 1215–1218, 2016. (Cited on pages 2 and 77.)
  - [36] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM*, WSDM ’09, pages 124–131, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-390-7. doi: 10.1145/1498759.1498818. URL <http://doi.acm.org/10.1145/1498759.1498818>. (Cited on pages 45, 59, and 75.)
  - [37] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2015. (Cited on page 86.)
  - [38] G. Hiranandani, H. Singh, P. Gupta, I. A. Burhanuddin, Z. Wen, and B. Kveton. Cascading linear submodular bandits: Accounting for position bias and diversity in online learning to rank. In *UAI*, 2019. (Cited on pages 4, 6, 78, 80, 81, 82, 83, 86, 87, 90, 91, 94, and 95.)
  - [39] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. URL <https://www.jstor.org/stable/2282952?seq=1>. (Cited on pages 10 and 24.)
  - [40] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in learning to rank online. In *ECIR*, pages 251–263, 2011. (Cited on pages 1, 77, and 78.)
  - [41] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM’11, pages 249–258, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0717-8. doi: 10.1145/2063576.2063618. URL <http://doi.acm.org/10.1145/2063576.2063618>. (Cited on pages 1, 9, 21, 22, and 23.)
  - [42] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing historical interaction data for faster

- 
- online learning to rank for IR. In *WSDM*, WSDM ’13, pages 183–192, New York, NY, USA, February 2013. ACM. doi: 10.1145/2433396.2433419. (Cited on pages 49, 77, and 78.)
- [43] K. Hofmann, S. Whiteson, and M. de Rijke. Fidelity, soundness, and efficiency of interleaved comparison methods. *ACM Trans. Inf. Syst.*, 31(4):17:1–17:43, Nov. 2013. ISSN 1046-8188. doi: 10.1145/2536736.2536737. URL <http://doi.acm.org/10.1145/2536736.2536737>. (Cited on pages 9 and 23.)
- [44] K. Hofmann, S. Whiteson, and M. Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Inf. Retr.*, 16(1):63–90, Feb. 2013. ISSN 1386-4564. doi: 10.1007/s10791-012-9197-9. URL <http://dx.doi.org/10.1007/s10791-012-9197-9>. (Cited on page 1.)
- [45] K. Hofmann, L. Li, and F. Radlinski. Online evaluation for information retrieval. *Found. Trends Inf. Retr.*, 10(1):1–117, June 2016. ISSN 1554-0669. doi: 10.1561/1500000051. URL <https://doi.org/10.1561/1500000051>. (Cited on page 9.)
- [46] J. Honda and A. Takemura. An asymptotically optimal policy for finite support models in the multiarmed bandit problem. volume 85, pages 361–391, Dec 2011. doi: 10.1007/s10994-011-5257-4. URL <https://doi.org/10.1007/s10994-011-5257-4>. (Cited on page 13.)
- [47] W. House. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. *White House, Washington, DC*, pages 1–62, 2012. (Cited on page 99.)
