Search engines that learn from their users

Schuth, A.G.

Publication date
2016
Document Version
Final published version

SEARCH ENGINES THAT LEARN FROM THEIR USERS

ANNE SCHUTH

Search Engines that Learn from Their Users

Anne Gerard Schuth

Search Engines that Learn from Their Users

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof.dr. D.C. van den Boom
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in
de Aula der Universiteit
op vrijdag 27 mei 2016, te 13:00 uur

door

Anne Gerard Schuth

geboren te Groningen

Promotiecommissie

Promotor:		
	Prof. dr. M. de Rijke	Universiteit van Amsterdam
Co-promotor:		
	Dr. S.A. Whiteson	University of Oxford
Overige leden:		
	Prof. dr. T. Joachims	Cornell University
	Dr. E. Kanoulas	Universiteit van Amsterdam
	Prof. dr. D. Kelly	University of North Carolina
	Dr. M.J. Marx	Universiteit van Amsterdam
	Prof. dr. M. Welling	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

SIKS Dissertation Series No. 2016-11
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

This research was supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 288024 (LiMoSINe project).

Cover by David Graus
Printed by Off Page, Amsterdam

ISBN: 978-94-6182-674-9

Acknowledgments

Almost nine years ago, I contacted a professor for a bachelor thesis project that had caught my eye. It was the start of a fruitful period during which I learned as much about the topic of my research and science in general as I did the people around me and myself. This period now comes to an end and I want to thank all the people that played an important role.

ILPS, the research group that adopted me while I was still a bachelor student, is a truly unique place to do research. While science is often seen as a solitary profession, I never experienced it as such. The atmosphere in the group has always been very supportive and cooperative. And for this I have to thank Maarten de Rijke, my supervisor, who built the group. I cannot thank Maarten enough. He taught me how to ask questions, which questions were important to ask, and how to answer them. His teaching, however, went far beyond science when we spent much of our weekly meetings talking about much broader topics than the papers we were writing. Thank you, Maarten.

Secondly, I want to thank Shimon Whiteson, my co-promotor, who has been a driving force for me. Coming from a more formal discipline, Shimon was there to point me in the right direction whenever my reasoning was flawed. Thank you for being persistent.

I am very honored to have Thorsten Joachims, Evangelos Kanoulas, Diane Kelly, Maarten Marx and Max Welling serving on my PhD committee.

I would also like to especially thank Maarten Marx for being very welcoming when I contacted him as a bachelor student. He first introduced me to research and showed me how much fun it could be. I also want to thank the other members of ILPS back then for allowing me to take a desk and have lunch and drinks with them: Balder, Bouke, Breyten, Edgar, Erik, Jiyin, Katja, Krisztian, Loredana, Marc, Manos, Nir, Valentin, and Wouter. Christof Monz, also in ILPS, then agreed to supervise my master thesis on statistical machine translation. I very much enjoyed working with him and the other SMTers: Bogomil, Simon, and Spyros. During the one year that I worked as scientific programmer, we had a great team with Arjan, Bart, Johan, Justin, and Lars.

I want to thank all the other people in and around ILPS that were part of my journey as a PhD student: Abdo, Adith, Aleksandr, Alexey, Amit, Anna, Arianna, Artem, Bob, Caroline, Cristina, Christophe, Chuan, Daan, Damiano, Damien, David, David, Diederik, Eva, Evangelos, Evgeny, Fares, Fei, Floor, Fons, Guangliang, Hamid, Harrie, Hendrike, Hosein, Ilya, Isaac, Ivan, Katya, Ke, Marlies, Marzieh, Masrour, Mustafa, Nikos, Petra, Praveen, Richard, Ridho, Shangsong, Tobias, Tom, Xinyi, and Zhaochun. I want to thank you for your friendship and for the countless evenings at Oerknal or wherever the conference train would bring us. Thank you Daan for looking out for others, David for disagreeing with me, Marlies for always being uplifting.

I want to thank all my co-authors. In particular Katja from whom I learned a lot. Harrie, who was always fast to produce and question results. I also want to thank Damien, Filip, Floor, Krisztian, Liadh and the students with whom I wrote papers.

I want to thank my students who have taught me more than I taught them.

I owe much to Petra and Caroline who took care of so much and were a pleasure to work with. Auke and Jeroen, thank you for taking care of our machines, but even more for all the discussions we had about infrastructure, I learned a lot.

During three months in Moscow, I had the pleasure of working with many great people at Yandex. In particular, I want to thank Damien, Eugene, and Pavel. In Cambridge, Filip and Katja were incredible mentors.

I want to thank the research community, of which I feel I have become part, for being awesome. It was great to meet all these wonderful people again and again wherever there was a conference.

Collaborations have been very valuable to me. I want to thank Joemon's research group in Glasgow for all the interactions we have had. I want to thank Sander Kieft at Sanoma for believing in our research. Agnes, Henning, Jakub, and Remko at Textkernel for showing me what search looked like in practice. Krisztian and Liadh for taking me on board of the Living Labs initiative. The collaborations with Seznam and SOARR that came from the Living Labs initiative have also been very valuable to me, thank you Jiří, Philipp, Narges for believing in the initiative.

I want to thank my fellow members of UvAPro and the FNWI PhD council. I enjoyed our many discussions. I also want to thank the ReThink UvA movement and in particular Sicco with whom I had countless discussions on what a university ideally should look like, and on how far we are from this ideal.

Ik wil graag mijn studiegenoten bedanken. Zij zorgden ervoor dat het meer dan een studie was: Aksel, Bram, Folkert, Gijs, Hanne, Hylke, Mark, Tjerk. En ik wil graag al mijn vrienden bedanken die bereid waren mijn eindeloze gepraat over mijn onderzoek aan te horen. Maar die er ook voor zorgden dat ik me er niet volledig in verloor. In het bijzonder bedank ik Bart, Bart-Jan, Guido, Willem Frederik, en Wisse.

Als laatste wil ik mijn familie bedanken. Mijn ouders, Paula en Gerard, omdat ze me altijd steunden. Wout en Rozalie voor het altijd tonen van interesse. Mijn oma, Sina, omdat ze het tolereert dat haar kleinzoon nog altijd studeert. Nina, mijn liefde, omdat ze er altijd is.

Contents

1 Introduction

Over half a billion web searches are performed every single day [46] by over half the world's population [204]. For many people, web search engines such as Baidu, Bing, Google, and Yandex are among the first resources they go to when any question arises. What is more, these web search engines have for many become the most trusted source of information, more so even than traditional media such as newspapers, news websites or news channels on television [54]. What web search engines present people with thus greatly influences what they believe to be true and consequently it influences their thoughts, opinions, decisions, and the actions they take. It matters a great deal what search engines present people with; more and more our world depends on them [184]. With this in mind, from an *information retrieval* (IR)[1] research perspective, two things are important. First, it is important to understand how well search engines perform and secondly this knowledge should be used to improve them. This thesis is about these two topics: *evaluation* of search engines and *learning* search engines.

Evaluation—understanding how well search engines perform—has always been a major area within IR research [155, page 250] and now that search engines have such a large impact on society, evaluating them is more important than ever. It is not only of commercial importance to the companies running web search engines, but now that more and more of the world's beliefs depend on it, evaluation is of great importance to society too. It is crucial to understand whether—or, to what degree—the trust people put into search engines is justified. Moreover, evaluation of search engines not only gives insight into the quality of the engine, it also enables reliable improvement of the search engine. As the British scientist Lord Kelvin stated, "if you cannot measure it, you cannot improve it" [190]. Evaluation serves as crucial guidance towards better search engines, towards search engines that serve their users better.

Traditionally, search engines were evaluated following the Cranfield approach [45, 155]. Using this approach, systems are evaluated in terms of document relevance for given queries, which is assessed by trained experts. While the Cranfield paradigm ensures high internal validity and repeatability of experiments, it has been shown that users' search success and satisfaction with an IR system are not always accurately reflected by standard IR metrics [187, 193].

One reason is that the judges typically do not assess queries and documents that reflect

[1] We maintain a list of acronyms at the back of this thesis, on page 163.

their own information needs, and have to make assumptions about relevance from an assumed user's point of view. Because the true information need can be difficult to assess, this can cause substantial biases. Conclusions drawn from analyses on data sets consisting of relevance assessments do not always agree with preferences from actual users of search engines [155].

A second difficulty with these data sets is that they are hard and expensive to produce and keep up to date. The manual effort required to obtain a substantial amount of high quality relevance judgments is enormous [7]. Doing so for an ever changing stream of queries coming from users is prohibitively expensive.

Besides the two issues mentioned before, data sets are actually not as easily reusable as they may seem at first glance. The Cranfield approach requires a diverse set of systems to produce rankings before any relevance judging takes place. A so-called pooling process combines the top k documents of each produced ranking and only these documents are actually assessed for relevance. As a consequence, only systems that are reasonably similar to the systems that produced the pool can ever be evaluated using the data set. A radically different system that might be much better than any of the systems contributing to the pool, would never be able to obtain a good score as none of the documents it ranks was ever assessed [196].

For all these reasons researchers have been looking for alternatives that do not suffer from these drawbacks. The Cranfield way of evaluating can be referred to as *offline evaluation* because there are no users interacting with a real, online system. In contrast, the alternative then is called *online evaluation*. Online evaluation uses the interactions of real and unsuspecting users of search engines for evaluation instead of professional human judges. These users have their natural information needs that made them decide to use the search engine. These users have a task in mind that they wish to solve. This task often extends beyond just search [114]. Search, for these users, is just a means to an end. These users come to the search engine, translate their information need into a keyword query and are determined to find what they need to complete their task. To this end they interact with the search engine. They may, for instance, reformulate their query or click around in result lists presented to them. All these traces of user interactions with the search engine are easily captured and can then be interpreted as implicit signals for evaluation. For instance, one could measure how often users need to reformulate their queries before they stop search, and presumably are satisfied with the results. Or one could measure where in a result list users click, the intuition being that it is better if users click higher in such a list because a user scanning the list from top bottom found it faster. Alternatively, one could measure the time it took until a user clicked a document, where the intuition again is that it is better if users spend less time to find what they were looking for. All these signals, and often many of them combined, are good indicators of performance of a search engine. Moreover, these interactions are the natural byproduct of users interacting with a search engine. When such interactions are used, explicit relevance judgments from trained assessors are no longer required. Without doing anything out of the ordinary, many search engines in fact already store all these interactions in their search logs. How to use these interactions for evaluation in a reliable and effective way is explored extensively in Part I of this thesis.

Search engines have become highly complex machines that incorporate hundreds or even

thousands of signals, each contributing to a part of the search solution. These signals are combined into a single ranking model—often called a *ranker*—that produces the rankings shown to users. A ranker is at the heart of every search engine. The signals—also referred to as features—that a ranker combines each serve a different purpose. Some reflect the importance of a document—a webpage in the case of web search—by looking at documents pointing to it and how important these documents are [139]. Other signals indicate whether the terms in the keyword query appear in the document and whether these terms are informative [146]. Many variants of such signals exist. A large body of research is aimed at improving the signals, while another large body of work looks into *learning to rank* (LTR): learning how to combine all these signals optimally [124]. The second part of this thesis is about the latter.

Traditionally, learning to rank, like evaluation, was done *offline*. Offline learning to rank is supervised machine learning: rankers are optimized based on their rankings for a fixed set of queries with an offline evaluation metric as target. All the downsides of offline evaluation, as described above, apply. It is expensive to produce data sets that contain human judgements, but more importantly, conclusions that are drawn from such data sets do not always agree with preferences of actual search engine users [155].

An alternative approach to learning optimal rankers uses online evaluation methods. Such an approach considers interactions of users with the search engines. A user's behavior such as clicks can reveal their preference. Since such interactions are readily available in large quantities, it makes sense to learn from them instead. Learning in this fashion, while interacting with users, is referred to as *online* learning to rank [82, 92, 144, 207] and is discussed in Part II of this thesis.

In the next section we outline the research in this thesis and the questions that are answered within it.

1.1 Research Outline and Questions

This thesis investigates *whether, how and to what degree search engines can learn from their users?* This thesis consists of three parts. All learning starts with evaluation: if you cannot measure it, you cannot improve it [190]. We therefore, in Part I, dive into evaluation of search engines before we investigate how this can be used for learning in Part II. This section sketches the outline of our research. Below, we introduce the research questions answered in the first two parts of thesis.

In Part III we investigate both online evaluation and learning *methodologies*. This part is not centered around research questions but rather around the design of these methodologies.

1.1.1 Evaluation of Search Engines

Part I discusses the evaluation of search engines.

Deployed search engines often have several teams of engineers tasked with developing potential improvements to the current production ranker. To determine whether the candidate rankers they develop are indeed improvements, such teams need experimental feedback about their performance relative to the production ranker. However, in order

to develop and refine those candidate rankers in the first place, they also need more detailed feedback about how the candidate rankers compare *to each other*. For example, to explore a parameter space of interest, they may be interested in the relative performance of multiple rankers in that space.

Several existing approaches could be used to generate this feedback. Firstly, assessors could produce relevance assessments from which offline metrics (e.g., MAP, nDCG, ERR [155]) could be computed. However, offline metrics do not tell the whole story since relevance assessments come from assessors, not users. Offline evaluation is described in detail in Section 2.2. Secondly, online experiments could generate user feedback such as clicks from which rankers could be evaluated. In particular, *interleaved comparison* [92, 93] methods enable such evaluations with greater data efficiency than A/B testing [144]. A/B testing is described in detail in Section 2.3.2 and interleaving is described in Section 2.3.3. In short, interleaving [93] is a highly sensitive online evaluation method that is often used at large scale commercial (web) search engines. One particular variant of interleaving, called Team Draft Interleaving, works as follows. It combines *two* rankings that are to be compared into a single ranking. This interleaved ranking is shown to a user and the ranker that contributed more documents that were clicked is preferred.

However, teams of engineers can easily produce enough candidate rankers that comparing all of them pairwise to each other using interleaving methods quickly becomes infeasible. To address this difficulty, we propose a new evaluation paradigm, which we call *multileaved comparison*, that makes it possible to compare more than two rankers at once. Multileaved comparisons can provide detailed feedback about how multiple candidate rankers compare to each other using much less interaction data than would be required using interleaved comparisons. We ask ourselves the following questions:

RQ1 Can multileaved comparison methods identify preferences between rankers faster than interleaved comparison methods?

RQ2 How does the sensitivity of multileaving methods compare to that of interleaving methods?

RQ3 Do multileaving methods improve over interleaving methods in terms of unbiasedness and online performance?

In particular, we first propose two specific implementations of multileaved comparisons. The first, which we call *team draft multileave* (TDM), builds on *team draft interleave* (TDI) [144], an interleaving method that assigns documents in the interleaved list to a team per ranker. Surprisingly, only a minor extension to TDI is necessary to enable it to perform multileaved comparisons, yielding TDM. However, despite its appealing simplicity, TDM has the important drawback that it requires multileavings, i.e., the result lists shown to the user, to be long enough to represent teams for each ranker.

Therefore, we propose a second method that we call *optimized multileave* (OM), which builds on *optimized interleave* (OI) [143], an interleaved comparison method that uses a *prefix constraint* to restrict the allowed interleavings to those that are "in between" the two rankers and then solves an optimization problem to ensure unbiasedness and to maximize sensitivity of the interleavings shown to users. OM requires deriving a new prefix constraint, new definitions of unbiasedness and sensitivity, a new credit function

upon which these definitions depend, and a new sampling scheme to make optimization tractable. We verify our motivation for introducing OM by answering the following question:

RQ4 Does OM scale better with the number of rankers than TDM?

Next, we propose *probabilistic multileave* (PM) which builds on *probabilistic interleave* (PI) [79]. We ask ourselves:

RQ5 How does PM compare to TDM and OM in terms of sensitivity, bias and scaling?

We empirically show that it is highly sensitive and unbiased and scales well to comparing many rankers. An important implication of this result is that historical interactions with multileaved comparisons can be reused, allowing for ranker comparisons that need much less user interaction data. Furthermore, we show that our method, as opposed to earlier sensitive multileaving methods, scales well when the number of rankers increases. Chapter 4 answers research questions RQ1 through RQ5.

The gold standard for information retrieval system evaluation is user satisfaction. However, as online user satisfaction is not directly observable, a significant amount of research has investigated how to summarize online behavior (such as clicks) into online metrics that best reflect user satisfaction.

Despite the advantages of interleaving and multileaving in terms of sensitivity, the most common online evaluation methodology is A/B testing [112]. Users of an online system are assigned to either a control or experimental condition, with the metric being computed on each population. However, large numbers of users are typically necessary to obtain reliable results as this approach has high variance. Interleaved evaluation is an alternative online approach previously shown to be much more sensitive. However, until now interleaved evaluation has not modeled user satisfaction as reliably as recent A/B metrics, resulting in low agreement with recent A/B metrics given realistic differences in IR system effectiveness.

We ask ourself the following two questions:

RQ6 How do A/B metrics compare to interleaving in terms of sensitivity and agreement?

RQ7 Can A/B metrics and interleaving be made to agree better without loosing sensivity?

Chapter 5 answers research questions RQ6 and RQ7.

1.1.2 Learning Search Engines

In Part II of this thesis, we turn to learning using the evaluation methods described above. Traditional approaches to evaluating or optimizing rankers are based on editorial data, i.e., manually created explicit judgments. Recent years have witnessed a range of alternative approaches for the purpose of evaluating or optimizing rankers, approaches that reduce or even avoid the use of explicit manual judgments.

One type of approach is based on so-called pseudo test collections, where judgments about query-document pairs are automatically generated by repurposing naturally occurring labels such as hashtags or anchor texts [10, 21].

Another type of approach is based on the use of implicit signals. The use of implicit signals such as click data to evaluate or optimize retrieval systems has long been a promising alternative or complement to explicit judgments. Evaluation methods that interpret clicks as absolute relevance judgments have often been found unreliable [95]. In some applications, e.g., for optimizing the click-through rate in ad placement and web search, it is possible to learn effectively from click data, using various learning to rank methods, often based on bandit algorithms [77, 207]. Click models can effectively leverage click data to allow more accurate evaluations with relatively little editorial data [30]. Moreover, interleaved comparison methods have been developed that use clicks not to infer absolute judgments but to compare rankers by observing clicks on interleaved result lists [144].

The state-of-the-art click-based optimization of IR systems has focused on optimizing a linear combination of the base rankers, thereby treating those rankers as black boxes [83, 207]. In this chapter, we try to break open those black boxes and examine whether online learning to rank can be leveraged to optimize those base rankers themselves. Surprisingly, even though a lot of work has been done on improving the weights of base rankers in a combined learner, there is no previous work on online learning of the parameters of base rankers and there is a lot of potential gain from this form of optimization. We investigate whether individual base rankers can be optimized using clicks. This question has two key dimensions. First, we aim to use clicks, an implicit signal, instead of explicit judgments. The topic of optimizing individual base rankers such as *term frequency times inverse document frequency* (TF.IDF), *best match 25* (BM25) or *divergence from randomness* (DFR) has received considerable attention over the years but that work has almost exclusively used explicit judgments. Second, we work in an online setting while previous work on optimizing base rankers has almost exclusively focused on a more or less traditional, Cranfield-style, offline setting.

Importantly, the problem of optimizing base rankers is not the limiting case of the problem of optimizing a linear combination of base rankers where one has just one base ranker. Unlike the scoring function that represents a typical online learning to rank solution, the scoring function for a single base ranker is not necessarily linear, meaning that the ranker is not necessarily a linear combination of raw ranking features. A clear example is provided by the well-known BM25 ranker [146], which has three parameters that are related in a non-linear manner: k_1, k_3 and b.

In Chapter 6, we pursue the problem of optimizing a base ranker using clicks by focusing on BM25. Currently, it is common practice to choose the parameters of BM25 according to manually tuned values reported in the literature, or to manually tune them for a specific setting based on domain knowledge or a sweep over a number of possible combinations using guidance from an annotated data set [147]. We propose an alternative by learning the parameters from click data. Our goal is not necessarily to improve performance over manually tuned parameter settings, but rather to obviate the need for manual tuning.

Specifically, the research questions we aim to answer are as follows.

RQ8 How good are the manually tuned parameter values of BM25 that are currently used? Are they optimal for all data sets on average? Are they optimal for individual data sets?

RQ9 Is it possible to learn good values of the BM25 parameters from clicks? Can we approximate or even improve the performance of BM25 achieved with manually tuned parameters?

Chapter 6 answers research questions RQ8 and RQ9.

Traditionally, learning in the context of information retrieval in general was done *offline* by optimizing for performance on a training set consisting of queries and relevance assessments produced by human assessors. As pointed out previously, such data sets are time consuming and expensive to produce and assessments are not always in line with actual user preferences [155]. User interactions, and in particular clicks, are readily available and have proven to be a valuable source of information when interpreted as a preference between either rankings [144] or documents [92]. In particular, as described above, when clicks are interpreted using interleaved comparison methods, they can reliably infer preferences between a pair of rankers [34, 92, 93].

Dueling bandit gradient descent (DBGD) [207] is an online learning to rank algorithm that learns from these interleaved comparisons. It uses the inferred preferences to estimate a gradient, which is followed to find a locally optimal ranker. At every learning step, DBGD estimates this gradient with respect to a *single* exploratory ranker and updates its solution if the exploratory ranker seems better. Exploring *more than one* ranker before updating towards a promising one could lead to finding a better ranker using fewer updates. For this purpose we can use multileaved comparisons such as TDM, as introduced above. In this way, our proposed method, *multileave gradient descent* (MGD), aims to speed up online learning to rank.

We propose two variants of MGD that differ in how they estimate the gradient. In *MGD winner takes all* (MGD-W), the gradient is estimated using one ranker randomly sampled from those that won the multileaved comparison. In *MGD mean winner* (MGD-M), the gradient is estimated using the mean of all winning rankers.

The research questions we aim to answer are the following.

RQ10 Can MGD learn faster from user feedback (i.e., using fewer clicks) than DBGD does?

RQ11 Does MGD find a better local optimum than DBGD?

RQ12 Which update approach, MGD-W or MGD-M, learns faster? Which finds a better local optimum?

Chapter 7 answers research questions RQ10 through RQ12.

1.1.3 Online Evaluation and Learning Methodology

Part III of this thesis discusses the methodology of online experimentation. This part is of a very different nature than the earlier two parts. As opposed to the earlier chapters, in this part we will no longer study algorithms. IR research has always been driven by a combination of algorithms, shared resources, and evaluation. In Part III we will introduce a new shared resource in the form of a learning framework. We will also introduce a new evaluation paradigm. Our research in this last part of the thesis will not be centered around

research questions but rather around designing these shared resources and designing the new evaluation methodology.

Performing online experiments in principle requires access to real users. This is often not available to researchers. And what is more, very new ideas are often not suitable for being exposed to real users. We therefore create an online learning to rank framework which allows for experimenting with simulated users. We implemented and distribute our framework in an open source software package called *learning and evaluating rankers online toolkit* (Lerot). In Lerot, presented in this thesis, we bundle all ingredients needed for experimenting with online learning to rank for IR. Lerot includes several online learning algorithms, interleaving methods and a full suite of ways to evaluate these methods. In the absence of real users, the evaluation method bundled in the software package is based on simulations of users interacting with the search engine. The software presented here has been used to verify findings of over six papers at major information retrieval venues over the last few years. We describe Lerot in detail in Chapter 8.

Finally, we turn to experiments with real users of search engines by introducing OpenSearch, the living labs for IR evaluation. The idea with this living lab is to provide a benchmarking platform for researchers to evaluate their ranking systems in a live setting with real users in their natural task environments. OpenSearch represents the first attempt to offer such an experimental platform to the IR research community in the form of a community challenge. In this thesis we describe how living lab experiments can be performed, how such experiments are actually run, what the resulting outcomes are, and provide a detailed analysis of the use-cases and a discussion of ideas and opportunities for future development research directions. OpenSearch is described in detail in Chapter 9.

1.2 Main Contributions

In this section we summarize the main contributions of this thesis. Our contributions come in the form of algorithmic, theoretical, empirical, and software contributions.

1.2.1 Algorithmic Contributions

We list five algorithmic contributions. The first two are online evaluation methods that extend interleaving methods. Algorithmic contributions 3 and 4 deal with bringing the outcomes of interleaving methods in line with A/B testing. Our last contribution of this type, contribution 5, is an online learning to rank method.

1. We introduce three implementations of multileaved comparisons methods *team draft multileave* (TDM), *optimized multileave* (OM), and *probabilistic multileave* (PM) (cf. Chapter 4).

2. Our extension of *probabilistic interleave* (PI) to *probabilistic multileave* (PM) is a multileaving method that is able to reuse historical interaction data (cf. Chapter 4).

3. We propose novel interleaving *credit functions* that are (1) designed to closely match the implementation and parameters of A/B metrics; or (2) are parameterized

to allow optimization towards agreement with arbitrary A/B metrics (cf. Chapter 5).

4. We further propose the first approach for automatically maximizing agreement between interleaving credit functions and A/B metrics (cf. Chapter 5).

5. Lastly, we propose several *multileave gradient descent* (MGD) approaches to using multileaved comparison outcomes in an online learning to rank method (cf. Chapter 7).

1.2.2 Theoretical Contributions

This thesis contains four theoretical contributions. We first propose a new online evaluation paradigm. Secondly, we propose a statistical method for assessing sensitivity of evaluation methods. Contribution 8 is insight into the parameter space of BM25. Our last theoretical contribution is in the area of online evaluation methodology.

6. A novel online ranker evaluation paradigm in which more than two rankers can be compared at once, called multileaved comparisons (cf. Chapter 4).

7. Starting with existing A/B and interleaving metrics, we propose a new, statistical method for assessing the sensitivity of these metrics from estimated effect sizes. The resulting method allows a detailed comparison between metrics in terms of the power of statistical tests at varying sample sizes. Our analysis shows that A/B tests typically require two orders of magnitude more data than interleaved comparisons (cf. Chapter 5).

8. The insight that we can potentially achieve significant improvements of state-of-the-art learning to rank approaches by learning the parameters of base rankers, as opposed to treating them as black boxes which is currently the common practice (cf. Chapter 6).

9. We introduce OpenSearch, the living lab for IR evaluation, a completely new evaluation paradigm for IR. Within the OpenSearch paradigm the idea is to perform experiments in situ, with real users doing real tasks using real-world applications (cf. Chapter 9).

1.2.3 Empirical Contributions

We list a total of ten empirical contributions. Empirical contributions 10 through 13 compare our new online evaluation algorithms to existing baselines. Contributions 14 through 16 are on the topic of bringing interleaving methods and A/B testing in agreement. Contributions 17 and 18 demonstrate the parameter space of BM25 and how BM25 can be optimized by learning from users. Our last contribution is that we empirically show that online learning to rank methods that use our multileaving methods outperform baselines.

10. We provide a thorough experimental comparison of TDM and OM against each other and against TDI and OI that shows that multileaved comparison methods can

find preferences between rankers much faster than interleaved comparison methods (cf. Chapter 4).

11. Our experiments also show that TDM outperforms OM unless the number of rankers becomes too large to handle for TDM, at which point OM performs better (cf. Chapter 4).

12. Our experiments show that, when the differences between evaluated rankers are varied, the sensitivity of TDM and OM is affected in the same way as for TDI and OI (cf. Chapter 4).

13. We show experimentally that PM is at least as sensitive as TDM (which in turn is more sensitive than OM), that PM is unbiased, and that PM scales well when the number rankers that are compared increases (cf. Chapter 4).

14. We demonstrate that interleaving credit functions can be automatically optimized, and that learned parameters generalize to unseen experiments. These results demonstrate for the first time that interleaving can be augmented with user satisfaction metrics, to accurately predict the outcomes of A/B tests that would require one to two orders of magnitude more data (cf. Chapter 5).

15. We find that current interleaved comparisons achieve from random up to 76% agreement with A/B user satisfaction metrics (cf. Chapter 5).

16. Our empirical results, obtained from experiments with 3 billion user impressions and 38 paired (A/B and interleaving) experiments demonstrate the effectiveness of approach to bringing the outcome of interleaving and A/B tests in agreement. In particular, we achieve agreement of up to 87%, while maintaining high sensitivity (cf. Chapter 5).

17. A demonstration of how parameters of an individual base ranker such as BM25 can be learned from clicks using the dueling bandit gradient descent approach (cf. Chapter 6).

18. We gained empirical insight into the parameter space of a base ranker such as BM25 (cf. Chapter 6).

19. Extensive empirical validation of our new online learning to rank methods that use multileaved comparisons methods show that MGD-W and MGD-M outperform the state of the art in online learning to rank (cf. Chapter 7).

1.2.4 Software Contributions

We have two software contributions, both are open source and have been contributed to by several others.

20. We contribute Lerot, an open source implementation of our online learning to rank framework that has all batteries included. The framework is easily extensible to compare the implemented methods to new online evaluation and online learning approaches (cf. Chapter 8).

21. We introduce OpenSearch, the first living labs for IR evaluation benchmarking platform and provide an open source implementation (cf. Chapter 9).

1.3 Thesis Overview

This section provides an overview of this thesis. We finish this section with reading directions.

The first chapter, to which this section belongs, gives an introduction to the subject of this thesis. This chapter also provides an overview of the research questions, the contributions and the origins of this work. Chapter 2 then introduces the background for all six research chapters that follow. The background material mostly provides background for the first two parts this thesis. Chapter 3 is a short chapter introducing the experimental setup used in Chapters 4, 6 and 7.

Part I of this thesis contains research chapters related to online evaluation of search engines. In particular, Chapter 4 introduces an extension to interleaving methods which is called multileaving. Chapter 5 then looks into the relation between A/B testing and interleaving.

Part II of this thesis uses the evaluation methods, discussed in the first part, for learning purposes. Chapter 6 uses interleaved comparisons for tuning parameters of BM25, a popular retrieval method. Chapter 7 uses multileaved comparison methods introduced in Chapter 4 as feedback for its learning method.

Part III is of a different nature than the earlier two parts. This part is more practical in nature and describes two evaluation methodologies and shared resources. Chapter 8 introduces an online simulation framework which acts as a shared resource for the community. Chapter 9 describes OpenSearch, a completely new evaluation paradigm that is an alternative to Cranfield-style evaluation and uses real users using real search engines.

Lastly, in Chapter 10 we draw conclusions and we give an outlook to future work.

Readers familiar with background on online evaluation, online learning to rank may skip over the respective parts of Chapter 2. Chapter 3 can be read only to understand the details of the experiments in Chapters 4, 6 and 7.

Part I and Part III can be read independently of other parts. However, Part II assumes Part I, and in particular Chapter 4, as prerequisite.

1.4 Origins

We list for each research chapter the publications on which it is based. For each publication we mention the role of each co-author. The thesis is based on in total 8 publications [15, 161–167].

Chapter 4 is based on *Multileaved Comparisons for Fast Online Evaluation* published at CIKM'14 by Schuth, Sietsma, Whiteson, Lefortier, and de Rijke [163]. Schuth performed the experiments for [163], all authors contributed to the design of the

algorithms and to the text. The chapter further builds on *Probabilistic Multileave for Online Retrieval Evaluation* published at SIGIR'15 by Schuth, Bruintjes, Büttner, van Doorn, Groenland, Oosterhuis, Tran, Veeling, van der Velde, Wechsler, Woudenberg, and de Rijke [165]. For this paper Büttner, van Doorn, Tran, Veeling and Wechsler delivered the first version of the text. Bruintjes, Groenland, Oosterhuis, van der Velde and Woudenburg were involved in the design and implementation of the experiments. Schuth wrote much of the final version and reimplemented the experiments. De Rijke contributed to the text.

Chapter 5 is based on *Predicting Search Satisfaction Metrics with Interleaved Comparisons* published at SIGIR'15 by Schuth, Hofmann, and Radlinski [166]. Schuth performed most of the experiments. All authors contributed equally to the text.

Chapter 6 is based on *Optimizing Base Rankers Using Clicks: A Case Study using BM25* published at ECIR'14 by Schuth, Sietsma, Whiteson, and de Rijke [162]. Schuth performed most of the experiments. All authors contributed to the text.

Chapter 7 is based on *Multileave Gradient Descent for Fast Online Learning to Rank* published at WSDM'16 by Schuth, Oosterhuis, Whiteson, and de Rijke [167]. All authors contributed to the design of the algorithms. Oosterhuis performed most of the experiments. Schuth wrote a first version and all authors contributed to the text.

Chapter 8 is based on *Lerot: an Online Learning to Rank Framework* published at the Living Labs Workshop at CIKM'13 by Schuth, Hofmann, Whiteson, and de Rijke [161]. All authors contributed to the conception of Lerot. Hofmann implemented the predecessor of Lerot. Hofmann and Schuth actually implemented Lerot. Schuth initiated writing and the other authors contributed to writing.

Chapter 9 is based on *Head First: Living Labs for Ad-hoc Search Evaluation* published at CIKM'14 by Balog, Kelly, and Schuth [15]. All authors contributed equally and author order was alphabetical. This chapter is also based on *Extended Overview of the Living Labs for Information Retrieval Evaluation (LL4IR) CLEF Lab 2015* by Schuth, Balog, and Kelly [164]. All authors contributed to the text. The experiments were run on the OpenSearch platform implemented mostly by Schuth.

This thesis also, but indirectly, builds on joint work on *interleaving methods for aggregated search* [36, 39, 41], *evaluation of recommender systems* [84], and *online learning to rank* [83, 86, 138].

Other work, not directly related to this thesis, did contribute to insight in the broader research areas of *evaluation* [159], *information extraction* [160], and *data integration* [129, 130, 137].

2 Background

In this chapter we set the stage for the research chapters in this thesis. In Section 2.1, we start with a historical introduction into the field of *information retrieval* (IR) including description of modern IR methods and retrieval models. We then continue with offline evaluation in Section 2.2, online evaluation in Section 2.3, offline learning to rank in Section 2.4, and finally online learning to rank in Section 2.5.

Section 2.1 serves as background to the whole thesis. Sections 2.2 and 2.3 are background to Part I. The Sections 2.4 and 2.5 serve as background to Part II.

2.1 Information Retrieval

We give a historical overview of the scientific field of IR starting in 300BC up, leading up to a modern view on the field followed by an overview of retrieval models.

2.1.1 A Brief History of Information Retrieval

The *Pinakes* may have been the first IR system. Callimachus (310/305-240 BC) compiled this catalog of the Library of Alexandria [55]. The Pinakes contained an—what we would now call—inverted index: an alphabetical list of topics with for each topic a list of papyrus scrolls about the topic and where they could be found. In total 500,000 scrolls were indexed this way in an indexing system that itself spanned 120 volumes, even for today's standards not a small collection.

The first automated IR system was developed in 1890 for the United States Census [156]. It was required by law to perform a census every 10 years but it was estimated that by 1890 the United States had grown so large that the census would take 13 years. An electrical tabulating machine was devised by Hollerith [87] that could perform all the calculations. Each citizen was represented by a punchcard, where each hole would represent properties such as age groups, gender, marital status etc. The machine then could operate on a batch of 10,000 such cards and sort them based on several criteria using boolean logic, much like components of modern IR systems.

In 1931 Goldberg [62] patented what he called a *statistical machine*, which was a search engine he had been working on since the twenties. The search engine used photoelectric cells and techniques much like *optical character recognition* (OCR) for searching in the metadata of documents that were stored on microfilm.

In 1945 Bush [26] published an article describing a *memory extender* (memex) he envisioned. The memex was a desk which would contain all the owners' personal documents on microfilm and provided screens and a keyboard to search through them. He also pioneered the idea of documents linking to each other and was later credited to have invented the hypertext system that now underlies the world wide web.

Mooers [135] was the first to introduce the term and problem of "information retrieval" at the 1950 meeting of the Association for Computing Machinery (ACM). He introduced it to this scientific community as he believed the development of IR should be taken up by members of this association. Mooers not only introduced the problem but he also had ideas about how IR systems should be evaluated, namely on two criteria: 1) whether the systems do their job; and 2) how expensive is it to operate such a system [134]. These areas that Mooers identified are still active research areas, and IR research is indeed carried out by the community that he had envisioned.

2.1.2 Modern Information Retrieval

Since Mooers coined the term, the field of IR rapidly evolved, driven by the need to search through the ever larger volumes of information that are being produced and stored. This was further accelerated with the advent of the internet and the increased access to digital equipment—such as personal computers and recently mobile devices—by the masses. Despite the intensified research in recent years, many of the methods developed in the early years of IR research are still very central to modern IR systems. We therefore describe them in some detail here, starting with a widely accepted definition of IR, which we borrow from Manning et al. [127, Chapter 1]:

> "Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

So, in essence, IR starts with an information need. An information need [186] is born in the head of a person and can be unrecognized by the person as such. If this person would use an IR system to satisfy the information need, we would refer to this person as a *user*. The act of retrieving information used to be a librarian's task. A user would explain the information need to a librarian who would then know how to search through a collection of books and articles. Nowadays the more common scenario is that a user translates their need into keywords and enters them into a search engine. Search engines are now the predominant instance of IR systems and in this thesis the terms "information retrieval" and "search" are interchangeable.

Modern IR as we know it started at the 1958 International Conference on Scientific Information, as recalled by Sparck Jones [172]. Following that conference, researchers considered automating retrieval tasks that were manual tasks until then [50, 151]. This progress combined with the growing amounts of scientific literature culminated in the Cranfield experiments by Cleverdon [43–45], setting the stage for much IR research today. The Cranfield evaluation paradigm is described in more detail in Section 2.2.1 and is highly popular until this date. Nevertheless, this thesis argues that, even though this paradigm has been instrumental in the growth of the field of IR, with the abundance of (web) search engines it is time to move at least partially away from Cranfield-style evaluation, towards

evaluation methodologies that take actual users with actual information needs into account. In fact, the type of evaluation this thesis proposes is moving closer to evaluating IR as it was defined by Manning et al. when compared to Cranfield-style evaluation. The style of evaluation this thesis proposes takes actual users with actual information needs into account, while Cranfield-stye evaluation requires human assessors that will assess documents for relevance with respect to some imagined information need. We discuss this in more detail in Section 2.3. The International Conference on Scientific Information in 1958 was also where Vickery [195] presented his view on relevance, which turned out to be influencial [132]. Vickery introduced a distinction between "relevance to a subject" and "user relevance", between topical relevance and relevance towards the information need of a user.

2.1.3 Retrieval Models

Perhaps the oldest well known retrieval model is the *boolean model* (BM) [116]. Incidentally, this model is still widely used. Documents are identified by a set of terms. An inverted index is then created for Boolean retrieval. For each term in the vocabulary this index lists which documents contain it. For each term in the query then, the BM obtains the set of documents that contain the term. The Boolean operators in the query are used to determine, using Boolean logic, what the set of documents is that answers the query. This is a simple, efficient and elegant solution to information retrieval. But it has several drawbacks as listed by Salton et al. [154, page 1]. Firstly, the size of the answer set is difficult to control. It can easily be too large a set of documents or it can be empty. Secondly, there is no ranking defined over the documents in the answer. Thirdly, terms in the query are all weighted equally. Lastly, Salton et al. state that formulating a Boolean query is not always very intuitive. However, perhaps the biggest problem of the BM is the mismatch with the "anomalous state of knowledge" [18] of users; users only have a vague notion of their information need and the sort of document that may satisfy it. It is therefore, for a user, very difficult to formulate a query in Boolean terms.

The *vector space model* (VSM) [153] addresses all the drawbacks of the BM as listed by Salton et al. The vector space model represents both documents and queries in a space of terms. Documents can then be ranked by how close they are to the query in this space. In this space, terms can be weighted. Term are trivially weighted by their *term frequency* (TF) [126] or more effectively terms can be weighted by the TF multiplied by their *inverse document frequency* (IDF) to arrive at the *term frequency times inverse document frequency* (TF.IDF) model [171]. The idea behind IDF is that the "specificity of a term can be quantified as an inverse function of the number of documents in which it occurs", as stated by Sparck Jones [171]. This idea remained very influential in later ranking models.

Salton et al. [154] introduced the *extended Boolean model* as an intermediate between the BM and the VSM. It preserved both the Boolean logic and queries as well as the ranking and term weighting from the VSM.

Robertson [145] formalized the concept of ranking documents in his *probability ranking principle*: a system's performance is optimal when it ranks documents in decreasing probability of relevance to the user. This led to the development of many new retrieval models of which *best match 25* (BM25) by Robertson and Walker [146] is still widely

used. BM25 is a weighted version of TF.IDF that corrects for document length.

More recent retrieval models are based on *language models* (LM) in which documents and queries are modeled as sequences drawn from a probability distribution over sequences. Documents are then ranked by the probability that document and query come from the same underlying distribution [76, 140]. *Divergence from randomness* (DFR) [8] is an even more recent retrieval model, based on the 2-Poisson indexing model by [69]. The idea behind DFR is that terms cary more meaning if their document frequency diverges from the collection frequency.

The retrieval models mentioned so far take only the terms of a document and query into account while there are additional sources that could be used. One of the earliest attempts to including additional sources was by Garfield [61]. He used citation counts to rank scientists and their work. He showed high correlation with Nobel prizes, indicating that citations were a useful external source of information. When the world wide web gained popularity, it was realized by several that hyperlinks could be used just like citations in scientific literature rankings. Two examples of such algorithms from that time were HITS [111] and PageRank [139], which was based on HITS. PageRank led to Page and Brin founding Google that since then grew into the most widely used web search engine. Since then, the most prominent instance of information retrieval has been web search.

Broder [23] explored how information needs in web search can differ. In his study less than half the queries were informational, which was earlier assumed to be the predominant information need. Figuring out the intent of the user and behind the query has become an important topic [117]. Why this is important is illustrated by Teevan et al. [188], who describe the *potential for personalization* as the gap in performance between a search engine responding to the average user and a search engine personalizing for each user. Additional sources that IR systems tap into for this purpose are *user sessions*; previous queries issued by the user [20, 203]. Other contexts of the user such as the location [19] of a user are also considered. With the advent of social networks, signals arising from these networks can also be integrated in ranking systems [16].

In much of this thesis, the retrieval models discussed above are considered as *features* or *ranking signals* that can be combined into *rankers*. In Section 2.4 we discuss the background of methods for combining features. Note how the earlier models, such as BM, VSM and BM25 take both the query and document into account while the later models such as HITS and PageRank only take the document into account. We call the former *query-document features* while we call the latter *document features*. Query intent, or other annotations of the query can be seen as *query features*.

We refer the reader to Manning et al. [127], Salton [152], Sanderson and Croft [156], Sparck Jones [173] who all write extensive overviews—each from their perspective and moment in time—of the history of IR in much more depth than can be covered in this thesis.

2.2 Offline Evaluation

The research discussed so far was mostly in the area of IR *systems*. However, the experimental evaluation of IR systems, i.e., rankers, has always been a central topic in IR research, as was already foreseen by Mooers [134]. Offline evaluation was the

predominant form of evaluation in IR for a long time. In particular Cranfield-style evaluation has been very influential in the formation of the field.

2.2.1 Cranfield-style Evaluation

Cranfield-style evaluation, as described by Cleverdon and Keen [43–45], uses a fixed document collection, a fixed set of queries, and relevance judgments for the documents in the collection with respect to the queries. These judgements are produced by trained assessors. These assessors are asked to follow carefully composed judging guidelines. Often a narrative for each information need underlying a query is composed. The judges then judge the relevance of documents with respect to that narrative [198]. The documents that are to be judged come from a pooling process: experimental systems that are to be evaluated produce their rankings for each query. For each query the top k documents from each of these rankings are taken and the union of the documents is judged. The result is that for experimental systems that took part in the pooling, all documents in the top k are judged for relevance. However, an experimental system that did not take part in the pooling does not necessarily have all documents in its top k judged. For this reason it is vital to have a diverse set of experimental systems contributing to the pooling. It is typically assumed that documents without judgments are not relevant, which is a relatively save assumption if the pooled systems were of high quality and diverse [197]. The Cranfield evaluation paradigm is the predominant form of evaluation in the *text retrieval conference* (TREC) [197] which has for the last 25 years organized the largest IR evaluation campaigns.

The relevance judgements are used to compute IR evaluation metrics so that systems can be compared. Originally, metrics such as recall—the fraction of relevant documents that is retrieved—and precision—the fraction of the retrieved documents that is relevant—were considered. But these metrics do not take the ranking of documents into account, nor do they allow for graded relevance. Many metrics have been introduced since. Popular metrics include *mean average precisions* (MAP) [68], *normalized discounted cumulative gain* (nDCG) [89] and *expected reciprocal rank* (ERR) [33]. Sanderson [155] gives a thorough overview of TREC and its metrics.

More task specific metrics have been proposed as well, such as intent aware metrics by Clarke et al. [42], diversification metrics by Agrawal et al. [1] and metrics for evaluating faceted search systems by Schuth and Marx [159].

In this thesis, we follow Hofmann [77] and use nDCG by Järvelin and Kekäläinen [89]. This metric allows for graded relevance and it takes rankings into account. Moreover, nDCG has been used in the validation of online evaluation (see Part I) and online learning systems [142, 207] (see Part II). Both lie close to the content of this thesis. Still following Hofmann [77], we use the formulation of nDCG as introduced by Burges et al. [24]. This is a slight variation of the original formulation by Järvelin and Kekäläinen so that differences at the highest two positions can also be measured. We explain our evaluation setup in detail in Section 3.4.

Offline evaluation has the benefit that it becomes easy to compare rankers and thus easy to try out new things, as long as the new experimental systems are close enough to the systems used in pooling [122]. Lipani et al. [123] recently worked on methods to reduce this pooling bias. With Cranfield-style evaluation, experiments also become

repeatable. The downside of this type of evaluation is that it expensive to obtain reliable relevance judgements. Moreover, these relevance judgements are not always in line with actual user preferences [75, 100, 155, 192, 206]. Carterette and Allan [29] and Sanderson and Joho [157] discuss approaches to building test sets for offline evaluation at low cost. Several attempts have been made to either simulate human queries or generate relevance judgments without the need of human assessors for a range of tasks. One recurring idea is that of pseudo test collections, which consist of automatically generated sets of queries and for every query an automatically generated set of relevant documents (given some document collection). The issue of creating and using pseudo test collections goes back at least to [183]. Azzopardi et al. [13] simulate queries for known-item search and investigate term-weighting methods for query generation. Asadi et al. [10] describe a method for generating pseudo test collections for training learning to rank methods for web retrieval; they use anchor text in web documents as a source for sampling queries, and the documents that these anchors link to are regarded as relevant documents for the anchor text (query). Berendsen et al. [21] use a similar methodology for optimizing microblog rankers and build on the idea that tweets with a hashtag are relevant to a topic covered by the hashtag and hence to a suitable query derived from the hashtag.

2.2.2 User Studies

Another way to evaluate rankers is through *user studies*. Such studies are usually conducted in a lab setting, as described extensively by Kelly [101]. Because one needs a lab and one needs to recruit users, these studies are expensive, hard to repeat and laborious and expensive to scale up. User studies do have the advantage that there is a large amount of control over the users. Additionally, many things that are otherwise unreachable can be measured. For instance, eye gaze can be tracked [63] and one can even measure brain activity [5]. User studies rarely scale beyond dozens to at most hundreds of subjects. Typically this number is determined by a power analysis. Still, one should be careful not to draw conclusions that generalize towards the whole population of users of a system [101].

2.3 Online Evaluation

Since the advent of web search, there has been a focus on online evaluation. The number of users interacting with search engines and the number of interactions of each of them has increased tremendously, from only professional users in libraries to over half a billion web searches per day [46][1], by over half the world's population [204]. It was realized that these interactions can be used for evaluation.

Deployed search engines often have several teams of engineers tasked with developing potential improvements to the current production ranker. To determine whether the candidate rankers they develop are indeed improvements, such teams need experimental feedback about their performance relative to the production ranker. They need to evaluate their rankers. One of the first and most obvious things to do is to use one of the methods described above. One would start with offline evaluation as described in Section 2.2 and

[1] These are only explicit searches by users, meaning that on a given day, over half a billion times a user navigated to a web search engine and typed in their keyword query.

maybe perform a user study as described in Section 2.2.2. Then when there is enough evidence that the candidate ranker is indeed an improvement over the current production ranker it should probably replace the production ranker. However, offline metrics do not tell the whole story since relevance assessments come from assessors, not users. And user studies do not tell the whole story since the number of subjects is rarely large enough to draw reliable enough conclusions.