- [48] M. Ibrahim and M. Carman. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. *ACM Trans. Inf. Syst.*, 34(4):20:1–20:38, Aug. 2016. ISSN 1046-8188. doi: 10.1145/2866571. URL <http://doi.acm.org/10.1145/2866571>. (Cited on page 9.)
- [49] R. Jagerman, I. Markov, and M. de Rijke. When people change their mind: Off-policy evaluation in non-stationary recommendation environments. In *WSDM*, pages 447–455, 2019. (Cited on pages 1, 3, 58, 60, and 67.)
- [50] R. Jagerman, H. Oosterhuis, and M. de Rijke. To model or to intervene: A comparison of counterfactual and online learning to rank from user interactions. In *SIGIR*, pages 15–24, 2019. (Cited on page 77.)
- [51] K. Jamieson, S. Katariya, A. Deshpande, and R. Nowak. Sparse dueling bandits. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics - Volume 38*, AISTATS’15, pages 416–424. PMLR, 2015. (Cited on page 12.)
- [52] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002. (Cited on pages 1 and 48.)
- [53] B. Jiang, C. Li, M. de Rijke, X. Yao, and H. Chen. Probabilistic feature selection and classification vector machine. *ACM Transactions on Knowledge Discovery from Data*, 13(2):Article 21, April 2019. (Cited on page 7.)
- [54] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, pages 133–142, 2002. (Cited on pages 1, 35, and 57.)
- [55] T. Joachims. Evaluating retrieval performance using clickthrough data. In *Text Mining*, pages 79–96. 2003. URL [https://www.cs.cornell.edu/people/tj/publications/joachims\\_02b.pdf](https://www.cs.cornell.edu/people/tj/publications/joachims_02b.pdf). (Cited on page 9.)
- [56] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst.*, 25(2), Apr. 2007. ISSN 1046-8188. doi: 10.1145/1229179.1229181. URL <http://doi.acm.org/10.1145/1229179.1229181>. (Cited on page 9.)
- [57] S. K. Karmaker Santu, P. Sondhi, and C. Zhai. On application of learning to rank for e-commerce search. In *SIGIR*, pages 475–484, New York, NY, USA, 2017. (Cited on page 38.)
- [58] S. Katariya, B. Kveton, C. Szepesvari, and Z. Wen. DCM bandits: Learning to rank with multiple clicks. In *ICML*, pages 1215–1224, 2016. (Cited on pages 35, 45, 46, 49, 57, 68, and 94.)
- [59] A. Kazerouni, M. Ghavamzadeh, Y. Abbasi, and B. Van Roy. Conservative contextual linear bandits. In *NIPS*, pages 3913–3922, 2017. (Cited on page 50.)
- [60] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’13, pages 1168–1176, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2488217. URL <http://doi.acm.org/10.1145/2487575.2488217>. (Cited on pages 10, 12, and 28.)
- [61] J. Komiyama, J. Honda, H. Kashima, and H. Nakagawa. Regret lower bound and optimal algorithm in dueling bandit problem. In *Proceedings of The 28th Conference on Learning Theory - Volume 40*, COLT’15, pages 1141–1154. PMLR, 03–06 Jul 2015. URL <http://jmlr.org/proceedings/>