Online experiments, as described by Kohavi et al. [113], however, can be used to reliably determine how users interact with a change in the ranker and whether the change really constitutes an improvement. Online experiments rely on real users of a real search engine. One variant, called *A/B testing*, compares two rankers by showing ranker A to one group of users and ranker B to another group and then tries to infer a difference between the systems from differences in observed behavior. We describe this variant in Section 2.3.2. The other variant, called interleaving, combines rankings from both A and B and shows the combined results to all users and then tries to infer differences between the systems by interpreting user behavior. We describe interleaving in Section 2.3.3. We then turn to the K-armed dueling bandits problem in Section 2.3.4 and quickly to counterfactual reasoning in Section 2.3.5. In Section 2.3.6 we discuss *click models* used to simulate users interacting with a search engine.

2.3.1 Interpreting User Interactions

We start, however, with an overview of how user interactions can be interpreted. The idea of using the interactions of users with search engines to either evaluate or improve the search engine is not new. Ruthven and Lalmas [149] write an extensive survey of *relevance feedback* methods, these are methods that allow users to *explicitly* mark documents as relevant in an initial ranking. These labels coming from the users are then used to generate a better ranking. Rocchio [148] was the first to formalize this idea. The type of feedback gathered though through a relevance feedback mechanism is *explicit*, in the sense that the user is explicitly asked to provide this feedback. Other ways of collecting explicit feedback is for instance through side-by-side comparisons [2, 189]. Explicit feedback has as drawback that it disturbs users in their normal interaction with search engines.

On the other hand, *implicit* feedback results from the natural interaction of users with the search system [102]. Implicit feedback consists for instance of query reformulations [71], mouse clicks [95], mouse movements [48, 66, 67, 73, 201], measurements of dwell-time [206]—the time users spend on a website—, or even the time a search result [115]. The advantage of implicit feedback over explicit feedback is that it is available in abundance. All user interactions with the search engine are typically recorded anyway so billions of such signals are generated every single day.

A major drawback, however, of implicit feedback is that it is inherently noisy, which makes interpretation challenging. What is more, user interactions are heavily biased by the systems that were used to collect them. This makes these sort of interactions much less fit for reuse, unlike judgements from human assessors in a Cranfield-style evaluation setting (see Section 2.2.1). There has been work that addresses this, which we discuss in Section 2.3.5.

Even though implicit user interactions are available in abundance, there is often a stiff competition among algorithms—and their developers—to be evaluated online [108, 109].

This is due to a combination of factors mentioned above: (1) the fact that it is hard to reuse user interactions for evaluation; and (2) the noise in this type of feedback which can be countered only by large amount of feedback. For this reason it is vital for an online evaluation method to be highly sensitive, that is, use as little user interaction as possible to draw reliable conclusions. Recently, Kharitonov et al. [108] worked on scheduling of online experiments such that the most promising would be evaluated first. Kharitonov et al. [109] introduced a method to prematurely stop an online experiment if it is known to be converged, which also avoids wasting valuable user interactions.

2.3.2 A/B Testing

The most common type of online evaluation today is *A/B testing*. A/B testing compares two rankers by showing ranker A to one group of users and ranker B to another group. A/B testing then tries to infer difference between the two rankers by computes metrics based on observed behavior of both groups. Kohavi et al. [113] describes this methodology in the context of web search. Standard assumptions allow experimenters to obtain unbiased online performance estimates, and confidence estimates or hypothesis testing are available via statistical methods such as the two-sample t-test.

A/B metrics are *absolute click metrics* that are computed for ranker A and B, after which a t-test is used to select a winner. For instance, this methodology has been effectively used to compare systems in terms of *click through rate* (CTR) (e.g., for news recommendation [121]). Carterette and Jones [30] studied the relationship between clicks coming from users and offline evaluations metrics. In particular, they were able to reliably predict nDCG from clicks.

While simple to measure, CTR has been shown to be a poor metric for measuring user satisfaction in search [95]. Consequently, a large body of work has developed online metrics that more accurately measure search satisfaction. An established signal is dwell time, where clicks followed by only short visits to the corresponding result document are considered "unsatisfied," i.e., the user is unlikely to have found the document as relevant [206]. Moving beyond a single time threshold for identifying user satisfaction, sophisticated click satisfaction classifiers combine a range of user signals, and have been shown to accurately detect satisfied clicks [110].

Other proposed online metrics also consider the effect of tabbed browsing (opening several results in browser tabs in quick succession) [88]. Conversely, the lack of a click (abandonment) is often taken as a signal of a lack of relevance, but this interpretation has posed a challenge for evaluating richer search engine result pages, where relevant information may be presented directly, without the need to click. A number of papers have proposed methods to determine when abandonment indicates satisfaction [119, 170]. Follow-on queries can also be considered indicative of a lack of success [71], as can skipping results be indicative of incorrect result order [200]. Of course, to accurately interpret user clicks, we must also consider which results users examined. If a user never looked at a search result, their lack of engagement on this result cannot be indicative of low relevance. A number of studies have shown that mouse movement can be an indicator of user examination of search results, and of specific sections within search results [48, 66, 67, 73, 201]. Similarly, in a mobile setting recording how long each part of the screen is visible can be considered an indicator of relevance [115].

An earlier survey of other implicit indicators of user satisfaction by Kelly and Teevan [102] provides further insight into implicit indicators of relevance that may be used in A/B metrics.

Finally, although the majority of online evaluation has focused on user satisfaction for individual queries, it has been argued that the correct unit of measurement is the user session, or a search task. A number of session based metrics have been proposed [70, 199].

Chapter 5 focuses on online evaluation and A/B testing in particular. In that chapter we also provide details of several common A/B metrics.

2.3.3 Interleaving

While providing flexibility and control, A/B tests typically require a large number of observations. Given typical differences in IR system performance in state of the art systems, many A/B metrics have been found to require millions of users [34, 166]. *Interleaved comparison methods*, originally proposed by Joachims [93], reduce the variance of measurement by combining documents retrieved by both the control and the treatment system. Projecting user clicks on the resulting interleaved document lists back to the original document rankings is then taken as an estimate of which system would be preferred by the user. This mixing substantially reduces variance and was found to reduce required sample sizes by up to two orders of magnitude [34, 144, 166] (see Chapter 5).

Interleaved comparison methods take as input two rankers and a query, and produce as output a combined result list to show to the user. The resulting clicks are then interpreted by the interleaving method to decide on a winning ranker. Many interleaving methods have been proposed over the years.

Balanced interleave *Balanced interleave* (BI) [95] randomly selects a ranker to start with. Then, it takes the first document from this ranker and, alternating, each ranker contributes its next document. This document is added to the interleaving only if it was not yet present. *balanced interleave* (BI) can produce biased results: in comparisons of two very similar rankers, it can favor one ranker regardless of where the user clicks.

Team draft interleave This bias was subsequently fixed in *team draft interleave* (TDI) [144]. This algorithm is most frequently used in practice, and has been empirically shown to be equally effective as BI [34, 93]. We describe *team draft interleave* (TDI) in detail below as the algorithm is used extensively throughout this thesis.

Document constraints interleave Lesser known interleaving methods include *document constraints interleave* (DCI) [72] which infers constraints on documents pairs based on the rank of the documents and on user interactions. For pairs of clicked and non-clicked documents, the method infers a requirement of the clicked document being ranked higher than the non-clicked document. DCI also infers a constraint that prefers click documents over the next non-clicked document. The input ranking that violates the least constraint is te preferred ranking.

Probabilistic interleave *Probabilistic interleave* (PI) by Hofmann et al. [79] is a more recent probabilistic generalization of TDI. This interleaving method has a non-zero

probability of any interleaving occurring as documents are sampled from a softmax distribution over the ranking instead of a deterministic ranking. Unlike TDI, team assignments are not recorded, instead, the algorithm marginalizes over all possible team assignments that could have occurred. *Probabilistic interleave* (PI) has the advantage that historical interaction data can be reused using importance sampling, for instance in an online learning to rank setting, as shown by Hofmann et al. [83].

Upper bound interleave Kharitonov et al. [106] also use historical click data, however with the goal to increase the sensitivity of interleaving, they refer to their method as *upper bound interleave* (UBI). *Upper bound interleave* (UBI) uses historical user interactions to train a click model which would predict the documents that are likely to be clicked by a user. This can be used to predict which interleavings are likely to contribute to the interleaving outcome. This provides the opportunity to select interleavings that can discriminate between two systems that are compared.

Optimized interleave *Optimized interleave* (OI) [143], like UBI, aims at increasing sensitivity. *Optimized interleave* (OI) does so by restricting the allowed interleavings to those that are the union of prefixes of the input rankings. In addition, it computes all possible probability distributions over these rankers that avoid bias. If there are multiple such distributions, then OI selects the distribution that maximizes sensitivity.

Generalized team draft interleave Kharitonov et al. [107] recently introduced *generalized team draft interleave* (GTDI) which jointly optimizes for credit assignment and an interleaving policy. This way, *generalized team draft interleave* (GTDI) achieves a higher sensitivity. Furthermore GTDI introduces a framework that allows for other layouts of results lists to be interleaved.

In this thesis, in Chapter 4, we extend TDI, OI, and PI to comparing more than two rankers at once. Previously, TDI was extended by Chuklin et al. [36] to handle non-uniform result lists that contain vertical documents such as images. TDI is also extended in Chapter 5 where we extend it to be more in agreement with A/B metrics. Below, we describe TDI in detail.

Team Draft Interleave

Given an incoming user query, TDI produces a result list as follows. The algorithm takes as input two ranked lists of documents for the query, $A = (a_1, a_2, \ldots)$ and $B = (b_1, b_2, \ldots)$. The goal is to produce a combined ranking $L = (l_1, l_2, \ldots)$. This is done in the same way that sports teams may be constructed in a friendly match, with two team captains taking turns picking players for their team.

The algorithm is detailed in Algorithm 1. It initializes the interleaved list L with any common prefix of A and B, if this exists. For this common prefix, no teams are assigned, as no preferences should be inferred.[2] Then, on line 5, the algorithm continues in phases by adding two documents to L: In each phase, on line 6, we first flip an unbiased coin to decide if ranker A or B is given priority. Assuming that ranker A is given priority, A

[2]This was shown to substantially increase sensitivity of the simpler original TDI algorithm [34, 142].

Algorithm 1 *Team draft interleave* (TDI) [144]

Require: Rankings $A = (a_1, a_2, \dots)$ and $B = (b_1, b_2, \dots)$
Init: $L \leftarrow (); TeamA \leftarrow \emptyset; TeamB \leftarrow \emptyset; i \leftarrow 1$
while $A[i] = B[i]$ **do** *// common prefix*
  $L \leftarrow L + A[i]$ *// append result to L without assigning teams*
  $i \leftarrow i + 1$ *// increment i*
while $(\exists i : A[i] \notin L) \wedge (\exists j : B[j] \notin L)$ **do** *// not at end of A or B*
  if $(|TeamA| < |TeamB|) \vee ((|TeamA| = |TeamB|) \wedge (RandBit() = 1))$ **then**
    $k \leftarrow \min_i\{i : A[i] \notin L\}$ *// top result in A not yet in L*
    $L \leftarrow L + A[k]$ *// append it to L*
    $TeamA \leftarrow TeamA \cup \{A[k]\}$ *// clicks credited to A*
  else
    $k \leftarrow \min_i\{i : B[i] \notin L\}$ *// top result in B not yet in L*
    $L \leftarrow L + B[k]$ *// append it to L*
    $TeamB \leftarrow TeamB \cup \{B[k]\}$ *// clicks credited to B*
Output: Interleaved ranking L, $TeamA$, $TeamB$

appends its highest ranked result that is not already in L to L (i.e., $l_1 \leftarrow a_1$ in the first instance), and assigns it to $TeamA$. Then, B selects its first result not already present in L (in the first instance either b_1 if it differs from a_1, and b_2 otherwise) and again appends it to L and $TeamB$. This repeats until all results in A or B have been consumed or until L reaches the desired length.

The interleaved ranking L is then shown to the user. Any clicks on documents contributed by A (in $TeamA$) are credited to A. Clicks on documents in $TeamB$ are credited to B. Over an observed sample of interleaving observations, a preference for A or B is then inferred based on which ranker was credited with more clicks. For any given set of results shown to users, the ranker where more contributed documents are clicked is considered to be preferred. In Algorithm 1, the results contributed by A are recorded in a set $TeamA$ and similarly for the results contributed by B. We refer the reader to work by Chapelle et al. [34] for a more in-depth discussion of this algorithm.

Suppose the user clicks on $C^q = c_1, c_2, \dots$ when presented with the results L in response to query q. Let $C^q_A = C^q \cap TeamA$ be the clicked documents in $TeamA$, and $C^q_B = C \cap TeamB$ be the clicked documents in $TeamB$. If $|C^q_A| > |C^q_B|$, then ranker A is considered to have won for this query, and if $|C^q_B > C^q_A|$ then ranker B is considered to have won. Otherwise, there was a tie.

The final outcome of the interleaving comparison experiment can thus be written as:

$$O_{TDI}(A, B) = \text{sgn}\left(\frac{1}{|Q|}\sum_{q \in Q} |C^q_A| - |C^q_B|\right), \tag{2.1}$$

where Q is the set of all query impressions (non-unique queries issued by all users during the interleaving experiment), and C^q_X denotes the set of clicks observed in $TeamX$ on q.

TDI is extensively used in Chapters 4 and 5. In Chapter 4 we extend interleaving so that

more rankers can be compared at once. In Chapter 5 we adapt interleaving to increase agreement with A/B testing metrics.

2.3.4 K-Armed Dueling Bandits Problem

Related to the online evaluation setting described so far is the *K-armed dueling bandits problem* by Yue et al. [209], which can be stated as follows. From a set of K rankers (bandits), the task is to find the best ranker using pairwise comparisons and to do so while incurring minimum regret. The pairwise comparisons use interleaved comparisons as introduced above in Section 2.3.3. These pairwise comparisons are used as they are easier to obtain than absolute metrics [93, 209]. Regret is proportional to the *probability* that users would have preferred the best ranking to the one shown to them. The aim is to minimize cumulative regret, the sum of regret over all query impressions. We define the best ranker as the *Condorcet winner* [194], the ranker that, when compared with every other ranker, is preferred in more interleaved comparisons.

Existing algorithms that aim at solving the *K-armed dueling bandits problem* (e.g., [208, 213, 214, 216]) all work by performing a series of pairwise interleaved comparisons, with a focus on finding the best ranker in a set of rankers. The order in which pairs of rankers are compared varies and is determined by these algorithms. One the first algorithms to address this problem is the *beat the mean* (BTM) by Yue and Joachims [208]. This algorithm keeps track of how often each ranker beats the mean of a set of rankers that is still being considered. *relative upper confidence bound* (RUCB) by Zoghi et al. [214] works by selecting randomly among those rankers that, when optimistic about them, appear to be Condorcet winners. RUCB then selects an opponent for this ranker by selecting the ranker that, when optimistic about that ranker, is most likely to beat this first ranker. MergeRUCB by Zoghi et al. [216] is a recent variant on RUCB that uses a divide an conquer strategy to make the algorithm scale to a large K. Zoghi et al. [215] introduce a version of the dueling bandits problem for when the Condorcet winner does not exist. They propose two algorithms aimed at minimizing regret with respect to the *Copeland winner* instead. This winner is guaranteed to exist.

The *dueling bandit gradient descent* (DBGD) algorithm by Yue and Joachims [207], which we describe in Section 2.5.1, can be seen as an algorithm to solve a variant of the *K-armed dueling bandits problem* that considers rankers in a continuous space of feature weights instead of a fixed set of K rankers

We address a similar but different problem in Chapter 4. In that chapter, we are not so much interested in finding the best ranker among a set of K rankers as we are interested in finding out how all rankers in the set compare to each other.

2.3.5 Counterfactual Analysis

Counterfactual analysis reuses historical user interactions, just like *probabilistic interleave* (PI) [79], described above in Section 2.3.3, does. As we described in Section 2.3.1, user interactions are biased by the system under which they were collected. In order to be able to reuse them, one needs to correct for the bias that was introduced. A common way to do so is by using a statistical technique called *importance sampling* [97]. Bottou and Peters [22] describe several importance sampling methods for counterfactual analysis. PI

also uses importance sampling to reinterpret historical interactions [79] and is therefore a particular counterfactual analysis method. Once there is a method for interpreting historical interactions, these can be used either for online evaluation methods [120, 158] or online learning methods [176, 180, 181]. Counterfactual analysis can be seen as an alternative for online evaluation. As opposed to online evaluation, users are not actually exposed to potentially inferior rankings.

2.3.6 Click Models

Yet another alternative to online evaluation is simulation. User interactions with search engines can be simulated by using click models. These interactions, in turn, can then be used either to validate evaluation methods, as we do in Part I of this thesis, or learning, as we do in Part II. In Section 3.3 we describe how we simulate clicks in this thesis. The advantage of simulating user interactions is that no actual users are exposed to potentially inferior rankings while it is still possible to experiment with for instance online learning to rank methods. And as with Cranfield-style evaluation, described in Section 2.2.1, experiments are repeatable. The downside is that every click model has its own assumptions that do not necessarily agree with reality. Additionally, click models, when they are used to simulate clicks, require relevance assessments which have their own drawbacks (see Section 2.2.1).

While we do not use them as such, an important use of click models is to *interpret clicks* in order to understand user behavior. In the setting we use them in, we assume that the parameters of the clicks models have been estimated from data or that they have been chosen such that they reflect actual user behavior. Once the click models are instantiated, they can be used to produce clicks instead of interpreting them.

Many alternative click models have been developed over the years, Chuklin et al. [40] provide an overview. We list a selection of well-known models here.

Cascade click model The *cascade click model* (CCM) [47] assumes users scan a result list from top to bottom and click on a result as soon as it seems to satisfy their information need.

Dependent click model Guo et al. [65] introduce the *dependent click model* (DCM), an extension to *cascade click model* (CCM) that allows for multiple clicks on a single results page. *Dependent click model* (DCM) is used extensively throughout this thesis as it produces simulated clicks in many of our experiments (see Section 3.3).

User browsing model The *user browsing model* (UBM) by Dupret and Piwowarski [53] assumes that the examination probability of a document also depends on previous clicks.

Dynamic bayesian network model The CCM has also been extended into the *dynamic bayesian network model* (DBN) by Chapelle and Zhang [32]. A difference is that *dynamic bayesian network model* (DBN) assumes that the probability of a user clicking is dependent on the actual relevance of a document.

Many more advanced models exist. For instance, Chuklin et al. [38] model clicks beyond the first result page and click models by Markov et al. [128] take vertical results into account.

Interestingly, Chuklin et al. [37] connect click models to information retrieval metrics as described in Section 2.2. Also Kharitonov [105] aims at improving both offline and online evaluation by modeling user behavior.

2.4 Offline Learning to Rank

Modern search engines base their rankings on combinations of dozens or even hundreds of features. These features are retrieval models as described above in Section 2.1.3. *Learning to rank* (LTR) for IR [124] is about finding an optimal combination of features using machine learning techniques. Where optimal is defined as optimal for some metric discussed in Section 2.2.

Traditionally, learning was done *offline* by optimizing for performance on a training set consisting of queries and relevance assessments produced by human assessors. Offline learning methods suffer from the drawback of requiring labeled data sets, which are expensive to produce and the models learned do not necessarily align with user satisfaction [155]. These drawbacks are the reason for us to focus on online learning to rank methods in this thesis, starting with Section 2.5. We first provide a short overview of offline learning to rank methods in this section.

The problem of LTR for IR can be described as follows. For a query q, each document is represented by a feature vector $\mathbf{x}$. Typically there would be dozens or even hundreds of such features. These features would either be only query dependent, only document dependent or they would model the relationship between queries and documents. Examples of features are BM25, LM, and PageRank (see Section 2.1.3). The task of the learner is then to find a model that combines these features such that, when this model is used to produce a ranking for an unseen query, user satisfaction is maximized. The model is a function that maps a set of documents, represented by features, to a ranking. This can be achieved in several ways. We provide details for three ways of doing is in this section.

User satisfaction is often approximated by an evaluation metric, such as described in Section 2.2. This in turn, would require relevance assessments y, acquired through the process described in Section 2.2.1. Training instances then, for a LTR method, consist of $(q, \mathbf{x}, y)$.

Offline learning to rank methods [124] can be supervised and semi-supervised [182]. Most methods are supervised and can be classified into *pointwise*, *pairwise*, and *listwise* LTR methods [27, 124]. We briefly describe each class of learning methods below.

2.4.1 Pointwise Learning to Rank

Pointwise LTR [124] methods learn a direct mapping from features $\mathbf{x}$ to label y. If the labels are binary, any machine learning classification method can be used. If the domain of the labels is the real values, regressions methods can be used. Fuhr [58], for instance, introduces one of the earliest LTR methods that minimizes square errors. This is a typical loss function used in regression based methods. Classification methods could use simple zero-one loss [136].

However, neither of these loss functions reflects the IR problem well. Absolute loss is not what matters. What matters is an optimal ranking of documents. It does, for instance,

matter much more that a good document is placed near the top of a ranking than that a label of an irrelevant document near the bottom of the ranking is incorrect.

Moreover, typically, for a query only few documents are relevant. This causes the training data sets to be highly unbalanced. This makes it a difficult problem for standard classification or regression problem that would score well by predicting that every documents is not relevant. This is difficult in the sense that, using offline evaluation metrics, such a naive baseline would perform extremely bad.

2.4.2 Pairwise Learning to Rank

Pairwise LTR [124] is about the ordering of *pairs* of documents. Training instances are pairs of feature vectors $(\mathbf{x}_{d_1}, \mathbf{x}_{d_2})$ and a binary label y to indicate whether document d_1 should be preferred over d_2 or the other way around. Pairwise LTR methods would typically take $\mathbf{x}_{d_{1,2}} = \mathbf{x}_{d_1} - \mathbf{x}_{d_2}$ as feature vector to reduce the problem to binary classification.

The advantage of pairwise over pointwise learning is that the loss function now counts ranking mismatches as opposed to absolute errors of individual documents. Also the unbalancedness of the training data is no longer a problem as there are now as many positive as negative examples (all document pairs twice). What remains is the problem of punishing incorrect document orderings in the top of the ranking as much as incorrect orderings at the bottom. Typical methods for pairwise learning to rank are Ranking SVM by Herbrich et al. [74], RankBoost by Freund et al. [57], RankSVM by Joachims [92], and RankNet by Burges et al. [24].

Additionally a new problem of pairwise LTR is that the complexity becomes quadratic in the number of documents, as all pairs of documents are considered. Sculley [168] addressed this problem by sampling from the pairs of documents. Another new problem, depending on the learned model, is that transforming pairwise preferences into a ranking may not be trivial.

An effective approach to pairwise LTR is RankSVM [92] which, when a linear kernel is used, produces a model that can easily be used to create rankings. Very related methods are developed by Fürnkranz and Hüllermeier [59].

2.4.3 Listwise Learning to Rank

Lastly, listwise LTR [124] aims at directly optimizing the whole ranking of documents. Listwise methods take feature vectors for all documents for a query as input and produce a ranking of documents. The loss function then can be an IR metric, as described in Section 2.2. The advantage listwise methods have over pointwise methods is that listwise methods capture the quality of the whole ranking instead of attempting to quantify the quality of a document in isolation. Advantages over pairwise LTR methods are that the importance of correct rankings at the top of a ranking can be taken into account. And that the problem does not become quadratic in the number of documents.

A disadvantage is that the objective function, an IR metric, may be very hard to learn as it may not be smooth or differentiable. Nevertheless, listwise LTR approaches are currently the state-of-the-art. Examples of successful methods are LambdaRank by Burges et al. [24] and LambdaMART by Burges et al. [25].

The above offline LTR methods are all supervised machine learning methods. And like any supervised learning method, these LTR methods require labeled data. This data is typically acquired through a Cranfield-style evaluation setup, described in Section 2.2.1. Offline LTR therefore suffers from the same drawbacks as Cranfield-style evaluation. Labels are expensive to obtain and do not necessarily reflect user preferences [75, 100, 155, 192, 206]. Learning models based on such labels thus results in an IR system that does not necessarily satisfy users maximally. Other offline LTR methods such as semi-supervised learning and active learning methods have been explored [49, 182, 205]. However, in contrast, in most of this thesis we focus on online learning methods as described in the next section.

2.5 Online Learning to Rank

Our focus in this thesis is on online learning to rank methods that learn from online evaluation methods, based on users' interactions with IR systems [82, 207], without requiring annotated data sets. One can formulate the online learning to rank for IR problem as a *reinforcement learning* (RL) problem [178]. The problem can be modeled as a *contextual bandit problem* that assumes that consecutive queries are independent of each other [120, 177]. In a contextual bandit problem, the algorithm has access to a *context vector*, which in the LTR setting would be the feature vector, as used in offline learning to rank.

A crucial difference between typical RL problems and the application to IR is that in the IR scenario the reward cannot be observed directly. Instead, user interactions with an IR system can be interpreted as a biased and noisy preference for either rankings [144] or documents [92].

Many methods for online learning to rank have been proposed. Hofmann et al. [82] explore learning from pairwise document preferences while most methods are based on the listwise learning paradigm where preferences between rankers are inferred from clicks. Such pairwise preferences can come from interleaving methods as discussed in Sections 2.3 and in particular 2.3.3. One influential learning to rank method is *dueling bandit gradient descent* (DBGD) [207], which is extended upon in Chapter 7 and therefore explained in more detail in Section 2.5.1. DBGD implements a stochastic gradient descent method to find the best ranker in an infinite space of rankers. This algorithm has been extended before by Hofmann et al. [83] such that it would reuse historical interaction data and not just live user interactions. Alternative methods, discussed in Secion 2.3.4, are based on the K-armed dueling bandit formulation by [209] and assume a finite space of possible rankers, the best of which needs to be found.

2.5.1 Dueling Bandit Gradient Descent

DBGD [207] is an online learning to rank method that learns from user feedback in the form of clicks. In particular, DBGD learns from a *relative* interpretation of this feedback produced by, e.g., TDI (see Section 2.3.3). The DBGD algorithm can be seen as an algorithm to solve a continuous variant of the *K-armed dueling bandits problem* which we introduced in Section 2.3.4.

DBGD, shown in Algorithm 2, assumes that rankers can be represented by weight

Algorithm 2 Dueling Bandit Gradient Descent (DBGD).

Require: α, δ, $\mathbf{w}_0^0$

1: **for** $t \leftarrow 1..\infty$ **do**
2: $q_t \leftarrow receive\_query(t)$ *// obtain a query from a user*
3: $\mathbf{l}_0 \leftarrow generate\_list(\mathbf{w}_t^0, q_t)$ *// ranking of current best*
4: $\mathbf{u}_t^1 \leftarrow sample\_unit\_vector()$
5: $\mathbf{w}_t^1 \leftarrow \mathbf{w}_t^0 + \delta\mathbf{u}_t^1$ *// create a candidate ranker*
6: $\mathbf{l}_1 \leftarrow generate\_list(\mathbf{w_t^1}, q_t)$ *// exploratory ranking*
7: $\mathbf{m}_t, \mathbf{t}_t \leftarrow TDI\_interleave(\mathbf{l})$ *// interleaving and teams*
8: $\mathbf{c}_t \leftarrow receive\_clicks(\mathbf{m}_t)$ *// show interleaving to the user*
9: $\mathbf{b}_t \leftarrow TDI\_infer(\mathbf{t}_t, \mathbf{c}_t)$ *// set of winning candidates*
10: **if** $\mathbf{w}_t^0 \in \mathbf{b}_t$ **then**
11: $\mathbf{w}_{t+1}^0 \leftarrow \mathbf{w}_t^0$ *// if current best wins or ties, no update*
12: **else**
13: $\mathbf{w}_{t+1}^0 \leftarrow \mathbf{w}_t^0 + \alpha\mathbf{u}_t^1$ *// update α step towards candidate*

vectors, starting with a randomly initialised weight vector $\mathbf{w}_0^0$, referred to as the *current best ranker*. For each query that is issued, on line 5, an exploratory *candidate ranker* $\mathbf{w}_t^1$ is created by slightly perturbing the weight vector of the current best ranker. Both the current best ranker and the candidate ranker create their rankings of documents for the issued query. These two rankings are interleaved using, e.g., TDI, on line 7. Then on line 8 this interleaving is shown to the user that issued the query and clicks are observed. The interactions of this user with the interleaving are interpreted by the interleaving method on line 9 to determine who won the comparison. If the candidate won, the weight vector of the current best ranker is updated with an α step towards the weight vector of the candidate ranker. If not, the weight vector is not updated. This process repeats indefinitely, yielding a continuously adapting system.

DBGD is used extensively in Part II of this thesis. Chapter 6 uses the algorithm to learn parameters of base rankers. Chapter 7 extends DBGD to using multileaved comparison methods—introduced in Chapter 4—instead of interleaved comparisons methods.

2.5.2 Reusing Historical Interaction Data

DBGD originally used *team draft interleave* (TDI) [144] (see Section 2.3.3) as a method of obtaining preferences between rankers. The introduction of *probabilistic interleave* (PI) by Hofmann et al. [79] paved the way also to new learning methods. PI allows for interpreting historical user interactions with an interleaved results for a *different* pair of rankers than was originally used. In other words, if a new pair of rankers is to be compared, a historical click can be used to obtain an unbiased preference. Hofmann et al. [83] used this idea in their *candidate preselection* (CPS), which extends DBGD, as follows. Where DBGD would, on line 5 in Algorithm 2, create a candidate ranker by stepping into a random direction, CPS would create many such candidates. Historical interactions in combination with PI would then be used to determine which of these candidates is most promising. The most promising candidate is then used and the algorithm functions the same as DBGD, except that additionally historical interactions need to be stored.

Hofmann et al. [83] show that by reusing historical interaction data in this way, the learning method can learn much faster than the baseline DBGD method, in particular when the user feedback is noisy.

3 Experimental Methodology

In this chapter we detail the experimental setup used in Chapters 4, 6 and 7. This setup has been used to validate both evaluation methods, as in Chapter 4, as well as learning methods, as in Chapters 6 and 7. The exact same setup or a variation on it has been used in numerous publications [36, 39, 41, 83, 84, 138, 162, 163, 165, 167, 216]. It has been implemented in a framework explained by Schuth et al. [161] and detailed in Chapter 8.[1]

We first describe the data sets that we use in Section 3.2. Then, in Section 3.3, we detail our click simulation framework. We then turn to evaluation metrics and statistical significance testing. Lastly we describe our evaluation metrics in Section 3.4.

3.1 Experimental Setup

We employ a click simulation framework analogous to that of [83]. We do so because we do not have access to a live search engine or a suitable click log. Note that, even if a click log was available, it would not be adequate since a learning algorithm or an algorithm that is to be evaluated is likely to produce result lists that do not appear in the log. But there are evaluation methods designed to address exactly this problem, e.g., the work by Li et al. [120], described in Section 2.3.5.

All our experiments that use this simulation setup assume a stream of independent queries coming from users interacting with the system we are training or evaluating. Users are presented with a result list in response to their query and may or may not interact with the list by clicking on one or more documents. The queries come from static data sets (Section 3.2) and the clicks from a click model (Section 3.3).

We measure performance in two ways: online, the way a user experienced it, and offline, measured on a held-out data set (Section 3.4).

3.2 Data Sets

Many of the experiments in this thesis are conducted on nine learning to rank data sets that are distributed as LETOR 3.0 and 4.0 [125, 141]. Per chapter, when applicable, we list which data sets are used exactly.

[1] All our experimental code is open source and available at `https://bitbucket.org/ilps/lerot` [161] (see also Chapter 8).

Table 3.1: Characteristics of the nine learning to rank data sets that are distributed as LETOR 3.0 and 4.0 [125, 141] and used throughout this thesis.

	Task	Data	#queries	#relevance	#docs	#features
NP2003	Named page finding	.GOV	150	2	1000	64
NP2004	Named page finding	.GOV	75	2	1000	64
HP2003	Homepage finding	.GOV	150	2	1000	64
HP2004	Homepage finding	.GOV	75	2	1000	64
TD2003	Topic distillation	.GOV	50	2	1000	64
TD2004	Topic distillation	.GOV	75	2	1000	64
MQ2007	Web search	.GOV2	1700	3	16-128	46
MQ2008	Web search	.GOV2	800	3	16-128	46
OHSUMED	Literature search	MedLine	106	3	150	45

Each learning to rank data set contains feature vectors representing the relationships between queries and documents. These feature vectors contain between 45 and 64 features. Examples of features are *best match 25* (BM25), *language model* (LM), and PageRank (see Section 2.1.3). Each of these features can be treated independently as rankers, by simply sorting on the feature value. While we use learning to rank data sets, we do not necessarily perform learning. These data sets can also be used to validate ranker evaluation, as in Chapter 4. If the data sets are used in an evaluation setting, individual features or combinations of features are typically taken as rankers.

The (manually assessed) relevance level of each document-query pair is also provided in the data set. Finally, all data sets are pre-split by query for 5-fold cross validation.

In the nine data sets, the following search tasks are implemented. The *OHSUMED* data set models a literature search task which is based on a query log of a search engine for the MedLine abstract database. This data set contains 106 queries that implement an informational search task. The remaining eight data sets are based on *text retrieval conference* (TREC) Web track tasks that run between 2003 and 2008.

The data sets *HP2003*, *HP2004*, *NP2003*, and *NP2004* implement navigational tasks, homepage finding and named-page finding respectively. *TD2003* and *TD2004* implement an informational task: topic distillation.

These last six data sets are based on the .GOV document collection, a crawl of the .gov domain, and contain between 50 and 150 queries and approximately 1000 judged documents per query. The more recent .GOV2 collection formed the basis of *MQ2007* and *MQ2008*; two data sets that contain 1700 and 800 queries respectively, but far fewer judged documents per query.

The data sets *OHSUMED*, *MQ2007* and *MQ2008* are annotated with graded relevance judgments (3 grades, from 0, not relevant, to 2, highly relevant). The other data sets have binary relevance labels (grade 0 for not relevant, 2 for relevant).

We summarize the characteristics of our data sets in Table 3.1.

3.3 Simulating Users

To produce clicks, we use a click simulation framework that is analogous to [83], which is explained in [161]. The framework produces clicks based on the *dependent click*

Table 3.2: Instantiations of the DCM [65] as used in this thesis.

		$P(click = 1\|R)$			$P(stop = 1\|R)$		
		0	1	2	0	1	2
perfect	(per)	0.0	0.5	1.0	0.0	0.0	0.0
navigational	(nav)	0.05	0.5	0.95	0.2	0.5	0.9
informational	(inf)	0.4	0.7	0.9	0.1	0.3	0.5
almost random	(a.ra)	0.4	0.5	0.6	0.5	0.5	0.5
random	(ran)	0.5	0.5	0.5	0.0	0.0	0.0

model (DCM) [65], also used by [77], that effectively explains the click behavior of web search users. The cascade click model explains position bias by assuming that users start examining a result from the top of the list. Then, when the user scans down the list, for each document they determine whether it looks promising enough to deserve a click. This is modeled with a click probability given some relevance label $P(click = 1|R)$. These relevance labels R come from human assessors, as explained in Section 2.2.1. After a click, a user decides whether their information need has been satisfied by the document just clicked. We model this with a stop probability $P(stop = 1|R)$. Table 3.2 lists the instantiations of the click model used in our experiments. Depending on our research questions, experiments use different instantiations.

The instantiations listed in Table 3.2 can be interpreted as follows.

perfect The *perfect* instantiation of the click model, in which exactly every relevant document is clicked, provides unrealistically reliable feedback, and is used to obtain an upper bound on performance. Users click on all the highly relevant documents, never on the irrelevant documents and half the time on the mildly relevant documents.

navigational The *navigational* click model, in which users almost only click relevant documents and usually stop when they have found a relevant document, reflects a user with a *navigational information need* [23]. Users with a navigational information need seek to find a particular url of a website they have in mind.

informational The *informational* click model, in which non-relevant documents are also clicked quite often and users stop after finding a relevant document only half of the time, represents a user with an *informational information need* [23]. Users with an informational information need seek to acquire some information about a particular topic. Users with such a need often inspect multiple documents.

almost random The *almost random* click model, in which there is only a very small difference in user behavior for relevant and non-relevant documents, is used to establish whether an evaluation method can identify preferences or alternatively whether a learning method can still learn, when the signal in clicks is extremely weak.

random The *random* instantiation of the click model is used to examine behavior of the evaluation methods when no information is present in the clicks.

For each instantiation, the table provides the click and stop probabilities given a relevance grade R. For example, under the navigational model, simulated users would be very likely to click on a highly relevant document ($P(click = 1|2) = 0.95$), and very likely to stop examining documents once they clicked on such a document ($P(stop = 1|2) = 0.9$). Under the informational model, users are less likely to stop, and click probabilities for the different relevance grades are much more similar, resulting in a higher level of noise. For data sets with binary relevance judgments, only the two extremes (relevance labels 0 and 2) are used.

See Section 2.3.6 for an overview of the many alternative click models that have been developed over the years. Despite the existence of all these models we use the relatively simple DCM, following Hofmann [77], as it makes very few assumptions.

3.4 Evaluation

We measure performance of rankers in two ways: *online*, the way a user experiences it, and *offline*, measured on a held-out data set.

First, to assess offline performance, we use the offline evaluation metric (see Section 2.2) *normalized discounted cumulative gain* (nDCG) [89] which is computed on held-out data. All data sets described in Section 3.2 are pre-split in five folds. Four folds are used for training purposes. The fifth fold is used for computing the metric.

We use the top $\kappa = 10$ documents for simulating clicks and computing the nDCG:

$$nDCG = \sum_{i=1}^{\kappa} \frac{2^{rel(\mathbf{r}[i])-1}}{\log_2(i+1)} InDCG^{-1}. \tag{3.1}$$

This metric calculates the gain over relevance labels $rel(\mathbf{r}[i])$ for each document, which is then normalized by the maximal nDCG possible, the ideal nDCG (InDCG). *Offline* performance is determined by computing the average nDCG score of the *current best ranker* over a held-out set.

During an experiment the user experience may be inferior compared to the production system that the user would otherwise see. Therefore, *online* performance is also assessed. We do this by computing the discounted cumulative nDCG over the results shown to the user. This can be seen as the inverse of regret as used in a *reinforcement learning* (RL) setting. For online performance, a discount factor of $\gamma = 0.995$ is used [77, 178]. We compute our discounted cumulative nDCG as

$$\sum_{n=1}^{N} \gamma^n \cdot nDCG_n, \tag{3.2}$$

where $nDCG_n$ denotes the nDCG of the nth query. This factor ensures that impressions beyond a horizon of $N = 1000$ query impressions have an impact of less than 1%. We repeat each experiment 25 times and average results over the 5 folds and these repetitions.

To verify whether differences are statistically significantly different, a two tailed Student's t-test is used [169].

Part I

Online Evaluation

4 Multileaved Comparisons

Evaluation methods for information retrieval systems come in three types: *offline evaluation*, using static data sets annotated for relevance by human judges; *user studies*, usually conducted in a lab-based setting; and *online evaluation*, using implicit signals such as clicks from actual users (see Sections 2.2 and 2.3). For the latter, preferences between rankers are typically inferred from implicit signals via *interleaved comparison* methods, which combine a pair of rankings and display the result to the user (see Section 2.3.3). We propose a new approach to online evaluation called *multileaved comparisons* that is useful in the prevalent case where designers are interested in the relative performance of more than two rankers. Rather than combining only a pair of rankings, multileaved comparisons combine an arbitrary number of rankings. The resulting user clicks then give feedback about how all these rankings compare to each other. We propose three specific multileaved comparison methods. The first, called *team draft multileave* (TDM), is an extension of *team draft interleave* (TDI). The second, called *optimized multileave* (OM), is an extension of *optimized interleave* (OI) and is designed to handle cases where a large number of rankers must be multileaved. We present experimental results that demonstrate that both team draft multileave and optimized multileave can accurately determine all pairwise preferences among a set of rankers using far less data than the interleaving methods that they extend. Lastly, we introduce *probabilistic multileave* (PM), an extension of *probabilistic interleave* (PI). PM can reliably infer preferences among multiple rankers, also using historical interaction data.

This chapter is based on two publications: Schuth, Sietsma, Whiteson, Lefortier, and de Rijke [163], Schuth, Bruintjes, Büttner, van Doorn, Groenland, Oosterhuis, Tran, Veeling, van der Velde, Wechsler, Woudenberg, and de Rijke [165].

4.1 Introduction

Deployed search engines often have several teams of engineers tasked with developing potential improvements to the current production ranker [112]. To determine whether the candidate rankers they develop are indeed improvements over the production ranker, such teams need experimental feedback about their performance relative to the production ranker. However, in order to develop and refine those candidate rankers in the first place, they also need more detailed feedback about how the candidate rankers compare *to each other*. For example, to explore a parameter space of interest, they may be interested in the

relative performance of multiple rankers in that space.

Several existing approaches could be used to generate this feedback. Firstly, assessors could produce relevance assessments from which offline metrics (e.g., MAP, nDCG, ERR [155]) could be computed. However, offline metrics do not tell the whole story since relevance assessments come from assessors, not users (see Section 2.3). Secondly, online experiments generate user feedback such as clicks from which rankers can be evaluated. In particular, *interleaved comparison* [92, 93] methods enable such evaluations with greater data efficiency than A/B testing [144]. But teams of engineers can easily produce enough candidate rankers that comparing all of them to each other using interleaving methods quickly becomes infeasible.

To address this difficulty, we propose a new evaluation paradigm, which we call *multileaved comparison*, that makes it possible to compare more than two rankers at once. Multileaved comparisons can provide detailed feedback about how multiple candidate rankers compare to each other using much less interaction data than would be required using interleaved comparisons.

In particular, we start by proposing two specific implementations of multileaved comparisons. The first, which we call *team draft multileave* (TDM), builds on *team draft interleave* (TDI) [144], an interleaving method that assigns documents in the interleaved list to a team per ranker. TDI is explained in detail in Section 2.3.3. Surprisingly, only a minor extension to TDI is necessary to enable it to perform multileaved comparisons, yielding TDM. However, despite its appealing simplicity, TDM has the important drawback that it requires multileavings, i.e., the result lists shown to the user, to be long enough to represent teams for each ranker.

Therefore, we propose a second method that we call *optimized multileave* (OM), which builds on *optimized interleave* (OI) [143], an interleaved comparison method that uses a *prefix constraint* to restrict the allowed interleavings to those that are "in between" the two rankers and then solves an optimization problem to ensure unbiasedness and maximize sensitivity of the interleavings shown to users. OM requires deriving a new prefix constraint, new definitions of unbiasedness and sensitivity, a new credit function upon which these definitions depend, and a new sampling scheme to make optimization tractable. Because it avoids the limitations of TDM, OM is better suited to handle larger numbers of rankers.

We present experimental results on several data sets that aim to answer the following research questions, repeated from Section 1.1.

RQ1 Can multileaved comparison methods identify preferences between rankers faster than interleaved comparison methods?

RQ2 Does OM scale better with the number of rankers than TDM?

RQ3 How does the sensitivity of multileaving methods compare to that of interleaving methods?

RQ4 Do multileaving methods improve over interleaving methods in terms of unbiasedness and online performance?

As an aside, we also investigate how multileaved comparison methods perform on decision problems other than finding all preferences between pairs of rankers in a set of rankers.

The interleaving and multileaving methods listed so far can only infer preferences among rankers based on interaction data that is generated using those very same rankers, thus preventing generalizations to new and unseen rankers. *Probabilistic interleave* (PI), however, is a recent interleaving method that can reuse *historical interaction data* collected using other rankers than the ranker being evaluated to infer preferences between rankers [79, 83]. This allows for comparisons that are more efficient in terms of how many user impressions are required for reliable comparisons.

In this chapter, we also propose an extension of PI to *probabilistic multileave* (PM): a multileaving method that is able to reuse historical interaction data.

As stated before, an evaluation method is *sensitive* if it quickly detects differences in the quality of rankings. That is, if rankers are of different quality, the evaluation method should detect those differences with fewest comparisons. An evaluation method is *unbiased* if all the rankers are equal in expectation when clicks are random. That is, the method should evaluate rankers fairly and only on the basis of their actual performance.

We answer the following research question, repeated from Section 1.1.

RQ5 How does PM compare to TDM and OM in terms of sensitivity, bias and scaling?

The main contributions of this chapter are as follows.

Paradigm We introduce a novel ranker evaluation paradigm in which more than two rankers can be compared at once.

Algorithms Three implementations of this new paradigm, TDM, OM, and PM are introduced.

Evaluation A thorough experimental comparison of TDM and OM against each other and against TDI and OI that shows that multileaved comparison methods can find preferences between rankers much faster than interleaved comparison methods.

Sensitivity When the differences between evaluated rankers are varied, the sensitivity of TDM and OM is affected in the same way as for TDI and OI. We show experimentally that PM is at least as sensitive as TDM, which in turn is more sensitive than OM.

Scaling Our experiments also show that TDM outperforms OM unless the number of rankers becomes too large to handle for TDM, at which point OM performs better. Our experiments show that, like OM but unlike TDM, PM scales well when the number rankers that are compared increases.

Bias We also show experimentally that PM is unbiased.

We have incorporated the above contributions in Section 1.2 where we give a complete overview of all contributions of this thesis.

An important implication of our results is that multileaving methods TDM, OM and PM can accurately determine a set of pairwise preferences among a set of rankers using

much less data than interleaving methods need. Furthermore, when using PM, historical interactions with multileaved comparisons can be reused, allowing for ranker comparisons that need much less user interaction data. This, in turn, means that users are exposed less often to inferior rankers and that more rankers can be compared with the same number of user interactions.

4.2 Related Work

We refer to Section 2.3 and in particular to Section 2.3.3 for background on this chapter.

Our work in this chapter differs from earlier work in that it does not rely on pairwise comparisons. As a result, when a set of rankers is evaluated, it is no longer necessary to separately compare each ranker pair. We obviate that need by introducing a new paradigm called *multileaved comparisons* that can evaluate a complete set of rankers in one comparison and thereby requires substantially less data.

4.3 Problem Definition

The problem we want to tackle can be formulated as follows: we have a set of rankers $\mathcal{R}$ (with $|\mathcal{R}| \geq 2$) whose performance we want to evaluate using click feedback. We may be interested in knowing how all rankers in $\mathcal{R}$ compare to each other, as doing so gives valuable feedback to the engineers who design new rankers. If we already have a working production ranker, we may also be interested in determining how each ranker in $\mathcal{R}$ compares to it. Alternatively, we may be interested in finding the best ranker in $\mathcal{R}$, which is an instance of the K-armed dueling bandit problem.

In this chapter, we focus on developing multileaved comparisons methods for the first task, comparing all rankers to each other, because it represents a scenario that is vital for enabling ranker development in deployed search engines. For completeness, in Section 4.6.1, we also evaluate our methods, designed to compare all rankers to each other, on the task variation in which they are asked to compare a set of rankers to a single production ranker. In Section 4.7, we discuss how our methods can be customized to this task variation or to the K-armed dueling bandit problem (see also Section 2.3.4).

To formalize the task of determining how all rankers in $\mathcal{R}$ compare to each other, we begin by defining ground truth as a *preference matrix* P, an $|\mathcal{R}| \times |\mathcal{R}|$ matrix in which each cell P_{ij} contains the difference in expected *normalized discounted cumulative gain* (nDCG) [89] between rankers R_i and R_j, normalized to lie between 0 and 1:

$$P_{ij} = 0.5(nDCG(R_i) - nDCG(R_j)) + 0.5, \tag{4.1}$$

where $nDCG(R_i)$ is the expected nDCG of ranker R_i across queries. The goal of an online evaluation method is then to use click feedback to learn a matrix $\hat{P}$ that approximates P. Its performance is thus measured using the error of $\hat{P}$ with respect to P. We propose a binary error metric that counts the number of times $\hat{P}$ is incorrect about which ranker has a higher expected nDCG:

$$E_{bin} = \frac{\sum_{i,j \in \mathcal{R} \wedge i \neq j} \operatorname{sgn}(\hat{P}_{i,j} - 0.5) \neq \operatorname{sgn}(P_{i,j} - 0.5)}{|\mathcal{R}| \cdot (|\mathcal{R}| - 1)}, \tag{4.2}$$

where $\mathrm{sgn}(\cdot)$ returns -1 for negative values, 1 for positive values and 0 otherwise, and the infix operator $\neq$ returns 1 whenever the signs are not equal.

4.4 Methods

Using interleaving methods, learning $\hat{P}$ requires interleaving each ranker pair (R_i, R_j) separately to estimate each P_{ij}, which means that many interleavings are required for learning. The goal of multileaved comparison methods is to reduce the cost of learning by constructing *multileavings* that, by combining documents from all rankers $\mathcal{R}$, can learn about all cells in P at once.

We propose three variants of multileaved comparison: *team draft multileave* (TDM), explained in Section 4.4.1, *optimized multileave* (OM), explained in Section 4.4.2, and *probabilistic multileave* (PM) in Section 4.4.3. OM is designed to avoid a limitation of TDM concerning the number of rankers that it can compare using a single query. PM has the same advantage over TDM and additional allows for reusing historical interaction data.

4.4.1 Team Draft Multileave

The first variant of multileaved comparisons is based on *team draft interleave* (TDI) [144], as introduced in Section 2.3.3. This interleaving method follows the analogy of selecting players (documents) for a team (ranking) for a friendly sports match. The construction of an interleaved list takes several rounds, until the interleaving is long enough. In each round, rankers select their most preferred document that is still available. It is added to their team and appended to the interleaving. The order in which rankers get to pick a document in a round is randomized. After a user interacts with documents in the interleaving, the team that owns a clicked document gets credit and the team with the most credit wins the comparison.

We propose *team draft multileave* (TDM), an extension that can compare more than two rankers at a time. Doing so is straightforward, as it only requires changing the number of teams that participate. TDM is described in Algorithm 3, which returns not only the multileaving, but also the teams to which the documents in the multileaving belong.

These team assignments are used after a user interacts with the interleaving to update the matrix $\hat{P}_{ij}$. We maintain an empirical mean for all $\hat{P}_{ij}$. We increase the preference $\hat{P}_{ij}$ if and only if there were more clicks on documents belonging to the team of ranker i than on documents belonging to the team of ranker j. Note that one reason why this may happen is that ranker j was not represented in the multileaving.

4.4.2 Optimized Multileave

While TDM is a natural way of dealing with more than two rankers, it requires multileavings to be long enough to represent teams for each ranker. Therefore, we propose *optimized multileave* (OM), based on *optimized interleave* (OI) [143], which does not have this drawback and thus may scale better with the number of rankers.

Algorithm 3 *Team draft multileave* (TDM).

Require: set of rankings $\mathcal{R}$, multileaving length k.

$L \leftarrow [\,]$ *// initialize new multileaving*
$\forall R_x \in \mathcal{R} : T_x \leftarrow \emptyset$ *// initialize teams for each ranking*
while $|L| < k$ **do**
  select R_x randomly s.t. $|T_x|$ is minimized
  $p \leftarrow 0$
  while $R_x[p] \in L$ **and** $p < k - 1$ **do**
    $p \leftarrow p + 1$
  if $R_x[p] \notin L$ **then**
    $L \leftarrow L + [R_x[p]]$ *// append document to multileaving*
    $T_x \leftarrow T_x \cup \{R_x[p]\}$ *// add document to team*
return L, T

We start in the following sub-section by constructing combinations of documents from the different rankings that satisfy a generalization of the prefix constraint of [143]; this results in a set of allowed multileavings. Then we assign a probability to each of these multileavings that determines how often it is shown to users. This probability distribution over multileavings is computed by solving for the simplex and unbiasedness constraints in the next sub-section. Subsequently, the probability distribution over multileavings that maximizes sensitivity is selected in the second to last sub-section of this method. When a multileaving is shown to a user, credit is assigned according to credit functions, described in the last sub-section, to each of the original rankings based on which documents the user clicks. We explain each step in more detail in the following sections.

Allowed Multileavings

The prefix constraint proposed by Radlinski and Craswell [143] states that any prefix (i.e., the top) of the constructed interleaving should be the union of prefixes of the two original rankings. We extend this to the case with more than two original rankings by defining the set of allowed multileavings $\mathcal{L}$ as follows:

$$\mathcal{L} = \{L_i : \forall k, \forall R_x \in \mathcal{R}, \exists m_x \text{ such that } L_i^k = \bigcup_{R_x \in \mathcal{R}} R_x^{m_x}\}. \tag{4.3}$$

Here, $\mathcal{R}$ is the set of original input rankings R_x that we want to compare, L_i^k is the top k documents of multileaving L_i, and $R_x^{m_x}$ is the top m_x documents in ranking R_x. Note that when there are only two rankings (A and B in the definition in [143]) in $\mathcal{R}$, then (4.3) coincides with the prefix constraint in [143]. In other words, (4.3) states that only multileavings are allowed for which any possible prefix is the union of prefixes of the input rankings.

Our constraint in (4.3) allows for at most $|\mathcal{R}|^{|L_i|}$ multileavings. Even with a relatively small $|\mathcal{R}|$ and $|L_i|$, this is more than can be handled by the optimization step described in the following sections. Therefore, we consider a sampling approach. Instead of materializing all multileavings allowed by (4.3), we construct only a small number of them using Algorithm 4. The result of this algorithm is a set $\mathcal{L}$ of multileavings that

Algorithm 4 Prefix constraint sampling for OM.

Require: set of rankings $\mathcal{R}$, multileaving length k, sample size η.

$\mathcal{L} \leftarrow \emptyset$ *// initialize empty set of multileavings*
while $|\mathcal{L}| < \eta$ **do**
  $L_i \leftarrow [\,]$ *// initialize new multileaving*
  while $|L_i| < k$ **do**
    select R_x randomly from $\mathcal{R}$
    $p \leftarrow 0$
    while $R_x[p] \in L_i$ **and** $p < k - 1$ **do**
      $p \leftarrow p + 1$
    if $R_x[p] \notin L_i$ **then**
      $L_i \leftarrow L_i + [R_x[p]]$ *// append document to multileaving*
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{L_i\}$ *// add constructed multileaving to set*
return $\mathcal{L}$

obey the prefix constraint (4.3) because documents from a ranking can be added to the multileaving only if all documents above it in the ranking have already been added.

The size of the set $\mathcal{L}$ of multileavings can be controlled by the parameter η. Keeping η small reduces the size of the resulting optimization problem but could introduce variance, since only a subset of allowed multileavings are considered. Besides that, due to the small number of multileavings considered, it may be the case that the optimization problem becomes overconstrained. As a result, it may no longer be possible to satisfy the unbiasedness constraint, leading to a second source of bias. We hypothesize, however, that this will not lead to severe degradation of the algorithm's performance, since ranker evaluation methods can perform well in practice even when they are biased [85].

Simplex Constraints

Every allowed multileaving $L_i \in \mathcal{L}$ is shown to the user with probability p_i. These probabilities have to satisfy a number of constraints. First of all, as in [143], they must satisfy the *simplex constraint* to form a valid probability distribution:

$$p_i \in [0, 1] \tag{4.4}$$

$$\sum_{i=1}^{|\mathcal{L}|} p_i = 1. \tag{4.5}$$

The constraints expressed in Equations 4.4 and 4.5 together ensure that the probability distribution p over multileavings $\mathcal{L}$ is in fact a probability distribution.

Unbiasedness Constraint

Furthermore, the multileavings satisfy the *unbiasedness constraint*: they should be shown to the user in such a way that none of the original rankings gets an unfair advantage. We instantiate this constraint by insisting that if the multileavings are presented to a randomly

clicking user (according to the probability distribution p over multileavings $\mathcal{L}$), all original rankings receive the same expected credit.

In [143], a randomly clicking user is assumed to pick a number k, and clicks every result in the top k of the presented list with the same probability. When a user clicks in this way, none of the original rankings should be preferred and they should all receive the same expected credit. We adapted the resulting constraint for the multileave case. Here, given a multileaving L_i, let d_{ij} denote its j-th document and let $\delta(d_{ij}, R_x)$ be the credit assigned to ranker R_x when d_{ij} is clicked. The following unbiasedness constraint directly extends [143] and expresses that, for every k, there should be some constant c_k such that when the user clicks every document in the top k, every original ranking receives the same expected credit c_k:

$$\forall k, \exists c_k \text{ such that } \forall x, \sum_{i=1}^{|\mathcal{L}|} \left(p_i \sum_{j=1}^{k} \delta(d_{ij}, R_x) \right) = c_k. \tag{4.6}$$

Optimizing for Sensitivity

Given the above constraints, multiple probability distributions over multileavings may still be possible, because the optimization problem may be underconstrained. Whether it is underconstrainted or overconstrained, however, depends on the number of sampled multileavings. As described in Section 4.4.2, if the number of samples is small, there might not even be a single solution to the optimization problem.

If the optimization problem is indeed underconstrained, there is the opportunity to prefer one probability distribution over multileavings over another. Following [143], we want to optimize the probabilities for maximal sensitivity. Intuitively, this means that probability distributions that distribute more mass to multileavings that can distinguish between rankers are preferred. We follow the alternative suggestion by [143], in that we minimize variance, as opposed to maximizing entropy.

The expected credit assigned to ranking R_x after the user clicks on documents in multileaving L_i is:

$$\mathbb{E}[\delta(R_x)] = \sum_{j=1}^{|L_i|} f(j) \cdot \delta(d_{ij}, R_x).$$

Here $f(j)$ is the probability with which a user clicks a document at position j. For simplicity and following [143], we assume that $f(j) = 1/j$. Given a multileaving L_i, we define the expectation over the variance in credit assigned to the different rankings as:

$$\mathbb{E}[Var_i] = \sum_{x=1}^{|\mathcal{R}|} \left(\left(\sum_{j=1}^{|L_i|} f(j) \cdot \delta(d_{ij}, R_x) \right) - \mu_i \right)^2, \tag{4.7}$$

$$\mu_i = \frac{1}{|\mathcal{R}|} \sum_{x=1}^{|\mathcal{R}|} \sum_{j=1}^{|L_i|} f(j) \cdot \delta(d_{ij}, R_x). \tag{4.8}$$

Then the aim of the optimization is to find the p_i's such that the sum of all variances is minimized:

$$\sum_{i=1}^{|\mathcal{L}|} p_i \cdot \mathbb{E}[Var_i]. \tag{4.9}$$

Note that we minimize the sum of all variances while taking all other constraints from Section 4.4.2 into account. In particular, if we did not ensure unbiasedness, we would find $p_i = 1$ for multileaving L_i with the lowest $\mathbb{E}[Var_i]$.

Assigning Credit

We have not yet defined the credit function δ. This function is used in a number of places in the multileaved comparison method: (1) ensuring unbiasedness, (2) optimizing for sensitivity, and (3) determining the outcome. The credit function should assign credit to an input ranking, given a clicked document in a multileaving. However, in the optimization step, there is no observed click yet. Therefore, we assume all documents are clicked.

Following Radlinski and Craswell [143], we define two possible credit functions. Intuitively, both assign more credit to rankings that rank clicked documents at a higher position. The first is *inverse rank* and analogous to the function with the same name in [143]:

$$\delta(d_{ij}, R_x) = \frac{1}{\text{rank}(d_{ij}, R_x)}. \tag{4.10}$$

Here, $\text{rank}(d_{ij}, R_x)$ is the rank of document d_{ij} in R_x if it is present in the ranking, and otherwise $|R_x| + 1$. Note that this is the rank in the full ranking R_x and not just the top k.

An alternative credit function is *negative rank*:

$$\delta(d_{ij}, R_x) = -\text{rank}(d_{ij}, R_x). \tag{4.11}$$

This credit function is analogous to the *linear rank difference* credit function from [143].[1] The difference between the credit functions in [143] and the ones defined here is that we cannot define our credit functions on a pair of rankings. Instead, our credit functions are defined as giving certain credit to a single ranking.

Optimized Multileaved Comparisons

Above, we described the ingredients of OM. Here and in Algorithm 5 we put them all together. When a multileaved comparison is performed, the following happens. Each of the rankers that are to be compared generates a ranking, given the user's query. A set of multileavings is generated from these rankings using Algorithm 4. Then, a probability distribution over these multileavings is computed that obeys the unbiasedness constraints in Section 4.4.2. Following [143] we use a linear constraint optimization solver to find a distribution that satisfies these constraints.[2] If there is more than one such distribution, we select the distribution that minimizes the variance in Section 4.4.2. A single multileaving is sampled from this distribution and shown to the user who issued the query.

[1] We use the term *negative rank* even when we refer to OI with *linear rank difference*.

[2] We use the *Gurobi optimization toolkit*, `http://www.gurobi.com`.

Algorithm 5 *Optimized multileave* (OM).

Require: set of rankings $\mathcal{R}$, multileaving length k, sample size η.

1: $\mathcal{L} \leftarrow prefix\_constraint\_sampling(\mathcal{R}, \eta)$ *// Algorithm 4*
2: $\mathcal{C} \leftarrow \emptyset$ *// initialize set of constraints*
3: $\forall L_i \in \mathcal{L} : \mathcal{C} \leftarrow \mathcal{C} \cup \{0 < p_i < 1\}$ *// add simplex constraints*
4: $\forall k \forall x : \mathcal{C} \leftarrow \mathcal{C} \cup \{\sum_{i=1}^{|\mathcal{L}|} \left(p_i \sum_{j=1}^{k} \delta(d_{ij}, R_x) \right) = c_k\}$ *// add unbiasedness constraints*
5: $\forall L_i \in \mathcal{L} : \mu_i \leftarrow \frac{1}{|\mathcal{R}|} \sum_{x=1}^{|\mathcal{R}|} \sum_{j=1}^{|L_i|} f(j) \cdot \delta(d_{ij}, R_x)$ *// compute means*
6: $s_i \leftarrow \sum_{x=1}^{|\mathcal{R}|} \left(\left(\sum_{j=1}^{|L_i|} f(j) \cdot \delta(d_{ij}, R_x) \right) - \mu_i \right)^2$ *// sensitivity*
7: $o \leftarrow \sum_{i=1}^{|\mathcal{L}|} p_i \cdot s_i$ *// optimization objective*
8: $p \leftarrow minimize(o, \mathcal{C})$ *// constrained optimization problem*
9: $L_i \leftarrow$ sample from $\mathcal{L}$ with probability p_i
10: **return** L_i

The user's clicks are used to assign credit to each ranker that participated in the comparison. As with TDM, we maintain an empirical mean for all $\hat{P}_{ij}$. We increase the preference $\hat{P}_{ij}$ if and only if the sum of credit for ranker i was larger than the sum of credit for ranker j.

4.4.3 Probabilistic Multileave

In this section we derive *probabilistic multileave* (PM), a multileaving version of *probabilistic interleave* (PI) [79]. Extending PI to multiple rankers is non-trivial due to three challenges. First, the interleaving method, which combines multiple rankings into one list, must be extended to accommodate more than two rankers in a probabilistic way; see Section 4.4.3. Secondly, we need to represent the outcomes of a multileaved comparison such that it allows for more than two rankers. Lastly, we need to extend the marginalization over all assignments[3] to the multileave setting without increasing the number of possible assignments beyond what we can reasonably handle.

Constructing Multileaved Lists

Probabilistic interleaving [79] constructs interleaved lists in a way similar to TDI. For each rank in the interleaved list a coin is flipped to decide which ranker assigns the next document. Instead of picking the top document from that ranking, like in TDI, the document is drawn from a softmax distribution. After a document has been chosen it is removed from all rankings it occurs in and all softmax distributions are renormalized. An extension of this approach to the setting with r rankers can be found in Algorithm 6. As in TDM, and unlike PI, we choose to introduce the notion of rounds on line 4: in each round *all* rankers are chosen at random to contribute a document. This way we increase

[3] Following TDM and PI, documents in a multileaved list are assigned to a rankers' teams in order to distribute credit from clicks.

Algorithm 6 *Probabilistic multileave* (PM).

Require: rankers $\mathcal{R}$, size k.

1: $L \leftarrow []$ *// initialize interleaved list*
2: **while** True **do**
3: $\quad \mathcal{R}' \leftarrow \mathcal{R}$
4: $\quad$ **while** $|\mathcal{R}'| > 0$ **do**
5: $\quad\quad \mathcal{R}_j \leftarrow$ draw from uniform distribution over $\mathcal{R}'$
6: $\quad\quad \mathcal{R}' \leftarrow \mathcal{R}' \setminus \{\mathcal{R}_j\}$
7: $\quad\quad d \leftarrow$ draw from $\mathcal{R}_j$ with $P(d|\mathcal{R}_j)$ *// see Equation 4.12*
8: $\quad\quad L.append(d)$
9: $\quad\quad$ **if** $|L| = k$ **then**
10: $\quad\quad\quad$ **return** L
11: $\quad\quad$ **for** $\mathcal{R}_j \in \mathcal{R}$ **do**
12: $\quad\quad\quad \mathcal{R}_j.remove(d)$ *// renormalize $P(d|\mathcal{R}_j)$*

the change of the multileaved list to include documents from the top of each ranker that is to be compared while ensuring non-zero probability for all possible multileavings.

Documents d are drawn from ranker $\mathcal{R}_j$ with probability

$$P(d|\mathcal{R}_j) = \frac{\frac{1}{r_j(d)^\tau}}{\sum_{d' \in D} \frac{1}{r_j(d')^\tau}}, \tag{4.12}$$

where $r_j(d)$ denotes the rank of the document in ranker $\mathcal{R}_j$. When documents are removed from $\mathcal{R}_j$, as is done on line 12, this changes the distribution.

Inferring Preferences

Once the multileaved list has been created and shown to a user and that user has clicked on zero or more documents, these clicks can be interpreted as preferences for rankers.

Hofmann et al. [79] propose an unbiased estimator of the expected outcome of a comparison outcome over many such query impressions as

$$E[o(C, A)] \approx \frac{1}{|Q|} \sum_{q \in Q} o(c_q, a_q). \tag{4.13}$$

Here, Q denotes a given set of queries, with clicks c_q and assignments a_q. In the PI implementation, $o(c, a) \in \{-1, 0, 1\}$ is used to denote the outcome of a comparison, which we change to $o_j(c, a) \in \mathbb{N}$ to denote the outcome (credit) for a single ranker $\mathcal{R}_j$. This outcome simply counts how many clicked documents were assigned to this ranker, as in TDI or TDM. Changing the outcome notation leads us to also compute the expected outcome per ranker $\mathcal{R}_j$ as

$$E[o_j(C, A)] \approx \frac{1}{|Q|} \sum_{q \in Q} o_j(c_q, a_q). \tag{4.14}$$

The PI algorithm then proposes a marginalization over all possible assignments. The intuition why this is a good idea for both PI and PM is that through a single interaction

Algorithm 7 Inferring preferences in PM.

Require: multileaved list L, rankers $\mathcal{R}$, clicks C.

1: $A \leftarrow \{\}$ *// assignment tree keeps track of outcome and probabilities*
2: $A' \leftarrow \{\langle o \leftarrow [0, ..., 0], p \leftarrow 0\rangle\}$ *// init assignment tree, $|o| = |\mathcal{R}|$*
3: **for** $d \in L$ **do**
4: $A \leftarrow A', A' \leftarrow \{\}$ *// next layer in assignment tree*
5: **for** $\mathcal{R}_j \in \mathcal{R}$ **do**
6: $p_j \leftarrow P(d|\mathcal{R}_j)$ *// see Equation (4.12)*
7: $\mathcal{R}_j.remove(d)$ *// renormalize $P(d|\mathcal{R}_j)$*
8: **for** $\langle o, p\rangle \in A$ **do**
9: **for** $\mathcal{R}_j \in \mathcal{R}$ **do**
10: **if** $random() > \frac{1}{|\mathcal{R}|} \cdot n^{\frac{1}{|L|}}$ **then**
11: **continue** *// sample, skip branch*
12: $p' \leftarrow p + \log(\frac{p_j}{2})$ *// log probability of assigning d to $\mathcal{R}_j$*
13: $o' \leftarrow o$ *// copy outcome vector from parent*
14: **if** $d \in C$ **then**
15: $o'_j \leftarrow o'_j + 1$ *// click on d, increment outcome for $\mathcal{R}_j$*
16: $A' \leftarrow A' \cup \{\langle o', p'\rangle\}$ *// append to next layer*
17: $o \leftarrow [0, ..., 0]$ *// init outcome, $|o| = |\mathcal{R}|$*
18: **for** $\langle o', p'\rangle \in A'$ **do**
19: $o \leftarrow o + o' \cdot \mathrm{e}^{p'}$ *// aggregate outcome vector o weighted with $\mathrm{e}^{p'}$*
20: **return** o

with users, inferences can be performed as if the $\mathcal{R}$ rankers were multileaved many more times. This allows for highly sensitive comparisons.

We closely follow the marginalization of PI, and marginalize over all possible assignments as

$$E[o_j(C, A)] \approx \frac{1}{|Q|} \sum_{q \in Q} \sum_{a_q \in \tilde{A}_q} o_j(c_q, a_q) \cdot P(a_q|L_q, q). \tag{4.15}$$

In this equation, $P(a_q|l_q, q)$ denotes the probability of an assignment a_q occurring given the interleaved list L_q for query q. We borrow this probability directly from the PI implementation.

A major difference with PI, however, is that we can no longer consider all possible assignments A as there are generally too many of them, namely $|\mathcal{R}|^{|L|}$, which, even with a small amount of rankers, is prohibitively large. Instead, we limit the number of assignments by taking a random and uniform sample from them. We denote the sample as $\tilde{A}$, and control the size $n \approx |\tilde{A}|$ by, deciding with probability $\frac{1}{|\mathcal{R}|} \cdot n^{\frac{1}{|L|}}$ whether we consider a *branch* in a *tree* of assignments or not. This sampling happens on line 10 in Algorithm 7 which lists our method for inferring preferences in PM. The algorithm builds a tree of assignments by considering assigning each document $d \in L$ (line 3) to all possible rankers $\mathcal{R}_j \in \mathcal{R}$ (line 9) with the probability mentioned before (line 10). The algorithm keeps track of the outcome for each ranker for each assignment branch (line 15). It also tracks the probability of each such assignment (line 12). Once the outcomes $o_j(c_q, a_q)$ have been computed for all assignments $a_q \in A'$, we aggregate them

weighted with their assigned probabilities on line 19 into a single outcome. From this outcome we construct the preference matrix $\hat{P}_{ij}$ just as is done in TDM.

Note that in our implementation, just as in PI, we use log probabilities and sums to avoid buffer underflow. Also following PI, as an optimization, we only actually compute assignments and probabilities up to the lowest clicks.

4.5 Experiments

In this section, we detail our experimental setup in as far as it is different from the setup described in Chapter 3.

We first describe the data sets that we use in Section 4.5.1. Then, in Section 4.5.2 we describe how we select rankers. In Section 4.5.3, we detail our click simulation framework, in Section 4.5.4 we describe our experiments, and in Section 4.5.5 we detail our parameter settings.

4.5.1 Data Sets

Our experiments for RQ1, RQ3, RQ4, and RQ5 are conducted on all the nine data sets described in Section 3.2.

4.5.2 Selecting Rankers

For experiments aimed at answering RQ1, RQ3, RQ4, and RQ5, we handpick a set of features that are known to perform well and treat each of them independently as a ranker. Among others, we select BM25, LMIR.JM, Sitemap, PageRank, HITS and TF.IDF. Most of our experiments are run with $|\mathcal{R}| = 5$ rankers; only those experiments that investigate the impact of the number of rankers use a different number of rankers. We compute nDCG [89] for each ranker to produce the ground truth P_{ij} for all ranker pairs i, j on the held-out test fold, as described in Section 4.3. Note that while, for instance, PageRank is generally not a good ranker on its own (because it does not consider the query), in the data sets that we use, candidate documents for a query are preselected using a procedure that does take the query into account. Some average nDCG values of rankers that we use are 0.46 (BM25), 0.43 (Hyperlink based), 0.11 (PageRank), 0.50 (Sitemap), and 0.39 (LMIR.JM).

To answer RQ2, that is, to understand the impact of the difference between evaluated rankers on interleaving and multileaved comparison methods, we use synthetic data generated in a controlled way. We first generate, for each query, a ranking with 10 documents, 4 to 6 of them being relevant, using 3 grades for relevance labels as in, e.g., *OHSUMED* above. Then, we derive additional rankings by altering the initial ranking depending on the expected difference between them (see Section 4.6.1).

4.5.3 Simulating Clicks

To produce clicks, we use a click simulation framework that is analogous to [83], which is explained in [161] and in Section 3.3. In this chapter, we use the following four

instantiations of the click model: *perfect*, *navigational*, *informational* and *random*. Details can be found in Table 3.2.

4.5.4 Experimental Runs

Our experiments consider the following evaluation methods.

TDI *team draft interleave* [144], pairwise comparisons (baseline).

TDM *team draft multileave*, our extension of TDI that performs multileaved comparisons.

OI *optimized interleave* [143], pairwise comparisons (baseline).

OM *optimized multileave*, our extension of OI that performs multileaved comparisons. We experiment with several sample sizes η and the credit function.

PI *probabilistic interleave* [79], pairwise comparisons (baseline)

PM *probabilistic multileave*, our extension of PI that performs multileaved comparisons.

Our experiments for RQ1, RQ3, RQ4, and RQ5 are performed as follows. We select a set of rankers to compare. We then repeatedly sample queries randomly with replacement from the pool of queries. This simulates a user arriving at our search engine and entering a query. We assume that there is no dependence between two consecutive queries. When a query has been selected, it is given to the online evaluation methods. For the pairwise (baseline) methods, we select a pair of rankers such that all ranker pairs i, j where $i \neq j$ are compared the same number of times. The multileaved comparison methods, on the other hand, compare all rankers at the same time. So, either an interleaving of two rankers or a multileaving of all rankers is shown to the user. We then simulate the user interacting with the result list and produce clicks according to the given instantiation of the click model. Using these clicks, for the pairwise (baseline) methods, we update $\hat{P}_{ij}$ only for the pair of rankers that we compared. For the multileaved comparison methods, we update all $\hat{P}_{ij}$ for all pairs of rankers.

For RQ2, we follow the above approach as closely as possible. However, since there is no notion of a *ranker* that generalizes over queries, we repeatedly ($N = 100$) issue the same set of rankings to produce clicks with the click model in order to obtain a stable $\hat{P}_{ij}$.

The main objective for all experiments is to find the $\hat{P}_{ij}$ that minimizes the error metric E_{bin} when compared to ground truth P_{ij} computed using nDCG (see Section 4.3). We also investigate other properties. We measure the *bias* of each method by using a random instantiation of the click model and comparing with E_{bin} to a ground truth where $P_{ij} = 0.5$ for all pairs of rankers. We also measure online performance in terms of nDCG of the rankings presented to the user. Lastly, we measure the effect of the number of rankers we compare and the effect of the length of the result list. We test for significant differences using a two tailed t-test.

4.5.5 Parameter Settings

Lastly, we describe parameters used in our experiments. For OM, we set the number of multileavings to $\eta = 1, 5, 10, 100$. For both OI and OM we test two types of credit function: *negative credit* and *inverse credit*. For OM, we use *inverse credit* by default

and for OI we use *negative credit* unless stated otherwise as these performed best for the respective methods. For all experiments except those that investigate the effects of these parameters, the number of rankers is $|\mathcal{R}| = 5$ and the results lists length is $k = 10$. Lastly, for PM the softmax decay is controlled by τ which we set to 3, following PI [79]. TDM does not have any parameters.

4.6 Results and Analysis

In this section we answer research questions RQ1 through RQ5, posed in Section 1.1. We start by comparing TDM and OM head to head in Section 4.6.1 and then continue by comparing PM to TDM, the winning method, in Section 4.6.2.

4.6.1 Team Draft Multileave and Optimized Multileave

Our main result with respect to TDM and OM is depicted in Figure 4.1. It shows the error measured with E_{bin} for the two baseline interleaving methods OI and TDI and for two of our new multileaving methods, OM and TDM. These results are obtained by aggregating over all the data sets that we consider. Table 4.1 provides an alternative view on the same results by splitting them per data set. We performed our analysis for three levels of increasing noise in the feedback: *perfect*, *navigational* and *informational* instantiations of the click model.

Interestingly, as can be seen in Figure 4.1, the multileaved extensions of the interleaving methods converge to an error close to their interleaving counterparts. Both OI and OM have difficulties coping with noise in user feedback: the error to which these methods converge increases when the noise increases. This is in contrast with TDI and TDM: with increasing noise they are capable of learning the ranker preference almost as well as with the *perfect* click model.

In response to RQ1, Figure 4.1 shows clearly that the error of both of our multileaving methods drops much faster than their interleaving counterparts. This indicates that multileaved comparison methods can learn preferences between multiple rankers with far less data (i.e., queries and clicks) than interleaved comparison methods.

Under perfect feedback, TDM and OM learn ranker preferences equally fast. When noise increases, OM initially learns these preferences faster than TDM does. Under noisy feedback, TDM keeps improving the learned preferences long after OM has plateaued.

Table 4.1 shows the error E_{bin} at 500 queries. We choose a rather low number of queries to emphasize learning speed. Note that the rightmost column is equal to the E_{bin} values in a slice of Figure 4.1 after 500 queries. For the multileaving methods, each $\hat{P}_{ij}$ has had 500 updates by then. The interleaving methods only performed 50 updates of $\hat{P}_{ij}$ for each pair of rankers. The results show that, in general, the multileaving methods have significantly less error than the interleaving methods. In particular, OM has less error than OI in 24 out 27 experiments. The exception to this rule are the three experiments on MQ2007. TDM has less error than TDI in 22 out of 27 experiments. In two experiments, TDM has a significantly higher error; those experiments are on perfect and navigational instantiations of the click model on the TD2003 data set. In both these

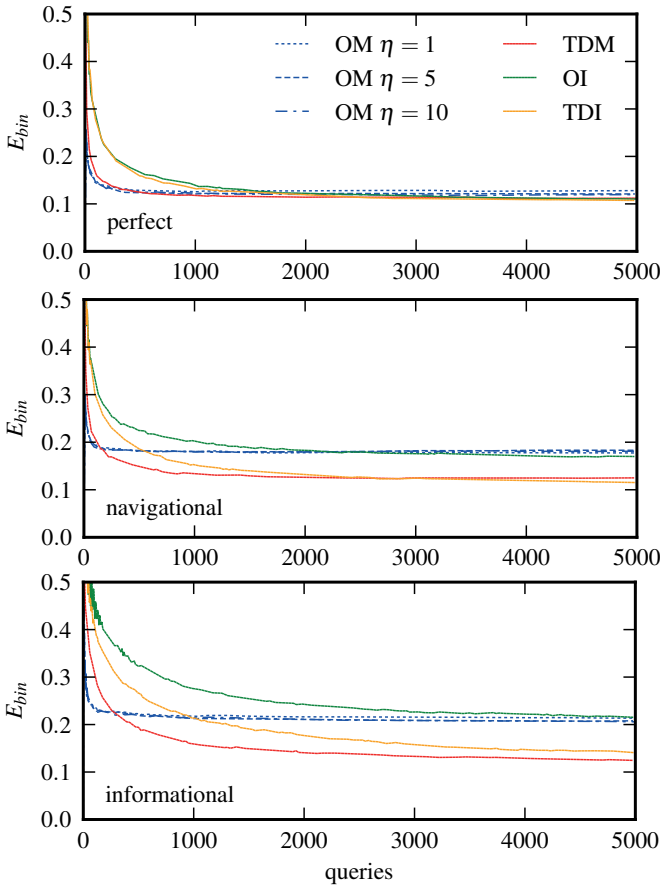

Figure 4.1: Average E_{bin} error of interleaved and multileaved comparison methods OM and TDM and their baselines. Averaged over 25 repetitions, 9 data sets with 5 folds each. The plots depict the error (see Equation 4.2) for three instantiations of the click model: perfect, navigational and informational. Result list length $l = 10$ and number of rankers $|\mathcal{R}| = 5$.

Table 4.1: E_{bin} at 500 queries for OM and TDM versus their respective baselines OI and TDI. Averaged over 25 repetitions and 5 folds. Standard deviation is between brackets. Per data set and instantiation of the click model, we print the best method in bold. Statistically significant improvements (losses) over the respective baselines are indicated by $^{\triangle}$ ($p < 0.05$) and $^{\blacktriangle}$ ($p < 0.01$) ($^{\triangledown}$ and $^{\blacktriangledown}$).

Method	HP2003	HP2004	MQ2007	MQ2008	NP2003	NP2004	OHSUMED	TD2003	TD2004	Average
perfect click model										
OM $\eta = 1$	**0.000** (0.00) $^{\blacktriangle}$	**0.000** (0.00) $^{\blacktriangle}$	0.324 (0.13) $^{\blacktriangledown}$	0.043 (0.05) $^{\blacktriangle}$	0.067 (0.07) $^{\blacktriangle}$	0.088 (0.09) $^{\triangle}$	0.340 (0.15) $^{\blacktriangle}$	0.194 (0.13) $^{\blacktriangle}$	**0.096** (0.07) $^{\blacktriangle}$	0.128 (0.15)
OM $\eta = 5$	**0.000** (0.00) $^{\blacktriangle}$	**0.000** (0.00) $^{\blacktriangle}$	0.285 (0.11) $^{\triangledown}$	**0.040** (0.05) $^{\blacktriangle}$	0.068 (0.07) $^{\blacktriangle}$	**0.084** (0.08) $^{\blacktriangle}$	0.341 (0.14) $^{\blacktriangle}$	0.195 (0.13) $^{\blacktriangle}$	0.106 (0.08) $^{\triangle}$	0.124 (0.15) $^{\triangle}$
OM $\eta = 10$	**0.000** (0.00) $^{\blacktriangle}$	**0.000** (0.00) $^{\blacktriangle}$	0.297 (0.11) $^{\blacktriangledown}$	0.049 (0.06) $^{\blacktriangle}$	0.071 (0.07) $^{\blacktriangle}$	0.090 (0.10) $^{\triangle}$	0.338 (0.14) $^{\blacktriangle}$	**0.186** (0.12) $^{\blacktriangle}$	0.107 (0.08) $^{\triangle}$	0.126 (0.15) $^{\triangle}$
OM $\eta = 100$	**0.000** (0.00) $^{\blacktriangle}$	**0.000** (0.00) $^{\blacktriangle}$	0.233 (0.11)	0.049 (0.05) $^{\blacktriangle}$	**0.067** (0.06) $^{\blacktriangle}$	0.093 (0.10)	0.338 (0.15) $^{\blacktriangle}$	0.193 (0.12) $^{\blacktriangle}$	0.114 (0.08)	0.136 (0.15)
OI	0.014 (0.04)	0.035 (0.05)	0.254 (0.12)	0.155 (0.10)	0.111 (0.09)	0.116 (0.09)	0.440 (0.16)	0.243 (0.12)	0.131 (0.10)	0.167 (0.16)
TDM	0.005 (0.02)	0.018 (0.04)	**0.166** (0.08) $^{\blacktriangle}$	0.050 (0.07) $^{\blacktriangle}$	0.086 (0.06) $^{\blacktriangle}$	0.097 (0.10)	**0.265** (0.19) $^{\blacktriangle}$	0.270 (0.10) $^{\triangledown}$	0.159 (0.07)	**0.124** (0.13)
TDI	0.007 (0.03)	0.024 (0.04)	0.305 (0.13)	0.134 (0.09)	0.114 (0.09)	0.122 (0.10)	0.350 (0.16)	0.235 (0.12)	0.143 (0.10)	0.159 (0.15)
navigational click model										
OM $\eta = 1$	**0.005** (0.02) $^{\blacktriangle}$	**0.006** (0.02) $^{\blacktriangle}$	0.530 (0.10) $^{\blacktriangledown}$	0.138 (0.09) $^{\blacktriangle}$	**0.062** (0.05) $^{\blacktriangle}$	**0.080** (0.07) $^{\blacktriangle}$	0.430 (0.11) $^{\blacktriangle}$	0.228 (0.15) $^{\triangle}$	0.159 (0.12)	0.182 (0.20)
OM $\eta = 5$	0.011 (0.03) $^{\triangle}$	0.011 (0.03) $^{\blacktriangle}$	0.518 (0.12) $^{\blacktriangledown}$	0.132 (0.09) $^{\blacktriangle}$	0.068 (0.06) $^{\blacktriangle}$	**0.080** (0.07) $^{\blacktriangle}$	0.438 (0.12) $^{\blacktriangle}$	**0.227** (0.15) $^{\triangle}$	**0.154** (0.11)	0.182 (0.20)
OM $\eta = 10$	0.018 (0.04)	0.012 (0.03) $^{\blacktriangle}$	0.503 (0.10) $^{\blacktriangledown}$	0.134 (0.10) $^{\blacktriangle}$	0.063 (0.06) $^{\blacktriangle}$	0.081 (0.08) $^{\blacktriangle}$	0.419 (0.13) $^{\blacktriangle}$	0.237 (0.14)	0.158 (0.11)	0.181 (0.19)
OM $\eta = 100$	0.013 (0.03)	0.019 (0.04) $^{\blacktriangle}$	0.462 (0.11) $^{\blacktriangledown}$	0.137 (0.09) $^{\blacktriangle}$	0.068 (0.06) $^{\blacktriangle}$	0.082 (0.08) $^{\blacktriangle}$	0.436 (0.13) $^{\blacktriangle}$	0.253 (0.16)	0.167 (0.10)	0.211 (0.20)
OI	0.022 (0.04)	0.044 (0.06)	0.398 (0.14)	0.229 (0.12)	0.116 (0.08)	0.137 (0.09)	0.635 (0.16)	0.269 (0.12)	0.166 (0.10)	0.224 (0.21)
TDM	0.021 (0.04)	0.026 (0.04)	**0.190** (0.09) $^{\blacktriangle}$	**0.082** (0.08) $^{\blacktriangle}$	0.086 (0.07) $^{\blacktriangle}$	0.106 (0.10) $^{\blacktriangle}$	**0.330** (0.17) $^{\blacktriangle}$	0.308 (0.14) $^{\triangledown}$	0.188 (0.08)	**0.149** (0.15)
TDI	0.017 (0.04)	0.030 (0.05)	0.322 (0.14)	0.198 (0.11)	0.126 (0.09)	0.154 (0.10)	0.386 (0.16)	0.272 (0.14)	0.169 (0.10)	0.186 (0.16)
informational click model										
OM $\eta = 1$	0.089 (0.03) $^{\blacktriangle}$	0.071 (0.05) $^{\blacktriangle}$	0.635 (0.12) $^{\blacktriangledown}$	**0.169** (0.10) $^{\blacktriangle}$	**0.064** (0.05) $^{\blacktriangle}$	**0.083** (0.07) $^{\blacktriangle}$	0.397 (0.09) $^{\blacktriangle}$	0.289 (0.17) $^{\blacktriangle}$	0.213 (0.10) $^{\blacktriangle}$	0.223 (0.20) $^{\blacktriangle}$
OM $\eta = 5$	0.095 (0.02) $^{\blacktriangle}$	0.081 (0.05) $^{\blacktriangle}$	0.602 (0.13) $^{\blacktriangledown}$	0.170 (0.12) $^{\blacktriangle}$	0.070 (0.06) $^{\blacktriangle}$	0.090 (0.07) $^{\blacktriangle}$	0.383 (0.10) $^{\blacktriangle}$	0.289 (0.16) $^{\blacktriangle}$	0.199 (0.10) $^{\blacktriangle}$	0.220 (0.20) $^{\blacktriangle}$
OM $\eta = 10$	0.096 (0.02) $^{\blacktriangle}$	0.087 (0.04) $^{\blacktriangle}$	0.583 (0.15) $^{\blacktriangledown}$	0.186 (0.11) $^{\blacktriangle}$	0.072 (0.05) $^{\blacktriangle}$	0.086 (0.07) $^{\blacktriangle}$	0.380 (0.11) $^{\blacktriangle}$	0.294 (0.16) $^{\blacktriangle}$	0.199 (0.09) $^{\blacktriangle}$	0.220 (0.19) $^{\blacktriangle}$
OM $\eta = 100$	0.100 (0.00) $^{\blacktriangle}$	0.101 (0.03) $^{\blacktriangle}$	0.518 (0.13) $^{\blacktriangledown}$	0.178 (0.11) $^{\blacktriangle}$	0.080 (0.06) $^{\blacktriangle}$	0.095 (0.07) $^{\blacktriangle}$	0.393 (0.11) $^{\blacktriangle}$	**0.282** (0.15) $^{\blacktriangle}$	0.200 (0.09) $^{\blacktriangle}$	0.243 (0.18) $^{\blacktriangle}$
OI	0.202 (0.12)	0.198 (0.13)	0.421 (0.15)	0.318 (0.14)	0.186 (0.11)	0.246 (0.11)	0.674 (0.14)	0.382 (0.13)	0.280 (0.13)	0.323 (0.20)
TDM	**0.060** (0.07) $^{\blacktriangle}$	**0.066** (0.06) $^{\blacktriangle}$	**0.276** (0.16) $^{\blacktriangle}$	0.177 (0.12) $^{\blacktriangle}$	0.139 (0.10) $^{\blacktriangle}$	0.147 (0.09) $^{\blacktriangle}$	**0.366** (0.19) $^{\blacktriangle}$	0.317 (0.13)	**0.198** (0.11) $^{\blacktriangle}$	**0.194** (0.16) $^{\blacktriangle}$
TDI	0.120 (0.10)	0.131 (0.09)	0.419 (0.15)	0.307 (0.14)	0.190 (0.10)	0.207 (0.11)	0.438 (0.17)	0.352 (0.16)	0.242 (0.14)	0.267 (0.17)

Table 4.2: E_{bin} when the number of rankers $|\mathcal{R}|$ is varied for OM and TDM and their baselines. Result list length $k = 10$, averaged over 10 repetitions and 5 folds of the NP2003 data set. Standard deviation of the error is given in brackets.

Method	$\|\mathcal{R}\| = 3$	$\|\mathcal{R}\| = 5$	$\|\mathcal{R}\| = 7$	$\|\mathcal{R}\| = 10$
OM $\eta = 10$	0.144 (0.16)	0.154 (0.12)	0.111 (0.06)	0.116 (0.04)
TDM	0.191 (0.18)	0.192 (0.09)	0.190 (0.06)	0.203 (0.05)
OI	0.189 (0.18)	0.200 (0.08)	0.255 (0.06)	0.316 (0.04)
TDI	0.143 (0.13)	0.214 (0.09)	0.246 (0.05)	0.284 (0.04)

exceptions convergence was reached long before 500 queries for all methods. While the multileaving methods still converged faster, they did so to a slightly higher error.

For OM we see in both Table 4.1 and Figure 4.1 that η, the sample size, does not seem to have a large effect on the error. Therefore, with a surprisingly small number of samples, effective and computationally efficient multileaving is possible. Consequently, in most of the analyses that follow, we report only on OM with $\eta = 10$.

Scaling the Number of Rankers with TDM and OM

The motivation for performing *multileaved* comparisons lies in the fact that it is possible to compare multiple rankers at once. Most of our experiments in this chapter use a set of 5 rankers but, in response to RQ4, in this section we analyze what happens when the number of rankers being compared increases.

Table 4.2 lists how each method performs when the number of rankers to be compared varies. We kept the result list length fixed at $k = 10$. Both interleaving methods OI and TDI are impacted greatly when the number of rankers increases. This is largely due to the fact that many more comparisons are needed and as such each $\hat{P}_{ij}$ receives fewer updates. By contrast, OM and TDM do not show significant degradation when the number of rankers increases.

We suspect that there may be an interaction between the number of rankers that are compared and the length of the result list shown to the user. Depending on the method, the result list length may limit the number of rankers that can be represented at once. We experimented with several settings where we varied the number of rankers to be compared and the result list length. We considered all combinations of $|\mathcal{R}| = 3, 5, 7, 10$ rankers and lengths $k = 3, 5, 7, 10$. Because of computational limitations, we had to limit ourselves to a single data set, a single user model, with fewer repetitions and fewer queries. We selected the NP2003 data set with the *informational* instantiation of the click model with 10 repetitions and 2.5K queries.

In Figure 4.2, we plot the error E_{bin} against the number of rankers per documents in the result list. The four rightmost data points, for instance, were produced using 10 rankers and result lists of length 3 only. The leftmost points are from the opposite scenario: 3 rankers were compared with document lists of length 10. Note that there are many ways in which $\frac{|\mathcal{R}|}{k}$ can be equal to 1, and that therefore there is a relatively wide spread of error.

We fitted lines for each evaluation method using least squares. Though these lines

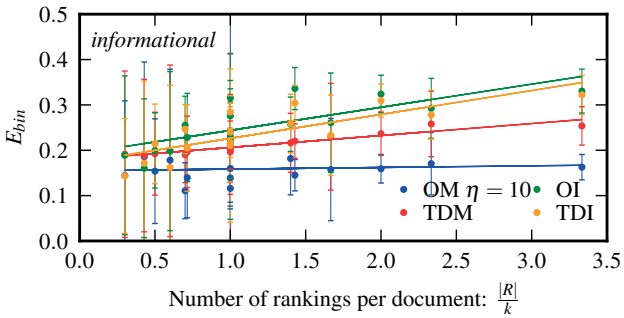

Figure 4.2: Scaling the number of rankers for OM and TDM and their baselines. Average E_{bin} against the number of rankers x per result list length k. Computed on all combinations of $|\mathcal{R}| = 3, 5, 7, 10$ and $k = 3, 5, 7, 10$. Averaged over 10 repetitions and 5 folds of the NP2003 data set. Standard deviation is indicated with error bars and lines are fitted using least squares.

are not perfect fits, they give a useful indication of the behavior of the methods when the ratio between the number of rankers and the number of documents increases. Figure 4.2 shows that the multileaving methods can cope better with an increase in this ratio than the interleaving baselines. The performance of OM is not impacted by an increase of this ratio; the two interleaving methods almost double their error when the ratio increases from $\frac{3}{10}$ to $\frac{10}{3}$.

While Table 4.2 shows that TDM is not impacted by the number of rankers, in Figure 4.2, we see that the error for TDM does increase when the ratio of rankers per result list length goes up. We attribute this to the fact that team draft methods always assign a document in an interleaving to a single input ranker. When there are (many) more rankers than documents to which they can be assigned, then most rankers cannot be distinguished from one another. Consequently, not all $\hat{P}_{ij}$ can be updated per comparison.

Sensitivity of TDM and OM

In this section, we investigate RQ2. We study the impact of the difference between evaluated rankers on interleaving and multileaved comparison methods using synthetic data as discussed in Section 4.5.2. We consider cases when the position of one or more document(s) changes from one ranking to another (we also investigated cases when one or more document(s) are replaced by new ones and obtained similar results). In doing so, we control two things: the *number* of documents moved as well as the *amplitude* of the move, i.e., how far away is the moved document located from its original position. While we only control the difference w.r.t. a single ranking and not between all pairs of rankings, by increasing the number and amplitude of the changes, we increase the space of possible rankings, effectively increasing the chance of them being different from each other.