## 6. Bibliography

---

- papers/v40/Komiyama15.html. (Cited on pages 12, 13, 14, and 23.)
- [62] J. Komiyama, J. Honda, and H. Nakagawa. Copeland dueling bandit problem: Regret lower bound, optimal algorithm, and computationally efficient algorithm. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1235–1244. JMLR.org, 2016. URL <http://dl.acm.org/citation.cfm?id=3045390.3045521>. (Cited on pages 12 and 33.)
- [63] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016. (Cited on page 99.)
- [64] B. Kveton, Z. Wen, A. Ashkan, H. Eydgahi, and B. Eriksson. Matroid bandits: Fast combinatorial optimization with learning. *arXiv preprint arXiv:1403.5045*, 2014. (Cited on pages 72 and 74.)
- [65] B. Kveton, C. Szepesvari, Z. Wen, and A. Ashkan. Cascading bandits: Learning to rank in the cascade model. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 767–776, 2015. (Cited on pages 5, 35, 45, 46, 47, 49, 57, 58, 60, 63, 64, 65, 67, 68, 71, 73, 78, 79, 80, 86, 94, and 95.)
- [66] B. Kveton, Z. Wen, A. Ashkan, and C. Szepesvari. Combinatorial cascading bandits. In *NIPS*, pages 1450–1458, 2015. (Cited on page 49.)
- [67] B. Kveton, C. Szepesvari, S. Vaswani, Z. Wen, T. Lattimore, and M. Ghavamzadeh. Garbage in, reward out: Bootstrapping exploration in multi-armed bandits. In *International Conference on Machine Learning*, pages 3601–3610. PMLR, 2019. (Cited on page 99.)
- [68] B. Kveton, M. Mladenov, C.-W. Hsu, M. Zaheer, C. Szepesvari, and C. Boutilier. Differentiable meta-learning in contextual bandits. *arXiv preprint arXiv:2006.05094*, 2020.
- [69] B. Kveton, M. Zaheer, C. Szepesvari, L. Li, M. Ghavamzadeh, and C. Boutilier. Randomized exploration in generalized linear bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 2066–2076, 2020. (Cited on page 99.)
- [70] P. Lagrée, C. Vernade, and O. Cappe. Multiple-play bandits in the position-based model. In *NIPS*, pages 1605–1613, 2016. (Cited on pages 35, 49, 57, and 68.)
- [71] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. (Cited on pages 2 and 78.)
- [72] T. Lattimore, B. Kveton, S. Li, and C. Szepesvari. Toprank: A practical algorithm for online stochastic ranking. In *NIPS*, pages 3945–3954, 2018. (Cited on pages 5, 36, 45, and 49.)
- [73] C. Li and M. de Rijke. Incremental sparse bayesian ordinal regression. *Neural Networks*, 106:294–302, October 2018. (Cited on page 7.)
- [74] C. Li and M. de Rijke. Cascading non-stationary bandits: Online learning to rank in the non-stationary cascade model. In *IJCAI*, pages 2859–2865, August 2019. (Cited on pages 6, 57, 78, 79, 87, and 94.)
- [75] C. Li and H. Ouyang. Federated unbiased learning to rank. In *SIGIR*. ACM, July 2021. Submitted. (Cited on pages 7 and 99.)
- [76] C. Li, A. Grotov, I. Markov, and M. de Rijke. Online learning to rank with list-level feedback for image filtering. *arXiv preprint arXiv:1812.04910*, December 2018. (Cited on page 7.)
- [77] C. Li, B. Kveton, T. Lattimore, I. Markov, M. de Rijke, C. Szepesvári, and M. Zoghi. BubbleRank: Safe online learning to re-rank via implicit click feedback. In *UAI*, July 2019. (Cited on pages 6, 11, 35, 57, 66, 78, 80, and 94.)
- [78] C. Li, H. Feng, and M. de Rijke. Cascading hybrid bandits: Online learning to rank for relevance and diversity. In *RecSys*, pages 33–42. ACM, September 2020. (Cited on pages 6 and 77.)
- [79] C. Li, I. Markov, M. de Rijke, and M. Zoghi. MergeDTS: A method for effective large-scale online ranker evaluation. *ACM Transactions on Information Systems*, 38(4):Article 40, August 2020. (Cited on pages 6, 9, 62, and 95.)
- [80] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, WWW '10, pages 661–670, 2010. (Cited on pages 19, 49, 78, and 85.)
- [81] S. Li, T. Lattimore, and C. Szepesvári. Online learning to rank with features. In *ICML*, pages 3856–3865, 2019. (Cited on pages 86, 94, and 95.)
- [82] F. Liu, J. Lee, and N. Shroff. A change-detection based framework for piecewise-stationary multi-armed bandit problem. In *AAAI*, pages 3651–3658, 2018. (Cited on page 68.)
- [83] S. Liu, F. Xiao, W. Ou, and L. Si. Cascade ranking for operational e-commerce search. *arXiv preprint arXiv:1706.02093*, 2017. (Cited on pages 3 and 38.)
- [84] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. (Cited on pages 1, 2, 35, 57, 77, and 78.)