For each interleaving and multileaved comparison method, we look at the impact on E_{bin} at 500 queries of the difference between rankings using the *informational* click

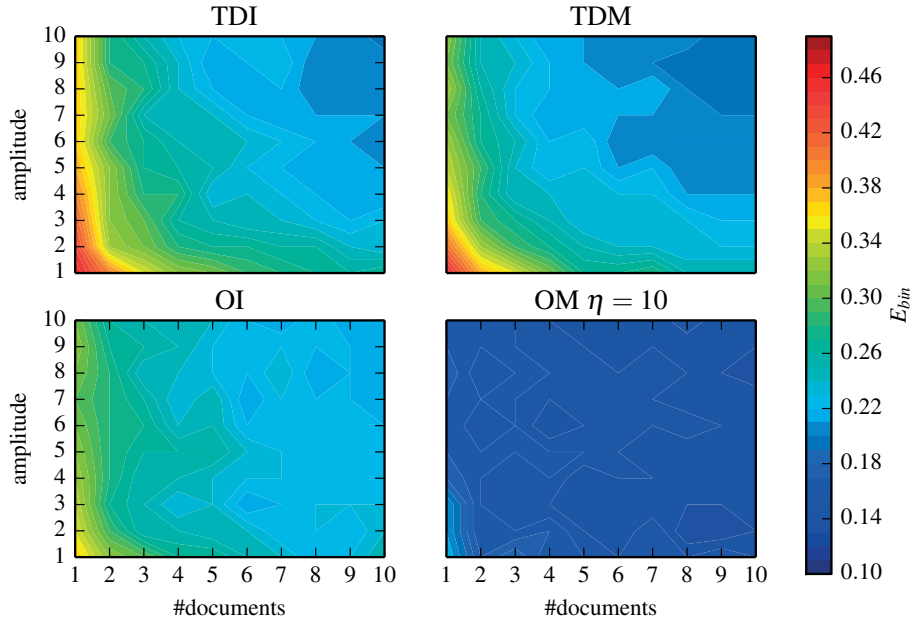

Figure 4.3: The effect on TDI, TDM, OI, and OM of differences between rankings, the number of moved documents and the amplitude of the move is controlled. E_{bin} at 500 queries, 100 issues, averaged over 125 repetitions. We used the informational click model.

model, with $|\mathcal{R}| = 5$ rankers, result lists of length $k = 10$ and 100 issues of each query. Results are depicted in Figure 4.3 as a heat map of E_{bin} depending on the number of documents moved and the amplitude of the move. We observe that E_{bin} decreases as the difference between rankings increases (whether this is the number of moves or the amplitude of the moves) in the same way for all methods, which means that differences between rankers affect all methods in the same way. We also observe that OM performs much better than other methods, which is in line with Figure 4.1 at the 100 query issue point.

Returning to RQ2, these results show that the sensitivity of multileaving methods is affected in the same way as for interleaving methods when the differences between rankers vary. Interestingly, this means that multileaved methods can distinguish between rankers just as well as interleaving methods even when the differences between them is very small. Hence, multileaved comparison methods can be used to explore a parameter space using very small steps.

Bias of TDM and OM

Next, we address RQ3. We evaluate fidelity requirement (2) from [85] which states that, under random clicks, rankers should tie in expectation. TDI was designed to fulfill this requirement. We run experiments with the *random* instantiation of the click model (see

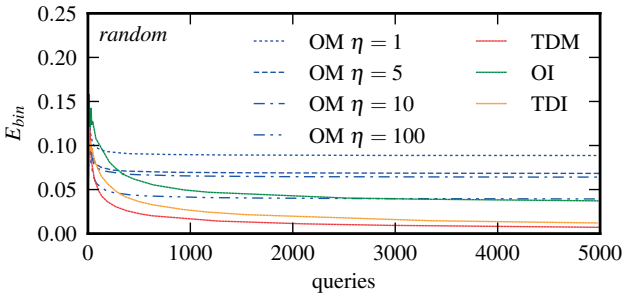

Figure 4.4: Bias of OM and TDM and their baselines as measured by measuring the incorrectly identified preferences under a random click model, with $|\mathcal{R}| = 5$ rankers and result list of length $k = 10$. Measured as E_{bin} versus a ground truth with no preferences, $P_{ij} = 0.5$ for all i, j. Averaged over 25 repetitions, 9 data sets with each 5 folds.

Section 4.5.3). When a user clicks on a result list without any preference for relevant documents, an online evaluation method that interprets these clicks should not detect any preferences among rankers. We measure how many preferences each comparison method detects when exposed to a random user by comparing the $\hat{P}_{ij}$ of the method to a ground truth that consists of $P_{ij} = 0.5$ for all i, j using E_{bin}.

The result is shown in Figure 4.4. For all methods, the error quickly drops to rather low values. Both TDI and TDM steadily converge to values near 0. Within a few hundred queries, their error is below 5%. In the long run, neither method detects differences among rankers when it should not. OI takes much longer to drop below 5% and plateaus higher than both team draft methods. For OM, it turns out that the number of multileavings that is sampled, η (see Section 4.4.2) has a big impact on the bias of the method. The larger the sample size, the less bias the OM method has. A more elaborate explanation of this effect can be found in Section 4.6.1. It may come as a surprise that both OI and OM have such a large bias since both these methods explicitly restrict themselves to producing unbiased result lists. The fact that the error increases when η goes up (see Table 4.1) can be explained by a bias-variance trade-off: when η goes up, the bias goes down at the cost of variance that is introduced.

Online Performance of TDM and OM

A general concern with online ranker evaluation is that users may be confronted with inferior systems. The degree to which this happens may vary per evaluation method. Again, in response to RQ3, we measure online performance of the four evaluation methods using nDCG [89]. Table 4.3 lists the nDCG for each evaluation method measured on the result list that was actually shown to the user. On average, TDM produces the highest online performance, i.e., users were the least affected by the evaluation in which they participated.

Interestingly, for OM, the nDCG score goes down when the sample size η goes up. This may be due to the fact that, when the number of sampled multileavings goes up, the

Table 4.3: Online performance of OM and TDM and their baselines measured with nDCG (higher is better). Averaged over 5K queries, 25 repetitions and 5 folds. Standard deviation is between brackets. Per data set, we print the best value in bold.

Method	HP2003	HP2004	MQ2007	MQ2008	NP2003	NP2004	OHSUMED	TD2003	TD2004	total
OM $\eta = 1$	0.522 (0.01)	0.465 (0.01)	0.289 (0.01)	0.377 (0.02)	0.500 (0.02)	0.445 (0.03)	0.396 (0.02)	0.183 (0.03)	0.180 (0.01)	0.373 (0.12)
OM $\eta = 5$	0.491 (0.01)	0.430 (0.01)	0.289 (0.00)	0.374 (0.02)	0.463 (0.02)	0.410 (0.02)	0.396 (0.02)	0.174 (0.02)	0.172 (0.01)	0.355 (0.11)
OM $\eta = 10$	0.486 (0.01)	0.425 (0.01)	0.289 (0.00)	0.373 (0.02)	0.460 (0.02)	0.407 (0.02)	0.394 (0.02)	0.173 (0.02)	0.170 (0.01)	0.353 (0.11)
TDM	0.536 (0.01)	**0.476** (0.01)	0.288 (0.01)	0.376 (0.02)	**0.513** (0.02)	**0.457** (0.03)	**0.398** (0.02)	**0.196** (0.03)	0.184 (0.01)	**0.380** (0.13)
OI	**0.539** (0.01)	0.476 (0.01)	**0.297** (0.00)	**0.383** (0.02)	0.506 (0.02)	0.432 (0.03)	0.388 (0.02)	0.173 (0.03)	**0.188** (0.01)	0.376 (0.13)
TDI	0.493 (0.01)	0.446 (0.01)	0.293 (0.00)	0.380 (0.02)	0.482 (0.02)	0.419 (0.03)	0.394 (0.02)	0.166 (0.02)	0.175 (0.01)	0.361 (0.12)

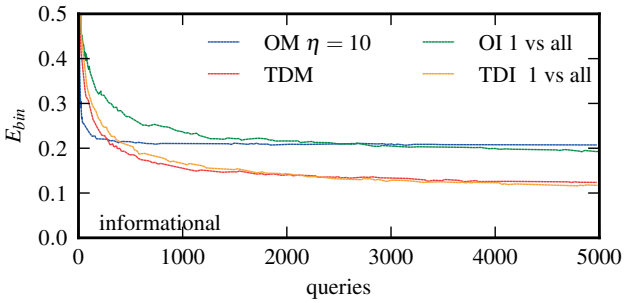

Figure 4.5: OM and TDM and their baselines on the task of comparing one ranker versus many rankers, measured with E_{bin} of $\hat{P}_{ij}$ against P_{ij} where we keep i fixed. Averaged over 25 repetitions, 9 data sets and 5 folds.

optimization problem is less overconstrained. As a consequence, it is easier to satisfy the unbiasedness constraint. Less biased multileavings are more "in between" the input rankings and therefore they do not represent a strong preference for one ranker. Such multileavings turn out to have a lower nDCG. TDM does not suffer from this problem. On some data sets, in particular HP2003, HP2004, NP2003 and NP2004, for OM the online performance drops considerably when η goes up. Incidentally, on these data sets, the error also increases when η goes up (see Table 4.1); less biased multileavings have a lower online performance.

Comparing to a Production Ranker using TDM and OM

Though we focus on efficiently comparing all rankers to each other, other variants are also useful in practice, as detailed in Section 4.3. Here, we investigate how online evaluation methods perform on one such variant: comparing a set of rankers to a single benchmark, e.g., a production ranker. Though our multileaving methods were not specifically designed for this variant, we can measure their performance on it by computing the error E_{bin} of $\hat{P}_{ij}$ against P_{ij} where we keep i fixed. The error is thus computed using only one row of the preference matrix. We perform this experiment on the *informational* instantiation of the click model and we average over 25 repetitions, 9 data sets and 5 folds.

Figure 4.5, which presents the result of this analysis, shows that multileaving methods outperform the interleaving methods. OM, in particular, continues to learn much more quickly than the alternatives. Unsurprisingly, when comparing Figure 4.5 to Figure 4.1, we see that the advantage of multileaving methods over interleaving methods diminishes when the task changes from learning all cells in $\hat{P}$ to learning just one row in $\hat{P}$. Note that the multileaving methods do still learn all cells in $\hat{P}$.

Parameters of OM and OI

In this section, we investigate some of the design choices made when extending OI to OM; where possible, we do so by comparing to the impact of our same choices on OI.

Table 4.4: Overconstrainedness of OM $\eta = 10$ averaged over 10 repetitions and 5 folds of the NP2003 data set. Values depict the proportion of experiments that was overconstrained when run with the indicated parameters. When the constrained optimization problem is always overconstrained (values close to 1), then there is no room for optimization.

k	$\lvert\mathcal{R}\rvert = 3$	$\lvert\mathcal{R}\rvert = 5$	$\lvert\mathcal{R}\rvert = 7$	$\lvert\mathcal{R}\rvert = 10$
3	0.393 (0.21)	0.976 (0.03)	0.999 (0.00)	1.000 (0.00)
5	0.934 (0.18)	0.996 (0.01)	0.999 (0.00)	1.000 (0.00)
7	0.984 (0.05)	0.997 (0.00)	0.999 (0.00)	1.000 (0.00)
10	0.995 (0.01)	0.998 (0.00)	1.000 (0.00)	1.000 (0.00)

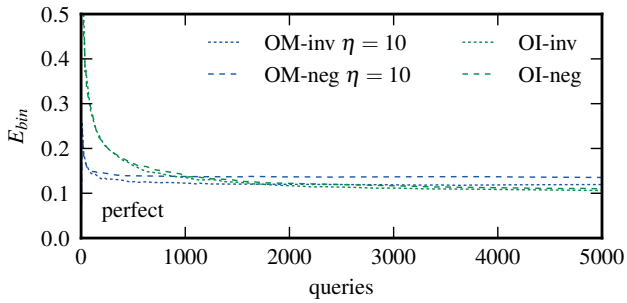

Figure 4.6: Impact of *negative* and *inverse* credit functions (see Section 4.4.2) in OI and OM on the perfect click model. Averaged over 25 repetitions, 9 data sets with 5 folds each.

As described in Section 4.4.2, we had to restrict the number of multileavings we can consider in the optimization problem of OM. As we saw in Section 4.6.1 and to a lesser extent in Section 4.6.1, the number of sampled multileavings η does have an impact on the performance of OM. We hypothesized that this is due to the optimization problem of OM becoming overconstrained when the number of multileavings is small. When we investigate this effect, we find the following. For smaller sample sizes, $\eta = 1, 5, 10$, the problem was almost always overconstrained on all of the nine data sets. With $\eta = 100$, the problem was overconstrained in 85% of the multileaved comparisons. For OI, we confirm the claim by Radlinski and Craswell [143] that the optimization problem is usually underconstrained: we found that the problem was overconstrained in only 1% of the interleavings.

The above findings were all for the scenario with $\lvert\mathcal{R}\rvert = 5$ rankers and $k = 10$ documents in the result lists. In Table 4.4, we see what happens when we vary $\lvert\mathcal{R}\rvert$ and k and keep $\eta = 10$. Computational limitations prevented us from evaluating what would happen with values larger than $\eta = 100$. As long as the number of rankers is small and the length of the multileaving is short, a small number of samples is enough to avoid having an overconstrained problem.

In Figure 4.6, we analyze the impact of the credit function (see Section 4.4.2) on OI

Table 4.5: Mean E_{bin} scores after 500 impressions for PM compared to baselines PI (first symbol) and TDM (second symbol). The symbol $^{\blacktriangle}$ means statistically better with $p < 0.01$ and $^{\triangle}$ for $p < 0.05$, whereas $^{\blacktriangledown}$ and $^{\triangledown}$ are their inverses.

	perfect	*navigational*	*informational*
PI	0.085 (0.08)	0.137 (0.11)	0.363 (0.15)
TDM	0.037 (0.06)	0.038 (0.05)	0.099 (0.09)
PM($n = 10^2$)	0.062 (0.07) $^{\blacktriangle\blacktriangledown}$	0.073 (0.07) $^{\blacktriangle\blacktriangledown}$	0.162 (0.10) $^{\blacktriangle\blacktriangledown}$
PM($n = 10^3$)	0.054 (0.05) $^{\blacktriangle\blacktriangledown}$	0.060 (0.06) $^{\blacktriangle\blacktriangledown}$	0.117 (0.09) $^{\blacktriangle\triangledown}$
PM($n = 10^4$)	0.046 (0.05) $^{\blacktriangle}$-	0.054 (0.05) $^{\blacktriangle\blacktriangledown}$	0.090 (0.08) $^{\blacktriangle}$-
PM($n = 10^5$)	0.046 (0.05) $^{\blacktriangle}$-	0.039 (0.05) $^{\blacktriangle}$-	0.087 (0.08) $^{\blacktriangle}$-

and OM. We see that OM performs best when using the *inverse credit function* while the effect of the credit function on OI is smaller than on OM. The observed degraded performance of the negative credit function for OM is explained by the fact that this credit function assumes a linear relation between the rank and credit. This effect is stronger in OM because the credit function does not model the difference but rather absolute values.

4.6.2 Probabilistic Multileave

Now we turn to experimentally evaluating PM, our probabilistic multileaving method, answering RQ5. We compare PM to TDM as in most of the experiments above TDM clearly outperformed OM. We evaluate sensitivity, bias and whether PM can scale when the number of rankers it is comparing increases.

Sensitivity of PM

We first look at sensitivity. In Figure 4.7 we see that for impressions from all three click models, TDM performs similarly to PM with a large number of samples n, although TDM performs slightly (but not significantly) better in the long run for the (unrealistic) *perfect* click model. The multileaving methods both perform much better than PI, as expected since the latter can only compare two rankers at a time.[4]

The binary error for both probabilistic multileaving and interleaving quickly goes down to almost zero. Performance after 500 queries for the same experiments is found in Table 4.5 where we also perform statistical significance testing. In the table we see that PM always has a lower error than PI and when there are enough samples ($n \geq 100$) statistically significantly so. However, when the number of samples is too small, PM is outperformed significantly by TDM. When the number of samples increases sufficiently, PM is on par with TDM in terms of sensitivity. Interestingly, when noise increases, performance of PM decreases less compared to TDM.

[4]The bump for PI in the first few query impressions, that is mostly visible under *informational* feedback, is due to the fact that it takes a while for PI to have preferences for all pairs of rankers.

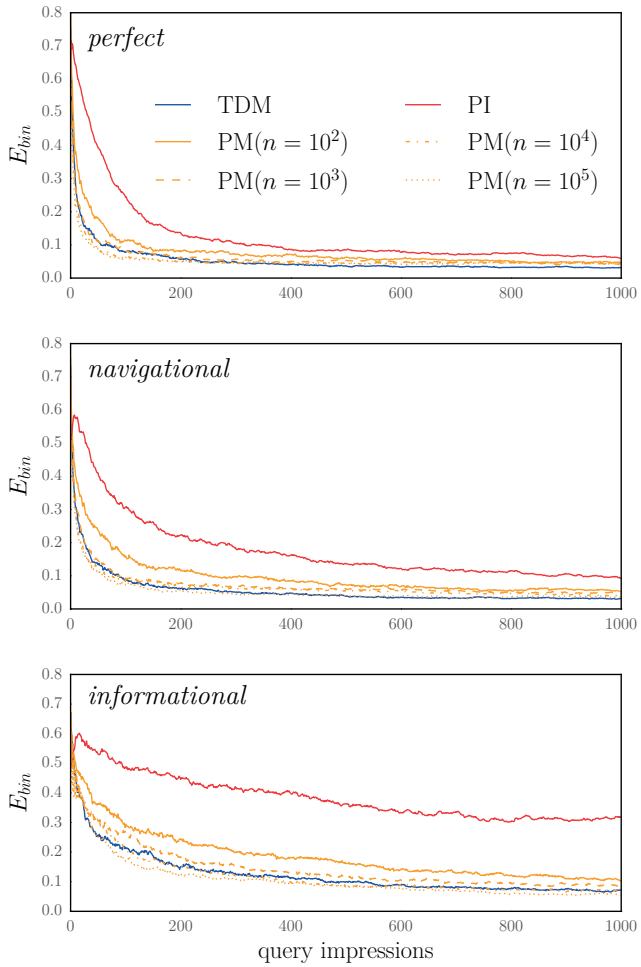

Figure 4.7: The binary error E_{bin} of PM, TDM and PI are plotted against the number of queries on which the error was evaluated. Clicks are generated by a *perfect*, *navigational*, and *informational* instantiations of the *dependent click model* (DCM) [65] (see Section 3.3).

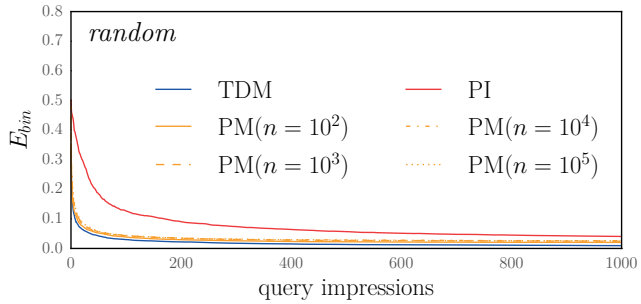

Figure 4.8: To measure bias of PM when compared to TDM and PI, the error is plotted against the number of queries. The error was evaluated by comparing to a ground truth of *no preferences* (i.e., $P_{ij} = 0.5$ for all i, j). Clicks are generated by a *random* instantiation of the *dependent click model* (DCM) [65] (see Section 3.3).

Bias of PM

In terms of bias, still answering RQ5, we see in Figure 4.8 that PM is on par with TDM. Both methods only need about 100 impressions from a random user to conclude that no preferences between rankers should be inferred. Again naturally, PI needs many more query impressions to draw the same conclusion because it needs to compare all pairs of rankers. It simply takes many comparisons to have done so reliably. We conclude that PM is as unbiased as TDM, irrespective of the number of samples.

Scaling the Number of Rankers with PM

Lastly, still answering RQ5, we investigate what happens when the number of rankers $|\mathcal{R}|$ that are being compared increases from the five rankers used until now. We test this with $|\mathcal{R}| = 20$ and find that after 500 navigational query impressions for PI the error $E_{bin} = 0.56$, for TDM this is $E_{bin} = 0.15$, and for PM($n = 10^4$) we find $E_{bin} = 0.13$. The advantage of multileaving over interleaving is clearly shown by these numbers. Both multileaving methods have a much lower error. Moreover, PM clearly outperforms TDM when the number of rankers increases and when the sample size for PM is large enough. We confirm a finding from Schuth et al. [163] who showed this to be an inherent disadvantage of TDM as it needs to represent all rankers with teams in the multileaving. PM, because it marginalizes over all possible team assignments, does not have this drawback and still performs well when the number of rankers goes up. For this reason, PM should be preferred over TDM.

4.7 Discussion off K-Armed Dueling Bandits

Multileaved comparison methods could form the basis of a new approach to tackling a generalization of the K-armed dueling bandit problem, in which the best ranker among a set is sought (see Section 2.3.4). By measuring the uncertainty associated with each $\bar{P}_{ij}$,

Algorithm 8 *Multileave RUCB* (MRUCB), for details see Zoghi et al. [214, Algorithm 1].

1: $\mathbf{W} \leftarrow \mathbf{0_{KxK}}$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad \mathbf{U} \leftarrow \frac{\mathbf{W}}{\mathbf{W}+\mathbf{W^T}} + \sqrt{\frac{\alpha \ln \mathbf{t}}{\mathbf{W}+\mathbf{W^T}}}$
4: $\quad$ Select S_0 uniformly randomly s.t. $\mathbf{U_{S_0,j}} \geq \frac{\mathbf{1}}{\mathbf{2}}$ for all j
5: $\quad$ **for** $i = 1, \ldots, m$ **do**
6: $\quad\quad S_i \leftarrow \arg\max_j \sum_{n=0}^{i-1} \mathbf{U_{j,S_n}}$
7: $\quad$ Compare rankers $R_{S_0} \ldots, R_{S_m}$ using a multileaving method
8: $\quad$ Update $\mathbf{W}$ to reflect all pairwise wins from the multileaved comparison

such a method could gradually exclude rankers from the multileaving that are deemed unlikely to be the best, thereby homing in on the most promising rankers. In this section, we take few steps towards addressing this problem.

To formalize the task of finding the best ranker among a set of rankers $\mathcal{R}$, we take the same ground truth as defined earlier in Section 4.3. Except now, following Zoghi et al. [216], we assume a *Condorcet winner* [194]. This Condorcet winner is a ranker which, without loss of generality, we label R_1. Then, $P_{1,j} > \frac{1}{2}$ for all $j > 1$ meaning that in expectation R_1 wins from all other rankers. We generalize the notion of regret by Zoghi et al. [216]. We define regret of comparing a set of rankers S to each other as

$$\frac{\sum_{j \in S} P_{1,j}}{|S|} + \frac{1}{2}.$$

This is the average sub-optimality of S, the set of rankers that is compared. This regret will go down to zero as soon as S consists of just the Condorcet winner. If interleaving methods are used for comparisons, at each comparison, S will consist of a pair of rankers, as in the original *dueling* bandit problem [216]. However, multileaving methods allow for S to consist of more than two rankers at a time. Cumulative regret at time T is defined as the sum of regret, as defined above, over the first T time steps.

The aim of an algorithm that solves this problem is to minimize such cumulative regret. It can do so by controlling which rankers are chosen to form S at each time step.

K-armed dueling bandit algorithms will have to be adapted to deal with 1) potentially selecting more than two rankers at a time for a comparison; and 2) with using the outcome of a multileaving comparison instead of an interleaving comparison. It is natural to extend the *relative upper confidence bound* (RUCB) algorithm by Zoghi et al. [214]. We do so in Algorithm 8 where we introduce *multileave RUCB* (MRUCB). On line 5 and 6 the algorithm selects m rankers greedily by comparing all upper bounds pairwise. When $m = 2$, this is equal to what RUCB does. Then on line 7, all these rankers are compared using a multileaving method. Again, when $m = 2$ this is equivalent to using an interleaving method and thus to RUCB. We leave both experimental and theoretical validation of this method to future work.

4.8 Conclusion

In this chapter, we presented a new paradigm for online evaluation of information retrieval systems. We have shown that it is possible to extend interleaved comparison methods to variants that, instead of comparing two rankers, compare multiple rankers at a time.

We introduced three implementations of this paradigm that extend state-of-the-art interleaving methods to their multileaving counterparts.

The first such method is *team draft multileave* (TDM) and is an extension of *team draft interleave* (TDI). Documents in the combined ranking shown to users are assigned to teams, so that when clicked, credit can be assigned to rankers.

The second is *optimized multileave* (OM) and extends *optimized interleave* (OI). We have shown in extensive experiments that both multileaving methods have their merits. OM learns preferences between rankers very quickly while TDM learns them slightly more slowly, though faster than either of the interleaving methods. However, TDM learns more accurate preferences in the long run than OM or either of the interleaving methods to which we compare. On the other hand, OM scales much better than TDM when the number of rankers increases. Thus, depending on the number of rankers to be compared, one might prefer one multileaving algorithm over the other but both should be preferred over interleaving algorithms when more than two rankers are to be compared.

Lastly, *probabilistic multileave* (PM) extends *probabilistic interleave* (PI) such that it can compare more than two rankers at once, while keeping PI's characteristic of being able to reuse historical interaction data. We empirically compared PM to PI as well as TDM. PM infers preferences between rankers by marginalizing over a *sample* of possible team assignments. We use a sample of controlled size to keep the computation tractable and show experimentally that given a large enough sample, our method is both as sensitive and as unbiased as TDM and more so than PI. That is, PM is capable of quickly finding meaningful differences between rankers and it does not infer preferences where it should not. An important implication of the introduction of PM is that historical interactions with multileaved comparisons can be reused, allowing for ranker comparisons that need much less user interaction data. Furthermore, we show that our method, as opposed to earlier sensitive multileaving methods, scales well when the number of rankers increases.

Finally, we are interested in integrating multileaving methods into learning methods analogous to the *dueling bandit gradient descent* (DBGD) method [209]. We address this in Part II of this thesis.

4.9 Future Work

As to future work, currently, in TDM, when documents belonging to the team of a ranker are clicked, preferences for this ranker over other rankers without clicks are inferred, even when those other rankers are not even represented by a team in the multileaving. This may happen when the number of rankers to be compared is larger than the number of documents in the multileaving. We aim to develop a variant of TDM that avoids this problem.

Another future direction is to customize TDM and OM to tasks other than comparing all rankers in a set to each other. When comparing all rankers to a production ranker, as

we do in Section 4.6.1, the definitions of unbiasedness and sensitivity could be adjusted to take into account the restricted goal of this task variant.

Proper evaluation of how multileaving methods could be applied in a K-armed dueling bandits problem setting, as quickly touched on in Section 4.7, is needed but left for future work.

5 Predicting A/B Testing with Interleaved Comparisons

The gold standard for *information retrieval* (IR) system evaluation is user satisfaction. However, as online user satisfaction is not directly observable, a significant amount of research has investigated how to summarize online behavior (such as clicks) into online metrics that best reflect user satisfaction.

Given a metric, the most common online evaluation methodology is *A/B testing*. Users of an online system are assigned to either a control or experimental condition, with the metric being computed on both populations. However, large numbers of users are typically necessary to obtain reliable results as this approach has high variance. Interleaved evaluation is an alternative online approach previously shown to be much more sensitive. Here, each user is presented a combination of results from both the control and treatment systems. However, until now interleaved evaluation has not modeled user satisfaction as reliably as recent A/B metrics, resulting in low agreement with recent A/B metrics given realistic differences in IR system effectiveness.

In this chapter we present an improvement to interleaving, showing how it can be optimized to maximize its agreement with any given A/B metric. Our results using 38 large-scale online experiments encompassing over 3 billion clicks in a commercial web search setting demonstrate substantial improvements in agreement.

This chapter is based on Schuth, Hofmann, and Radlinski [166].

5.1 Introduction

Evaluation has long played a key role in information retrieval. Traditionally, systems were often evaluated following the Cranfield approach [155]; see Section 2.2.1. Using this approach, systems are evaluated in terms of document relevance for given queries, which is assessed by trained experts. While the Cranfield paradigm ensures high internal validity and repeatability of experiments, it has been shown that the users' search success and satisfaction with an IR system are not always accurately reflected by standard IR metrics [187, 193]. One reason is that the relevance judges typically do not assess queries and documents that reflect their own information needs, and have to make assumptions about relevance from an assumed users point of view. Because the true information need can be difficult to assess, this can cause substantial biases [75, 192, 206].

To address the gap between offline evaluation and true use of IR systems, *online* evaluation has been used to directly measure observable user behavior on alternative systems; see Section 2.3. The biggest challenge for online evaluation is to identify metrics that accurately reflect user satisfaction. This has motivated a large amount of research on online metrics. While early online evaluation focused on simple metrics such as click-through rate (*click through rate* (CTR), the fraction of queries for which users click a result) or the ranks of clicked documents [93], more sophisticated metrics have been recently developed. These include observing which results users skip over [200], the time between search engine visits [52], and focusing on "satisfied" (long-duration) clicks (which we refer to as SAT clicks) [56].

Given an IR system and an appropriate online metric, the standard experimental procedure for comparing systems is *A/B testing* [113]; see Section 2.3.2. This means that a controlled experiment is conducted on users of a running system. A random sample of users is exposed to the treatment system, a second sample is exposed to the control system. Given that the assignment to systems is random and the experimental units (e.g., users) can be assumed independent of each other, any differences in online performance measured on the two samples can be attributed to differences between treatment and control system. If the measured differences are statistically significant, we can make highly confident decisions on which system to deploy. Unfortunately, the variance in user behavior is typically high, which results in low sensitivity of such A/B tests. This means that, to reach high confidence levels, large samples need to be collected over a long period of time (e.g., millions of samples) [34]. Considering the effects of exposing users to potentially lower quality systems over long periods of time, it can be seen that A/B tests can be extremely expensive.

An alternative online evaluation approach, interleaving, was developed to improve on the A/B design [93] (see Section 2.3.3). It avoids many of the sources of variance by combining results from both the treatment and control systems, for all queries. In particular, the results returned by the two systems are combined in a way that is fair to both, in the sense that neither system would be preferred in expectation if users were to click on documents at random. Observed user clicks on the combined result list are then credited to one of the systems to infer which system would be preferred by the user [144]. In comparison to A/B tests, interleaved comparisons have been shown to be substantially more sensitive. For instance, in an empirical comparison of five A/B tests and corresponding interleaving experiments, Chapelle et al. [34] observed that A/B tests required 145 times more data than interleaving to achieve statistical significance.

While high sensitivity makes interleaving very attractive for online evaluation, existing methods have primarily focused on observed clicks, and ignore the richer user satisfaction signals that have been incorporated into A/B metrics. As a result, it was unclear to what degree interleaved comparisons agree with user satisfaction, as measured by specifically designed A/B tests. This is challenging to address because by their very nature interleaving methods change rankings and attribute credit in a non straightforward way, making it far from trivial to align them with A/B metrics. The research in this chapter is the first to address this limitation of interleaving methods.

We present theoretical and experimental results that aim to answer the following research questions, repeated from Section 1.1.

RQ6 How do A/B metrics compare to interleaving in terms of sensitivity and agreement?

RQ7 Can A/B metrics and interleaving be made to agree better without loosing sensivity?

We make the following contributions in this chapter.

Sensitivity Starting with existing A/B and interleaving metrics (defined in Section 5.3), we propose a new statistical method for assessing the sensitivity of these metrics from estimated effect sizes (Section 5.4). The resulting method allows a detailed comparison between metrics in terms of the power of statistical tests at varying sample sizes. Our analysis shows that A/B tests typically require two orders of magnitude more data than interleaved comparisons.

Agreement Turning to the agreement between existing metrics, we find that current interleaved comparisons achieve from random up to 76% agreement with A/B user satisfaction metrics.

Credit Formulation Motivated by the results of our analysis, we propose novel interleaving *credit functions* that are (a) designed to closely match the implementation and parameters of A/B metrics, or (b) are parameterized to allow optimization towards agreement with arbitrary A/B metrics (Section 5.5). We further propose the first approach for automatically maximizing agreement between such parameterized interleaving credit functions and A/B metrics.

Optimization We demonstrate that interleaving credit functions can be automatically optimized, and that learned parameters generalize to unseen experiments. These results demonstrate for the first time that interleaving can be augmented with user satisfaction metrics, to accurately predict the outcomes of A/B tests that would require one to two orders of magnitude more data.

Large Scale Evaluation Finally, our empirical results, obtained from experiments with 3 billion user impressions and 38 paired (A/B and interleaving) experiments demonstrate the effectiveness of our proposed approach (Section 5.6). In particular, we achieve agreement of up to 87%, while maintaining high sensitivity.

We have incorporated the above contributions in Section 1.2 where we give a complete overview of all contributions of this thesis.

We now present related work to the extend that we have not presented that in Chapter 2 yet, followed by a detailed background on A/B metrics and interleaving in Section 5.3. This leads to an analysis of the power of different approaches in Section 5.4, followed by the details of our method in Section 5.5. Section 5.6 presents our results, followed by conclusions.

5.2 Related Work

We discussed prior work on measuring user satisfaction with online IR systems in Section 2.3. Relevant for this chapter is the background on A/B testing and absolute relevance

metrics: metrics that measure a single number for a given ranking system (see Section 2.3.2). Also relevant is Section 2.3.3 in which we presented online evaluation using interleaving.

Finally, below we discuss approaches for optimizing online evaluation metrics as relevant to this chapter (see Section 5.2.1).

5.2.1 Optimizing Interleaving Metrics

The interleaving approaches described above measure which ranker is more likely to attract user clicks in a fair, paired comparison. However, as described in Section 2.3.2, raw clicks can be misleading. While the pairwise nature of interleaving removes position bias and leads to much more reliable comparisons between systems [34, 144], other effects must also be considered. Previous research has shown that with interleaving there may be biases due to highlighting in search result titles [211] and other caption effects such as title and snippet length [80]. Proposed methods to mediate these biases were shown to improve agreement with offline evaluation [80, 211], but optimizing agreement with online metrics remains an open challenge.

Also, the above approaches may improve interleaving by removing some click bias, they still aim to be *unbiased* rather than *predictive of satisfaction*. In this chapter, we show how to create an interleaving evaluation that instead aims to predict the outcome of an A/B test for any given A/B metric, while maintaining the sensitivity improvements of interleaving. In particular, we take into account whether clicks are indicators of success by reimplementing the classifier learned by Kim et al. [110].

The goal of this chapter is also related to prior work on optimizing the sensitivity of interleaving algorithms, where interleaving algorithms were learned to be more statistically powerful [210], or to satisfy given choices about the value of any given preference observed [143]. In contrast, our work is the first that focuses on optimizing "correctness" of an interleaving outcome as captured in terms of agreement with A/B metrics, while maintaining high sensitivity. Our results show that in this way agreement between interleaving and any given A/B metric can be dramatically improved.

5.3 Background

In this section we describe—as far as they have not yet been covered in Section 2.3—the most commonly used A/B metrics, and the interleaved evaluation approach that we build on in the remainder of this chapter. We take the presented A/B metrics as the *ground truth* user satisfaction metrics we aim to predict with much smaller interleaving samples.

5.3.1 Common A/B Metrics

As described in Section 2.3.2, a large number of A/B metrics have been developed. Most have in common that clicks are the basic observed interaction with users. Thus this is our focus too. We note that many common A/B metrics can be categorized as taking into account particular attributes of clicking behavior. The most common attributes include

(1) estimating clicks as indicative of satisfaction or not, (2) giving particular importance to clicks at the top position of Web search results, and (3) measuring the time spent by the user prior to clicking. Consequently, we implement the following A/B metrics. An overview is given in Table 5.1.

Click-through Rate

Click through rate (CTR) is often used as a baseline A/B metric, e.g., by Chapelle et al. [34]. It can be implemented as the average number of clicks per search result page, or as the fraction of pages for which there are any clicks. We follow the second definition. In Table 5.1, we use $|C^q|$ to denote the number of clicks for query q.

Click Rank

It was noted by Chapelle et al. [34] that of all the A/B metrics studied in a large scale comparison of A/B tests and interleaving evaluation, the A/B metric that most reliably agreed with known experimental outcomes was the fraction of search results pages with a click at the top position. As such, we also use two types of metrics: those which only consider clicks at the top position (named *@1*) and those that consider all clicks (the others). In equations and in Table 5.1 we use $\mathrm{rank}_A(c)$ to denote the rank of click c in the results returned by ranker A.

User Satisfaction

While clicks are often directly interpreted as a user preference, they are known to be both noisy and biased. To remedy the noise, a common approach is to only consider *satisfied* (SAT) clicks with dwell time above a fixed cutoff of 30 seconds [206].

However, only using time as a threshold for satisfaction is problematic as some queries naturally require users to spend more time than others. Recently, Kim et al. [110] showed that taking more user signals into account leads to better prediction of user satisfaction. For this chapter, we partially re-implement the SAT click classifier from that work. Our classifier uses the dwell time, document readability, document topic and query topic features suggested by Kim et al. [110]. In particular, the features beyond dwell time are assumed to partially explain the dwell time necessary for a given query and document. We combine these features using quantile regression forests [131]. The model is trained to predict the probability of a SAT click, given user signals. It can be turned into a classifier by selecting a decision threshold, e.g., based on the distribution over classes in the training set. With training on approximately 3,000 manually labeled clicks, our classifier obtains an accuracy of 77%, which is marginally lower than the 81% reported by Kim et al. [110]. The major difference between the implementations is that we do not represent dwell time distributions per topic. Instead, we use raw dwell time values directly as input for our classifier.

The output of this SAT click classifier is used throughout this chapter. For a given click c, we define $\mathrm{sat}(c)$ as the estimated probability that c indicates user satisfaction. For succinctness, we also define $\mathrm{is\_sat}(c) := true$ whenever $sat(c) > 0.8$ (the threshold based on the class distribution). Half of the A/B metrics we consider use the $\mathrm{is\_sat}(c)$

Table 5.1: A/B metrics implemented as ground truth for comparisons with interleaving. See Section 5.3.1 for notation. Metrics marked with the symbol † ignore all queries for which there was no click of the required type. $|C^q|$ denotes the number of clicks for query q, $\mathrm{rank}(c)$ denotes the position of the click. The indicator function $\mathbf{1}(\cdot)$ is used and evaluates to 1 when the argument is true, and 0 otherwise.

A/B Metric	Description	Implementation $\frac{1}{Q_A}\sum_{q\in Q_A}\ldots$
AB	Number of queries that received at least one click.	$\mathbf{1}(\lvert C^q\rvert > 1)$
AB@1	Number of queries that received at least one click on the first position.	$\mathbf{1}((\sum_{c\in C^q}\mathbf{1}(\mathrm{rank}(c)=1)) > 1)$
AB_S	Number of queries that received at least one SAT click.	$\mathbf{1}((\sum_{c\in C^q}\mathbf{1}(\mathrm{is\_sat}(c))) > 1)$
AB_S@1	Number of queries that received at least one SAT click on the first position.	$\mathbf{1}((\sum_{c\in C^q}\mathbf{1}(\mathrm{rank}(c)=1)\cdot\mathbf{1}(\mathrm{is\_sat}(c))) > 1)$
AB_T †	Time from the query being issued until the first click.	$\min_{c\in C^q}\mathrm{time}(c)$
AB_T@1 †	Time to the first click on the top position.	$\min_{c\in C^q}\mathrm{time}(c)\cdot\mathbf{1}(\mathrm{rank}(c)=1)$
$AB_{T,S}$ †	Time to the first click classified as SAT.	$\min_{c\in C^q}\mathrm{time}(c)\cdot\mathbf{1}(\mathrm{is\_sat}(c))$
$AB_{T,S}$@1 †	Time to the first click on the top position classified as SAT.	$\min_{c\in C^q}\mathrm{time}(c)\cdot\mathbf{1}(\mathrm{rank}(c)=1)\cdot\mathbf{1}(\mathrm{is\_sat}(c))$

signal to filter out clicks c that are not deemed satisfied by our classifier. These A/B metrics are marked with subscript S.

Time to Click

Another commonly used metric is the time that the user spends on the search result page before clicking a document. As time spent is the key cost to search system users, reducing this time is considered good (e.g. [34]). Our metrics that measure time to click are marked with subscript T. In equations we use $\mathrm{time}(c)$ to denote the time from the user issuing the query until the click c.

Combining all possible choices of A/B metrics leads to the eight metrics shown in Table 5.1. The first four (AB, AB@1, AB_S, AB_S@1) focus on the presence of a click, while the other four capture the time to the first click of a particular type, if such a click occurred (AB_T, AB_T@1, $AB_{T,S}$ and $AB_{T,S}$@1).

5.3.2 Interleaving

In this chapter, we use *team draft interleave* (TDI) [144] as our interleaving baseline. This algorithm is most frequently used in practice, and has been empirically shown to be equally

effective as *balanced interleave* (BI) [34, 93]. We have detailed TDI in Section 2.3.3 and extended upon TDI in Chapter 4. Our focus is on replacing the credit function; replacing the right hand side of Equation 2.1. We introduce our methods in Section 5.5.

5.4 Data Analysis

Many of the common A/B metrics that we introduced in Section 5.3.1 have been developed recently. Therefore, it is not clear to what degree interleaved comparisons agree with these metrics. In this section, we conduct an empirical analysis of the sensitivity and directional agreement between these A/B metrics and TDI. We start by describing the data we use in this section and in the remainder of this chapter (5.4.1). We then propose a new approach for comparing the sensitivity (in terms of statistical power) of A/B and TDI comparisons, and use this method to analyze the relative power of the different approaches (5.4.2). This also lets us estimate the probability of agreement between approaches at varying sample sizes. The results of our analysis are presented in Section 5.4.3. They motivate why an improved approach is needed, as discussed in Section 5.4.4.

5.4.1 Data

For our experiments, we start with a set of 38 pairs of rankers for which both an A/B comparison and a TDI interleaving comparison were performed. Our data consists of records of users interacting with a web search engine, Bing. All ranker comparisons reflect changes that are typical for the normal development of a commercial web search engine. They consist of changes to the ranking function used to order web search results, in terms of parameters of the ranking function, modified ranking features, and so forth. The comparisons were all run in the first 9 months of 2014, in the United States locale. We only considered traffic on `bing.com`, with *en-US* as language, only the web vertical, only the first result page, only external traffic and we filtered out bots. The experimental unit consisted of assigning users to individual ranking conditions uniformly at random.

The A/B and interleaving comparisons were run for varying durations, usually around one week for A/B comparisons and around 4 days for interleaving comparisons. Additionally, A/B comparisons were typically run with higher volume, resulting in about 80 times more queries for each A/B comparison than each interleaving comparison. In all, this data consists of over 3 billion clicks. Depending on the experiment, between 2% and 30% of interleaved queries with clicks had at least one click on a result assigned to one of the teams.

5.4.2 Estimating Power and Agreement

We now propose an approach for assessing the relative power of A/B measurement compared to TDI, and further show how this approach can be used to estimate agreement between approaches at varying sample sizes.

As described earlier, A/B tests perform controlled experiments. Users are exposed to either treatment or control result rankings, rendering this a *between subject* experiment. In interleaving, each user is exposed to results from both rankers, rendering them

within subject experiments. We can measure the importance of this difference using a power computation, which tells us how many independent samples we need to obtain a statistically significant outcome for each approach, as follows.

We start with A/B comparisons, following the standard methodology described in [113]: Two independent samples are collected by exposing a random fraction of users to treatment A, and another to treatment B. An A/B metric is used to assess each sample, and we are interested in determining whether there is a statistically significant difference between A and B in terms of this metric. This question is typically addressed using a two-sample t-test.

Power of A/B Comparisons

Assume that the target metric is approximately normally distributed (this is reasonable due to the central limit theorem), with means μ_A and μ_B and equal variance σ_{AB}. Formally, we have $A \sim \mathcal{N}(\mu_A, \sigma_{AB})$ and $B \sim \mathcal{N}(\mu_B, \sigma_{AB})$. We are interested in detecting whether $\mu_A \stackrel{?}{=} \mu_B$. This gives us the null hypothesis $H_0 : \mu_A = \mu_B$ and the alternative hypothesis $H_1 : \mu_A \neq \mu_B$. We also choose the probability of a Type-I error that we are willing to accept, e.g., $\alpha = 0.05$. The t-test then assumes that H_0 is true and assesses the probability of observing differences of at least the observed sample difference $|\hat{A} - \hat{B}|$ under H_0.

While Type-I errors are controlled in the significance test, here we are interested in the *power* (also called sensitivity) of the conducted test. Assuming H_1 is actually true, power quantifies the probability of correctly rejecting H_0. It is affected by the true effect size

$$\delta_{AB} = (\mu_A - \mu_B)/\sqrt{1/n_A + 1/n_B}\sigma, \tag{5.1}$$

where n_A, n_B are the respective sample sizes.

We can assess the power of a test as follows. Under H_1 (samples are drawn from normal distributions with means $\mu_A \neq \mu_B$ and shared variance σ_{AB}) we observe sample A, B and compute the test statistic [118]

$$t(A, B) = \frac{(\bar{A} - \bar{B})}{\sqrt{\frac{1}{n_A} + \frac{1}{n_B}}} / \sqrt{\frac{\sum(A_i - \bar{A})^2 + \sum(B_j - \bar{B})^2}{v_{AB}}}. \tag{5.2}$$

The test statistic $t(A, B)$ follows a non-central t distribution $\bar{A} - \bar{B} \sim nct(\delta_{AB}, v_{AB})$, with non-centrality parameter δ_{AB} (the effect size) and degrees of freedom $v_{AB} = n_A + n_B - 2$.

H_0 is correctly rejected when $|t(A, B)| \geq C_0$. Where C_0 is the critical value for the test statistic that corresponds to the chosen α level. The power of the test is the probability $P(reject(H_0)|H_1) = P(|t(A, B)| \geq C_0)$ and can be computed (solved using linear programming[1])

$$P(|t(A, B)| \geq C_0) = \int_{C_0}^{\infty} nct(\delta_{AB}, v_{AB})dy. \tag{5.3}$$

[1] We use the python implementation `statsmodels.stats.power.tt_ind_solve_power`, `http://statsmodels.sourceforge.net`

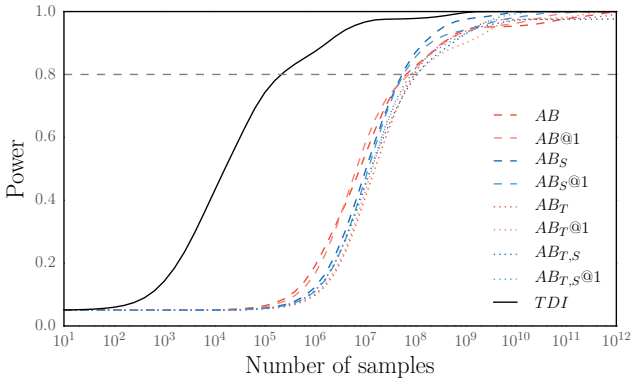

Figure 5.1: Power as a function of sample size, computed using the observed effect sizes for 38 interleaving and A/B comparisons, averaged over all comparisons (assuming two-sided t-test with $\alpha = 0.05$, as described in Section 5.4.2).

Power of Interleaving Comparisons

The analysis for interleaving is closely related, but relies on the typically more powerful (one-sample) paired t-test. Instead of independent samples, we now observe a single sample I of paired comparisons, assumed to be normally distributed with $I \sim \mathcal{N}(\mu_I, \sigma_I)$. We want to detect whether $\mu_I = 0$ with $H_0 : \mu_I = 0$ and $H_1 : \mu_I \neq 0$. Given μ_I and sample size n_I, the test statistic $t(I)$ follows a non-central t-distribution $t(I) \sim nct(\delta_I, v_I)$ with non-centrality parameter $\delta_I = \sqrt{n\mu_I/\sigma_I}$ and $v_I = n_I - 1$ degrees of freedom. The power calculation is[2]

$$P(|t(I)| \geq C_0) = \int_{C_0}^{\infty} nct(\delta_I, v_I)dy. \tag{5.4}$$

Probability of Agreement

Given Equations 5.3 and 5.4, we can compute the probability of comparison outcomes at varying sample sizes. For example, the probability that an A/B comparison with parameters $\mu_A, \mu_B, \sigma_{AB} : \mu_A - \mu_B > 0$ agrees with the true A/B outcome at sample sizes n_A, n_B is computed by plugging into Equation 5.3 and computing $P(\bar{A} - \bar{B} > 0) = P(t(A, B) > 0)$. Correspondingly, the probability that an interleaving comparison with parameters μ_I, σ_I would agree with the true A/B outcome is computed using Equation 5.4 so that $P(\bar{I} > 0) = P(t(I) > 0)$.

5.4.3 Data Analysis Results

In this section we answer RQ6. We apply the analysis methodology described above to the set of 38 ranking algorithms described in Section 5.4.1. In Figure 5.1 we show

[2]We use the python implementation `statsmodels.stats.power.tt_solve_power`.

Table 5.2: Agreement of A/B metrics on our data. We measure agreement with TDI, with a sub-sample of A/B of the same size as the TDI comparison, and with a sub-sample of A/B with the same size as the original A/B comparison (this is an upper-bound on agreement for each A/B metric). Values in bold are statically significantly different from 50%.

		self-agreement	
A/B Metric	TDI	AB_{Sub}	AB_{Up}
AB	0.63	0.63	**0.94**
AB@1	**0.71**	0.62	**0.95**
AB_S	**0.71**	0.61	**0.96**
AB_S@1	**0.76**	0.60	**0.95**
AB_T	0.53	0.58	**0.91**
AB_T@1	0.45	0.59	**0.90**
$AB_{T,S}$	0.47	0.59	**0.88**
$AB_{T,S}$@1	0.42	0.60	**0.87**

the power obtained using A/B comparisons and interleaving comparisons at increasing sample sizes. We see that on average across the set of 38 experiments, 80% power[3] with A/B experiments is obtained with between 10^7 and 10^8 observations (queries). On the other hand, the same power is obtained with between 10^5 and 10^6 observations with TDI. This difference of approximately two orders of magnitude is consistent with previous work [34].

Having presented the relative power of the approaches, we return to the key question: Do the metrics agree on which ranker is better? We use the method developed in the previous section to estimate agreement between A/B and interleaving comparisons, and A/B comparisons at varying sample sizes. The first column in Table 5.2 summarizes the agreement rates of TDI and A/B metrics. We see that agreement rates are generally low. These observed levels of agreement of TDI with A/B metrics are a *baseline* that we want to improve upon in this chapter. In the results in Section 5.6, this baseline is referred to as simply TDI.

Recall that our goal is to predict the outcome of a large A/B comparison given the much smaller amount of data used in interleaving comparisons. As such, another reasonable baseline is to assess how well a smaller A/B comparison predicts the outcome of the full A/B comparison. We can answer this question using the methodology for assessing the probability of agreement developed above (Section 5.4.2), by plugging in the observed effect sizes and setting the sample size to that of the corresponding interleaving comparison. This is our second *baseline*, which we refer to as AB_{Sub}. Results are given in the second column of Table 5.2. Generally, these A/B metrics computed on small subsamples have low agreement with the experiment outcome given the complete data, of around 60%, never statistically significantly different from 50%.

We also compute an upper bound of agreement with A/B metrics by actually measuring how well a subsample of the same size as the full A/B, AB_{Up}, would agree with itself,

[3]Controlled experiments are typically designed for 80–95% power.

using the same methodology. This can be seen as a measure of how predictable a metric is. The last column of Table 5.2 shows that the time-based metrics are much less predictable than count-based A/B metrics. Given these lower and upper bounds on A/B agreement with itself, we can restate our goal as follows. In this chapter, we aim to augment TDI to improve over the above two baselines (which we introduced above and refer to as TDI and AB_{Sub}) and to close the gap with the upper bound AB_{Up}.

5.4.4 Implications

Summarizing the results above motivates the rest of the work in this chapter: A/B metrics have been developed guided by real analysis of user behavior. Yet they usually have relatively low power. Interleaving has much higher power, but low agreement with most A/B metrics, being blind to richer behavioral signals. Thus, we aim to optimize interleaving to increase agreement with A/B metrics, while maintaining the statistical power of the technique. The A/B metric is treated as the gold standard to which interleaving must be compared.

As noted in the Section 5.2, this goal is similar to that addressed by [210]. However, they only focused on optimizing sensitivity, while we focus on optimizing correctness in the sense of agreeing with A/B metrics. It is similar to [80, 211] in the sense that our method can reduce click bias in interleaved comparisons. However, the earlier work only considered agreement with offline metrics.

5.5 Methods

In this section we describe how to incorporate user signals into TDI comparisons, to increase agreement with A/B metrics. We first formalize the notion of interleaving credit in a way that allows us to incorporate user signals (see Section 5.5.1). We then design a set of credit functions that closely match user satisfaction A/B metrics (see Section 5.5.2). Because agreement between interleaving and A/B metrics is not necessarily maximized by mirroring A/B parameters, we then introduce parameterized credit functions (see Section 5.5.3), and combined credit functions (see Section 5.5.4) designed to be automatically tuned to maximize agreement. Finally, our methodology for maximizing agreement is detailed in Section 5.5.5.

5.5.1 Formalizing Interleaving Credit

Formally, for all pairs of rankers A and B, we aim to find an interleaving method that agrees with the *sign* of the differences in A/B metrics that we found in an A/B comparison. The sign of such a difference should be interpreted as a preference for either A, B, or neither. We denote such a preference, the comparison outcome O_{AB}, of the metric AB as:

$$O_{AB}(A, B) = \text{sgn}(AB(A) - AB(B)). \tag{5.5}$$

Instances of A/B metrics are *click through rate*, *clicks at one*, and *time to click* (see Table 5.1 for details).

As opposed to A/B metrics, interleaving methods are directly defined on pairs of rankers. Following the same notation, the outcome of an interleaving comparison with TDI can thus be denoted as:

$$O_{TDI}(A, B) = \text{sgn}(TDI(A, B)). \tag{5.6}$$

Interleaving preferences, when using TDI (cf., Section 5.3.2), come from differences between credit acquired by each ranker:

$$TDI(A, B) = \frac{1}{|Q|} \sum_{q \in Q} \delta(C_A^q) - \delta(C_B^q). \tag{5.7}$$

Here, Q is a set of query impressions and $\delta(C_A)$ a credit function (see the next Section 5.5.2 for instances of this function) that attributes credit to ranker A depending on user interactions with the result list. Next, we introduce a new set of credit functions that is designed to mirror the use of user signals in A/B comparisons.

5.5.2 Matching A/B Credit

We now present instantiations of credit functions $\delta(C_A)$ designed to match the A/B metrics in Section 5.3.1. All our interleaving credit functions are defined on a set of clicks assigned to a ranker (e.g., for ranker A these are $c \in C_A$), for a query impression. Clicks are associated with user signals.

The details of our *matching* credit functions are given in the first part of Table 5.3. The following signals are used:

- $|C_A|$ the number of clicks for ranker A for a query impression. See Section 5.3.1.
- $\text{rank}_A(c)$ is the rank of the clicked document in the original ranking A (before interleaving: i.e., rankers A and B can have different documents at rank 1). See Section 5.3.1 for a description of this signal as used in A/B metrics.
- $\text{is\_sat}(c)$ is a binary indicator that is *true* if the SAT classifier identified the click as SAT click. See Section 5.3.1 for details on the SAT click classifier.
- $\text{sat}(c)$ is the probability of the click being a SAT click. Again, details are in Section 5.3.1.
- $\text{time}(c)$ is the time from query submission to the observed click, in seconds. See Section 5.3.1 for the corresponding A/B signal.

Previously proposed interleaved comparison methods, such as TDI, use the credit function TDI shown in the table. It can be interpreted as a close match to the A/B metric AB, because it estimates whether a given ranker would have obtained a click in an A/B comparison.

The credit functions $TDI@1$, TDI_S, and $TDI_S@1$ are designed to closely match the A/B metrics $AB@1$, AB_S, and $AB_S@1$. For clicks at rank one, we consider whether a clicked document would have been placed first by the original ranker, as this reflects the most accurately whether the ranker would be likely to receive a click at the top rank in an A/B comparison. For SAT clicks, we use the same classifier as for our A/B comparisons above, as in Section 5.3.1.

The four time-based credit functions TDI_T, $TDI_T@1$, $TDI_{T,S}$, and $TDI_{T,S}@1$ are designed to match the time-based A/B metrics. E.g., TDI_T matches the A/B metric AB_T. However, we use the average time to click for a ranker, as it tends to be more robust than the time to the first click. The remaining three metrics implement filters on which clicks contribute, parallel to the click-based metrics described above.

5.5.3 Parameterized Credit Functions

Next, we propose a second set of interleaving credit functions that can be parameterized to allow automatic calibration to maximize agreement with A/B metrics. Effectively calibrating these credit functions would allow users of interleaved comparisons to automatically identify credit functions that maximize agreement with arbitrary A/B metrics.

For instance, we define a credit function that captures user satisfaction. We filter out clicks c that have a low satisfaction probability $\mathrm{sat}(c)$ by thresholding this probability using a threshold t_s. This leads to the following credit function:

$$\delta(C_A)^{t_s} = \sum_{c \in C_A} \mathbf{1}(\mathrm{sat}(c) > t_s). \tag{5.8}$$

The threshold, t_s in this case, of such a parametrized credit function can be tuned to maximize agreement with A/B metrics. We define two such parameterized functions, the first click-based, as shown above, the second time-based. We list our parameterized credit functions in the second part of Table 5.3.

5.5.4 Combined Credit Functions

Now that we have several credit functions, as listed in the first two sections of Table 5.3, we can take it a step further and start combining them. We propose to combine the interleaving credit functions in a weighted linear combination:

$$TDI^{\mathbf{w}}(A, B) = \frac{1}{|Q|} \sum_{q \in Q} \sum_{w_i \in \mathbf{w}} w_i \delta_i(C_A^q) - w_i \delta_i(C_B^q), \tag{5.9}$$

where $\mathbf{w}$ denotes the weights used to combine several credit functions. We thus define the interleaving preference as a weighted sum of credit functions we introduced earlier.

In the original TDI, we have a single credit function as defined in the first row of Table 5.3 and a weight vector of $\mathbf{w} = (1)$.

5.5.5 Maximizing Agreement with A/B Metrics

We return to our initial goal, to optimize the agreement between interleaving metrics and A/B metrics, and present a method for automatically tuning interleaving credit functions to maximize agreement with a given A/B metric. Together with the parameterized and combined credit functions presented above, this allows us to tune interleaving to an arbitrary A/B metric. Our approach treats the A/B metric as a black box presented by an experimenter who presumably selected this metric for some reason.

Table 5.3: Definitions for interleaving credit functions. The $\delta(C_A)$ functions give credit to ranker A based on attributes of the clicked documents assigned to ranker A. The last row computes a combination of credit functions above it. To obtain credit for ranker B, the function can be called as $\delta(C_B)$. For details see Section 5.5.2.

Credit functions designed to match A/B metrics (cf., 5.5.2)		$\delta(C_A) =$
TDI	Number of clicks on ranker A	$\lvert C_A \rvert$
TDI@1	Number of clicks on documents that A ranks first	$\sum_{c \in C_A} \mathbf{1}(\text{rank}_A(c) = 1)$
TDI_S	Number of SAT clicked documents contributed by A	$\sum_{c \in C_A} \mathbf{1}(\text{is\_sat}(c))$
TDI_S@1	Number of SAT clicked document ranked first by A	$\sum_{c \in C_A} \mathbf{1}(\text{is\_sat}(c)) \cdot \mathbf{1}(\text{rank}_A(c) = 1)$
TDI_T	Time to clicks on documents contributed by A	$\sum_{c \in C_A} \text{time}(c)$
TDI_T@1	Time to clicks on documents ranked first by A	$\sum_{c \in C_A} \mathbf{1}(\text{rank}_A(c) = 1) \cdot \text{time}(c)$
$TDI_{T,S}$	Time to SAT clicks on documents contributed by A	$\sum_{c \in C_A} \mathbf{1}(\text{is\_sat}(c)) \cdot \text{time}(c)$
$TDI_{T,S}$@1	Time to SAT clicks on documents ranked first by A	$\sum_{c \in C_A} \mathbf{1}(\text{rank}_A(c) = 1) \cdot \mathbf{1}(\text{is\_sat}(c)) \cdot \text{time}(c)$
Parameterized credit functions (cf., 5.5.3)		$\delta(C_A) =$
$TDI_S^{t_s}$, $t_s \in \{0.1 \ldots 0.9\}$	Number of clicks with SAT probability $\geq t_s$, on documents contributed by ranker A	$\sum_{c \in C_A} \mathbf{1}(\text{sat}(c) \geq t_s)$
$TDI_{T,S}^{t_s}$, $t_s \in \{0.1 \ldots 0.9\}$	Time to clicks with SAT probability $\geq t_s$, on documents contributed by ranker A $\geq t_s$	$\sum_{c \in C_A} \mathbf{1}(\text{sat}(c) \geq t_s) \cdot \text{time}(c)$
Combined credit function (cf., 5.5.4)		$\delta(C_A) =$
$TDI_{T,S}^{\mathbf{w}}$, $w_i \in \{0.1 \ldots 0.9\}$	Weighted combination of the credit functions above	$\sum_{w_i \in \mathbf{w}} w_i \delta_i(C_A)$

Algorithm 9 Maximizing Agreement

Require: Ranker pairs $C = ((A_1, B_1), \ldots, (A_n, B_n))$, A/B metric AB

1: **Init**: test agreements $A \leftarrow []$, weights $W \leftarrow []$
2: **for all** $n \leq N$ **do** *// N repetitions*
3: $S \leftarrow sample\_with\_rep(C, |C|)$ *// bootstrap sample, train set*
4: $\hat{\mathbf{w}} \leftarrow \arg\max_{\mathbf{w}} \sum_{(A,B)\in S} \mathbf{1}(O^{\mathbf{w}}_{TDI}(A, B) = O_{AB}(A, B))$
5: $O \leftarrow C \setminus S$ *// 'out of bag' sample, test set*
6: $A \leftarrow A + \frac{1}{|O|} \sum_{(A,B)\in O} \mathbf{1}(O^{\hat{\mathbf{w}}}_{TDI}(A, B) = O_{AB}(A, B))$
7: $W \leftarrow W + \hat{\mathbf{w}}$ *// append weight vector*
8: **Output**: weights $mean(W)$, agreement $mean(A)$

The weights introduced in Equation 5.9 can be optimized such that we maximize the agreement of this interleaving outcome with a given A/B metric:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \sum_{(A,B)\in S} \mathbf{1}(O^{\mathbf{w}}_{TDI}(A, B) = O_{AB}(A, B)). \tag{5.10}$$

I.e., we maximize the number of times the outcome of an A/B comparison agrees with the outcome of an interleaving comparison for all of ranker comparisons S, in our case the 38 comparisons described in Section 5.4.1.

To implement and validate the maximization step in Equation 5.10, we use the bootstrap procedure presented in Algorithm 9. This takes as input a set C of pairs of rankers that have been compared (such as those described in described in Section 5.4.1) and an A/B metric such as AB. For N repetitions, a bootstrap sample S of size $|C|$ is taken from C. On this sample we compute $\hat{\mathbf{w}}$ using Equation 5.10 for all $\mathbf{w}$ we consider. We validate the agreement that this weight vector $\hat{\mathbf{w}}$ gives on unseen data and we report the mean $\hat{\mathbf{w}}$ and mean agreement. In our experiments $N = 100$ and we consider $\mathbf{w} = (w_1, \ldots, w_n)$, where $w_i \in \{0, 0.01 \ldots 1\}$.

We use the same procedure to optimize the parameters t_s of the parameterized credit functions described in Section 5.5.3. Instead of computing the argmax on line 4 over all $\mathbf{w}$, we compute the optimal $\hat{t_s}$:

$$\hat{t_s} = \arg\max_{t_s} \sum_{(A,B)\in S} \mathbf{1}(O^{t_s}_{TDI}(A, B) = O_{AB}(A, B)). \tag{5.11}$$

5.6 Experiments and Results

In Section 5.4, we examined the agreement between TDI and A/B metrics, and the sensitivity of both types of comparison methods. Depending on the A/B metric, agreement ranges from random up to 75%, while sensitivity of TDI is on average two orders of magnitude higher than that of A/B metrics. In this section we evaluate our new interleaving

Table 5.4: Agreement of *matching* interleaving credit functions (designed to match A/B metric parameters). Boldface indicates values significantly different from 0.5 (two-sided binomial test, $p = 0.05$). On the diagonal are metrics for which parameters are designed to match (gray background). Best agreement per A/B metric (row) is underlined.

	Interleaving Credit							
A/B Metric	TDI	TDI@1	TDI_S	TDI_S@1	TDI_T	TDI_T@1	$TDI_{T,S}$	$TDI_{T,S}$@1
AB	0.63	0.66	**0.84**	0.66	0.61	0.61	0.58	0.53
AB@1	**0.71**	**0.68**	**0.76**	0.63	0.63	0.47	0.55	0.55
AB_S	**0.71**	**0.68**	**0.87**	**0.68**	**0.68**	0.58	0.61	0.55
AB_S@1	**0.76**	**0.68**	**0.82**	0.63	**0.74**	0.53	0.61	0.50
AB_T	0.53	0.55	0.47	0.55	**0.71**	0.55	**0.68**	0.58
AB_T@1	0.45	0.47	0.45	0.58	0.63	0.58	0.61	0.61
$AB_{T,S}$	0.47	0.55	0.53	**0.71**	0.66	0.66	0.58	0.53
$AB_{T,S}$@1	0.42	0.50	0.53	0.66	0.61	0.66	0.58	0.58

credit functions. First, we analyze what level of agreement can be reached by matching interleaving credit functions with the parameters of A/B metrics. Second, we evaluate our parameterized credit functions, and our method for optimizing agreement with A/B metrics.

5.6.1 Matching A/B Credit

In our first set of experiments, we evaluate our *matching* credit functions. These are designed to match the parameters of the A/B metric that we wish to optimize, as explained in Section 5.5.2. For instance, for the target A/B metric AB_S we classify observed clicks on interleaving impressions using the same classifier used by the A/B comparison, and only assign interleaving credit for satisfied clicks. As we study 8 A/B metrics, this gives rise to 8 possible variants of TDI with matching credit functions.

Table 5.4 shows the agreement between each A/B metric and each variant of TDI. In the first column, we see the agreement between baseline TDI and each A/B metric, computing each as previously defined. The lowest agreement is observed between the original TDI and the A/B metric $AB_{T,S}$@1, at 42%. The highest agreement is observed between TDI_S and the A/B metric it is designed to optimize (AB_S), with 87%. Given the small sample of 38 comparisons, only the agreement rates above 68% are statistically significantly different from random agreement, and are shown in bold in the table. These compare favorably with typical inter-judge agreement rates in offline evaluations of around 65% [196], and with the bounds on A/B self-agreement AB_{Sub}, AB_{Up} in Table 5.2.

We note that using different credit functions often increases agreement between A/B metrics and TDI. However, interestingly the maximal agreement is often not seen when the A/B metric matches the credit function used for interleaving. This can be observed by comparing the metrics that match in terms of their parameters (indicated by the gray cells in Table 5.4), to the ones that achieved highest agreement (underlined). For example, agreement with AB_S is maximized by TDI_S, but agreement with $AB_{T,S}$ is maximized by

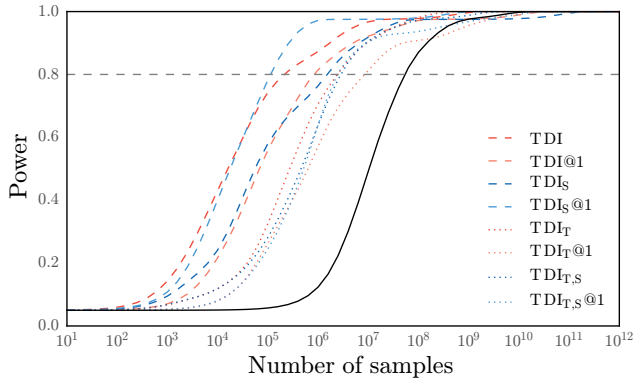

Figure 5.2: Power for TDI with matching credit functions (assuming two-sided t-test with $\alpha = 0.05$, as described in Section 5.4.2). The black line denotes AB_S, the A/B metric with most power.

TDI_S@1. A reason for this is the interplay between bias and noise. By more aggressively removing noise in the interleaving comparison (in this case, by only considering SAT clicks at the top position), we may increase agreement with related A/B metrics, even those for which there is bias due to a slight mismatch between the interleaving and A/B metric.

Our results show that agreement between interleaving and A/B comparisons can be substantially improved by matching interleaving credit parameters to those of the target A/B metrics. We also need to ensure that in doing so, we do not decrease the sensitivity of interleaving. Intuitively, removing observations (e.g., clicks beyond the first position) may reduce sensitivity. On the other hand, if the removed observations are noisy, the interleaving signal may actually become more discriminative, and sensitivity can be increased.

Figure 5.2 shows the power for TDI with replaced scoring functions. We see that TDI with matching credit functions typically has lower power than standard TDI. In particular, sensitivity decreases for time-based metrics, which may also explain the relatively lower agreement between time-based interleaving credit functions and A/B metrics. However, the power of these variants of TDI is still 1 to 2 orders of magnitude larger than the power of the A/B metric with the most power. Sensitivity is increased by TDI_S, the credit function that also shows highest agreement. This result indicates that focusing interleaving credit on low-noise clicks is a very promising way to achieve both high sensitivity and good agreement with user satisfaction metrics.

The results of the analysis in this section motivate the next set of questions. Given a target A/B metric, what is the best credit function that should be applied to TDI? Just as the correct credit function may not be the same as the target A/B metric, the parameters of the credit function may need to be tuned. And, once we automatically optimize the parameters of interleaving credit functions, to what degree do optimal values generalize to unseen ranker comparisons? We address these questions next.

Table 5.5: Agreement for $TDI_S^{t_s}$, $TDI_{T,S}^{t_s}$, and $TDI_{T,S}^{\mathbf{w}}$. Parameters t_s and $\mathbf{w}$ are chosen to maximize agreement with the A/B metrics on held out data, standard deviation is reported in brackets. Higher agreement than TDI is indicated in bold face. Statistically significant improvements over TDI are indicated by ▲ ($p < 0.01$) (losses ▼).

		(a) $TDI_S^{t_s}$		**(b)** $TDI_{T,S}^{t_s}$		**(c)** $TDI_{T,S}^{\mathbf{w}}$		
A/B Metric	TDI	Agree	t_s	Agree	t_s	Agree	w_1	w_2
AB	0.63	**0.82** ▲	0.76 (0.09)	0.53 ▼	0.52 (0.40)	**0.84** ▲	1.00 (0.00)	0.00 (0.00)
AB@1	0.71	**0.79** ▲	0.74 (0.19)	0.54 ▼	0.40 (0.32)	**0.75** ▲	1.00 (0.00)	0.05 (0.22)
AB_S	0.71	**0.84** ▲	0.76 (0.09)	0.48 ▼	0.29 (0.31)	**0.85** ▲	1.00 (0.00)	0.00 (0.00)
AB_S@1	0.76	**0.84** ▲	0.68 (0.24)	0.48 ▼	0.37 (0.32)	**0.82** ▲	1.00 (0.00)	0.02 (0.14)
AB_T	0.53	0.47 ▼	0.67 (0.28)	**0.67** ▲	0.54 (0.27)	**0.68** ▲	0.99 (0.11)	0.90 (0.30)
AB_T@1	0.45	**0.49** ▲	0.57 (0.35)	**0.62** ▲	0.61 (0.34)	**0.56** ▲	0.96 (0.22)	0.79 (0.41)
$AB_{T,S}$	0.47	0.46	0.46 (0.38)	**0.61** ▲	0.41 (0.30)	**0.63** ▲	0.91 (0.30)	0.88 (0.33)
$AB_{T,S}$@1	0.42	**0.52** ▲	0.30 (0.39)	**0.62** ▲	0.42 (0.34)	**0.50** ▲	0.06 (0.65)	0.25 (0.41)

5.6.2 Parameterized Credit Functions

One way to increase agreement of TDI with A/B metrics is to take an interleaving credit function with parameters (see Section 5.5.3) and tune the parameters towards a given A/B metric. For instance, previous work has shown that it is possible to estimate the probability that a given click indicates user satisfaction [110]. While an A/B metric such as AB_S must incorporate a threshold below which clicks are not considered to indicate user satisfaction, the threshold for TDI need not be the same. Rather, we can find the optimal threshold t_s for $TDI_S^{t_s}$ at which to consider a click as satisfied. This optimization procedure might lead to reduced variance, and thereby increase agreement with A/B metrics.

We use the maximization procedure described in Section 5.5.5 and in particular Equation 5.11 to find an optimal threshold for each A/B metric we consider. Note that, as opposed to experiments in the previous section, here we obtain averages over $N = 100$ iterations of the maximization procedure, instead of averages over the 38 comparisons. This allows us to perform statistical significance testing using a one-sample two-sided student's t-test. In our result table we indicate statistically significant improvements over TDI by ▲ ($p < 0.01$) (losses ▼). Also, as opposed to before, we now measure to what extend our optimized credit functions generalize to unseen data.

The results for maximizing agreement of $TDI_S^{t_s}$ are shown in column (a) in Table 5.5. In the table, we see substantially and signficantly increased agreement rates of up to 84% for the A/B metrics that only depend on clicks, reducing disagreement rates by between 6% and 20%. E.g., in Table 5.5, for $t_s = 0.76$ in the first row means that clicks predicted to have less than a 76% chance of indicating satisfaction are ignored. This causes 58% of clicks to be ignored on average. We see that across many folds, the optimal value is between 0.67 and 0.85, and we found that the precise value does not impact the outcome significantly.

Interestingly, the optimal threshold at which clicks should be included in the score

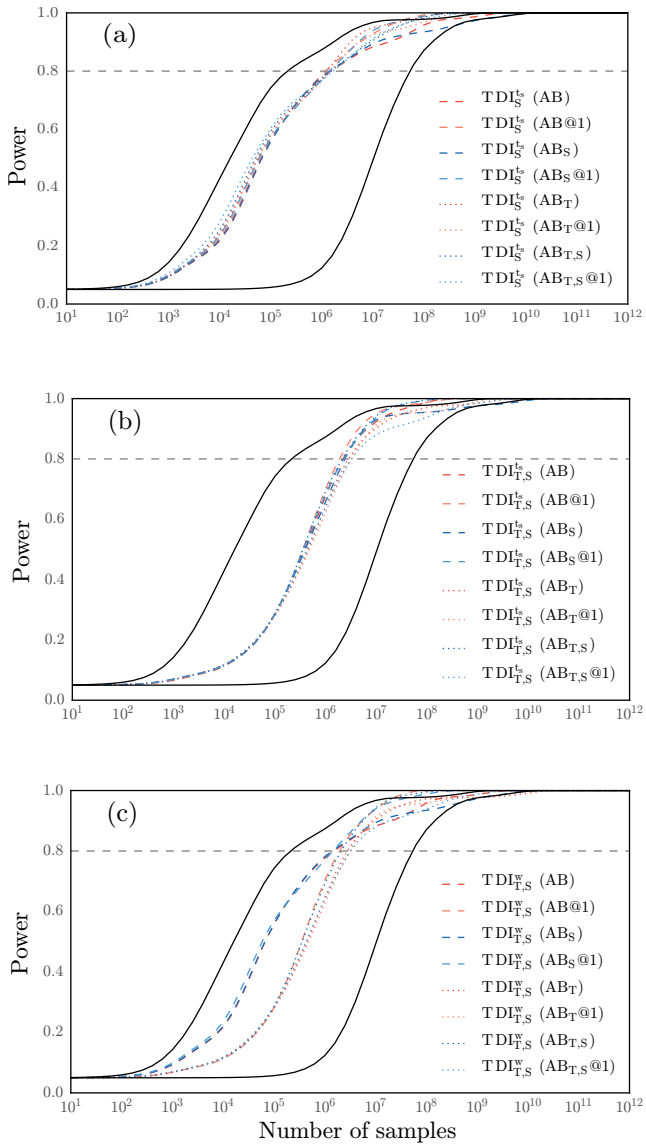

Figure 5.3: Power for (a) $TDI^{t_s}_S$, (b) $TDI^{t_s}_{T,S}$, and (c) $TDI^{\mathbf{w}}_{T,S}$. Parameters t_s and $\mathbf{w}$ maximize agreement with A/B metrics (in brackets) on held out data (see Table 5.5). The upper black line denotes TDI, the lower AB_S, the A/B metric with most power. Note that the differences between methods measured at 0.8 power are typically of interest.

calculation are around $t_s = 0.75$, which is lower than the $t_s = 0.8$ which is used in the AB_S and $AB_S@1$ A/B metrics. It is equally interesting that learning such a threshold changes agreement between TDI and time-based A/B metrics much less but generally decreases it, and selects a t_s threshold that is much lower.

In contrast, if we take the credit function $TDI^{t_s}_{T,S}$, that does take time as well as satisfaction into account and if we learn the same threshold t_s, then in column (b) of Table 5.5 we see a very different result than before in column (a). Now, the agreement between TDI and A/B metrics that incorporate time increases substantially and significantly (from 42–53% to 61–67%, a net disagreement error reduction of between 4% and 22%) while disagreement increases significantly with the non-time based metrics. Now, for the time based metrics we also outperform the baseline AB_{Sub} which uses as many query impressions as TDI does (see Table 5.2).

These observed changes in agreement exhibit the same pattern as seen when tuning a simpler threshold on satisfaction (as reported in column (a) in Table 5.5). In particular, tuning a feature of an interleaving method that does not represent a measure included in the A/B metric, reduces agreement with this A/B metric. We hypothesize that this is due to the maximization procedure failing to find a optimal value that generalizes well to unseen data, as the target A/B metrics that are based only on clicks have low correlation with the credit function we are optimizing. These results motivate our next approach: optimize a combination of interleaving credit functions that best matches a given A/B metric.

But first, we look at what happens to the power of TDI when we optimize parameters of a credit function. Results are in Figures 5.3 (a) and 5.3 (b). We see that again, the power for our adjusted interleaving credit functions lies between standard TDI and the A/B metric with the highest power. In other words, we increased agreement while maintaining an advantage in terms of power over A/B comparisons.

5.6.3 Combined Credit Functions

As we saw in the previous section and in columns (a) and (b) in Table 5.5, for different types of A/B metrics we need different interleaving credit functions to increase agreement. Optimizing a single parameter (t_s) for a single credit function proved not powerful enough. In this section we use the maximization procedure described in Section 5.5.5 and in particular Equation 5.10 to find weights $\mathbf{w}$ for a weighted combination of already optimized credit functions that maximizes agreement. That is, for each A/B metric we take the threshold t_s that in the previous section maximized agreement. Note that we only optimize a weighted combination of *two* credit functions, namely, we learn w_1 for $\hat{TDI^{t_s}_S}$ and w_2 for $\hat{TDI^{t_s}_{T,S}}$. The intuition behind this simple model is that it should be able to capture attributes of each of the A/B metrics.

We obtain the results presented in column (c) in Table 5.5. Where in the previous section, in columns (a) and (b) in Table 5.5, we obtained average agreement of 65% and 57% respectively, now we obtain an average agreement of 70%. Agreement with all individual A/B metrics increased significantly from 42-76% to 50-85%. Interestingly, we see that the weights w_1 and w_2 that are the result of the optimization procedure are mostly selecting ($w_2 \approx 0$) the $\hat{TDI^{t_s}_S}$ credit function for the click based A/B metrics. While the time based A/B metrics additionally put weight ($w_2 \gg 0$) on the time based credit

function $TDI^{\hat{f}_s}_{T,S}$.

Lastly, turning to the sensitivity, in Figure 5.3 (c) we see that also for the combined credit functions sensitivity stayed 1 to 2 orders of magnitude higher compared to A/B metrics.

5.7 Conclusion

In this chapter, we showed how to optimize interleaving outcomes to agree better with a given target A/B metric, while maintaining the sensitivity advantage of interleaved comparisons over A/B tests. We started by analyzing the agreement of *team draft interleave* (TDI) with a set of 8 A/B metrics based on combinations of click count, click positions, satisfied clicks, and time to click signals. To enable this analysis, we introduced a method for comparing A/B and interleaved comparison metrics in terms of power and agreement across varying sample sizes. We found that, while TDI is very sensitive, its agreement with user satisfaction A/B metrics on realistic ranking evaluations is low, from random up to 76%.

Results of this analysis motivated our approach. We proposed to replace the default credit function of TDI with novel credit functions that take richer user signals into account. In particular, we designed sets of credit functions that (1) match the parameters of A/B metrics, (2) are parameterized, and (3) combine (parameterized) credit functions. To automatically tune the parameters of these last credit functions, we further introduced a bootstrapping algorithm that can automatically maximize agreement with arbitrary A/B metrics.

Our empirical results, obtained on 38 paired experiments with a total of 3 billion clicks, showed that our approach can substantially and significantly increase agreement with A/B metrics. In particular, learning a combination of parameterized credit functions resulted in agreement of up to 85%, improving the agreement with A/B metrics by up to 22% (almost halving these disagreements). We also showed that the sensitivity for all our adapted versions of TDI is still 1 to 2 orders of magnitude higher than that of A/B metrics.

Despite the 3 billion clicks we used, we only studied 38 ranker comparisons. This constitutes the largest shortcoming of our work. Our 38 data points were often too few to draw generalizable conclusions.

The most important implication of our results is that it enables, for the first time, the integration of rich user satisfaction signals with highly sensitive interleaved comparison methods. This will dramatically reduce the required sample sizes, and therefore cost, of such online evaluations.

Lastly, there is a sample cost of getting the data needed to train our methods. Our methods our supervised training methods that can only be applied when AB tests interleaved comparisons have been performed on the same ranker pairs.

5.8 Future Work

There are a number of opportunities for further work that our results open up. First, if it were possible to generate larger data set, beyond the 38 ranker comparisons we

currently use, it may be possible to learn more sophisticated credit functions with even higher agreement with the target A/B metric. This would allow us to combine more credit functions than we currently do without overfitting the learned models. More data would also open the way to development of yet more sophisticated (learned) credit functions, e.g., to take into account session-level or task-level features. Also, our approach does not currently take magnitude and uncertainty in the A/B test label of individual experiments into account. Lastly, we would like to measure agreement with statistically significant A/B outcomes. Again, we would require more ranker comparisons for such an analysis.

Part II

Online Learning to Rank

6

Learning Parameters for Existing Rankers using Users Interactions

In this chapter we study the problem of optimizing an individual base ranker using clicks. Surprisingly, while there has been considerable attention for using clicks to optimize linear combinations of base rankers such as the models described in Section 2.1.3, the problem of optimizing an individual base ranker using clicks has been ignored. The problem is different from that of optimizing linear combinations of base rankers as the scoring function of a base ranker may be highly non-linear. For the sake of concreteness, we focus on the optimization of a specific base ranker, viz. *best match 25* (BM25) [146]. We start by showing that significant improvements in performance can be obtained when optimizing the parameters of BM25 for individual data sets. We also show that it is possible to optimize these parameters from clicks, i.e., without the use of manually annotated data, reaching or even beating manually tuned parameters.

This chapter is based on a paper by Schuth, Sietsma, Whiteson, and de Rijke [162].

6.1 Introduction

Traditional approaches to evaluating or optimizing rankers are based on manually created explicit relevance judgments (see Section 2.2). Recent years have witnessed a range of alternative approaches for the purpose of evaluating or optimizing rankers, which reduce or even avoid the use of explicit manual judgments. One type of approach is based on pseudo test collections, where judgments about query-document pairs are automatically generated by repurposing naturally occurring labels such as hashtags or anchor texts [10, 13, 21].

Another type of approach is based on the use of implicit signals (see Section 2.3). The use of implicit signals such as click data to evaluate or optimize retrieval systems has long been a promising alternative or complement to explicit judgments [30, 90, 92, 96, 144]. Evaluation methods that interpret clicks as absolute relevance judgments have often been found unreliable [144]. In some applications, e.g., for optimizing the click-through rate in ad placement and web search, it is possible to learn effectively from click data, using various learning to rank methods, often based on bandit algorithms. Click models can effectively leverage click data to allow more accurate evaluations with relatively little editorial data. Moreover, interleaved comparison methods have been developed that use

clicks not to infer absolute judgments but to compare rankers by observing clicks on interleaved result lists [79] (see Section 2.3.3).

The vast majority of work on click-based evaluation or optimization has focused on optimizing a linear combination of base rankers, thereby treating those rankers as black boxes [82, 83, 207]. See Section 2.1.3 for an overview of these retrieval models. Rankers, as discussed earlier, are functions that map documents to a ranking given a query. Rankers can either be based on a combination of these retrieval models or a single retrieval model can be taken as a ranker. In this chapter we do the latter. We take single retrieval models and refer to them as base rankers.

In this chapter, we try to break open the black boxes that base rankers often are and examine whether online learning to rank can be leveraged to optimize the base rankers themselves. Surprisingly, even though a lot of work has been done on improving the weights of base rankers in a combined learner, there is no previous work on online learning of the parameters of base rankers and there is a lot of potential gain from this new form of optimization. We investigate whether individual base rankers can be optimized using clicks. This question has two key dimensions. First, we aim to use clicks, an implicit signal, instead of explicit judgments. The topic of optimizing individual base rankers such as a model called *term frequency times inverse document frequency* (TF.IDF) [171], BM25 [146], or *divergence from randomness* (DFR) [8] has received considerable attention over the years but that work has almost exclusively used explicit judgments. Second, we work in an online setting while previous work on optimizing base rankers has almost exclusively focused on a more or less traditional, Cranfield-style, offline setting (see Section 2.2.1).

Importantly, the problem of optimizing base rankers is not the limiting case of the problem of optimizing a linear combination of base rankers where one has just one base ranker. Unlike the scoring function that represents a typical online learning to rank solution, the scoring function for a single base ranker is not necessarily linear. A clear example is provided by the well-known BM25 ranker [146], which has three parameters that are related in a non-linear manner: k_1, k_3 and b.

In this chapter, we pursue the problem of optimizing a base ranker using clicks by focusing on BM25. Currently, it is common practice to choose the parameters of BM25 according to manually tuned values reported in the literature, or to manually tune them for a specific setting based on domain knowledge or a sweep over a number of possible combinations using guidance from an annotated data set [60, 185]. We propose an alternative by learning the parameters from click data. Our goal is not necessarily to improve performance over manually tuned parameter settings, but rather to obviate the need for manual tuning.

Specifically, the research questions we aim to answer in this chapter are as follows, and repeated from Section 1.1.

RQ8 How good are the manually tuned parameter values of BM25 that are currently used? Are they optimal for all data sets on average? Are they optimal for individual data sets?

RQ9 Is it possible to learn good values of the BM25 parameters from clicks? Can we approximate or even improve the performance of BM25 achieved with manually

tuned parameters?

The contributions of this chapter are the following.

Learn Parameters of Base Rankers The insight that we can potentially achieve significant improvements of state-of-the-art learning to rank approaches by learning the parameters of base rankers, as opposed to treating them as black boxes which is currently the common practice.

Case Study with BM25 A demonstration of how parameters of an individual base rankers such as BM25 can be learned from clicks using the dueling bandit gradient descent approach.

Parameter Space of BM25 We provide insight into the parameter space of a base ranker such as BM25.

We have incorporated the above contributions in Section 1.2 where we give a complete overview of all contributions of this thesis.

6.2 Related Work

Related work comes in two main flavors: (1) work on ranker evaluation or optimization that does not use traditional manually created judgments, and (2) specific work on optimizing BM25 .

We refer to Section 2.3 for an extensive overview of online evaluation methods, also used in this chapter. Section 2.5 provides background on online learning to rank, relevant to this chapter. In particular, in that section Algortihm 2 lists the *dueling bandit gradient descent* (DBGD) [207] learning method that is used in this chapter. DBGD requires pairwise preferences as feedback. This can be an interleaved comparison method (see Section 2.3.3). In this chapter, we use *probabilistic interleave* (PI) [79], an interleaved comparison method that uses clicks not to infer absolute judgments but to compare base rankers by observing clicks on interleaved result lists; we use this relative feedback not only to optimize a linear combination of base rankers, as has been done before, but also to optimize an individual ranker. Our optimization method uses this relative feedback in a dueling bandit algorithm, where *pairs of rankers* are the arms that can be pulled to observe a click as relative feedback [82, 83, 207].

Our case study into optimizing an individual base ranker using clicks focuses on BM25, a parameterized (with parameters k_1, k_3 and b) combination of *term frequency* (TF), *inverse document frequency* (IDF) and query term frequency (cf. Section 6.3.1). A good general introduction to this ranker was written by Robertson and Zaragoza [147], while detailed coverage of early experiments aimed at understanding the model's parameters can be found in a paper by Sparck Jones et al. [174]. Improvements to standard BM25 have previously been investigated by Svore and Burges [179], who apply BM25 to different document fields and then use a machine learning approach to combine the results on these different fields. However, there the parameters of BM25 are still set at a fixed value. Most similar to the work presented here is work by Taylor et al. [185] and Gao

et al. [60]. There, however, the parameters of BM25 are optimized based on relevance labels, not clicks, in an offline learning setup, so that the parameters learned cannot be adapted while search takes place. Interestingly, over the years, different values of the key parameters in BM25 are used as manually tuned "default" values. E.g., Qin et al. [141] use $k_1 = 2.5$, $k_3 = 0$, $b = 0.8$ for the .gov collection. They use $k_1 = 1.2$, $k_3 = 7$, $b = 0.75$ for the OHSUMED collection, while Robertson and Walker [146] use $k_1 = 2.0$, $b = 0.75$.

6.3 Methods

Today's state-of-the-art ranking models combine the scores produced by many base rankers and compute a combination of them to arrive at a high-quality ranking. In its simplest form, this combination can be a weighted sum:

$$s(q, d) = w_1 \cdot s_1(q, d) + \cdots + w_n \cdot s_n(q, d), \tag{6.1}$$

where w_i is the weight of the base ranker $s_i(q, d)$ that operates on the query q and document d. The base rankers may have internal parameters that influence their performance. We focus on one particular base ranker, BM25, which has three parameters that determine the weight applied to term frequency, inverse document frequency and other query or document properties in the BM25 scoring function.

Below, we first recall BM25 in full detail and then describe how we use clicks to optimize BM25's parameters.

6.3.1 Implementation of BM25

Several variants of BM25 are used in the literature. We use the variant that is used to compute the BM25 feature in the LETOR data set [141] (see Section 3.2). Given a query q and document d, the BM25 score is computed as a sum of scores for every term q_i in the query that occurs at least once in d:

$$BM25(q, d) = \sum_{q_i : tf(q_i, d) > 0} \frac{idf(q_i) \cdot tf(q_i, d) \cdot (k_1 + 1)}{tf(q_i, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avgdl})} \cdot \frac{(k_3 + 1) \cdot qtf(q_i, q)}{k_3 + qtf(q_i, q)} \tag{6.2}$$

The terms used in this formula are:

- $idf(q_i)$ (*inverse document frequency*): computed as

$$idf(q_i) := \log \left(\frac{N - df(q_i) + 0.5}{df(q_i) + 0.5} \right), \tag{6.3}$$

  where N is the total number of documents in the collection and $df(q_i)$ is the number of documents in which the term q_i occurs at least once;

- $tf(q_i, d)$ (*term frequency*): the number of times the term q_i occurs in the document d;

- $qtf(q_i, q)$ (*query term frequency*): the number of times the term q_i occurs in the query q;

- $\frac{|d|}{avgdl}$: the length of the document, normalized by the average length of the documents in the collection;

- k_1, b, and k_3: the parameters of BM25 that we want to optimize. Usually, k_1 is set to a value between 1 and 3, b is set somewhere around 0.8 and k_3 is set to 0. Note that when k_3 is set to 0 the entire right part of the product in Equation 6.2 cancels out to 1 and can thus be ignored.

6.3.2 Learning from Clicks

Most learning to rank approaches learn from explicit, manually produced relevance assessments [124]. These assessments are expensive to obtain and usually produced in an artificial setting. More importantly, it is not always feasible to obtain the assessments needed. For instance, if we want to adapt a ranker towards a specific user or a group of users, we cannot ask explicit feedback from these users as it would put an undesirable burden upon these users.

Instead, we optimize rankers using clicks. It has been shown by Radlinski et al. [144] that interpreting clicks as absolute relevance judgments is unreliable. Therefore, we use a dueling bandit approach: the *candidate preselection* (CPS) method. This method was shown to be state-of-the-art by Hofmann et al. [83]. It is an extension of the *dueling bandit gradient descent* (DBGD) method, proposed in [207]. DBGD has been described extensively in Section 2.5 and is listed in Algorithm 2. Briefly, DBGD works as follows. The parameters that are being optimized are initialized. When a query is presented to the learning system, two rankings are generated: one with the parameters set at the current best values, another with a perturbation of these parameters. These two rankings are interleaved using probabilistic interleave [79, 81], which allows for the reuse of historical interactions. The interleaved list is presented to the user and we observe the clicks that the user produces, which are then used to determine which of the two generated rankings was best. If the ranking produced with the perturbed set of parameters wins the interleaved comparison, then the current best parameters are adapted in the direction of the perturbation. CPS is a variant of DBGD that produces several candidate perturbations and compares these on historical click data to decide on the most promising candidate. Only the ranking produced with the most promising perturbation is then actually interleaved with the ranking generated with the current best parameters and exposed to the user.

The difference between the current best ranker and the perturbed ranker is controlled by the parameter δ. The amount of adaptation of the current best ranker in case the perturbed ranker wins is controlled by a second parameter, α. Together, these parameters balance the speed and the precision with which the algorithm learns. If they are too big, the learning algorithm may oscillate, skip over optimal values and never converge to the optimum. If they are too small, the learning algorithm will not find the global optimum at all or not in a reasonable amount of time.

We aim to learn the BM25 parameters k_1, b and k_3 from clicks, using the learning method described above. Because the parameters are of very different orders of magnitude, with b typically ranging between 0.45 and 0.9 and k_1 typically ranging between 2 and 25, we chose to use a separate δ and α for each parameter. This is necessary because what may be a reasonable step size for k_1 will be far too large for b. Therefore we have,

for example, a separate δ_{k_1} and δ_b. This allows us to govern the size of exploration and updates in each direction.

6.4 Experiments

In this section, we detail our experimental setup in as far as it is different from the setup described in Chapter 3.

We design our experiments to answer RQ8 and RQ9. We investigate whether we can optimize the parameters of a base ranker, BM25, from clicks produced by users interacting with a search engine. Below, we first describe the data we use to address this question. Then we describe how our click-streams are generated, and our evaluation setup.

6.4.1 Data Sets

For all our experiments we use features extracted from the .gov collection that is also included in the LETOR data set [141], described in Section 3.2. The six sets of queries and relevance assessments we use in this chapter are based on TREC Web track tasks run from 2003 and 2004.

The data sets HP2003, HP2004, NP2003, and NP2004 implement navigational tasks: homepage finding and named-page finding, respectively. TD2003 and TD2004 implement an informational task: topic distillation. All six data sets contain between 50 and 150 queries and approximately 1,000 judged documents per query. These data sets have binary relevance.

We index the original .gov collection to extract the low-level features such as term frequency and inverse document frequency that are needed for BM25. While indexing, we do not perform any pre-processing (e.g., no stemming, no stop word removal). We only extract features for the documents in the LETOR data set [141]. All the data sets we use are split by query for 5-fold cross validation.

6.4.2 Clicks

To produce clicks, we use a click simulation framework that is analogous to [83], which is explained in [161] and in Section 3.3. Our framework simulates clicks using the *dependent click model* (DCM) [65].

Again following [83], we instantiate $P(C|R)$ and $P(S|R)$ as in Table 3.2. We only use the extreme instantiations as the data sets used in this chapter only have binary relevance. In this chapter, we use four instantiations of the click model: the *perfect* click model; the *navigational* click model; the *informational*; and the *almost random* click model.

6.4.3 Parameter Settings

We employ the learning approach described in Section 6.3.2. For CPS we use the parameters suggested by [83]: we use $\eta = 6$ candidate perturbations and we use the $\lambda = 10$ most recent queries. We initialize the weights of both the BM25 model and the LETOR features randomly. For the learning of BM25 in isolation, we set $\alpha_b = 0.05$ and $\delta_b = 0.5$. We computed the average ratio between k_1 and δ across the parameter values

that were optimal for the different data sets, and set α_{k_1} and δ_{k_1} accordingly. This ratio was 1 to 13.3, so we set $\alpha_{k_1} = 0.665$ and $\delta_{k_1} = 6.65$. These learning parameters have been tuned on a held out development set.