- 
- [85] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani. Post-learning optimization of tree ensembles for efficient ranking. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’16*, pages 949–952, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340694. doi: 10.1145/2911451.2914763. URL <https://doi.org/10.1145/2911451.2914763>. (Cited on pages 10 and 21.)
- [86] C. Lucchese, F. M. Nardini, R. K. Pasumarthi, S. Bruch, M. Bendersky, X. Wang, H. Oosterhuis, R. Jagerman, and M. de Rijke. Learning to rank in theory and practice: From gradient boosting to neural networks and unbiased learning. In *SIGIR 2019: 42nd international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1419–1420, ACM, July 2019. (Cited on page 1.)
- [87] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *SIGKDD*, pages 1222–1230, 2013. (Cited on page 35.)
- [88] A. Moffat, P. Bailey, F. Scholer, and P. Thomas. Incorporating user expectations and behavior into the measurement of search effectiveness. *ACM Trans. Inf. Syst.*, 35(3):24:1–24:38, June 2017. ISSN 1046-8188. doi: 10.1145/3052768. URL <http://doi.acm.org/10.1145/3052768>. (Cited on page 9.)
- [89] T. Moon, L. Li, W. Chu, C. Liao, Z. Zheng, and Y. Chang. Online learning for recency search ranking using real-time user feedback. In *CJMK*, pages 1501–1504, 2010. (Cited on page 49.)
- [90] K. Nelissen, M. Snoeck, S. V. Broucke, and B. Baesens. Swipe and tell: Using implicit feedback to predict user engagement on tablets. *ACM Trans. Inf. Syst.*, 36(4):35:1–35:36, June 2018. ISSN 1046-8188. doi: 10.1145/3185153. URL <http://doi.acm.org/10.1145/3185153>. (Cited on page 9.)
- [91] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978. (Cited on pages 78 and 83.)
- [92] F. S. Pereira, J. Gama, S. de Amo, and G. M. Oliveira. On analyzing user preference dynamics with temporal social networks. *Machine Learning*, 107(11):1745–1773, 2018. (Cited on page 58.)
- [93] P. Perrault, V. Perchet, and M. Valko. Exploiting structure of uncertainty for efficient matroid semi-bandits. In *ICML*, pages 5123–5132. PMLR, 2019. (Cited on page 95.)
- [94] L. Qin and X. Zhu. Promoting diversity in recommendation by entropy regularizer. In *IJCAI*, pages 2698–2704, 2013. (Cited on page 81.)
- [95] L. Qin, S. Chen, and X. Zhu. Contextual combinatorial bandit and its application on diversified online recommendation. In *SDM*, pages 461–469, 2014. (Cited on pages 90 and 95.)
- [96] T. Qin and T.-Y. Liu. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013. URL <https://arxiv.org/abs/1306.2597>. (Cited on pages 1, 10, and 21.)
- [97] T. Qin, T.-Y. Liu, J. Xu, and H. Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010. (Cited on pages 1, 35, and 57.)
- [98] F. Radlinski and T. Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *AAAI*, pages 1406–1412, 2006. (Cited on page 36.)
- [99] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML*, pages 784–791, 2008. (Cited on pages 35, 49, 57, 67, 68, 94, and 95.)
- [100] M. Richardson, E. Dominowska, and R. Rago. Predicting clicks: Estimating the click-through rate for new ads. In *WWW*, pages 521–530, 2007. (Cited on page 37.)
- [101] C. Riquelme, G. Tucker, and J. Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018. (Cited on page 99.)
- [102] A. Saha and A. Gopalan. Battle of bandits. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence, UAI’18*. AUAI.org, 2018. URL <http://auai.org/uai2018/proceedings/papers/290.pdf>. (Cited on page 14.)
- [103] T. Saracevic. Relevance: A review of and a framework for the thinking on the notion in information science. *J. Am. Soc. Inform. Sci.*, 26(6):321–343, 1975. (Cited on page 1.)
- [104] A. Schuth, K. Hofmann, S. Whiteson, and M. de Rijke. Lerot: An online learning to rank framework. In *Proceedings of the 2013 Workshop on Living Labs for Information Retrieval Evaluation, LivingLab ’13*, pages 23–26, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2420-5. doi: 10.1145/2513150.2513162. URL <http://doi.acm.org/10.1145/2513150.2513162>. (Cited on page 23.)
- [105] A. Slivkins and E. Upfal. Adapting to a changing environment: the brownian restless bandits. In