6.4.4 Evaluation and Significance Testing

As evaluation metric, we use *normalized discounted cumulative gain* (nDCG) [89] on the top 10 results, measured on the test sets, following, for instance, [179, 185] (see Section 3.4). For each learning experiment, for each data set, we run the experiment for 2,000 interactions with the click model. We repeat each experiment 25 times and average results over the 5 folds and these repetitions. We test for significant differences using the paired t-test in answering RQ8 and the independent measures t-test for RQ9.

6.5 Results and Analysis

We address our two research questions, RQ8 and RQ9, in the following two subsections.

6.5.1 Measuring the Performance of BM25 with Manually Tuned Parameters

In order to answer RQ8, we compute the performance of BM25 with the parameters used in the LETOR data set [141]. The parameter values used there differ between the two document collections in the data set. The values that were chosen for the .gov collection were $k_1 = 2.5$, $b = 0.8$ and $k_3 = 0$. The values that were chosen for the OHSUMED collection were $k_1 = 1.2$, $b = 0.75$ and $k_3 = 7$. We refer to these values as the manually tuned .gov or OHSUMED parameter values, respectively. Note that the manually tuned .gov parameter values were tuned to perform well on average, over all data sets.

The results of running BM25 with the .gov manual parameters (as described in Section 6.4) are in the first row of Table 6.1. We also experiment with different values of k_1, b and k_3. We first tried a range of values for k_1 and b. For k_1 the range is from -1 to 30 with steps of 0.1 and for b the range is for -0.5 to 1 with steps of 0.05. The results are in Table 6.1. For each of the data sets, we include the parameter values that gave maximal nDCG scores (in bold face). For each value of k_1 and b, we show the performance on each data set and the average performance over all data sets.

The results show that when we average over all data sets, no significant improvements to the manually tuned .gov parameter values can be found. This is to be expected, and merely shows that the manual tuning was done well. However, for four out of six data sets, a significant improvement can be achieved by deviating from the manually tuned .gov parameter values *for that particular data set*. Furthermore, if we take the average optimal nDCG, weighted with the number of queries in each data set, we find an overall performance of 0.644 nDCG versus 0.613 nDCG when manually tuned parameters would be used. Thus, it pays off to optimize the parameters for specific data sets.

In cases where both k_1 and b were different from the manually tuned .gov values, we also consider the results of combining k_1 with the manually tuned .gov value for b and vice-versa. E.g., when $k_1 = 4.0$ and $b = 0.5$, for the NP2004 data set the value of b has a

Table 6.1: nDCG scores for various values of BM25 parameters k_1 and b, optimized for different data sets. The first and last row give the scores with parameters that have been manually tuned for the .gov and OHSUMED collections, respectively [141]. Other parameter values are chosen to produce maximal scores, printed in boldface, for the different data sets listed in the first column. For all results, $k_3 = 0$. Statistically significant improvements (losses) over the values manually tuned for .gov are indicated by $^{\triangle}$ ($p < 0.05$) and $^{\blacktriangle}$ ($p < 0.01$) ($^{\triangledown}$ and $^{\blacktriangledown}$).

	k_1	b	HP2003	HP2004	NP2003	NP2004	TD2003	TD2004	Overall
.gov	2.50	0.80	0.674	0.629	0.693	0.599	0.404	0.469	0.613
HP2003	7.40	0.80	**0.692**	0.650	0.661$^{\blacktriangledown}$	0.591	0.423$^{\blacktriangle}$	0.477	0.614
HP2004	7.30	0.85	0.688	**0.672**$^{\triangle}$	0.657$^{\blacktriangledown}$	0.575	0.423$^{\blacktriangle}$	0.482$^{\triangle}$	0.613
	2.50	0.85	0.671	0.613	0.682	0.579$^{\triangledown}$	0.404	0.473	0.605$^{\triangledown}$
	7.30	0.80	0.690	0.647	0.661$^{\blacktriangledown}$	0.592	0.423$^{\blacktriangle}$	0.477	0.613
NP2003	2.60	0.45	0.661	0.572$^{\triangledown}$	**0.719**	0.635	0.374$^{\blacktriangledown}$	0.441$^{\blacktriangledown}$	0.607
	2.50	0.45	0.660	0.572$^{\triangledown}$	0.718	0.635	0.374$^{\blacktriangledown}$	0.441$^{\blacktriangledown}$	0.607
	2.60	0.80	0.675	0.629	0.692	0.601	0.403	0.470	0.613
NP2004	4.00	0.50	0.663	0.584	0.705	**0.647**$^{\triangle}$	0.386$^{\triangledown}$	0.446$^{\blacktriangledown}$	0.609
	2.50	0.50	0.663	0.573$^{\triangledown}$	0.713	0.635	0.381$^{\blacktriangledown}$	0.444$^{\blacktriangledown}$	0.607
	4.00	0.80	0.680	0.645	0.683	0.605	0.414$^{\triangle}$	0.474	0.616
TD2003	25.90	0.90	0.660	0.597	0.515$^{\blacktriangledown}$	0.478$^{\blacktriangledown}$	**0.456**$^{\triangle}$	0.489$^{\triangle}$	0.550$^{\blacktriangledown}$
	2.50	0.90	0.676	0.607	0.672	0.560$^{\blacktriangledown}$	0.405	0.471	0.600$^{\blacktriangledown}$
	25.90	0.80	0.645	0.576	0.535$^{\blacktriangledown}$	0.493$^{\blacktriangledown}$	0.445	0.482	0.549$^{\blacktriangledown}$
TD2004	24.00	0.90	0.664	0.604	0.520$^{\blacktriangledown}$	0.481$^{\blacktriangledown}$	0.449$^{\triangle}$	**0.491**$^{\triangle}$	0.553$^{\blacktriangledown}$
	2.50	0.90	0.676	0.607	0.672	0.560$^{\blacktriangledown}$	0.405	0.471	0.600$^{\blacktriangledown}$
	24.00	0.80	0.645	0.578	0.538$^{\blacktriangledown}$	0.496$^{\blacktriangledown}$	0.446	0.482	0.550$^{\blacktriangledown}$
OHSUMED	1.20	0.75	0.662$^{\triangledown}$	0.589$^{\blacktriangledown}$	0.703	0.591	0.398	0.461$^{\blacktriangledown}$	0.605$^{\triangledown}$

bigger impact than the value of k_1: changing k_1 back to the manually tuned value causes a decrease of nDCG of 0.012 points, while changing b to the manually tuned value gives a decrease of 0.042 points. However, in other cases the value of k_1 seems to be more important. E.g., for the TD2003 data set we can achieve an improvement of 0.041 points by changing k_1 to 25.9, while keeping b at the manually tuned 0.8.

The bottom row in Table 6.1 shows the results of a BM25 ranker with the manually tuned OHSUMED parameter values. This ranker performs worse than the manually tuned .gov values averaged over all data sets, which, again, shows that it makes sense to tune these parameters, rather than just taking a standard value from the literature.

For the third parameter k_3, we performed similar experiments, ranging the parameter value from 0 to $1,000$. There were no significant differences in the resulting nDCG scores. The small differences that were present were in favor of the manually tuned value 0. The fact that k_3 hardly has any influence on the performance is to be expected, considering the fact that k_3 weights the query term frequency (cf. Equation 6.2), the number of times one word appears in the query. For most query terms, the query term frequency will be 1.

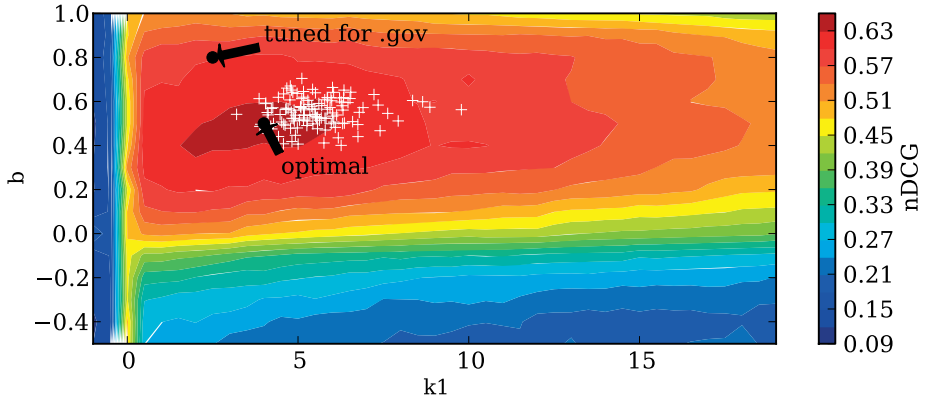

Figure 6.1: Optimization landscape for two parameters of BM25, k_1 and b, for the NP2004 data set measured with nDCG. White crosses indicate where individual runs of the learning algorithm plateaued when learning from clicks produced with the *perfect* click model. For the other five data set we experimented with we obtained a similar landscape with the peak on a different location.

Hence, the weight of this feature does not greatly affect the result.

Using these results, we are ready to answer our first research question RQ8. The manually tuned .gov values for k_1 and b are quite good when we look at a combination of all data sets. When looking at different data sets separately, significant improvement can be reached by deviating from the values that were manually tuned for the entire collection. This shows that tuning of the parameters to a specific setting is a promising idea. Considering the last parameter k_3, the standard value was optimal for the data sets we investigated.

6.5.2 Learning Parameters of BM25 Using Clicks

In this section we answer RQ9: can we learn the parameters of BM25 from clicks? We aim to learn the parameters per data set from clicks. Our primary goal is not to beat the performance of the manually tuned .gov parameters. Should optimizing a base ranker such as BM25 prove successful, i.e., reach or even beat manually tuned values, the advantage is rather that optimizing the parameters no longer requires human annotations. Furthermore, learning of the parameters eliminates the need for domain-specific knowledge, which is not always available, or sweeps over possible parameter values, which cost time and cannot be done in an online setting.

To begin, we visualize the optimization landscape for the two BM25 parameters that matter: k_1 and b. We use the data obtained from the parameter sweep described in Section 6.5.1. The optimization landscape is unimodal and generally smooth when averaged over many queries, as illustrated by Figure 6.1 for the NP2004 data set. We find similar landscapes with peaks at different locations (listed in Table 6.1) for other data sets.

This observation suggests that a gradient descent approach such as DBGD is a suitable

learning algorithm. Note, however, that the online learning algorithm will never actually observe this landscape: it can only observe *relative* feedback from the interleaved comparisons and moreover, this feedback is observed on a *per query* basis.

Next, we optimize k_1 and b for each individual data set using the four instantiations of our click model: *perfect*, *navigational*, *informational*, and *almost random*. The learning curves are depicted in Figure 6.2. Irrespective of the noise in the feedback, the learning method is able to dramatically improve the performance of BM25. For the *perfect* click model the final performance after 2000 queries is either on par with the manually tuned values used by Qin et al. [141] or above. We can, however, not always recover the gain we observe in the parameter sweep in Table 6.1 completely when learning from clicks.

For the NP2004 data set we plot the final parameter values that have been learned using the *perfect* click model in Figure 6.1. The final parameter values are clustered near the optimal value indicating that the learning method is indeed capable of finding the peak in the landscape. Final parameters for individual runs using each data set are depicted in Figure 6.3. We see that for each data set, the parameters converge to a different region. We also see that the manually tuned parameters are not included in any of these regions.

Performance generally degrades when clicks become less reliable. However, performance of the navigational click model is not much lower than the performance of the perfect click model. This is a promising result, since feedback from actual users will be noisy and our learning method should be able to deal with that.

The above experiments are all initialized with random starting parameters. If one knew a good starting point, learning could be sped up. E.g., we also initialized learning with the manually tuned .gov parameters ($k_1 = 2.5$ and $b = 0.8$) and observed that the plateau that was found was not different from the one we found with random initialization. It was, however, found in fewer than 200 queries, depending on the data set.

In conclusion, we can give a positive answer to the first part of RQ9. Learning good values for the BM25 parameters from user clicks is possible. As to the second part of RQ9, the optimized parameters learned form clicks lead to a performance of BM25 that approaches, equals or even surpasses the performance achieved using manually tuned parameters for all data sets.

6.6 Conclusion

In this chapter we investigated the effectiveness of using clicks to optimize base rankers in an online learning to rank setting. State-of-the-art learning to rank approaches use a linear combination of several base rankers to compute an optimal ranking. Rather than learning the optimal weights for this combination, we optimize the internal parameters of these base rankers. We focussed on the base ranker BM25 and aimed at learning these parameters of BM25 in an online setting using clicks.

Our results show that learning good parameters of BM25 from clicks is indeed possible. As a consequence, it is not necessary to hand tune these parameters or use human assessors to obtain labeled data. Learning with a dueling bandit gradient descent approach converges to near-optimal parameters after training with relative click feedback on about 1,000 queries. Furthermore, the performance of BM25 with these learned parameters approaches, equals or even surpasses the performance achieved using manually tuned

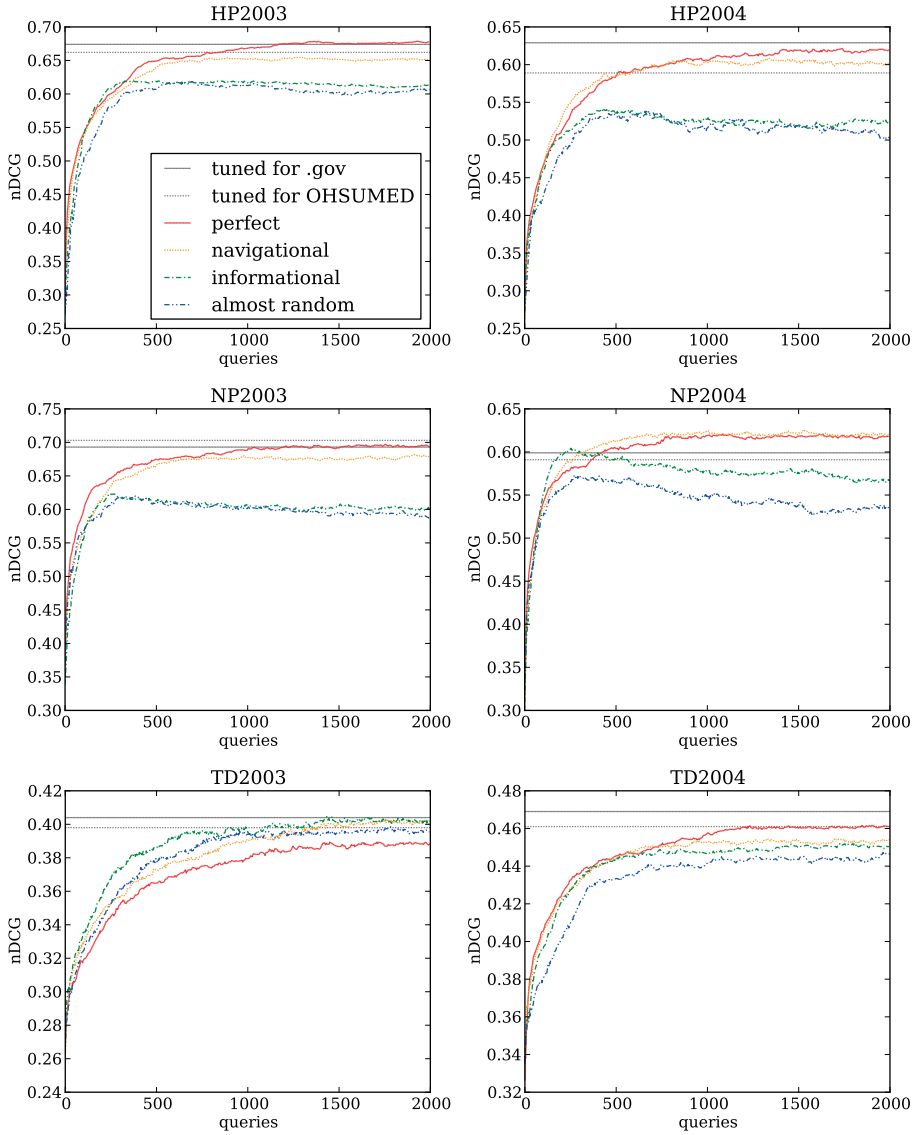

Figure 6.2: Learning curves when learning the parameters of BM25 using DBGD from clicks. Measured in nDCG on a holdout data set averaged over 5-fold cross validation and 25 repetitions. The clicks used for learning are produced by the *perfect*, *navigational*, *informational* and *almost random* click model. The horizontal gray lines indicate the performance for the manually tuned .gov (solid) and OHSUMED (dotted) parameters.

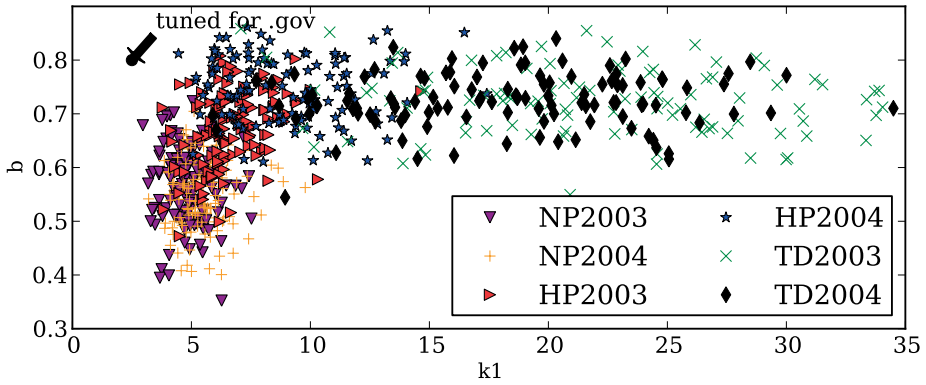

Figure 6.3: Parameter settings where individual runs of the learning algorithm plateaued when learning with the *perfect* click model for 2,000 queries.

parameters for all data sets. The advantage of our approach lies in the fact that the parameters can be learned automatically from implicit feedback, and that the parameters can be tuned specifically for different settings. The parameters learned from the data in the different data sets differ greatly from each other. More importantly, the performance gains we observed by tuning the parameters for a specific data set were significant for many of the data sets considered. We could, however, not always recover this gain completely when learning from clicks.

6.7 Future Work

For future work, it is interesting to see how the click-optimized versions of BM25 can improve the performance of a state-of-the-art learning to rank algorithm when BM25 is used as one query-document feature among many.

There are several ways in which the process of optimizing a base ranker can be integrated with state-of-the-art online learning to rank:

- First learn the optimal parameters of the base rankers in isolation, then use the optimized base rankers and learn the optimal weights of a linear combination of these base rankers.
- First learn the optimal weights of a linear combination of base rankers using either optimized or default parameter values for the individual base rankers and then learn the optimal parameters of the individual base rankers based on user clicks in reply to the outcome of the ensemble learning-to-rank algorithm.
- Learn the parameters of the base rankers and the weights of the ensemble together based on user clicks in reply to the outcome of the ensemble.

It would be interesting to see whether we can integrate this with the learning methods in Chapter 7. Those methods use the multileaved comparisons from Chapter 4. This allows

for exploring many more alternative rankers. This in turn may allow for learning the parameters of base rankers and the combination of base rankers simultaneously.

Additionally, we would like to investigate whether and to what extent parameters of other base rankers can be learned through the same procedure as we used in this chapter.

7 Learning from Multileaved Comparisons

Modern search systems are based on dozens or even hundreds of ranking features such as the retrieval models described in Section 2.1.3. The *dueling bandit gradient descent* (DBGD) algorithm has been shown to effectively learn combinations of these features solely from user interactions (see Algorithm 2 on page 29). DBGD explores the search space by comparing a possibly improved ranker to the current production ranker. To this end, it uses *interleaved comparison methods*, which can infer with high sensitivity a preference between two rankings based only on interaction data (see Section 2.3.3). Recently introduced *multileaved comparison methods*, which can compare a set of rankings instead of just a pair, have been found to be even more sensitive, i.e., they require even less interaction data for reliable ranker comparisons (see Chapter 4).

In this chapter we propose an online learning to rank algorithm called *multileave gradient descent* (MGD) that extends DBGD to learn from multileaved comparison methods. We show experimentally that MGD allows for better selection of candidates than DBGD without the need for more comparisons involving users. An important implication of our results is that orders of magnitude less user interaction data is required to find good rankers when multileaved comparisons are used within online learning to rank. As a consequence, fewer users need to be exposed to possibly inferior rankers and our method allows search engines to adapt more quickly to changes in user preferences.

This chapter is based on work by Schuth, Oosterhuis, Whiteson, and de Rijke [167].

7.1 Introduction

Modern search engines base their rankings on combinations of dozens or even hundreds of features [124]. *Learning to rank*, i.e., finding an optimal combination of such features, is an active area of research (see Section 2.4).

Traditionally, learning was done *offline* by optimizing for performance on a training set consisting of queries and relevance assessments produced by human assessors. However, as described in Section 2.5, such data sets are time consuming and expensive to produce. Moreover, these assessments are not always in line with actual user preferences [155]. And since data of users interacting with a search engine are often readily available, research is focussing more on learning *online* instead [82, 92, 144, 207]. Online learning to rank methods optimize combinations of rankers while interacting with users of a search engine. While interacting with the search engine, users leave a trace of interaction data, e.g.,

query reformulations, mouse movements, and clicks, that can be used to infer preferences. Clicks have proven to be a valuable source of information when interpreted as a preference between either rankings [144] or documents [92]. In particular, when clicks are interpreted using interleaved comparison methods, they can reliably infer preferences between a pair of rankers [34, 92, 93].

Dueling bandit gradient descent (DBGD) [207] is an online learning to rank algorithm that learns from these interleaved comparisons. It uses the inferred preferences to estimate a gradient, which is followed to find a locally optimal ranker. At every learning step, DBGD estimates this gradient with respect to a *single* exploratory ranker and updates its solution if the exploratory ranker seems better. Exploring *more than one* ranker before updating towards a promising one could lead to finding a better ranker using fewer updates. However, when using interleaved comparisons, this would be too costly, since it would require pairwise comparisons involving users between all exploratory rankers. Instead, we propose to learn from comparisons of multiple rankers at once, using a single user interaction. In this way, our proposed method, *multileave gradient descent* (MGD), aims to speed up online learning to rank.

We propose two variants of MGD that differ in how they estimate the gradient. In *MGD winner takes all* (MGD-W), the gradient is estimated using one ranker randomly sampled from those who won the multileaved comparison. In *MGD mean winner* (MGD-M), the gradient is estimated using the mean of all winning rankers.

In this chapter, we answer the following research questions, repeated from Section 1.1.

RQ10 Can MGD learn faster from user feedback (i.e., using fewer clicks) than DBGD does?

RQ11 Does MGD find a better local optimum than DBGD?

RQ12 Which update approach, MGD-W or MGD-M, learns faster? Which finds a better local optimum?

Our contributions in this chapter are the following.

Online Learning to Rank Methods Two approaches, MGD-W and MGD-M, to using multileaved comparison outcomes in an online learning to rank method.

Extensive Evaluation Extensive empirical validation of our new methods via experiments on nine learning to rank data sets, showing that MGD-W and MGD-M outperform the state of the art in online learning to rank.

We have incorporated the above contributions in Section 1.2 where we give a complete overview of all contributions of this thesis.

In Section 7.2 we discuss related work. In Section 7.3 we introduce multileave gradient descent. In Section 7.4 we detail our experimental setup. Section 7.5 provides both results and their analysis. We conclude in Section 7.6.

7.2 Related Work

Related work for this chapter comes in two areas. First, there are learning to rank methods that have been introduced in Section 2.5. Particularly important for this chapter is the *dueling bandit gradient descent* (DBGD) method, which is described in more detail in that section and listed in Algorithm 2 on page 29. Second, there are evaluation methods in Section 2.3, of which the interleaving and multileaving methods play a crucial role in this chapter. Interleaving methods are described in more detail in Section 2.3.3.

In thesis, interleaving methods have been extended to *multileaving* methods that allow for comparisons of more than two rankers at once (see Chapter 4 and the work by Schuth et al. [163]). In particular, we extended *team draft interleave* (TDI) [144] to *team draft multileave* (TDM) and *optimized interleave* (OI) [143] to *optimized multileave* (OM). TDM forms part of our motivation of this chapter and is discussed in detail in Chapter 4 where multileaving methods are introduced.

If n rankers must be compared, an interleaving method such as TDI needs $n \cdot (n-1)$ queries to determine how they all relate to each other. By contrast, for TDM, a user's query is passed to all n rankers at once. These rankers each produce their rankings, which are integrated into a single ranking using a team selection process similar to that of TDI. However, there now are n teams that take turns.[1] This implies that, in case n is larger than the number of slots in the interleaved list, some teams may not be represented. Inferring which team wins is now done by counting the number of clicked documents for each team. The result is a partial ordering over the n rankers. Thus, only a single query, instead of $n \cdot (n-1)$ queries, is needed to compare all n rankers. Of course, many queries are still needed for a *reliable* comparison, and potentially more so than with TDI. However, it was shown that this tradeoff can be quite favorable for TDM [163]. Note that TDM reduces to TDI for $n = 1$.

Our work in this chapter is different from the work mentioned above in that our online learning to rank methods are the first to learn from multileaving comparison feedback. So far, online learning to rank methods only learned from pairwise preferences between either documents or rankers. Our methods are the first to learn from n-way preferences between rankers.

7.3 Multileave Gradient Descent

In this section, we propose a new algorithm called *multileave gradient descent* (MGD).

7.3.1 Extending DBGD with Multileaving

MGD is shown in Algorithm 10. As in DBGD, which is shown in Algorithm 2, MGD learns from online feedback and uses a *current best ranker*, which is updated based on user feedback. For each query, MGD uses the *current best ranker* to create a ranking. Subsequently, on lines 4 through 7, n exploratory candidate rankers are generated along with their corresponding rankings. Unlike DBGD, which is restricted to a single candidate

[1] As in TDI, documents belonging to a prefix that is common to all n rankings are not assigned to teams.

Algorithm 10 Multileave Gradient Descent (MGD).

Require: n, α, δ, $\mathbf{w}_0^0$, $update(\mathbf{w}, \alpha, \{\mathbf{b}\}, \{\mathbf{u}\})$

1: **for** $t \leftarrow 1..\infty$ **do**
2: $q_t \leftarrow receive\_query(t)$ *// obtain a query from a user*
3: $\mathbf{l}_0 \leftarrow generate\_list(\mathbf{w}_t^0, q_t)$ *// ranking of current best*
4: **for** $i \leftarrow 1...n$ **do**
5: $\mathbf{u}_t^i \leftarrow sample\_unit\_vector()$
6: $\mathbf{w}_t^i \leftarrow \mathbf{w}_t^0 + \delta\mathbf{u}_t^i$ *// create a candidate ranker*
7: $\mathbf{l}_t^i \leftarrow generate\_list(\mathbf{w}_t^i, q_t)$ *// exploratory ranking*
8: $\mathbf{m}_t, \mathbf{t}_t \leftarrow TDM\_multileave(\mathbf{l}_t)$ *// multileaving and teams*
9: $\mathbf{c}_t \leftarrow receive\_clicks(\mathbf{m}_t)$ *// show multileaving to the user*
10: $\mathbf{b}_t \leftarrow TDM\_infer(\mathbf{t}_t, \mathbf{c}_t)$ *// set of winning candidates*
11: **if** $\mathbf{w}_t^0 \in \mathbf{b}_t$ **then**
12: $\mathbf{w}_{t+1}^0 \leftarrow \mathbf{w}_t^0$ *// if current best among winners, no update*
13: **else**
14: $\mathbf{w}_{t+1}^0 \leftarrow update(\mathbf{w}_t^0, \alpha, \mathbf{b}_t, \mathbf{u}_t)$ *// Algorithm 11 or 12*

Algorithm 11 MGD update function: winner takes all (MGD-W).

Require: $\mathbf{w}$, α, $\mathbf{b}$, $\mathbf{u}$ *// in* $\mathbf{b}$ *are only winners*

1: $\mathbf{b}^j \leftarrow pick\_random\_uniformly(\mathbf{b})$
2: **return** $\mathbf{w} + \alpha\mathbf{u}^j$

ranker during comparison, MGD can handle multiple candidate rankers because it uses multileaving, which on line 8 creates a single document list out of the n rankings. After observing user clicks on this ranker, on line 9, a set of rankers that won the comparison is inferred. In our case, TDM is used and thus the set of winners contains the candidate(s) that received the greatest number of clicks. If the *current best ranker* is among the winners, then no candidate is considered to be better and no update is performed. However, if not, the *current best ranker* is updated accordingly on line 14, using one of two update methods described in Section 7.3.2. In this way, MGD incrementally improves the *current best ranker*.

By comparing multiple candidates at each iteration the probability of finding a better candidate ranker than the current best is expected to increase. Furthermore, adding more rankers to the comparison increases the expected value of the resulting ranker, since the candidate rankers will also compete with each other. Correspondingly, the intuition behind MGD is that the use of multileaving improves the learning speed compared to DBGD. It should be noted, though, that the quality of the document list presented to the user may decrease: as MGD is more exploratory than DBGD, i.e., multileaving lets more candidate rankers add documents to the list, thus the *current best ranker* is exploited less than in the DBGD case.

7.3.2 Multileave Approaches to Gradient Descent

DBGD generates each candidate ranker by sampling a unit sphere uniformly and adding

Algorithm 12 MGD update function: mean winner (MGD-M).

Require: $\mathbf{w}$, α, $\mathbf{b}$, $\mathbf{u}$ *// in* $\mathbf{b}$ *are only winners*

1: **return** $\mathbf{w} + \alpha \frac{1}{|\mathbf{u}|} \sum_{\mathbf{b}^j \in \mathbf{b}} \mathbf{u}^j$

the resulting unit vector to the *current best ranker*. For the MGD approaches in this chapter this procedure was repeated n times to create a set of n candidate rankers. However, this approach might have the drawback that it can produce identical or very similar candidate rankers. Thus it is possible that during an iteration identical candidates are compared, potentially compromising the exploratory benefits of using MGD. But since the dimensionality of the feature space is expected to be much greater than the number of candidates, it is most unlikely that the set contains similar rankers.

The simple update method of DBGD is only applicable to a single winning candidate ranker. Conversely, MGD requires an approach to infer an update from a winning set of candidate rankers. We introduce two approaches for performing updates: *MGD winner takes all* (MGD-W) and *MGD mean winner* (MGD-M) displayed in Algorithms 11 and 12, respectively. MGD-W picks a random candidate ranker out of the set of winners and performs the DBGD update as if it were the only winner. This has the disadvantage that all other winning candidates are discarded, but it has the advantage that the update is performed towards a candidate that was part of the comparison. MGD-M, on the other hand, takes the mean of the winning candidate rankers and performs the DBGD update as if the mean was the only winner. In contrast with MGD-W, MGD-M uses all the winning candidates in its update. However, the update is performed towards the mean of the winning rankers, thereby assuming that the mean of all winners is preferred over the *current best ranker*, despite the two not having been directly compared. Thus, updates could actually harm the current best ranker. However, this risk also exists for MGD-W since user interaction is expected to contain noise and can result in poor candidate ranker winning a comparison. Note that both methods reduce to DBGD for $n = 1$.

An alternative way of comparing many candidate rankers without having to do many comparisons with users involved, is DBGD with *candidate pre-selection* (CPS) [83]. However, this method reuses historical interaction data which it requires to be generated stochastically using potentially unsafe rankings [143]. MGD is a new way of comparing candidates that does not have this drawback.

7.4 Experiments

In this section, we detail our experimental setup in as far as it is different from the general setup described in Chapter 3. Our experiments are designed to answer the research questions posed in Section 7.1. We are interested in whether and how our newly introduced algorithm MGD (RQ10) learns faster than DBGD; (RQ11) converges to a better optimum compared to DBGD; and (RQ12) how the two variants MGD-W and MGD-M compare to each other.

All our experiments assume a stream of independent queries coming from users interacting with the system we are training. Users are presented with a result list in response to their query and may or may not interact with the list by clicking on one or

more documents. The queries come from static data sets (Section 7.4.1) and the clicks from a click model (Section 7.4.2). In Section 7.4.3 we describe the experiments we run. Our evaluation measures are described in Section 7.4.4.

7.4.1 Data Sets

Our experiments in this chapter are conducted on all nine data sets described in Section 3.2.

7.4.2 Simulating Clicks

We use the setup described by Hofmann [77] to simulate user interactions. For details see Section 3.3. In this chapter, we use four instantiations of the *cascade click model* (CCM) as listed in Table 3.2: *perfect*, *navigational*, *informational* and *almost random*.

7.4.3 Experimental Runs

To evaluate the effect of the number of candidates n that are being contrasted in a multileave experiment, both flavors of multileave gradient descent, MGD-W and MGD-M, are run with $n \in \{1, 2, 6, 9, 20\}$. We included $n = 9$ to capture the case where all documents in the top top $\kappa = 10$ come from different rankers. We write MGD-W-n (MGD-M-n) to indicate settings in which we run MGD-W (MGD-M) with n candidates.

In our experiments we contrast the performance of MGD-W and MGD-M with each other as well as with the DBGD baseline. A run of both MGD methods with $n = 1$ is included to verify wether this setting has no significant difference with DBGD. All but one of our experiments consist of 1,000 iterations (i.e., simulated user impressions). We repeat each experiment 25 times on each data fold resulting in 125 runs over each data set. We run one set experiment with many more iterations to test convergence. This experiment was run with 100,000 query impressions and the same number of repetitions. In total, our results are based on over 86M simulated query impressions.

The parameters of the MGD algorithm are set according to the current standard for DBGD [207]. Accordingly, the candidates were generated with $\delta = 1$, updates for DBGD were performed with $\alpha = 0.01$, and zeros used for initialization of $\mathbf{w}_0^0$. For MGD we increased the learning rate to $\alpha = 0.03$ by tuning it on *NP2003* (see Section 7.5.4).

7.4.4 Evaluation

As evaluation metric, we use *normalized discounted cumulative gain* (nDCG) [89] on the top 10 results, measured on the test sets, following, for instance, [179, 185] (see Section 3.4). For each learning experiment, for each data set, we run the experiment for 1,000 interactions with the click model.

We measure *offline* performance by computing the average nDCG score of the *current best ranker* over a held-out set. Furthermore, since the user experience with MGD may be inferior to the existing DBGD algorithm, *online* performance is also assessed, by computing the cumulative nDCG over the results shown to the user.

To verify whether differences are statistically significantly different, a two tailed Student's t-test is used both for the offline and the the online condition.

Table 7.1: Offline score (nDCG) after 1,000 query impressions of each of the algorithms for the 3 instantiations of the DCM [65] (see Table 3.2). Bold values indicate maximum performance per data set and click model. Statistically significant improvements (losses) over the DBGD baseline are indicated by $^{\triangle}$ ($p < 0.05$) and $^{\blacktriangle}$ ($p < 0.01$) ($^{\triangledown}$ and $^{\blacktriangledown}$).

	HP2003	NP2003	TD2003	HP2004	NP2004	TD2004	MQ2007	MQ2008	OHSUMED
perfect click model									
DBGD	0.766	0.710	0.299	0.730	0.715	0.303	0.381	0.476	0.443
MGD-W-2	0.771	0.705	0.314	0.731	0.726	0.306	0.392 $^{\blacktriangle}$	0.480	0.445
MGD-W-4	0.771	0.712	0.318	0.742	**0.732**	0.310	0.396 $^{\blacktriangle}$	0.481	**0.447**
MGD-W-6	0.778	0.712	0.314	0.745	0.725	0.308	0.398 $^{\blacktriangle}$	0.479	0.444
MGD-W-9	0.774	0.713	0.314	0.744	0.725	0.311	0.400 $^{\blacktriangle}$	0.481	0.430 $^{\triangledown}$
MGD-W-20	0.776	0.710	0.314	**0.749** $^{\triangle}$	0.726	0.308	0.396 $^{\blacktriangle}$	0.480	0.438
MGD-M-2	0.771	0.712	0.312	0.743	0.730	0.311	0.392 $^{\blacktriangle}$	0.480	0.443
MGD-M-4	0.777	0.711	0.317	0.742	0.729	0.315 $^{\blacktriangle}$	0.400 $^{\blacktriangle}$	0.482	**0.447**
MGD-M-6	0.779	**0.716**	0.320	0.747 $^{\triangle}$	0.725	0.312 $^{\triangle}$	0.402 $^{\blacktriangle}$	0.481	**0.447**
MGD-M-9	**0.780**	0.714	**0.322** $^{\triangle}$	0.747 $^{\triangle}$	0.726	0.311	0.406 $^{\blacktriangle}$	**0.484**	0.437
MGD-M-20	0.777	0.714	0.321 $^{\triangle}$	0.747 $^{\triangle}$	0.724	**0.316** $^{\blacktriangle}$	**0.408** $^{\blacktriangle}$	**0.484**	0.446
navigational click model									
DBGD	0.725	0.672	0.281	0.676	0.693	0.281	0.370	0.460	0.433
MGD-W-2	0.766 $^{\blacktriangle}$	0.702 $^{\blacktriangle}$	0.306 $^{\triangle}$	0.732 $^{\blacktriangle}$	0.715 $^{\triangle}$	0.303 $^{\blacktriangle}$	0.372	0.466	**0.438**
MGD-W-4	0.769 $^{\blacktriangle}$	0.708 $^{\blacktriangle}$	0.314 $^{\blacktriangle}$	0.735 $^{\blacktriangle}$	0.720 $^{\blacktriangle}$	**0.307** $^{\blacktriangle}$	0.380 $^{\blacktriangle}$	0.469	0.437
MGD-W-6	**0.772** $^{\blacktriangle}$	0.705 $^{\blacktriangle}$	0.312 $^{\blacktriangle}$	0.738 $^{\blacktriangle}$	0.721 $^{\blacktriangle}$	0.304 $^{\blacktriangle}$	0.382 $^{\blacktriangle}$	0.468	0.431
MGD-W-9	0.771 $^{\blacktriangle}$	0.708 $^{\blacktriangle}$	0.304 $^{\triangle}$	0.738 $^{\blacktriangle}$	**0.725** $^{\blacktriangle}$	0.304 $^{\blacktriangle}$	0.388 $^{\blacktriangle}$	0.470 $^{\triangle}$	0.431
MGD-W-20	0.771 $^{\blacktriangle}$	0.710 $^{\blacktriangle}$	0.314 $^{\blacktriangle}$	0.738 $^{\blacktriangle}$	0.721 $^{\blacktriangle}$	0.304 $^{\blacktriangle}$	0.386 $^{\blacktriangle}$	0.470	0.432
MGD-M-2	0.766 $^{\blacktriangle}$	0.703 $^{\blacktriangle}$	0.302	0.726 $^{\blacktriangle}$	0.717 $^{\triangle}$	0.301 $^{\blacktriangle}$	0.376	0.467	0.435
MGD-M-4	0.768 $^{\blacktriangle}$	0.705 $^{\blacktriangle}$	0.312 $^{\blacktriangle}$	0.738 $^{\blacktriangle}$	0.721 $^{\blacktriangle}$	0.305 $^{\blacktriangle}$	0.385 $^{\blacktriangle}$	0.468	0.435
MGD-M-6	**0.772** $^{\blacktriangle}$	0.707 $^{\blacktriangle}$	0.309 $^{\blacktriangle}$	0.736 $^{\blacktriangle}$	0.723 $^{\blacktriangle}$	0.305 $^{\blacktriangle}$	0.387 $^{\blacktriangle}$	**0.473** $^{\blacktriangle}$	0.437
MGD-M-9	0.771 $^{\blacktriangle}$	0.710 $^{\blacktriangle}$	0.317 $^{\blacktriangle}$	**0.741** $^{\blacktriangle}$	0.724 $^{\blacktriangle}$	0.302 $^{\blacktriangle}$	**0.391** $^{\blacktriangle}$	0.472 $^{\triangle}$	0.436
MGD-M-20	0.769 $^{\blacktriangle}$	**0.711** $^{\blacktriangle}$	**0.318** $^{\blacktriangle}$	**0.741** $^{\blacktriangle}$	0.720 $^{\blacktriangle}$	0.306 $^{\blacktriangle}$	0.390 $^{\blacktriangle}$	0.472 $^{\triangle}$	0.437
informational click model									
DBGD	0.460	0.418	0.167	0.401	0.489	0.197	0.323	0.419	0.407
MGD-W-2	0.677 $^{\blacktriangle}$	0.585 $^{\blacktriangle}$	0.223 $^{\blacktriangle}$	0.625 $^{\blacktriangle}$	0.629 $^{\blacktriangle}$	0.233 $^{\blacktriangle}$	0.338 $^{\triangle}$	0.427	0.418
MGD-W-4	0.722 $^{\blacktriangle}$	0.636 $^{\blacktriangle}$	0.253 $^{\blacktriangle}$	0.667 $^{\blacktriangle}$	0.662 $^{\blacktriangle}$	0.258 $^{\blacktriangle}$	0.343 $^{\blacktriangle}$	0.440 $^{\blacktriangle}$	0.422 $^{\triangle}$
MGD-W-6	0.727 $^{\blacktriangle}$	0.652 $^{\blacktriangle}$	0.249 $^{\blacktriangle}$	0.674 $^{\blacktriangle}$	0.669 $^{\blacktriangle}$	0.272 $^{\blacktriangle}$	0.344 $^{\blacktriangle}$	0.441 $^{\blacktriangle}$	0.423 $^{\triangle}$
MGD-W-9	0.727 $^{\blacktriangle}$	0.656 $^{\blacktriangle}$	0.264 $^{\blacktriangle}$	0.679 $^{\blacktriangle}$	0.670 $^{\blacktriangle}$	0.259 $^{\blacktriangle}$	0.339 $^{\blacktriangle}$	0.434 $^{\triangle}$	0.418
MGD-W-20	0.729 $^{\blacktriangle}$	0.649 $^{\blacktriangle}$	0.252 $^{\blacktriangle}$	0.675 $^{\blacktriangle}$	0.664 $^{\blacktriangle}$	0.260 $^{\blacktriangle}$	0.341 $^{\blacktriangle}$	0.434 $^{\triangle}$	0.415
MGD-M-2	0.696 $^{\blacktriangle}$	0.606 $^{\blacktriangle}$	0.241 $^{\blacktriangle}$	0.634 $^{\blacktriangle}$	0.645 $^{\blacktriangle}$	0.246 $^{\blacktriangle}$	0.333	0.430	0.421 $^{\triangle}$
MGD-M-4	0.736 $^{\blacktriangle}$	0.659 $^{\blacktriangle}$	0.276 $^{\blacktriangle}$	0.685 $^{\blacktriangle}$	0.682 $^{\blacktriangle}$	0.271 $^{\blacktriangle}$	0.350 $^{\blacktriangle}$	0.443 $^{\blacktriangle}$	0.427 $^{\blacktriangle}$
MGD-M-6	0.742 $^{\blacktriangle}$	0.667 $^{\blacktriangle}$	0.275 $^{\blacktriangle}$	0.692 $^{\blacktriangle}$	0.687 $^{\blacktriangle}$	0.278 $^{\blacktriangle}$	0.351 $^{\blacktriangle}$	0.448 $^{\blacktriangle}$	0.427 $^{\blacktriangle}$
MGD-M-9	0.745 $^{\blacktriangle}$	**0.681** $^{\blacktriangle}$	0.283 $^{\blacktriangle}$	**0.710** $^{\blacktriangle}$	0.698 $^{\blacktriangle}$	0.284 $^{\blacktriangle}$	**0.361** $^{\blacktriangle}$	**0.454** $^{\blacktriangle}$	0.425 $^{\blacktriangle}$
MGD-M-20	**0.752** $^{\blacktriangle}$	0.677 $^{\blacktriangle}$	**0.295** $^{\blacktriangle}$	0.703 $^{\blacktriangle}$	**0.703** $^{\blacktriangle}$	**0.291** $^{\blacktriangle}$	0.356 $^{\blacktriangle}$	0.452 $^{\blacktriangle}$	**0.430** $^{\blacktriangle}$

Table 7.2: Online score (discounted cumulative nDCG, see Section 7.4.4) for the 3 instantiations of the DCM [65] (see Table 3.2). Bold values indicate maximum performance per data set and click model. Statistically significant improvements (losses) over the DBGD baseline are indicated by $^{\triangle}$ ($p < 0.05$) and $^{\blacktriangle}$ ($p < 0.01$) ($^{\triangledown}$ and $^{\blacktriangledown}$).