## 6. Bibliography

---

- COLT*, pages 343–354, 2008. (Cited on page 68.)
- [106] A. Slivkins, F. Radlinski, and S. Gollapudi. Learning optimally diverse rankings over large document collections. In *ICML*, pages 983–990, 2010. (Cited on pages 94 and 95.)
- [107] A. Slivkins, F. Radlinski, and S. Gollapudi. Ranked bandits in metric spaces: Learning diverse rankings over large document collections. *JMLR*, 14(1):399–436, 2013. (Cited on page 49.)
- [108] A. Strehl, J. Langford, L. Li, and S. M. Kakade. Learning from logged implicit exploration data. In *NIPS*, pages 2217–2225, 2010. (Cited on page 49.)
- [109] Y. Sui, V. Zhuang, J. W. Burdick, and Y. Yue. Multi-dueling bandits with dependent arms. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence*, UAI’17. AUAI.org, 2017. URL <http://auai.org/uai2017/proceedings/papers/155.pdf>. (Cited on pages 5, 12, 13, 14, 22, 23, 24, and 26.)
- [110] Y. Sui, M. Zoghi, K. Hofmann, and Y. Yue. Advancements in dueling bandits. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI’18, pages 5502–5510. AAAI Press, 2018. ISBN 978-0-9992411-2-7. URL <http://dl.acm.org/citation.cfm?id=3304652.3304790>. (Cited on page 11.)
- [111] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285–294, 1933. URL <https://www.jstor.org/stable/pdf/2332286.pdf?seq=1>. (Cited on pages 2, 10, 36, and 95.)
- [112] T. Urvoy, F. Clerot, R. Féraud, and S. Naamane. Generic exploration and k-armed voting bandits. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML’13, pages II–91–II–99. JMLR.org, 2013. URL <http://dl.acm.org/citation.cfm?id=3042817.3042904>. (Cited on pages 12, 22, and 29.)
- [113] A. Vorobev, D. Lefortier, G. Gusev, and P. Serdyukov. Gathering additional feedback on search results by multi-armed bandits with respect to production ranking. In *WWW*, pages 1177–1187, 2015. (Cited on pages 36 and 49.)
- [114] X. Wang, N. Golbandi, M. Bendersky, D. Metzler, and M. Najork. Position bias estimation for unbiased learning to rank in personal search. In *WSDM*, pages 610–618, 2018. (Cited on pages 36 and 49.)
- [115] H. Wu and X. Liu. Double Thompson sampling for dueling bandits. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS’16, pages 649–657, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. URL <http://dl.acm.org/citation.cfm?id=3157096.3157169>. (Cited on pages 5, 10, 12, 13, 22, 23, 26, and 31.)
- [116] Q. Wu, N. Iyer, and H. Wang. Learning contextual bandits in a non-stationary environment. In *SIGIR*, pages 495–504, 2018. (Cited on page 58.)
- [117] Y. Wu, R. Shariff, T. Lattimore, and C. Szepesvári. Conservative bandits. In *ICML*, pages 1254–1262, 2016. (Cited on page 50.)
- [118] Y. Yan, Z. Liu, M. Zhao, W. Guo, W. P. Yan, and Y. Bao. A practical deep online ranking system in e-commerce recommendation. In *ECML PKDD*, pages 186–201, 2018. (Cited on page 49.)
- [119] J. Y. Yu and S. Mannor. Piecewise-stationary bandit problems with side observations. In *ICML*, 2009. (Cited on pages 58 and 68.)
- [120] Y. Yue and C. Guestrin. Linear submodular bandits and their application to diversified retrieval. In *NIPS*, pages 2483–2491, 2011. (Cited on pages 4, 35, 78, 81, 83, 87, 90, 94, and 95.)
- [121] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML*, ICML ’09, pages 1201–1208, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553527. URL <http://doi.acm.org/10.1145/1553374.1553527>. (Cited on pages 1 and 19.)
- [122] Y. Yue and T. Joachims. Beat the mean bandit. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML’11, pages 241–248, USA, 2011. Omnipress. ISBN 978-1-4503-0619-5. URL <http://dl.acm.org/citation.cfm?id=3104482.3104513>. (Cited on pages 22 and 23.)
- [123] Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The k-armed dueling bandits problem. *J. Comput. Syst. Sci.*, 78(5):1538–1556, Sept. 2012. ISSN 0022-0000. doi: 10.1016/j.jcss.2011.12.028. URL <http://dx.doi.org/10.1016/j.jcss.2011.12.028>. (Cited on page 10.)
- [124] D. Zhou, L. Li, and Q. Gu. Neural contextual bandits with upper confidence bound-based exploration. *arXiv preprint arXiv:1911.04462*, 2019. (Cited on page 99.)
- [125] J. Zimmert and Y. Seldin. An optimal algorithm for stochastic and adversarial bandits. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 467–475. PMLR, 16–18 Apr 2019. (Cited on page 13.)
- [126] M. Zoghi. *Dueling Bandits for Online Ranker Evaluation*. PhD thesis, University of

- 
- Twente, 2017. URL <https://research.utwente.nl/en/publications/dueling-bandits-for-online-ranker-evaluation>. (Cited on pages 1, 2, 10, 11, and 21.)
- [127] M. Zoghi, S. Whiteson, R. Munos, and M. de Rijke. Relative upper confidence bound for the k-armed dueling bandit problem. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, volume II of *ICML'14*, pages II–10–II–18. JMLR.org, 2014. URL <http://dl.acm.org/citation.cfm?id=3044805.3044894>. (Cited on pages 12, 13, 24, and 57.)
- [128] M. Zoghi, S. A. Whiteson, M. de Rijke, and R. Munos. Relative confidence sampling for efficient on-line ranker evaluation. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 73–82, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2351-2. doi: 10.1145/2556195.2556256. URL <http://doi.acm.org/10.1145/2556195.2556256>. (Cited on pages 10, 12, 21, 23, and 30.)
- [129] M. Zoghi, Z. Karnin, S. Whiteson, and M. de Rijke. Copeland dueling bandits. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 307–315, Cambridge, MA, USA, 2015. MIT Press. URL <http://dl.acm.org/citation.cfm?id=2969239.2969274>. (Cited on pages 12, 22, and 31.)
- [130] M. Zoghi, S. Whiteson, and M. de Rijke. Mergerucb: A method for large-scale online ranker evaluation. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 17–26, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3317-7. doi: 10.1145/2684822.2685290. URL <http://doi.acm.org/10.1145/2684822.2685290>. (Cited on pages 2, 5, 10, 12, 14, 18, 19, 21, 22, 23, 24, and 26.)
- [131] M. Zoghi, T. Tunys, L. Li, D. Jose, J. Chen, C. M. Chin, and M. de Rijke. Click-based hot fixes for underperforming torso queries. In *SIGIR*, pages 195–204. ACM, 2016. (Cited on pages 1, 35, 38, 49, 57, and 77.)
- [132] M. Zoghi, T. Tunys, M. Ghavamzadeh, B. Kveton, C. Szepesvari, and Z. Wen. Online learning to rank in stochastic click models. In *ICML*, pages 4199–4208, 2017. (Cited on pages 5, 35, 36, 37, 40, 41, 45, 46, 47, 49, 57, 58, 59, 66, 67, 68, and 94.)
- [133] S. Zong, H. Ni, K. Sung, N. R. Ke, Z. Wen, and B. Kveton. Cascading bandits for large-scale recommendation problems. In *UAI*, pages 835–844, 2016. (Cited on pages 6, 49, 78, 80, 82, 86, 87, 91, 93, 94, and 95.)