	HP2003	NP2003	TD2003	HP2004	NP2004	TD2004	MQ2007	MQ2008	OHSUMED
perfect click model									
DBGD	95.88	97.79	36.28	97.93	102.92	**42.92**	60.11	78.17	70.43
MGD-W-2	110.77 $^{\blacktriangle}$	101.71 $^{\blacktriangle}$	41.19 $^{\blacktriangle}$	100.92	106.69 $^{\blacktriangle}$	38.15 $^{\blacktriangledown}$	60.49	78.79	72.58 $^{\blacktriangle}$
MGD-W-4	112.96 $^{\blacktriangle}$	103.42 $^{\blacktriangle}$	42.36 $^{\blacktriangle}$	104.44 $^{\blacktriangle}$	108.94 $^{\blacktriangle}$	38.50 $^{\blacktriangledown}$	61.33 $^{\triangle}$	78.52	72.73 $^{\blacktriangle}$
MGD-W-6	113.37 $^{\blacktriangle}$	104.13 $^{\blacktriangle}$	43.00 $^{\blacktriangle}$	104.79 $^{\blacktriangle}$	110.02 $^{\blacktriangle}$	38.13 $^{\blacktriangledown}$	61.22 $^{\triangle}$	78.76	72.73 $^{\blacktriangle}$
MGD-W-9	114.66 $^{\blacktriangle}$	**105.79** $^{\blacktriangle}$	43.53 $^{\blacktriangle}$	**107.22** $^{\blacktriangle}$	110.27 $^{\blacktriangle}$	38.39 $^{\blacktriangledown}$	60.62	78.12	70.32
MGD-W-20	**116.25** $^{\blacktriangle}$	104.96 $^{\blacktriangle}$	44.43 $^{\blacktriangle}$	106.23 $^{\blacktriangle}$	109.94 $^{\blacktriangle}$	39.79 $^{\triangledown}$	60.28	78.07	72.42 $^{\blacktriangle}$
MGD-M-2	111.23 $^{\blacktriangle}$	101.42 $^{\blacktriangle}$	41.26 $^{\blacktriangle}$	101.67 $^{\triangle}$	108.24 $^{\blacktriangle}$	37.51 $^{\blacktriangledown}$	61.43 $^{\triangle}$	79.08	72.74 $^{\blacktriangle}$
MGD-M-4	113.91 $^{\blacktriangle}$	103.48 $^{\blacktriangle}$	42.64 $^{\blacktriangle}$	103.58 $^{\blacktriangle}$	109.70 $^{\blacktriangle}$	38.39 $^{\blacktriangledown}$	**61.85** $^{\blacktriangle}$	**79.11**	**72.84** $^{\blacktriangle}$
MGD-M-6	113.46 $^{\blacktriangle}$	104.25 $^{\blacktriangle}$	43.22 $^{\blacktriangle}$	105.31 $^{\blacktriangle}$	109.80 $^{\blacktriangle}$	38.68 $^{\blacktriangledown}$	61.00	78.97	72.78 $^{\blacktriangle}$
MGD-M-9	115.81 $^{\blacktriangle}$	105.09 $^{\blacktriangle}$	44.02 $^{\blacktriangle}$	106.79 $^{\blacktriangle}$	**110.88** $^{\blacktriangle}$	38.38 $^{\blacktriangledown}$	60.64	77.88	70.97
MGD-M-20	115.67 $^{\blacktriangle}$	104.75 $^{\blacktriangle}$	**44.50** $^{\blacktriangle}$	107.05 $^{\blacktriangle}$	110.51 $^{\blacktriangle}$	40.18 $^{\triangledown}$	61.57 $^{\blacktriangle}$	78.69	72.51 $^{\blacktriangle}$
navigational click model									
DBGD	78.79	85.83	32.21	80.61	90.92	37.46	58.07	76.04	66.99
MGD-W-2	105.53 $^{\blacktriangle}$	95.55 $^{\blacktriangle}$	38.54 $^{\blacktriangle}$	92.59 $^{\blacktriangle}$	100.96 $^{\blacktriangle}$	35.72	59.24 $^{\triangle}$	77.18	71.34 $^{\blacktriangle}$
MGD-W-4	110.07 $^{\blacktriangle}$	100.22 $^{\blacktriangle}$	41.58 $^{\blacktriangle}$	100.37 $^{\blacktriangle}$	105.81 $^{\blacktriangle}$	37.90	59.57 $^{\triangle}$	78.18 $^{\blacktriangle}$	72.00 $^{\blacktriangle}$
MGD-W-6	109.97 $^{\blacktriangle}$	101.36 $^{\blacktriangle}$	41.66 $^{\blacktriangle}$	101.00 $^{\blacktriangle}$	107.07 $^{\blacktriangle}$	38.21	60.18 $^{\blacktriangle}$	77.88 $^{\blacktriangle}$	72.62 $^{\blacktriangle}$
MGD-W-9	**113.17** $^{\blacktriangle}$	101.71 $^{\blacktriangle}$	43.03 $^{\blacktriangle}$	102.54 $^{\blacktriangle}$	106.63 $^{\blacktriangle}$	39.04	**60.71** $^{\blacktriangle}$	77.91 $^{\blacktriangle}$	72.69 $^{\blacktriangle}$
MGD-W-20	112.18 $^{\blacktriangle}$	101.85 $^{\blacktriangle}$	42.24 $^{\blacktriangle}$	102.82 $^{\blacktriangle}$	107.02 $^{\blacktriangle}$	39.28	60.55 $^{\blacktriangle}$	77.77 $^{\blacktriangle}$	72.39 $^{\blacktriangle}$
MGD-M-2	106.27 $^{\blacktriangle}$	94.34 $^{\blacktriangle}$	39.21 $^{\blacktriangle}$	94.70 $^{\blacktriangle}$	103.34 $^{\blacktriangle}$	36.40	59.65 $^{\blacktriangle}$	77.72 $^{\blacktriangle}$	71.04 $^{\blacktriangle}$
MGD-M-4	109.44 $^{\blacktriangle}$	99.22 $^{\blacktriangle}$	41.02 $^{\blacktriangle}$	99.01 $^{\blacktriangle}$	105.82 $^{\blacktriangle}$	38.09	60.25 $^{\blacktriangle}$	78.16 $^{\blacktriangle}$	72.49 $^{\blacktriangle}$
MGD-M-6	110.70 $^{\blacktriangle}$	100.56 $^{\blacktriangle}$	42.45 $^{\blacktriangle}$	102.04 $^{\blacktriangle}$	106.04 $^{\blacktriangle}$	38.28	60.31 $^{\blacktriangle}$	77.84 $^{\blacktriangle}$	**72.79** $^{\blacktriangle}$
MGD-M-9	112.30 $^{\blacktriangle}$	**102.95** $^{\blacktriangle}$	42.74 $^{\blacktriangle}$	**103.96** $^{\blacktriangle}$	**107.41** $^{\blacktriangle}$	**39.55**	60.36 $^{\blacktriangle}$	77.89 $^{\blacktriangle}$	72.71 $^{\blacktriangle}$
MGD-M-20	111.38 $^{\blacktriangle}$	102.23 $^{\blacktriangle}$	**43.10** $^{\blacktriangle}$	102.88 $^{\blacktriangle}$	106.78 $^{\blacktriangle}$	38.79	60.21 $^{\blacktriangle}$	**78.23** $^{\blacktriangle}$	72.53 $^{\blacktriangle}$
informational click model									
DBGD	48.23	50.42	22.46	43.36	59.58	27.76	55.60	71.94	62.99
MGD-W-2	72.76 $^{\blacktriangle}$	64.09 $^{\blacktriangle}$	25.94 $^{\blacktriangle}$	62.61 $^{\blacktriangle}$	69.48 $^{\blacktriangle}$	26.52	55.83	73.33 $^{\triangle}$	66.39 $^{\blacktriangle}$
MGD-W-4	78.49 $^{\blacktriangle}$	73.02 $^{\blacktriangle}$	29.34 $^{\blacktriangle}$	70.73 $^{\blacktriangle}$	78.97 $^{\blacktriangle}$	28.21	56.04	74.31 $^{\blacktriangle}$	67.70 $^{\blacktriangle}$
MGD-W-6	81.96 $^{\blacktriangle}$	73.66 $^{\blacktriangle}$	30.84 $^{\blacktriangle}$	70.90 $^{\blacktriangle}$	81.44 $^{\blacktriangle}$	29.12	56.24	**74.86** $^{\blacktriangle}$	67.75 $^{\blacktriangle}$
MGD-W-9	85.14 $^{\blacktriangle}$	77.57 $^{\blacktriangle}$	30.26 $^{\blacktriangle}$	73.60 $^{\blacktriangle}$	82.86 $^{\blacktriangle}$	28.45	56.09	73.64 $^{\triangle}$	68.48 $^{\blacktriangle}$
MGD-W-20	84.44 $^{\blacktriangle}$	74.65 $^{\blacktriangle}$	29.91 $^{\blacktriangle}$	69.06 $^{\blacktriangle}$	81.78 $^{\blacktriangle}$	28.56	56.18	73.09	67.51 $^{\blacktriangle}$
MGD-M-2	70.70 $^{\blacktriangle}$	66.03 $^{\blacktriangle}$	27.33 $^{\blacktriangle}$	61.78 $^{\blacktriangle}$	71.30 $^{\blacktriangle}$	27.29	56.28	73.35 $^{\triangle}$	67.15 $^{\blacktriangle}$
MGD-M-4	84.38 $^{\blacktriangle}$	76.41 $^{\blacktriangle}$	28.98 $^{\blacktriangle}$	72.82 $^{\blacktriangle}$	82.12 $^{\blacktriangle}$	28.44	56.58	74.45 $^{\blacktriangle}$	**68.75** $^{\blacktriangle}$
MGD-M-6	85.51 $^{\blacktriangle}$	76.37 $^{\blacktriangle}$	31.55 $^{\blacktriangle}$	73.52 $^{\blacktriangle}$	83.05 $^{\blacktriangle}$	28.59	**56.88**	74.26 $^{\blacktriangle}$	67.97 $^{\blacktriangle}$
MGD-M-9	**87.84** $^{\blacktriangle}$	**81.04** $^{\blacktriangle}$	**32.45** $^{\blacktriangle}$	75.81 $^{\blacktriangle}$	**86.05** $^{\blacktriangle}$	**30.21** $^{\triangle}$	55.91	74.50 $^{\blacktriangle}$	68.65 $^{\blacktriangle}$
MGD-M-20	86.73 $^{\blacktriangle}$	80.99 $^{\blacktriangle}$	32.07 $^{\blacktriangle}$	**76.72** $^{\blacktriangle}$	82.92 $^{\blacktriangle}$	29.68	56.57	73.99 $^{\blacktriangle}$	68.50 $^{\blacktriangle}$

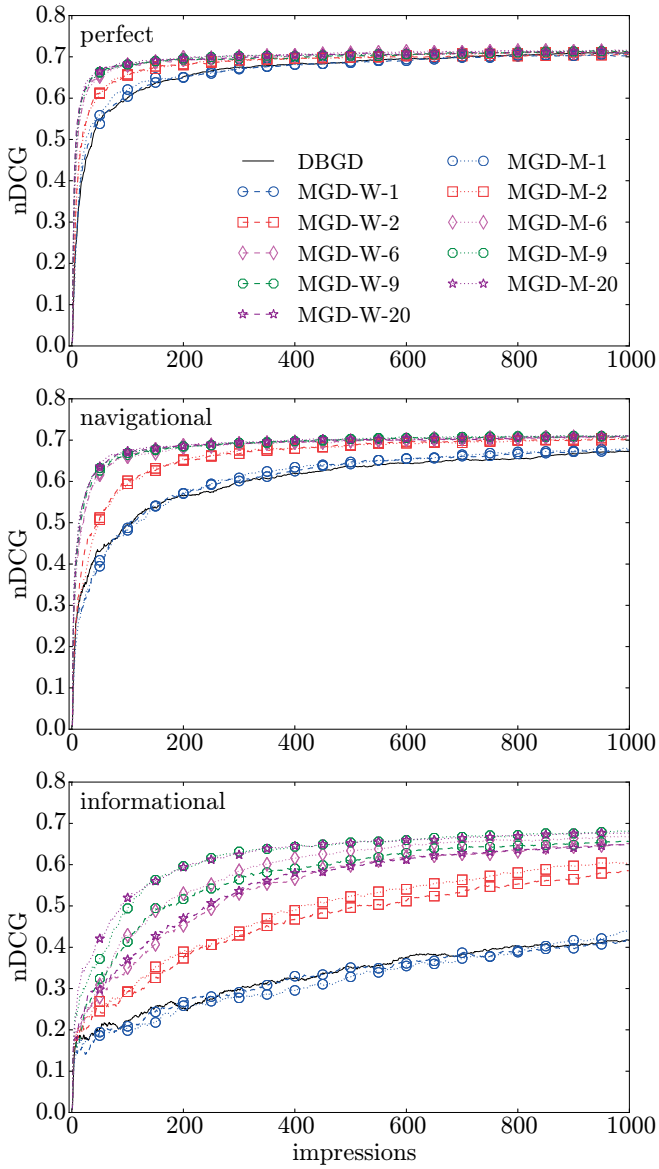

Figure 7.1: Offline performance (nDCG) on MGD-W and MGD-M with varying number of candidates compared to DBGD on the *NP2003* data set for the *perfect*, *navigational* and *informational* click model.

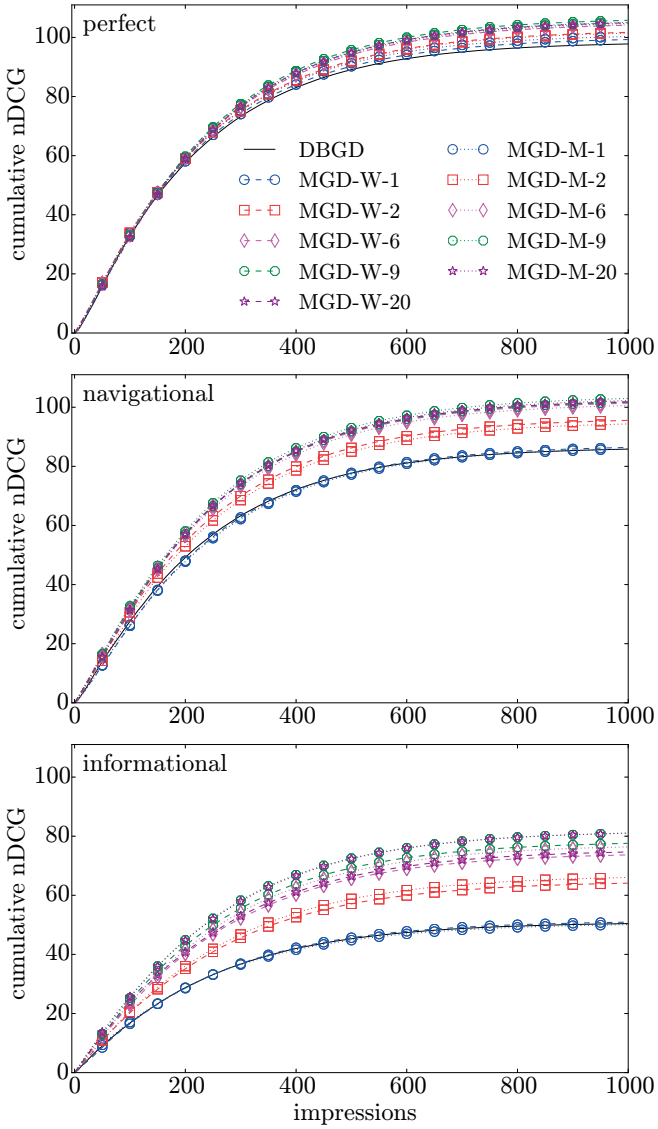

Figure 7.2: Online performance (discounted cumulative nDCG) on MGD-W and MGD-M with varying number of candidates compared to DBGD on the *NP2003* data set for *perfect*, *navigational* and *informational* click model instantiations.

7.5 Results and Analysis

In this section we present the results of our experiments and answer the research questions posed in Section 7.1. Furthermore, in Section 7.5.4, we investigate the effect of n, the number of candidates, and α, the learning rate.

7.5.1 Learning Speed

We start by answering RQ10: whether MGD learns faster than DBGD. The plots in Figure 7.1 show how offline performance, measured as nDCG on a held-out fold, increases as the learning methods observe more queries. These plots are based only on queries from *NP2003* and are illustrative of performance on all other data sets. We see that when n, the number of candidates that are being multileaved, increases, offline performance of both MGD-M-n and MGD-W-n improves monotonically. Furthermore, systems with more candidates learn much faster.

In the case of *perfect* feedback, there is less of an effect as there is less to gain over an already well performing baseline. But when the noise in user feedback increases, the advantage of MGD over DBGD becomes stronger. Interestingly, for $n = 20$, the MGD methods obtain an nDCG value on *informational* feedback that is close to the converged performance on *perfect* feedback. In other words, the inclusion of more candidates counters the noise introduced by the click model. In Table 7.1 we see the same effect for all data sets: generally, under *perfect* feedback converged performance does not change much; however, if the feedback is noisier, then the more candidates are added, the more MGD improves over the baseline. Offline performance for MGD goes up dramatically for noisy feedback compared to the baseline and the standard deviation (between brackets in the table) drops dramatically. This indicates much more stable performance for MGD. As a sanity check, we see in Figure 7.1 that both MGD-W-1 and MGD-M-1 perform very close to the DBGD baseline. This is to be expected because, besides their learning rates, both methods are algorithmically identical to the baseline for $n = 1$.

Offline performance, however, only tells half the story in an online learning to rank setting. Users are exposed to interleaved and multileaved lists that are used by the systems to infer preferences. Since the quality of these lists may vary, it is critical to measure the impact on users. Note that the quality of these lists varies due to a combination of two factors: the quality of the rankers learned so far and the impact of the interleaving or multileaving method. We measure online performance by computing the nDCG score of the lists that users observe and discounting it over time (see Section 7.4.4). The online and offline metrics together thus tell us how the system dealt with the *exploration-exploitation trade off* [178]. Figure 7.2 displays the online performance for all systems, again on a single data set. Just like with offline performance, MGD outperforms DBGD more when the noise in the feedback increases. In fact, Table 7.2 shows that under *perfect* feedback, online performance for one data set actually decreases compared to the baseline. This suggests that, for feedback without noise, increasing exploration, which is a direct consequence of adding candidates, is not as helpful for maximizing online performance. In other words, while adding candidates increases offline performance, in the absence of feedback noise it may harm online performance through the introduction of excessive exploration. However, generally, whether there is noise in the click feedback or not, MGD

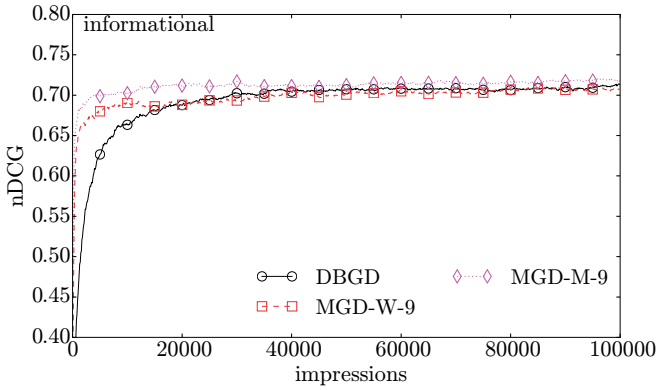

Figure 7.3: Offline performance (nDCG) on MGD-W and MGD-M with 9 candidates compared to DBGD on the *NP2003* data set an *informational* click model.

outperforms DBGD. Also for online performance, the standard deviation of MGD is much lower than for DBGD, irrespective of the noise.

Our answer to RQ10 is thus that MGD with increasing numbers of candidates learns increasingly faster than DBGD in terms of offline performance. In terms of online performance, MGD is on par with or outperforms DBGD when feedback has more realistic levels of noise.

7.5.2 Convergence

In this section, we answer RQ11: whether MGD converges to a better optimum than DBGD. To do so, we investigate converged offline performance only. Table 7.1 shows converged performance after 1,000 query impressions for all the data sets that we consider. Note that for *NP2003* these values correspond to the points on the right vertical axis of Figure 7.1. These graphs are also illustrative of all data sets and show that increasing the number of candidates results in a better converged offline performance. In the case of *perfect* feedback, the effect of the number of candidates is not very strong; for most data sets and algorithms no significant improvement over the baseline is apparent. However, when noise in the feedback increases, and thus when feedback becomes more realistic compared to the *perfect* instantiation, the effect becomes much stronger as more candidates are used. In general, as Table 7.1 shows, this effect is significant and substantial as soon as more than one candidate is used. For many data sets, performance of MGD after 1,000 query impressions is almost on par with DBGD trained without noise. However, as Figure 7.1 and Table 7.1 show, MGD with enough candidates *always* outperforms DBGD after 1,000 queries.

The graphs in Figure 7.1 clearly suggest that not all systems converge within 1,000 impressions. For this reason, we ran an additional longer experiment with 100,000 queries with *informational* feedback. Figure 7.3 shows the results with the same setup as in Figure 7.1 but over a larger number of queries. The graph shows that even after 100,000 queries DBGD has not converged, and MGD still performs better. Nonetheless, the

Table 7.3: Performance, in terms of nDCG and discounted cumulative nDCG, of MGD-M, MGD-W and normalized MGD-M each with 9 candidates for the three instantiations of the *dependent click model* (DCM) [65] (see Table 3.2). Run on the *NP2003* data set; performance evaluated after 1,000 impressions.

		perfect	***navigational***	***informational***
offline	**MGD-W**	0.713 (0.05)	0.708 (0.05)	0.656 (0.07)
	MGD-M	0.714 (0.05)	0.710 (0.05)	0.681 (0.06)
	Norm MGD-M	0.711 (0.04)	0.710 (0.04)	0.667 (0.06)
online	**MGD-W**	105.785 (5.88)	101.708 (5.19)	77.568 (10.24)
	MGD-M	105.087 (4.71)	102.953 (5.62)	81.037 (8.74)
	Norm MGD-M	105.844 (5.21)	102.686 (5.32)	81.676 (9.46)

difference between the algorithms decreases over time, until they converge to a similar level of performance. Thus, both algorithms seem to converge to the same optimum but DBGD requires many more queries than MGD to do so.

Hence, we answer RQ11 as follows: MGD converges to an optimum which is at least as good as the optimum DBGD finds. However, on the data sets that we have examined, MGD does so much faster, as shown in Section 7.5.1.

7.5.3 Comparing Outcome Interpretations

In this section we answer RQ12: how MGD-W and MGD-M compare to each other. Figure 7.1, which shows the learning curves of both methods for varying click models, indicates that, in terms of offline performance, there is no substantial difference between MGD-W and MGD-M for the *perfect* and *navigational* click models. However, for the *informational* click model, which has noisier feedback, MGD-M consistently outperforms MGD-W. The same applies to all data sets we considered (see Table 7.1). In the offline setting, MGD-M is the better approach as it is more capable of handling noise than MGD-W.

In terms of *online* performance, MGD-M also usually outperforms MGD-W (see Figure 7.2 and Table 7.2). Again, the effect is stronger when there is more noise in the feedback. Generally, MGD-M has lower standard deviation than MGD-W indicating that it is more stable.

Note that the *informational* click model has a high probability to produce multiple clicks because its stop probabilities are low (see Table 3.2). This typically leads to multiple winners of a TDM comparison, which in turn allows MGD-M to be different from MGD-W. Thus, a potential reason for MGD-M to outperform MGD-W is that the mean of several unit vectors is shorter than a unit vector. As a result, MGD-M updates the current best weight vector with smaller steps. In other words, for DBGD and MGD-W we hypothesize that $|w_t^0 - w_{t+1}^0| = \alpha \cdot \delta$, while for MGD-M $|w_t^0 - w_{t+1}^0| \leq \alpha \cdot \delta$.

We tested this hypothesis by normalizing the mean vector to a unit vector before updating using MGD-M. Algorithm 12 was effectively changed such that $|w_t^0 - w_{t+1}^0| = \alpha \cdot \delta$. The result is depicted in Table 7.3, where we see how MGD-M with normalized

update directions indeed performs slightly worse than MGD-M without normalization for the *informational* click model in terms of offline performance, confirming that some of its advantage indeed comes from the smaller update step. Nonetheless, MGD-M with normalization still either performs on par with or better than MGD-W. Not all of its performance advantage can be attributed to smaller updates. This implies that the *direction* of the update taken by MGD-M is better than that of MGD-W.

To answer RQ12, while in general both MGD methods outperform DBGD, MGD-M is better at handling high noise levels, making it more effective than MGD-W overall. The advantage of MGD-M over MGD-W comes from both the update direction and a smaller update size.

7.5.4 Number of Candidates and Learning Rate

In this section, we investigate some remaining questions.

Number of Candidates

In Section 7.5.1 we have already discussed the interplay between the amount of noise in the feedback and the optimal number of candidates in MGD. Figure 7.4 shows the effect of increasing the number of candidates even further to a maximum of 1,000 candidates. Note that as soon as the number of candidate rankers goes beyond the length of the result shown to users, the only effect of increasing it even further is that the probability of including the current best ranker decreases.[2] We see in Figure 7.4, Table 7.1 and Table 7.2 that *both* offline and online performance generally go up when the number of candidates goes up. However, beyond approximately 10 candidates this either stabilizes or fluctuates slightly, depending on the amount of noise in the click model. This matches $\kappa = 10$, the result list length in our experiments. We increase the noise further than we did until now by including results for an *almost random* click model instantiation. Still in Figure 7.4 (the green curves near the bottom in both plots), we see that the more noise we add, the more MGD benefits from adding candidates.

In conclusion, both offline and online performance increase with the number of candidates when noise is present, but this effect appears to be limited by the length of the result list shown to users.

Learning Rate

Our MGD algorithms are sufficiently different from DBGD to warrant a new investigation of the learning rate α. The results in Section 7.5.3 suggest that some of MGD-M's superior performance over MGD-W could be explained by the smaller steps this algorithm takes. To further investigate this effect, we vary the learning rate.

Figure 7.5, which shows a sweep over learning rates, again shows a considerable difference between MGD-M and MGD-W. Furthermore, for most algorithms online performance increases when α goes up while offline performance drops slowly. With a learning rate close to zero, MGD performs notably worse than DBGD because multileaving

[2] This is an artifact of the way we generate candidates and the fact that we use TDM as our multileaving method.

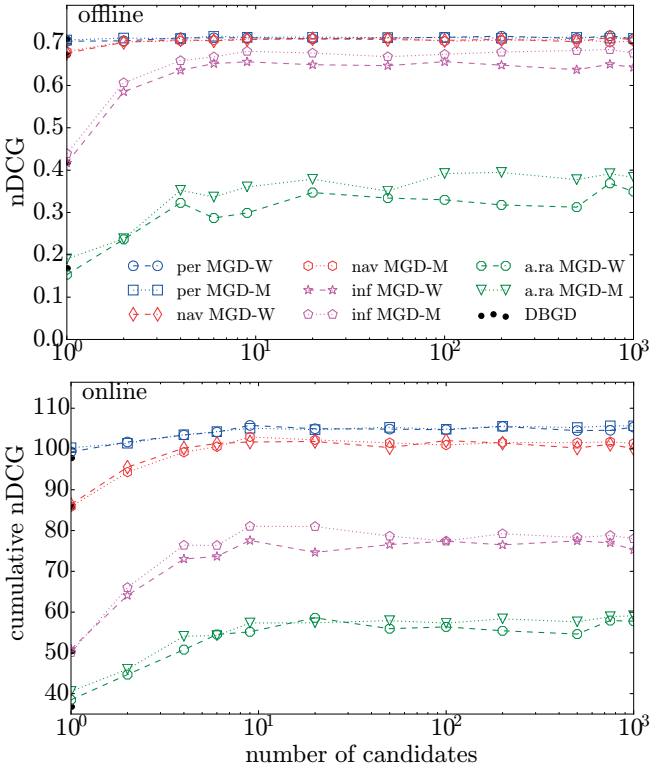

Figure 7.4: Sweep over the number of candidates in terms of offline and online performance for MGD-M and MGD-W after 1,000 impressions. Performed on the *NP2003* data set using *all four* instantiations of the click model. DBGD is displayed by the black dots on the left axis. Note the log scale on the horizontal axis.

interferes more with the ranking presented to the user, while the low learning rates prevents it from adapting quickly. Conversely, when the learning rate increases, MGD greatly outperforms DBGD in terms of online performance for all three click models. This illustrates the tradeoff MGD makes: multileaving distorts the ranking shown to the user, but when the learning rate increases it compensates by adapting to the user faster. So, interestingly, also when there is no noise in the feedback, MGD can greatly outperform DBGD if the learning rate is chosen appropriately. Note that, for all our earlier experiments, we chose a fixed value of $\alpha = 0.03$ for MGD based on these plots. This point denotes a reasonable tradeoff between offline and online performance. This is a different optimum than DBGD and, since DBGD is equal to MGD with a single candidate, it seems the optimal learning rate depends on the number of candidates. Ideally, one would find a learning rate that is optimal for each number of candidates. Doing so would only increase MGD's performance advantage.

In sum, this experiment shows that DBGD and MGD have different optimal learning rates and that MGD can greatly outperform DBGD, both offline and online, when the

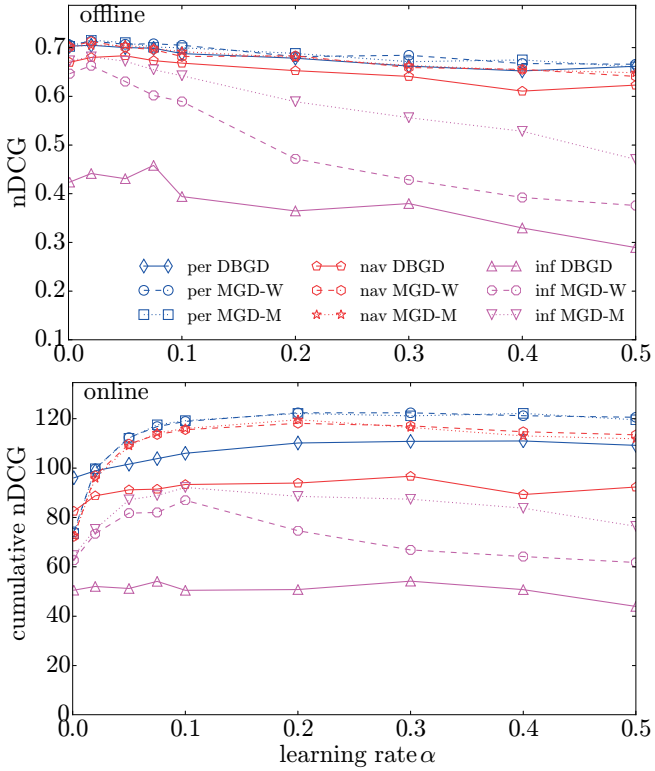

Figure 7.5: Sweep over learning rate values in terms of offline and online performance after 1,000 impressions for MGD-M and MGD-W with 9 candidates and DBGD. Performed on the *NP2003* data set using three different click models with varying degrees of noise.

learning rate is chosen appropriately.

7.6 Conclusion

We proposed an extension of *dueling bandit gradient descent* (DBGD), an online learning to rank method. DBGD is limited to exploring only a single candidate ranker at a time. Where DBGD uses interleaved comparisons to infer *pairwise* preferences, our newly introduced method—*multileave gradient descent* (MGD)—learns from comparisons between a set of rankers to infer n-*way* preferences between n candidate ranker improvements. We proposed two specific ways of using these preferences for updating a current best ranker. The first variant, MGD-W, picks a ranker to update towards at random from among the rankers that win a comparison; the second variant, MGD-M, updates towards the mean of all winners of the comparison.

Our empirical results, based on extensive experiments on nine learning to rank data sets encompassing 86M user interactions, show that either variant dramatically improves

over the DBGD baseline. In particular, when the noise in user feedback increases, we find that both MGD-W and MGD-M are capable of learning better rankers much faster than the baseline does. When the number of candidate rankers we consider increases from 1 (as in the baseline), offline performance—measured on held-out data—and online performance—measured on the results shown to users—consistently go up until it converges at around 10 candidate rankers. After 1,000 query impressions with *noisy* feedback, MGD performs almost on par with DBGD trained on feedback *without any noise*. We further show that MGD obtains at least the same converged performance that DBGD ultimately obatins, but that it does so using orders of magnitude less user interaction data. From the two variants we compared, MGD-M performs either equal to, or outperforms MGD-W. The advantage of MGD-M over MGD-W comes from both the update direction and smaller update size.

An important implication of our results is that orders of magnitude less user interaction data is required to find good rankers when multileaved comparisons are used as feedback mechanism for online learning to rank. This results in far fewer users being exposed to inferior rankers and it allows search engines to adapt faster to changes in user preferences.

7.7 Future Work

Our findings give rise to several directions that remain to be explored.

Firstly, we sampled candidate rankers randomly uniformly from a unit sphere around the current best ranker. Alternatively, one could consider selecting rankers such that all directions are covered, which may speed up learning even further.

Secondly, currently we have two strategies for interpreting multileave comparison outcomes, MGD-M and MGD-W. We could consider an additional strategy that takes a weighted combination of all the compared rankers, potentially even down weighting the directions for loosing candidate rankers.

Thirdly, we noticed that often, in particular closer to convergence, many of the compared rankers become very similar. One could consider adapting the multileaving algorithm to not attempt to infer preferences between rankers that produce the same rankings, but rather, consider all these to be the same rankers.

It may seem that increasing the number of candidates increases the amount of required computation. However, when using *team draft multileave* (TDM), the number of rankers that can contribute a document to the multileaved list is bound by its length. If it is decided beforehand which rankers can contribute a document than only these rankers actually have to materialize their rankings.

One could imagine learning from feedback using *probabilistic multileave* (PM), the probabilistic multileave method introduced in Chapter 4, which would not limit the number of rankers that can be compared at once. An initial study of this idea has recently been published by Oosterhuis et al. [138].

Our ideas in this chapter have been experimentally validated using the simulation setup described in Chapter 3 using the framework described in Chapter 8. Validating them on real users using real search engines, for instance using OpenSearch, the methodology described in Chapter 9, is left for future work.

Finally, a theoretical analysis of the convergence properties of MGD and its variants, in comparison to DBGD, would give valuable insights in the broader applicability.

Part III

Resources and Methodology

8 Lerot: Simulating Users

This third part of the thesis is of a different nature than the earlier two parts. This part is more practical and describes two evaluation methodologies and shared resources.

The research presented so far, in Parts I and II of this thesis, was on methods and algorithms for online evaluation and online *learning to rank* (LTR). See Sections 2.3 and 2.5 for background on these two topics. What online evaluation and online LTR have in common is that both require *users* interacting with a search engine. This is something that is not typically available to academic researchers and even researchers or engineers with access to users may not always be willing to try out a new idea by exposing real users to it.

For this purpose, in this chapter, we propose an online evaluation framework that allows for simulating users interacting with a search engine. Most of our experiments, in Chapters 4, 6, and 7, have been conducted using this framework. In this online evaluation framework, implemented in the software package *learning and evaluating rankers online toolkit* (Lerot), we have bundled all ingredients needed for experimenting with online evaluation and learning to rank for *information retrieval* (IR). Lerot includes several online learning algorithms, interleaving methods and a full suite of ways to evaluate these methods. In the absence of real users, the evaluation method bundled in the software package is based on simulations of users interacting with the search engine.

The framework presented in this chapter has been used to verify findings of more than fifteen papers at major information retrieval venues over the last few years [36, 39, 41, 77–79, 81–84, 138, 162, 163, 165, 167, 216]. The experimental setup in the papers that use Lerot has been described in Chapter 3 of this thesis.

This chapter is based on a paper by Schuth, Hofmann, Whiteson, and de Rijke [161].

8.1 Introduction

As discussed in Section 2.4, adapting IR systems to a specific user, group of users, or deployment setting has become possible and popular due to *learning to rank* (LTR) techniques [124]. Generally speaking, a LTR method learns function that maps a document-query pair described by a feature vector to a value that is used to rank documents for a given query. We refer to such a function with instantiated weights as a *ranker*. Most current approaches learn *offline*, i.e., before deployment rankers are estimated from manually annotated training data.

As detailed in Section 2.5, in contrast, an *online* learning to rank method learns directly from interactions with users, e.g., using click feedback. For instance, an online learning to rank approach such as *dueling bandit gradient descent* (DBGD) [82, 207] or *multileave gradient descent* (MGD) (see Chapter 7) aims to find a high quality ranker while interacting with a user. In each step, the current best ranker is perturbed, and then the original and perturbed rankers are compared using an *interleaved comparison method* [144] or using an *multileaved comparison method* (see Chapter 4). The original and candidate rankers are combined and presented to the user, whose clicks determine which ranker wins the comparison. If a perturbed ranker wins, the original ranker is adjusted slightly in its direction. See Section 2.5 for details on these learning algorithms (. These and many more algorithms can only be experimentally validated through interactions with users. However, users are typically not available to academic researchers and even researchers or engineers with access to users may not always be willing to try out a new idea by exposing real users to it.

Lerot, the implementation of the framework presented in this chapter, offers a solution for evaluating and experimenting with online learning to rank algorithms using simulations of users. Simulation experiments make it possible to expose simulated users to arbitrary search results lists, without the risk of adversely affecting the experience of real users in a production system. This allows researchers to try out new ideas that could otherwise not be tried out. Simulation experiments with Lerot may typically complement or precede experimentation in a setup for online learning to rank with real users. In Chapter 9 of this thesis we discuss the scenario that does involve real users.

While there are several other libraries and frameworks for learning to rank such as SVMRank[1] [94], RankLib[2] and Sofia-ml[3] [168], these all focus on *offline* learning to rank. By contrast, Lerot focuses on *online* learning to rank (see Section 2.5). Lerot implements DBGD and extensions of DBGD such as *candidate preselection* (CPS) [83] and MGD (see Chapter 7) in an easily decomposable fashion.

In this chapter, we present all the components that are included in Lerot. The framework has *all batteries included* (except for the data), to replicate experiments; no code needs to be written.

Lerot and its predecessors have been used to verify the findings in numerous publications [36, 39, 41, 77–79, 81–84, 138, 162, 163, 165, 167, 216] at major IR venues. The framework is easily extensible to compare the implemented methods to new online evaluation and online learning approaches.

Our contribution in this chapter is the following.

Implementation We contribute Lerot, an open source implementation of our online learning to rank framework that has all batteries included. The framework is easily extensible to compare the implemented methods to new online evaluation and online learning approaches.

[1] `http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`
[2] `http://people.cs.umass.edu/~vdang/ranklib.html`
[3] `https://code.google.com/p/sofia-ml/`

Listing 8.1: Minimal example of an online learning experiment that uses a list wise learning algorithm and a cascade user model to simulate clicks. Arguments for many classes have been omitted for readability, they are included in the actual example code that is part of the package.

```
import sys, random
import retrieval_system, environment, evaluation,
                                                    query
learner = retrieval_system.ListwiseLearningSystem(
                                                   [...])
user_model = environment.CascadeUserModel([...])
evaluation = evaluation.NdcgEval([...])
train = query.load_queries(sys.argv[1], [...])
test = query.load_queries(sys.argv[2], [...])
while True:
    q = train[random.choice(train.keys())]
    l = learner.get_ranked_list(q)
    c = user_model.get_clicks(l, q.get_labels())
    s = learner.update_solution(c)
    print evaluation.evaluate_all(s, test)
```

8.2 Framework

In broad terms, Lerot can be used to run two types of experiments: *online learning experiments* and *online evaluation experiments*. Learning experiments operate in a continuous space of possible solutions and evolve rankers over time to find an optimal one. Evaluation experiments, on the other hand, operate on a fixed set of rankers and are designed to identify the best ranker among this set using, for instance, interleaved comparisons. Evaluation experiments are discussed at length in Part I of this thesis. Learning experiments are discussed in Part II.

This chapter mostly focuses on describing the learning experiments as they encompass the evaluation component as well.

A minimal example[4] of a learning algorithm embedded in a simulation with a user model is shown in Listing 8.1. The example defines a learner (see Section 8.2.1), a user model (see Section 8.2.3), an evaluation method (see Section 8.2.4), and lists of training and test queries with labels. Potentially, if real users would be available, they would be the source of the training queries and the clicks; we describe a setting where real users are available in Chapter 9. In Lerot, we provide an environment to connect Lerot to the setting we describe in that chapter.

In the absence of users, the queries are sampled from a data set and the clicks are generated by a click model that uses relevance judgements. The queries `q` are observed in a random order, a ranked list `l` is produced by a learning algorithm such as DBGD or MGD. This ranking `l` is sent to the click model and the clicks `c` it produces, in turn, are

[4]This code serves as an illustration (but is included in Lerot in `src/scripts/run-example.py`). A version of this algorithm that also interprets the configuration files explained in Section 8.3.2 is preferred over simple version.

observed by the learner so that it can update the solution. The updated solution `s` is then evaluated on the test queries. In theory, this process continues indefinitely.

8.2.1 Learning Algorithms

The `learner` in Listing 8.1 can be instantiated in many ways. Our framework has implementations for the following algorithms.

1. Learning from document-pairwise feedback [77, 92, 168, 212];
2. Learning from listwise feedback, such as *dueling bandit gradient descent* (DBGD) [207] (see Section 2.5.1); and
3. Extensions of DBGD, such as *candidate preselection* (CPS) [83] which we discuss in Section 2.5.2; and
4. *Multileave gradient descent* (MGD) as introduced in Chapter 7 of this thesis.

All the above learning algorithms have the exact same interface; they implement three functions, two of which are called in our example in Listing 8.1.

- `l = get_ranked_list(q)`
  Returns a list `l` of documents in response to query `q`.
- `s = update_solution(c)`
  Updates the current solution using clicks `c` and returns the updated solution `s`.
- `s = get_solution()`
  Returns the updated solution `s`.

Listing 8.2 shows the implementation of the learning algorithm used by DBGD; we provide pseudocode of this algorithm in Algorithm 2.[5] During initialization (which we omitted from the listing) `self.ranker` is randomly initialized. When query `q` is observed, it arrives at `get_ranked_list()`. In DBGD this function creates a `candidate` ranker that is a variation of `self.ranker`. Both these rankers are given to an interleaved comparison method (see Section 8.2.2) and after storing all intermediate results, the interleaved list `l` is returned. As soon as the user interacts with this list, the clicks `c` arrive at the function `update_solution()`. This function delegates the computation of the outcome `o` of the interleaved comparison to the interleaving method. Based on `o`, `self.ranker` is updated towards the `candidate`.

8.2.2 Interleaved Comparison Methods

In recent years, several methods for *interleaved comparisons* have been developed. As we pointed out in Section 2.3.3, interleaving methods can be viewed as online evaluation methods that can be applied—as opposed to Cranfield-style evaluation—without manual labeling of relevant documents. Instead, the clicks of real users (or, in our case, simulated

[5] The actual implementation of DBGD included in Lerot is slightly more involved as it is configurable.

Listing 8.2: Listwise learning algorithm, that in combination with Listing 8.1, constitutes DBGD [207] (see Section 2.5.1 and in particular Algorithm 2).

```
class ListwiseLearningSystem(AbstractLearningSystem):
    def __init__(self): [...]

    def get_ranked_list(self, q):
        u = utils.sample_unit_sphere()
        candidate = self.ranker + self.delta * u
        l, a = self.comparison.interleave(
                self.ranker, candidate, q, self.n)
        self.l, self.a, self.q, self.u = l, a, q, u
        return l

    def update_solution(self, c):
        o = self.comparison.infer_outcome(
                        self.l, self.a, c, self.q)
        if o > 0:
            self.ranker += self.alpha * self.u
        return self.ranker

    def get_solution(self):
        return self.ranker
```

clicks) of the search engine are interpreted to compare two ranking algorithms. We describe interleaved comparison methods in detail in Section 2.3.3.

In the context of online learning, interleaved comparisons are mainly used to decide whether a candidate ranker is an improvement over the current best ranker or not. In comparison to absolute click metrics typically used in A/B testing, interleaved comparison methods reduce variance (briefly, this is because they perform within-subject as opposed to between-subject comparisons) [144], and make different assumptions about how clicks should be interpreted (as relative, as opposed to absolute feedback) [77]. We discuss this subject at length in Chapter 5 of this thesis.

In the Lerot framework, the following interleaving methods have been implemented.

- *Balanced interleave* (BI) [95, 144];
- *Team draft interleave* (TDI) [144];
- *Document constraints interleave* (DCI) [72];
- *Probabilistic interleave* (PI) [79];
- *Optimized interleave* (OI) [143]; and
- *Vertical aware team draft interleave* (TDI-VA) [36].

These methods also have the exact same interface; they implement the two functions that are called in Listing 8.2.

- `l, a = interleave(A, B, q, n)`
  Returns an interleaved list of documents `l` with length `n` of rankings produced by systems `A` and `B` for query `q`. The return value `a` can be used to store, for instance, team assignments in the case TDI is used.

- `o = infer_outcome(l, a, c, q)`
  Returns the outcome `o` in $(-1, 1)$ of the interleaved comparison based on clicks `c` for query `q`, and interleaved list `l`. If $o < 0$ then system `A` wins the comparison, else if $o > 0$ then `B` wins the comparison, otherwise the systems tie.

The *probabilistic interleave* (PI) method requires the ranking systems to be probabilistic; the others expect a deterministic ranker. The TDI-VA method requires documents to be annotated with the vertical to which they belong.

Additionally, the three multileaving methods introduced in Chapter 4 are included in the framework as well.

- *Team draft multileave* (TDM) [163];
- *Optimized multileave* (OM) [163]; and
- *Probabilistic multileave* (PM) [165].

Multileaving methods are very similar to interleaving methods but they have a slightly different interface, simply because they extend beyond the two rankers A and B expected by the interleaved comparison methods:

- `l, a = multileave([R1, ..., Rn], q, n)`
  Returns a multileaved list of documents `l` with length `n` of rankings produced by systems `R1` through `Rn` for query `q`. The return value `a` can be used to store, for instance, team assignments in the case TDM is used.

- `o = infer_outcome(l, a, c, q)`
  Returns the outcome `o` of the multileaved comparison based on clicks `c` for query `q`, and multileaved list `l`. The outcome `o` is a list with values for each ranker that participated in the comparisons. These values define a ranking over rankers.

8.2.3 User Models

A user model is used to simulate user's clicking behavior. Click models are a subclass of such models that are aimed at predicting what users would click on given a result list with relevance judgments in response to a query. We describe several click models in Section 2.3.6 and Chuklin et al. [40] provide an extensive overview.

The click models need data sets that are annotated with relevance to condition their click on. In Section 8.3.2 we list some data sets that are suited. In Lerot, we have implemented the following click models:

- *Dependent click model* (DCM) [64, 65], a generalization of the cascade click model [47]. This is the click model used in most experiments in this thesis that were based on simulations;

- *Random click model* (RCM) which mimics a user not having any preference, this click model is to validate the absence of bias in evaluation methods; and

- *Federated click model* (FCM) [35], in particular we implement the *attention bias model* which model the attraction of vertical results in a search result list.

These models implement the `user_model` in Listing 8.1 and, again, these models all have the exact same interface; they implement the following function.

- `c = get_clicks(l, r)`
  Returns a list of clicks `c` on documents in result list `l` given a list of relevance labels `r` for these documents.

Like TDI-VA, the FCM requires documents to be annotated with their vertical. The click models only model the assumptions regarding how users examine result pages. They still have to be instantiated to match the situation that has to be simulated. For several instantiations of DCM, see work by Hofmann [77], and for instantiations of FCM, see work by Chuklin et al. [36].

8.2.4 Evaluation

Evaluation can be done both online and offline, as described in Section 3.4. Online evaluation measures what a user experiences; i.e., the quality of interleaved lists that the user (or user model) interacts with. It is measured as a discounted sum over time. Offline evaluation measures how the current best ranker would perform on a held-out data set. Metrics implemented in Lerot are *normalized discounted cumulative gain* (nDCG) [24] and a slight variation LetorNDCG [89]. Listing 8.1 illustrates how and when offline metrics would be calculated in a learning setting. Extending Lerot with more metrics is a matter of creating a new class that implements the following two functions.

- `score = evaluate_ranking(l, q)`
  Returns a `score` for the ranking `l` with respect to query `q`.

- `mean_score = evaluate_all(s, queries)`
  Returns the `mean_score` for all `queries` if they where ranked with solution `s`.

8.3 Implementation

Lastly, we provide details on the implementation of Lerot. Lerot is implemented in Python and consists of several packages (retrieval_system, comparison, evaluation, etc). Each package has an abstract class that defines the expected interface (as described in Section 8.2) of the classes that implement it. Extending the framework is a matter of implementing such a class and changing the configuration file (see Section 8.3.2) to point to the new class. Lerot is available under the GNU Lesser General Public License.

8.3.1 Installation

Prerequisites needed for running Lerot are the following versions of Python and Python packages.

- Python 2.7;
- PyYaml;
- Numpy;
- Scipy;
- Celery (when MetaExperiment is used to distribute over several machines, see Section 8.3.3); and
- Gurobi[6] (when either OI or OM is used as the comparison method).

Once Python (and pip) have been installed, Lerot can be installed using these commands:

```
$ git clone https://bitbucket.org/ilps/lerot.git
$ cd lerot
$ pip install -r requirements.txt
$ python setup.py install
```

This will copy the source of Lerot into a directory called `lerot` in your current working directory and it will be available system wide to import into python. All requirements will also automatically be installed.

8.3.2 Configuration

Lerot can be flexibly configured using yaml files. A full example of a configuration file can be found in Listing 8.3. For instance, to pick *dependent click model* (DCM) as the user model, `user_model` can be pointing to the `environment.CascadeUserModel` class.

Lerot requires training and test query files in SVMLight format (plain or gzipped) [91]. The framework has been shown to run with the LETOR 3.0 and LETOR 4.0 collections [125], and the Yahoo! Learning to Rank Challenge [31] and MSLR-WEB30k data sets. These data sets all consist of a number of query-documents pairs, each represented by a sparse feature vector and the relevance for each document with respect to the query is judged by professional human annotators. Relevance scales can be binary or graded. We described these data sets in more detail in Section 3.2.

The data set mentioned in Listing 8.3, e.g., MQ2007 from the LETOR 4.0 collections, can be downloaded and unpacked as follows:

```
$ mkdir data
$ wget http://research.microsoft.com/en-us/um/
    beijing/projects/letor/LETOR4.0/Data/MQ2007.rar
    -O data/MQ2007.rar
$ unrar x data/MQ2007.rar data/
```

[6] Download from `http://www.gurobi.com` with a free academic trial license.

Listing 8.3: An example configuration file for a learning experiment. This particular configuration defines a setup that uses DBGD [207] (see Section 2.5.1). It also defines that clicks are simulated by a DCM [64, 65] (see Section 3.3).

```
training_queries: data/MQ2007/Fold1/train.txt
test_queries: data/MQ2007/Fold1/test.txt
feature_count: 46
num_runs: 1
num_queries: 10
query_sampling_method: random
output_dir: outdir
output_prefix: Fold1
user_model: environment.CascadeUserModel
user_model_args:
    --p_click 0:0.0,1:0.5,2:1.0
    --p_stop 0:0.0,1:0.0,2:0.0
system: retrieval_system.ListwiseLearningSystem
system_args:
    --init_weights random
    --sample_weights sample_unit_sphere
    --comparison comparison.ProbabilisticInterleave
    --delta 0.1
    --alpha 0.01
    --ranker ranker.ProbabilisticRankingFunction
    --ranker_arg 3
    --ranker_tie random
evaluation:
    - evaluation.NdcgEval
```

8.3.3 Running

After a configuration file has been created and the data has been prepared, a learning experiment can be run as follows:

```
$ python src/scripts/learning-experiment.py
                              -f config/config.yml
```

With `--help`, we can see all the options it accepts. Settings from the configuration file can be overwritten using the command line. When Lerot is run, a backup copy of the actual configuration it runs is always kept alongside the results it produces.