# Summary

People use interactive systems, such as search engines, as the main tool to obtain information. To satisfy the information needs, such systems usually provide a list of items that are selected out of a large candidate set and then sorted in the decreasing order of their usefulness. The result lists are generated by a ranking algorithm, called ranker, which takes the request of user and candidate items as the input and decides the order of candidate items. The quality of these systems depends on the underlying rankers.

There are two main approaches to optimize the ranker in an interactive system: using data annotated by humans or using the interactive user feedback. The first approach has been widely studied in history, also called offline learning to rank, and is the industry standard. However, the annotated data may not well represent information needs of users and are not timely. Thus, the first approaches may lead to suboptimal rankers. The second approach optimizes rankers by using interactive feedback. This thesis considers the second approach, learning from the interactive feedback. The reasons are two-fold: (1) Everyday, millions of users interact with the interactive systems and generate a huge number of interactions, from which we can extract the information needs of users. (2) Learning from the interactive data have more potentials to assist in designing the online algorithms.

Specifically, this thesis considers the task of learning from the user click feedback. The main contribution of this thesis is proposing a safe online learning to re-rank algorithm, named BubbleRank, which addresses one main disadvantage of online learning, i.e., the safety issue, by combining the advantages of both offline and online learning to rank algorithms. The thesis also proposes three other online algorithms, each of which solves unique online ranker optimization problems. All the proposed algorithms are theoretically sound and empirically effective.



# Samenvatting

Mensen gebruiken interactiesystemen, zoals een zoekmachine, als hun belangrijkste hulpmiddel om informatie te verkrijgen. Om aan de informatiebehoefte te voldoen, bieden dergelijke zoekmachines meestal een lijst met items aan. Deze items zijn geselecteerd uit een grote set van kandidaten en zijn vervolgens gesorteerd in afnemende volgorde van relevantie. Deze resultatenlijst wordt gegenereerd door een ranking-algoritme, *ranker* genaamd, dat de informatiebehoefte van de gebruiker en de kandidaat-items als input neemt en vervolgens de volgorde van de kandidaat-items bepaalt. De kwaliteit van deze systemen is afhankelijk van de onderliggende rankers.

Er zijn twee belangrijke benaderingen om de rankers in een interactiesysteem te optimaliseren: De eerste manier is door gebruik te maken van de gegevens die door mensen zijn geannoteerd. De tweede is door gebruik te maken van de interactie-feedback van gebruikers. De eerste benadering is uitgebreid bestudeerd, ook wel bekend als offline-learning-to-rank, en is de standaard binnen de industrie. Echter, geannoteerde gegevens geven mogelijk niet de juiste informatiebehoeften van de gebruikers weer en zijn niet actueel. De eerste benadering kan dus leiden tot sub-optimale rankers. De tweede benadering optimaliseert rankers door gebruik te maken van interactie-feedback. Dit proefschrift behandelt de tweede benadering. Namelijk, het leren van feedback van gebruikers-interactie. De redenen zijn tweeledig: (1) Elke dag hebben miljoenen gebruikers interactie met interactie-systemen waarmee zij een enorm aantal interacties genereren. Uit deze interactie-data van de gebruikers kan de informatiebehoeften van de gebruikers worden afgeleid. (2) Het leren van de interactie-gegevens biedt meer mogelijkheden bij het ontwerpen van de online-algoritmen.

Dit proefschrift gaat specifiek in op de taak om te leren van feedback van gebruikers op basis van clicks. De belangrijkste bijdrage van dit proefschrift is het introduceren van een algoritme voor veilig online learning to re-rank, genaamd BubbleRank, dat een belangrijk nadeel van online leren aanpakt, namelijk het veiligheidsprobleem. Het doet dit door de voordelen van zowel offline als online learning to rank te combineren. Het proefschrift stelt ook drie andere online algoritmen voor, die elk een uniek optimalisatieprobleem voor online rankers oplossen. Alle voorgestelde algoritmen zijn theoretisch correct en empirisch effectief.