Running experiments with many repetitions over several data sets and user models is computationally expensive. With Lerot, it is possible to distribute computation over many machines. Lerot uses Celery to handle the bookkeeping of the distribution. The configuration file has to be extended with some additional information regarding the data sets and user models over which the experiment should be run. An example configuration is included with Lerot.

```
$ python src/scripts/meta-experiment.py
                         -f config/meta-config.yml
```

Again, we can see all the options it accepts with `--help`. In order to rerun the last experiment, e.g., in case some parts failed, we can specify `--rerun`.

8.4 Conclusion

Online learning to rank is a rapidly evolving area in information retrieval and the experimental validation of online learning to rank algorithms is vital to the field. While several libraries exist for *offline* learning to rank, in this chapter we introduced Lerot, the first framework for online learning to rank.

In this chapter, we described in detail the workings of Lerot. We also described all functions that need to be implemented in order to extend Lerot to new learning algorithms, to new user models and to new evaluation metrics.

The framework has been used in many recent publications and reproducing results from those papers only requires a user of the framework to run it with the appropriate configuration file.

8.5 Future Work

Lerot invites several directions of development. First, it allows for experiments with simulated users. The user models it currently implements reflect our current understanding of user behavior; they can easily be extended or replaced by evaluations under different sets of assumptions. Chuklin et al. [40] provide an overview of click models, we mention several in Section 2.3.6. Including these and even newer models would be valuable.

Second, Lerot provides components that implement complete online learning to rank solutions for use as part of an evaluation setup that involves real users. To this end it connects to ideas introduced in Chapter 9. This connection so far has not been extensively used and integration with all learning method remains to be resolved. For instance, the current setup of Lerot expects sequential events in the sense that user interactions are expected immediately. Real users take their time to interact with search engines. This delayed feedback has not been studied yet and has not been properly integrated into Lerot either.

9
OpenSearch: Actual Users

In Parts I and II of this thesis, we introduced methods and algorithms for online evaluation and online *learning to rank* (LTR). Earlier in *this* part of the thesis, in Chapter 8, we introduced a methodology to validate the methods and algorithms from the first two parts experimentally. Lerot, the framework we introduced in that chapter, uses *simulations* for this purpose. While this provides a flexible platform for repeatable experimentation, it is not the same as learning from real users of real search engines. The simulations in Lerot have assumptions that are violated in practice by real users.

In this chapter we propose OpenSearch, a new living labs evaluation paradigm for *information retrieval* (IR) that *does* allow for experimentation with real users of real search engines. We define OpenSearch as follows:

> OpenSearch is a new evaluation paradigm for IR. The experimentation platform *is* an existing search engine. Researchers have the opportunity to replace components of this search engine and evaluate these components using interactions with real, unsuspecting users of this search engine.

This definition is generic, on purpose, to allow for the replacement of, and experimentation with, any part of a search engine (including, e.g., result presentation). Our immediate focus, however, lies at the very core of a search engine: the ranking method.

Our new paradigm is a first step towards more realistic evaluation of IR and is the first of its kind. Our aim was have this paradigm adopted by the IR research community as we believe it to be important that evaluation of IR is as close to reality as possible for it to be meaningful. To do so, we created an actual implementation of a shared platform for all researchers in the IR research community. We believe we succeeded as our platform, and thereby the OpenSearch paradigm, has been adopted by both the*living labs for IR evaluation lab at CLEF* (LL4IR) [164] initiative and the upcoming OpenSearch track at TREC [197]. Through these two platforms, researchers now have direct access to real users of real search engines. This allows these researchers to perform research that was not possible using traditional evaluation paradigms.

We provide and describe an instantiation of OpenSearch, a benchmarking platform, for researchers to evaluate their ranking systems in a live setting with real users in their natural task environments. Our initiative is the first to offer such an experimental platform to the IR research community in the form of a community challenge.

This chapter is based on two papers by Balog, Kelly, and Schuth [15] and Schuth, Balog, and Kelly [164]. Details on our implementation and the results from the LL4IR

experiments run with the OpenSearch paradigm can be found in the paper by Schuth, Balog, and Kelly [164]. In this chapter we present the idea behind OpenSearch. We provide details on the architecture with which it is entangled. We describe the limitations of our current setup and we finish with conclusions and directions for future work.

9.1 Introduction

As we explained in Section 2.2.1, the Cranfield methodology [45] offers researchers a way to perform cross-comparable evaluation of IR systems, using a document collection, queries, and relevance assessments. Ever since the introduction of the methodologyl, as we argue in Section 2.3, researchers have strived to make IR evaluations more "realistic," i.e., centered on real users, their needs, and behaviors. Living labs have been proposed as a way for researchers to perform *in situ* evaluations, with real users performing real tasks using real-world applications [103]. This concept has already been used for a number of years as an important instrument for technology development in industrial settings. For example, A/B testing procedures are employed heavily by major web search providers [113]. This form of evaluation, however, is currently only available to those working at the said organizations. Living labs already exist for other and widely varying fields of research [6]. However, OpenSearch is the first living lab aimed specifically at IR evaluation for any researcher.

We quote Azzopardi and Balog [12] with their definition of a living lab: "The basic idea of living labs for IR is that rather than individual research groups independently developing experimental search infrastructures and gathering their own groups of test searchers for IR evaluations, a central and shared experimental environment is developed to facilitate the sharing of resources." The potential benefits of living labs to the IR community are profound, including the availability of interaction and usage data for researchers and greater knowledge transfer between industry and academia [14]. Progress towards realizing actual living labs for IR evaluation, in an academic setting, has nevertheless been limited until the LL4IR initiative and the OpenSearch track at *text retrieval conference* (TREC). Azzopardi and Balog [12] discuss a number of search and recommendation tasks in an online shopping environment and present an idealized architecture based on web services. There are many challenges associated with operationalizing these ideas, including architecture, hosting, maintenance, security, privacy, participant recruiting, and scenarios and tasks for use development [12].

In this chapter we present OpenSearch, a concrete implementation of a living lab for IR evaluation benchmarking platform. We argue that mid-sized organizations with a search engine that lack their own R&D department are good potential collaborators. Such collaborators, which we refer to as *sites*, have the opportunity to gain much improved retrieval approaches (and, as a consequence, increased revenue). We have so far conducted experiments with two specific sites for ad-hoc search: product search on an e-commerce site and web search on large scale web search engine. These sites represent a setting with at least two major challenges: (1) relatively low search volume (for the product search site); and (2) means to facilitate experimentation by "third parties" in live, production systems. We postulate that focusing on *head queries*, i.e., queries most frequently issued to the site organizations' search engines, can help overcome these challenges. The choice

of head queries is critical because it removes a harsh requirement of providing rankings in real-time for query requests. Instead, experimental search systems (developed by benchmark participants) can generate ranked results lists for these queries offline. These participant rankings can then be used by the live system when head queries are next issued. Finally, feedback is made available to experimental search systems to facilitate improved offline ranking generation. Data exchange between live systems and participants is facilitated by a web-based API.

In summary, we make the following contributions in this chapter.

Paradigm We introduce OpenSearch, the first living labs for IR evaluation, a completely new evaluation paradigm for IR.

Implementation An implementation of OpenSearch that was used to run the LL4IR evaluation lab at CLEF 2015 and that will be used to run TREC OpenSearch in 2016. This includes the development of the architecture as well as implementation of the OpenSearch API, made available as open source software.[1]

We have incorporated the above contributions in Section 1.2 where we give a complete overview of all contributions of this thesis.

The remainder of this chapter is organized as follows. In Section 9.2 we briefly discuss related work. Next, in Section 9.3, we introduce our evaluation platform and methodology. Limitations and directions for future research are discussed in Section 9.5. Finally, we conclude in Section 9.6.

9.2 Related Work

The need for more realistic IR evaluation, involving real users, was reiterated at recent IR workshops [4, 14, 98] and also stated in Section 2.3 of this thesis. Approaches that attempt to incorporate user behavior into batch-style evaluations can be divided into two main categories. One is to create effectiveness measures that better model user behavior, e.g., [28, 89, 133, 150]. Another approach is to develop models that simulate user behavior and then validate these models against actual usage data, e.g., [9, 11, 17, 202] (see also Section 2.3 and Chapter 8). These ideas have been implemented in a number of community benchmarking efforts, including the TREC Interactive [51], HARD [3], and Session tracks [99], and the INEX Interactive track [191]. While user simulation is a great instrument for fine-tuning systems, it cannot substitute for the real user as simulations necessarily make assumptions that are violated in practice. Crowdsourcing, using, e.g., Mechanical Turk, enables the sourcing of individuals in the online community to perform various relevance assessment and annotation tasks [7]. However, these individuals do not constitute *real* users performing *real* tasks driven by a *real* information need. Living labs such as OpenSearch, as discussed in this chapter, offer this potential.

In the *information-seeking support space* (ISSS) the living labs notion was first proposed in the context of IR by Kelly et al. [103]: "Such a lab might contain resources and tools for evaluation as well as infrastructure for collaborative studies. It might also

[1] `https://bitbucket.org/living-labs/ll-api`

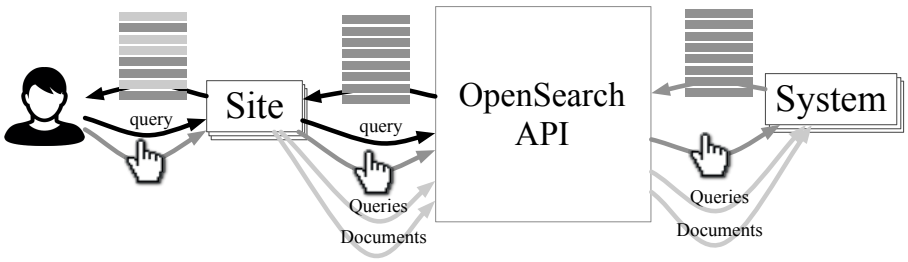

Figure 9.1: Schematic representation of interaction with the OpenSearch API. Users issue queries to sites. A site shares a set of queries and for each query a set of documents with the OpenSearch API. Systems of participants can take these queries and documents and upload rankings for each query. When a user issues a participating query to the site then the site asks the OpenSearch API for a ranking. These can immediately be returned, interleaved and shown to the user. User interactions are again shared with the API and the system.

function as a point of contact with those interested in participating in ISSS studies." Azzopardi and Balog [12] provided greater insight into what this might be in the IR space: "A living lab would provide a common data repository and evaluation environment giving researchers (in particular from academia) the data required to undertake meaningful and applicable research." Kelly et al. [104] then showed a practical interpretation of this for personal desktop search. However, until OpenSearch, there have been no attempts at operationalizing a living labs benchmark in the IR space. The nearest to this has been the CLEF NEWSREEL lab[2] and the Plista contest,[3] addressing the problem of news recommendation. Participants are expected to implement their recommender system as a service that can handle a large number of (recommendation) requests. Their response to a request is shown to a user and resulting clicks are then made available to participants so that they can update their system. One major difference between this and OpenSearch is the task itself: OpenSearch focuses on retrieval as opposed to recommendation. There are also important architectural differences stemming from the nature of our experimental environment; in our setup participants do not get full control over the results shown to the user, they are always interleaved with those of the production system.

9.3 OpenSearch Architecture

OpenSearch, our livings labs evaluation methodology, heavily depends on a novel architecture which we describe in this section. We start with an overview, we then introduce the organization of our lab.

[2] http://www.clef-newsreel.org
[3] http://contest.plista.com

9.3.1 Overview

Before we provide an overview of our architecture we will first introduce the terminal we use in the context of OpenSearch.

Site We use the term *site* to refer to a search engine that provides the interaction data. These sites are real search engines with real users that collaborate with OpenSearch in that for a selection of queries they show rankings coming from the OpenSearch API.

Participant We refer to teams of researchers that use OpenSearch as their experimentation platform as *participant*.

User We reserve the term *user* for user of our *sites*. These are real user of these real search engines. The users are not (necessarily) aware they are part of an experiment, as opposed to for instance crowd workers.

For each of the sites participants take part in a live evaluation process. For this they use a set of frequent queries as training queries and a separate set of frequent queries as test queries. Candidate documents are provided for each query along with historical information associated with the queries. When participants produce their rankings for a query, they upload these through the provided OpenSearch API. The commercial provider then interleaves a given participant's ranked list with their own ranking, and presents the user with the interleaved result list. See Section 2.3.3 for background on interleaving. Participants take turns in having their ranked list interleaved with the commercial provider's ranked list. This process of interleaving a single experimental system with the production system at a time is orchestrated by the OpenSearch API, such that each participant gets about the same number of impressions. The actions performed by the users of the commercial provider's system are then made available to the participant (whose ranking was shown) through the API, i.e., the interleaved ranking, resulting clicks, and (aggregated) interleaving outcomes.

Figure 9.1 shows the OpenSearch architecture and how a participant's system interacts with the sites (the search engines) through the provided API. As can be seen, frequent queries (Queries) with candidate documents for each query (Documents) are sent from a site through the API to the experimental systems of participants. These systems upload their rankings for each query to the API. When a user of the site issues one of these frequent queries, then the site requests a ranking from the API and presents it interleaved to the users. Any interactions such as clicks of the user with this ranking are sent back to the API. Systems can then obtain these interactions from the API and update their ranking if they wish. Darker arrows indicate that there is a need for realt-time interactions. Lighter arrows do not have this requirement.

Participants are provided with example code and guidelines to ease the adaptation to the OpenSearch setup.[4]

[4] `http://doc.trec-open-search.org/en/latest/guide-participant.html`

Head Queries

The distribution of search queries typically follows a power law [175], where a relatively small set of head queries are frequently posed by many users and there is a long tail of queries that appear in the logs only a few times (often only once). In OpenSearch, our living labs setting, we focus exclusively on head queries for a number of reasons:

1. This allows us to evaluate experimental search systems on the same set of queries.
2. These queries have a stable volume level, even for mid-sized sites (cf. [15]).
3. Historical click and usage data is available in meaningful quantities.
4. Most importantly, because we use head queries, participants' systems do not need to respond in real-time to user queries (cf. [15]). Since the set of queries is fixed beforehand, participants can upload their rankings to the API at any time and at any speed. After uploading their is no need for them to be online. The API will always immediately show users the latest ranking from a participant and participants can upload their rankings any time they want.

9.3.2 Lab Organization

Much like any other (information retrieval or machine learning) evaluation campaign, we split our data into training and test parts. Given that we perform online evaluation, it is slightly more involved than simply providing disjoint data sets. We describe our setup here.

Training Phase

During the training phase, participants are free to update their rankings using feedback information. This feedback information is made available to them as soon as it arrives at the OpenSearch API. Their rankings can be updated at any time and as often as desired. Both click feedback and aggregated outcomes are made available directly and are updated constantly.

Test Phase

Just before the test phase starts, participants receive another set of frequent queries as test queries. Again, the associated historical click information as well as candidate results for these queries are made available. After downloading the test queries, participants can only upload their rankings until the test phase starts or only once after it started. The reason for this is that per test phase we only want to evaluate a single underlying system per participant.

The test rankings are, as soon as they are uploaded, treated in the same way as training queries. That is, they are interleaved with the commercial providers' rankings for several weeks. As for the training phase, in the test phase each participant is given an approximately equal number of impressions. A major difference is that for the test queries, the click feedback is not made available. Aggregated outcomes are provided only after the test phase ends.

9.3.3 Evaluation Metric

The overall evaluation of participants' systems is based on the final system performance, and additionally on how the systems performed at each query issue. The primary metric used is aggregated interleaving outcomes, and in particular we are interested in the fractions of winning system comparisons.

Interleaved Comparisons

In Section 2.3.3 of this thesis we provided details on how interleaving, and in particular *team draft interleave* (TDI) [144], works. There are two reasons for using interleaved comparisons. Firstly, interleaved comparisons were shown to be two orders of magnitude more sensitive than other ways of performing online evaluation such as A/B testing [34, 166]. Secondly, interleaved comparisons ensure that at least half the ranking shown to users comes from the production system. This reduces the risk of showing bad rankings to users.

Aggregated Outcomes

We use the following aggregated interleaving metrics, where *Outcome* serves as the primary metric for comparing participants rankings. These aggregations are constantly updated for training queries. For the test phase they are only computed after the phase is over.

#Wins is defined as the number of wins of the participant against the production system, where a *win* is defined as the experimental system having more clicks on results assigned to it by TDI than clicks on results assigned to the production system;

#Losses is defined as the number of losses against the production system;

#Ties is defined as the number of ties with the production system;

#Impressions is the total number of times when rankings (for any of the test queries) from the participant have been displayed to users of the production system; and

Outcome is defined as the fraction of wins, so *#Wins*/(*#Wins*+*#Losses*).

An *Outcome* value below the *expected outcome* (typically 0.5) means that the participant system performed worse than the production system (i.e., overall it has more losses than wins). Significance of outcomes is tested using a two-sided binomial test which uses the expected outcome; p-values are reported.

Note that using these metrics, we are in theory only able to say something about the relationship between the participant's system and the production system. However, Radlinski et al. [144] show experimentally that it is not unreasonable to assume transitivity. This allows us to also draw conclusions about how systems compare to each other. Ideally, instead of interleaving, we would have used multileaved comparison methods [163, 165] which would directly give an ordering over rankers by comparing them all at once for each query. However, multileaved comparisons could potentially impact users more than they would be impacted by interleaving. Moreover, multileaved comparison methods are not yet as established as an evaluation method.

9.4 Implementation and Results

Details on our implementation and the results from the LL4IR experiments run with the OpenSearch paradigm can be found in the paper by Schuth, Balog, and Kelly [164].

9.5 Limitations

OpenSearch is a new evaluation paradigm and represents an important step towards making the living labs evaluation accessible to the wider IR research community. Nevertheless, it is not without limitations. Next, we briefly consider some of these limitations and look at ways in which they could be addressed.

L1) *Head queries only.* While head queries constitute a considerable portion of a site's traffic, they are representative of only one type of request, that is, popular information needs.

L2) *Lack of context.* The search algorithm has no knowledge of the searcher's context, such as location, previous searches, etc. This means that currently there is no room for personalization of results.

L3) *No real-time feedback.* While the proposed API does provide detailed feedback, it is not immediate. Thus, it cannot directly be used in the given search session.

L4) *Limited control.* Experimentation is limited to single searches, where results are interleaved with those of the production system. I.e., there is no control over the entire result list.

L5) *Ultimate measure of success.* Having better search facilities is usually only a means to an end—it is not the ultimate goal. For example, in the e-commerce case, and from the company's perspective, the ultimate measure of success is the profit made on purchases. Evaluation metrics should reflect this overall goal.

L1–L4 could be overcome by a live architecture, in which control is given to benchmark participants over entire sessions, with real-time access to context and feedback. However, it is still a very much open question how to ensure availability, response time, and quality of the experimental methods in production environments. Safety mechanisms are needed for "experiment shutdown" in which case methods can default back to the production system. L5 could be addressed by providing a "utility" score for documents (products); this could already be done with the existing architecture.

9.6 Conclusion

The OpenSearch methodology introduced in this chapter offers great potential to evaluate information retrieval systems in live settings with real users. The *living labs for IR evaluation lab at CLEF* (LL4IR) represents the first attempt at an implementation of the OpenSearch idea.

The first edition of LL4IR at *conference and labs of the evaluation forum* (CLEF) 2015 focused on two *sites* that provided interactions with real users: *product search* and *web search*. Our product search site was a commercial e-commerce website, REGIO. Our web search site a commercial web search engine, Seznam. A major contribution of the LL4IR lab is the development of the necessary OpenSearch API infrastructure, which is made publicly available.

Overall, we regard our effort successful in showing the feasibility and potential of this form of evaluatio‘ both sites, there was an experimental system that outperformed the corresponding production system significantly. It is somewhat unfortunate that in both cases that experimental system was a baseline approach provided by the lab organizers; nevertheless, it demonstrates the potential benefits to site owners as well.

The OpenSearch API infrastructure developed for the LL4IR Lab offers the potential to host ongoing IR evaluations in a live setting. As such, these “challenges” will continue on an ongoing basis at both CLEF and as the OpenSearch track at TREC 2016, with an expanding number of sites as well as refinements to the existing sites.

9.7 Future Work

One particular issue, that surfaced in our LL4IR lab at CLEF 2015, for the product search site is the frequent changes in inventory. This appears to be more severe than we first anticipated and represents some challenges, both technical and methodological. This is not only a problem for sites with inventories that change rapidly. Also for other documents collections the available documents for queries may change fast, depending on the type of query. Solving this issue in a general enough way, so that it can be applied to any search engine that decides to participate, will lead to much wider applicability of the OpenSearch methodology.

Another important direction for future research is work towards the inclusion of context. One way to include context in the current setup would be by providing several *common* contexts alongside the query. Participants could then prepare rankings for each of these possible contexts. Examples of common contexts would be common preceding queries, types of users or localization on a high enough granularity.

Lastly, currently our OpenSearch efforts only focused on replacing the ranking component of a search engine. We would be very interested in investigating the replacement of other components of the search engine. Obvious candidates are query suggestion, query autocompletion and snippet generation.

10 Conclusions

In this thesis we investigated *whether, how and to what degree search engines can learn from their users*. We started in Part I by investigating how user interactions with search engines can be used to evaluate these search engines. In particular, in Chapter 4 we introduced a new online evaluation paradigm called multileaving. With multileaving, many rankers can be compared at once by combining document lists from these rankers into a single result list and attributing user interactions with this list to the rankers. Then, in Chapter 5, we investigated the relation between A/B testing and interleaved comparison methods.

In Part II we turned to online learning to rank. We learned from the evaluation methods introduced and extended upon in the previous part. In Chapter 6 we learned the parameters of base rankers based from user interactions. In Chapter 7 we used the multileaving methods from Chapter 4 as feedback in our learning method, leading to much faster convergence.

The last part of this thesis is of a different nature than the earlier two parts. Progress in *information retrieval* (IR) research was always driven by a combination of algorithms, shared resources, and evaluation. In Part III we focussed on the latter two. We introduced a new shared resource and a new evaluation paradigm. Lerot, in Chapter 8, is an online evaluation framework that allows us to simulate users interacting with a search engine. Lerot has been released as open source software and is currently used by researchers around the world. Chapter 9, also in the last part, introduced a new evaluation paradigm involving real users of real search engines. In that chapter we also described an implementation of this paradigm that has been adopted widely by the research community.

In this concluding chapter we first look back at the questions asked in the first chapter of this thesis, in Section 1.1, and summarize the answers to our research questions asked in Parts I and II. We summarize the research of Part III were we did not have research questions. We summarize all our findings once more in Section 10.2 and close by looking forward in Section 10.3.

10.1 Main Findings

We investigated *whether, how and to what degree search engines can learn from their users*. Part I discussed the evaluation of search engines. In particular we focused on

online evaluation, where preferences are interpreted from users interacting with the search engine. It was shown earlier that *interleaved comparison* [92, 93] methods enable such evaluations with greater data efficiency than A/B testing [144]. In Chapter 4 we introduced extensions of interleaving methods called multileaving. Multileaved comparison methods make it possible to compare more than two rankers at once. Multileaved comparisons can provide detailed feedback about how multiple candidate rankers compare to each other using much less interaction data than would be required using interleaved comparisons. In particular, we first proposed two specific implementations of multileaved comparisons. The first, which we call *team draft multileave* (TDM), builds on *team draft interleave* (TDI) [144]. The second method is called *optimized multileave* (OM) and builds on *optimized interleave* (OI) [143]

We asked ourselves the following questions:

RQ1 Can multileaved comparison methods identify preferences between rankers faster than interleaved comparison methods?

RQ2 How does the sensitivity of multileaving methods compare to that of interleaving methods?

RQ3 Do multileaving methods improve over interleaving methods in terms of unbiasedness and online performance?

In response to RQ1, our experimental results clearly showed that the error of both of our multileaving methods drops much faster than their interleaving counterparts. This indicates that multileaved comparison methods can learn preferences between multiple rankers with far less data (i.e., queries and clicks) than interleaved comparison methods. Addressing RQ2, our results showed that the sensitivity of multileaving methods is affected in the same way as for interleaving methods when the differences between rankers vary. Interestingly, this means that multileaved methods can distinguish between rankers just as well as interleaving methods even when the differences between them are small. Hence, multileaved comparison methods can be used to explore a parameter space using very small steps. With regard to RQ3, TDM was shown to be unbiased. OM, our other multileaving method, had as large a bias as OI, its interleaving counterpart. TDM showed the highest online performance, i.e., users were the least affected by the evaluation in which they participated.

We asked the following:

RQ4 Does OM scale better with the number of rankers than TDM?

In response to this question, we analyzed what happens when the number of rankers being compared increases. Both interleaving methods OI and TDI are impacted greatly when the number of rankers increases. This is largely due to the fact that many more comparisons are needed and as such each pair of rankers receives fewer comparisons. By contrast, OM and TDM do not show significant degradation when the number of rankers increases.

Next, we proposed *probabilistic multileave* (PM) which builds on *probabilistic interleave* (PI) [79]. We asked ourself the following question:

RQ5 How does PM compare to TDM and OM in terms of sensitivity, bias and scaling?

We showed empirically that PM is highly sensitive and unbiased. An important implication of this result is that, like with PI, historical interactions with multileaved comparisons can be reused, allowing for ranker comparisons that need much less user interaction data. Furthermore, we show that our method, as opposed to earlier sensitive multileaving methods, scales well when the number of rankers increases.

Despite the advantages of interleaving and multileaving in terms of sensitivity, the most common online evaluation methodology is still A/B testing [112]. A downside of A/B testing is that large numbers of users are typically necessary to obtain reliable results as this approach has high variance. Interleaved comparisons have much lower variance and need fewer interactions. However, there is low agreement in terms of preferences with recent A/B metrics given realistic differences in IR system effectiveness.

In Chapter 5 we asked ourself the following questions:

RQ6 How do A/B metrics compare to interleaving in terms of sensitivity and agreement?

RQ7 Can A/B metrics and interleaving be made to agree better without losing sensivity?

To answer RQ6 we proposed a new statistical method for assessing the sensitivity of these metrics from estimated effect sizes. The resulting method allows for a detailed comparison between metrics in terms of the power of statistical tests at varying sample sizes. Our analysis showed that A/B tests typically require two orders of magnitude more data than interleaved comparisons. While answering RQ7, we proposed novel interleaving *credit functions* that are (1) designed to closely match the implementation and parameters of A/B metrics, or (2) are parameterized to allow optimization towards agreement with arbitrary A/B metrics. Our empirical results, obtained on 38 paired experiments with a total of 3 billion clicks, showed that our approach can substantially and significantly increase agreement of interleaving with A/B metrics while maintaining the advantages in terms of sensitivity. Our adapted interleaved comparisons methods are still one to two orders of magnitude more sensitive when compared to A/B testing.

In Part II of this thesis, we turned to learning using the online evaluation methods from Part I. In Chapter 6 we pursued the problem of optimizing a base ranker using clicks by focusing on *best match 25* (BM25). Currently, it is common practice to choose the parameters of BM25 according to manually tuned values reported in the literature, or to manually tune them for a specific setting based on domain knowledge or a sweep over a number of possible combinations using guidance from an annotated data set [147]. We proposed an alternative by learning the parameters from click data using a learning algorithm called *dueling bandit gradient descent* (DBGD) [207].

Specifically, the research questions we aimed to answer were as follows.

RQ8 How good are the manually tuned parameter values of BM25 that are currently used? Are they optimal for all data sets on average? Are they optimal for individual data sets?

RQ9 Is it possible to learn good values of the BM25 parameters from clicks? Can we approximate or even improve the performance of BM25 achieved with manually tuned parameters?

In response to RQ8 we provided insight into the parameter space of a base ranker such as BM25. We showed that each data set requires a different set of parameters for optimal performance. Answering RQ9 we showed that significant improvements can be achieved if parameters of base rankers are learned as opposed to treating them as black boxes which is currently the common practice.

Dueling bandit gradient descent (DBGD) [207] is an online learning to rank method. At every learning step, DBGD estimates a gradient to follow with respect to a *single* exploratory ranker and updates its solution if the exploratory ranker seems better according to an interleaved comparison. We proposed, instead, to use multileaved comparisons methods as introduced in Chapter 4 so that we can explore gradients in *multiple* directions at once. Our proposed method, *multileave gradient descent* (MGD), aims to speed up online learning to rank.

We asked ourself the following questions:

RQ10 Can MGD learn faster from user feedback (i.e., using fewer clicks) than DBGD does?

RQ11 Does MGD find a better local optimum than DBGD?

Our answer to RQ10 is that MGD with increasing numbers of candidates learns increasingly faster than DBGD in terms of offline performance. In terms of online performance, MGD is on par with or outperforms DBGD when feedback has realistic levels of noise. We answer RQ11 as follows: MGD converges to an optimum which is at least as good as the optimum DBGD finds. However, MGD does so much faster.

We introduced two variants of MGD that differ in how they estimate the gradient. In *MGD winner takes all* (MGD-W), the gradient is estimated using one ranker randomly sampled from those that won the multileaved comparison. In *MGD mean winner* (MGD-M), the gradient is estimated using the mean of all winning rankers. We asked the following question:

RQ12 Which update approach, MGD-W or MGD-M, learns faster? Which finds a better local optimum?

To answer RQ12, while in general both MGD methods outperform DBGD, MGD-M is better at handling high noise levels, making it more effective than MGD-W overall. The advantage of MGD-M over MGD-W comes from both the update direction and a smaller update size.

Lastly, Part III of thesis was of a different nature than the earlier two parts. As opposed to the earlier chapters, in this part we no longer studied algorithms. IR research has always been driven by a combination of algorithms, shared resources, and evaluation. In Part III we introduced a new shared resource in the form of a learning framework. We also introduced a new evaluation paradigm. Our research in this last part of the thesis was not centered around research questions but rather around designing these shared resources and designing a new evaluation methodology.

In Chapter 8, we introduced Lerot, an online evaluation framework which allows us to simulate users interacting with a search engine. Several libraries existed for *offline* learning

to rank, Lerot is the first framework for *online* learning to rank and online evaluation. In Chapter 8, we described in detail the workings of Lerot. We also described all functions that need to be implemented in order to extend Lerot to new learning algorithms, to new user models and to new evaluation metrics. The framework has been used in over 15 recent publications and reproducing results from those papers only requires a user of the framework to run it with the appropriate configuration file. This has the potential of rapidly furthering the research field.

In Chapter 9 we introduced OpenSearch, a new evaluation paradigm for IR involving real users of real search engines. We describe in detail the *living labs for IR evaluation lab at CLEF* (LL4IR) which represents the first implementation of the OpenSearch methodology. A major contribution of the LL4IR lab is the development of the necessary API infrastructure, which is made publicly available. This OpenSearch API infrastructure offers the potential to host ongoing IR evaluations in a live setting. As such, these "challenges" will continue on an ongoing basis at both *conference and labs of the evaluation forum* (CLEF) and as the OpenSearch track at *text retrieval conference* (TREC) , with an expanding number of participating search engines.

10.2 Summary of Findings

We now summarize our findings in this thesis. We introduced a new and highly sensitive online evaluation paradigm called *multileaved comparisons*. This paradigm allows for the comparison of many rankers at once which in turn allows for search engines that learn from their users much more efficient than was possible before. This urge for efficiency also led us to investigate the relationship between A/B testing and interleaving. Interleaving is much more efficient in that it requires much less user interaction data to infer a preferences between two rankers. While this is a desired property, the correctness of the preference is also important. We introduced a method to compare two rankers using interleaving such that the preference is in line with the preference of an A/B testing metric. This method and our multileaved comparison method are both highly sensitive ways of inferring preferences from users interacting with a search engine. This result implies that orders of magnitude less user interaction data is required to compare rankers allowing for much faster ranker development. Moreover, it leads to less users being exposed to inferior rankers.

We have interpreted these efficient ranker preferences as guidance for learning algorithms. We have shown how parameters that used to be hand tuned can instead be learned automatically from users. We have further shown how our multileaved comparison methods can guide a highly efficient online learning to rank algorithm towards an optimal ranker. An important implication of these results is that good rankers can be found efficiently by interacting with users. This results in far fewer users being exposed to inferior rankers and it allows search engines to adapt faster to changes in user preferences.

Besides the above algorithmic contributions, we also contributed a shared resource and a new evaluation paradigm. We introduced Lerot, an online learning to rank framework, a shared resource for the IR research community. The framework allows for reproducible experimentation with online learning to rank and with online evaluation. Lastly, we introduced OpenSearch, an entirely new IR evaluation paradigm that allows any researcher to experiment on a real search engine and with real users. OpenSearch comes with an

implementation and is adopted widely by the IR community.

10.3 Future Work

The research presented in this thesis has many important implications, as described above. However, there are a number of opportunities for further work. In this section we select the most prominent directions, summarize them and expand on them.

10.3.1 Online Evaluation

Adapting our newly introduced multileaving paradigm to online evaluation tasks other than comparing all rankers in a set to each other could prove promising. When, for instance, one would instead compare all rankers to a single production ranker the definitions of unbiasedness and sensitivity could be adjusted to take into account the restricted goal of this task variant. Proper investigation of how multileaving methods could be applied in a K-armed dueling bandits problem setting constitutes another promising research direction. Both these directions could lead to breakthroughs on their respective sub-tasks of online ranker evaluation.

Our research that brought ranker preferences inferred by interleaved comparison method closer to those inferred by A/B tests, was the first of its kind. Many directions for future work remain. Firstly, more data in the form of ranker comparisons would open the way to development of yet more sophisticated (learned) credit functions, e.g., to take into account session-level or task-level features. Secondly, our approach does not currently take magnitude and uncertainty in the A/B test preferences into account. For future work, we would like to measure agreement with statistically significant A/B outcomes. Again, we would require more ranker comparisons for such an analysis.

Both our contributions to online evaluation of IR only consider so called *single shot queries*: a query followed by a result list and interactions with this list. In reality, a user typically issues many queries over a longer period of time. Incorporating information from sessions into interleaving or multileaving methods is a completely open and very promising area of research.

10.3.2 Online Learning to Rank

For future work, it is interesting to see how the click-optimized versions of BM25 can improve the performance of a state-of-the-art learning to rank algorithm when BM25 is used as one query-document feature among many features. Furthermore, it would be interesting to see whether we can integrate the learning of parameters for base rankers such as BM25 while we learn how to combine such base rankers at the same time. Potentially, our multileave gradient descent methods could be of help here as this allows for exploring many more alternative rankers. Additionally, we would like to investigate whether and to what extent parameters of other base rankers can be learned through the same procedure as we used in Chapter 6.

Our new learning method that uses multileaved comparison methods invites many directions for future work. Firstly, we sampled candidate rankers randomly uniformly

from a unit sphere around the current best ranker. Alternatively, one could consider selecting rankers such that all directions are covered, which may speed up learning even further. Secondly, currently we have two strategies for interpreting multileave comparison outcomes, MGD-M and MGD-W. We could consider an additional strategy that takes a weighted combination of all the compared rankers, potentially even down-weighting the directions for losing candidate rankers. Thirdly, we noticed that often, in particular closer to convergence, many of the compared rankers become very similar. One could adapt the multileaving algorithm to not attempt to infer preferences between rankers that produce the same rankings, but rather, consider all these to be the same rankers. It may seem that increasing the number of candidates increases the amount of required computation. However, when using TDM, the number of rankers that can contribute a document to the multileaved list is bound by its length. If it is decided beforehand which rankers can contribute a document then only these rankers actually have to materialize their rankings. Finally, a theoretical analysis of the convergence properties of MGD and its variants, in comparison to DBGD, would give valuable insights in the broader applicability of our new method.

Both our contributions to online learning to rank only consider fairly simple ranking models. In *offline* learning to rank much more sophisticated models are the current state-of-the-art. Learning such complicated models from user feedback remains future work.

Our learning methods make the simplifying assumption that users arrive at the search engine sequentially, never more than a single user at the same time. What is more, our approaches assume that the user finishes their interactions with a result page before the next query is issued. Naturally these are unrealistic assumptions and lifting these would be interesting future work.

10.3.3 Online Learning and Evaluation Methodology

Lerot, our online learning and evaluation framework, is highly successful as a shared research tool. Currently, the source code of Lerot has been forked 16 times, the repository sees between 100 and 700 unique users per month and 15 people have contributed to the code.

Lerot invites several directions of development. First, it allows for experiments with simulated users. The user models it currently implements reflect our current understanding of user behavior; they can easily be extended or replaced by evaluations under different sets of assumptions. Secondly, Lerot provides components that implement complete online learning to rank solutions for use as part of an evaluation setup that involves real users. To this end it connects to OpenSearch, also introduced in this thesis. This connection so far has not been extensively used and integration with all learning methods remains to be resolved. For instance, the current setup of Lerot expects sequential events in the sense that user interactions are expected immediately after the query issue, before a new query is issued by any other user. Real users do take their time to interact with search engines. This delayed feedback has not been studied yet and has not been properly integrated into Lerot either.

More even than Lerot, OpenSearch has rapidly been adopted by the IR community, witnessed by the lab at CLEF and the OpenSearch track at TREC 2016.

An important direction for future research around OpenSearch is work towards the inclusion of context. One way to include context in the current setup would be by providing several *common* contexts alongside the query. Participants could then prepare rankings for each of these possible contexts. Examples of common contexts would be common preceding queries, types of users or localization on a high enough granularity. Currently, our OpenSearch efforts only focused on replacing the ranking component of a search engine. We would be interested in investigating the replacement of other components of the search engine. Obvious candidates are query suggestion, query autocompletion and snippet generation.

This would, for the very first time, allow academic researchers to experiment on such components and would open up entire new and exciting research directions.

Bibliography

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM '09*. ACM Press, 2009. (Cited on page 17.)

[2] K. Ali and C. Chang. On the relationship between click-rate and relevance for search engines. *Proc. of Data-Mining and Information Engineering*, 2006. (Cited on page 19.)

[3] J. Allan. HARD track overview in TREC 2003 high accuracy retrieval from documents. Technical report, 2005. (Cited on page 137.)

[4] J. Allan, W. B. Croft, A. Moffat, and M. Sanderson. Frontiers, Challenges, and Opportunities for Information Retrieval: Report from SWIRL 2012 the Second Strategic Workshop on Information Retrieval in Lorne. *SIGIR Forum*, 46(1), 2012. (Cited on page 137.)

[5] M. Allegretti, Y. Moshfeghi, M. Hadjigeorgieva, F. E. Pollick, J. M. Jose, and G. Pasi. When Relevance Judgement is Happening?: An EEG-based Study. In *SIGIR '15*. ACM Press, 2015. (Cited on page 18.)

[6] E. Almirall and J. Wareham. Living Labs: arbiters of mid- and ground-level innovation. *Technology Analysis & Strategic Management*, 23(1), 2011. (Cited on page 136.)

[7] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for Relevance Evaluation. *SIGIR Forum*, 42(2), 2008. (Cited on pages 2 and 137.)

[8] G. Amati. *Probability models for information retrieval based on divergence from randomness*. PhD thesis, University of Glasgow, 2003. (Cited on pages 16, 92, and 164.)

[9] P. Arvola, J. Kekäläinen, and M. Junkkari. Expected Reading Effort in Focused Retrieval Evaluation. *Information Retrieval*, 13(5), 2010. (Cited on page 137.)

[10] N. Asadi, D. Metzler, T. Elsayed, and J. Lin. Pseudo test collections for learning web search ranking functions. In *SIGIR '11*. ACM, 2011. (Cited on pages 5, 18, and 91.)

[11] L. Azzopardi. The Economics in Interactive Information Retrieval. In *SIGIR '11*, 2011. (Cited on page 137.)

[12] L. Azzopardi and K. Balog. Towards a Living Lab for Information Retrieval Research and Development. A Proposal for a Living Lab for Product Search Tasks. In *CLEF '11*, 2011. (Cited on pages 136 and 138.)

[13] L. Azzopardi, M. de Rijke, and K. Balog. Building simulated queries for known-item topics: an analysis using six european languages. In *SIGIR '07*. ACM, 2007. (Cited on pages 18 and 91.)

[14] K. Balog, D. Elsweiler, E. Kanoulas, L. Kelly, and M. D. Smucker. Report on the CIKM Workshop on Living Labs for Information Retrieval Evaluation. *SIGIR Forum*, 48(1), 2014. (Cited on pages 136 and 137.)

[15] K. Balog, L. Kelly, and A. Schuth. Head First: Living Labs for Ad-hoc Search Evaluation. In *CIKM '14*. ACM Press, 2014. (Cited on pages 11, 12, 135, and 140.)

[16] S. Bao, G.-R. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su. Optimizing web search using social annotations. In *WWW '07*. ACM Press, 2007. (Cited on page 16.)

[17] F. Baskaya, H. Keskustalo, and K. Järvelin. Time Drives Interaction: Simulating Sessions in Diverse Searching Environments. In *SIGIR '12*, 2012. (Cited on page 137.)

[18] N. Belkin, R. Oddy, and H. Brooks. ASK for information retrieval: Part I. Background and theory. *Journal of documentation*, 1982. (Cited on page 15.)

[19] P. N. Bennett, F. Radlinski, R. W. White, and E. Yilmaz. Inferring and using location metadata to personalize web search. In *SIGIR '11*. ACM Press, 2011. (Cited on page 16.)

[20] P. N. Bennett, R. W. White, W. Chu, S. T. Dumais, P. Bailey, F. Borisyuk, and X. Cui. Modeling the impact of short- and long-term behavior on search personalization. In *SIGIR '12*. ACM Press, 2012. (Cited on page 16.)

[21] R. Berendsen, M. Tsagkias, W. Weerkamp, and M. de Rijke. Pseudo Test Collections for Training and Tuning Microblog Rankers. In *SIGIR '13*. ACM, 2013. (Cited on pages 5, 18, and 91.)

[22] L. Bottou and J. Peters. Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research*, 2013. (Cited on page 24.)

[23] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2), 2002. (Cited on pages 16 and 33.)

[24] C. J. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05*, 2005. (Cited on pages 17, 27, and 131.)

[25] C. J. Burges, R. Ragno, and Q. Le. Learning to Rank with Nonsmooth Cost Functions. In *NIPS '06*, 2006. (Cited on page 27.)

[26] V. Bush. As We May Think, 1945. (Cited on page 14.)

[27] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From Pairwise Approach to Listwise Approach. In *ICML '07*. ACM, 2007. (Cited on page 26.)

[28] B. Carterette. System Effectiveness, User Models, and User Utility: A Conceptual Framework for Investigation. In *SIGIR '11*, 2011. (Cited on page 137.)

[29] B. Carterette and J. Allan. Incremental test collections. In *CIKM '05*. ACM, 2005. (Cited on page 18.)

[30] B. Carterette and R. Jones. Evaluating search engines by modeling the relationship between relevance and clicks. In *NIPS '07*. MIT Press, 2008. (Cited on pages 6, 20, and 91.)

[31] O. Chapelle and Y. Chang. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research*, 2011. (Cited on page 132.)

[32] O. Chapelle and Y. Zhang. A dynamic bayesian network click model for web search ranking. 2009. (Cited on pages 25 and 164.)

[33] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *CIKM '09*. ACM, 2009. (Cited on pages 17 and 164.)

[34] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale validation and analysis of interleaved search evaluation. *ACM Transactions on Information Systems (TOIS)*, 30(1), 2012. (Cited on pages 7, 21, 22, 23, 68, 70, 71, 72, 73, 76, 106, and 141.)

[35] D. Chen, W. Chen, and H. Wang. Beyond ten blue links: enabling user click modeling in federated web search. In *WSDM '12*. ACM, 2012. (Cited on pages 131 and 164.)

[36] A. Chuklin, A. Schuth, K. Hofmann, P. Serdyukov, and M. de Rijke. Evaluating Aggregated Search Using Interleaving. In *CIKM '13*. ACM Press, 2013. (Cited on pages 12, 22, 31, 125, 126, 129, 131, and 166.)

[37] A. Chuklin, P. Serdyukov, and M. de Rijke. Click model-based information retrieval metrics. In *SIGIR '13*. ACM Press, 2013. (Cited on page 26.)

[38] A. Chuklin, P. Serdyukov, and M. de Rijke. Modeling clicks beyond the first result page. In *CIKM '13*. ACM Press, 2013. (Cited on page 25.)

[39] A. Chuklin, K. Zhou, A. Schuth, F. Sietsma, and M. de Rijke. Evaluating Intuitiveness of Vertical-Aware Click Models. In *SIGIR '14*, 2014. (Cited on pages 12, 31, 125, and 126.)

[40] A. Chuklin, I. Markov, and M. de Rijke. Click Models for Web Search, 2015. (Cited on pages 25, 130, and 134.)

[41] A. Chuklin, A. Schuth, K. Zhou, and M. de Rijke. A comparative analysis of interleaving methods for aggregated search. *ACM Transactions on Information Systems (TOIS)*, 2015. (Cited on pages 12, 31, 125, and 126.)

[42] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR '08*, SIGIR '08. ACM, 2008. (Cited on page 17.)

[43] C. W. Cleverdon. Report on the testing and analysis of an investigation into the comparative efficiency of indexing systems. 1962. (Cited on pages 14 and 17.)

[44] C. W. Cleverdon. The Cranfield tests on index language devices. *Aslib proceedings*, 1967. (Cited on

page 14.)

[45] C. W. Cleverdon and M. Keen. Factors Determining the Performance of Indexing Systems. Technical report, 1966. (Cited on pages 1, 14, 17, and 136.)

[46] ComScore. comScore Releases January 2015 U.S. Desktop Search Engine Rankings, 2015. URL http://www.comscore.com/Insights/Market-Rankings/comScore-Releases-January-2015-US-Desktop-Search-Engine-Rankings. Accessed 17-October-2015. (Cited on pages 1 and 18.)

[47] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM '08*. ACM Press, 2008. (Cited on pages 25, 130, and 163.)

[48] F. Diaz, R. W. White, G. Buscher, and D. Liebling. Robust Models of Mouse Movement on Dynamic Web Search Results Pages. In *CIKM '13*. ACM Press, 2013. (Cited on pages 19 and 20.)

[49] P. Donmez and J. G. Carbonell. Active Sampling for Rank Learning via Optimizing the Area under the ROC Curve. In *Advances in Information Retrieval*, 2009. (Cited on page 28.)

[50] L. B. Doyle. Semantic Road Maps for Literature Searchers. *Journal of the ACM*, 8(4), 1961. (Cited on page 14.)

[51] S. T. Dumais and N. J. Belkin. The TREC interactive tracks: Putting the user into search. *TREC: Experiment and evaluation in information retrieval*, 2005. (Cited on page 137.)

[52] G. E. Dupret and M. Lalmas. Absence time and user engagement: Evaluating ranking functions. In *WSDM '13*, 2013. (Cited on page 68.)

[53] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *SIGIR '08*, 2008. (Cited on pages 25 and 166.)

[54] Edelman. Trust Around the World, 2015. URL http://www.edelman.com/insights/intellectual-property/2015-edelman-trust-barometer/trust-around-world/. Accessed 17-October-2015. (Cited on page 1.)

[55] S. Eliot and J. Rose. *A Companion to the History of the Book*. 2009. (Cited on page 13.)

[56] S. Fox, K. Karnawat, M. Mydland, S. T. Dumais, and T. White. Evaluating Implicit Measures to Improve Web Search. *ACM Transactions on Information Systems (TOIS)*, 23, 2005. (Cited on page 68.)

[57] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4, 2003. (Cited on page 27.)

[58] N. Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems*, 7(3), 1989. (Cited on page 26.)

[59] J. Fürnkranz and E. Hüllermeier. *Preference learning*. 2010. (Cited on page 27.)

[60] N. Gao, Z.-H. Deng, H. Yu, and J.-J. Jiang. ListOPT: Learning to Optimize for XML Ranking. In *Advances in Knowledge Discovery and Data Mining*, LNCS. Springer, 2011. (Cited on pages 92 and 94.)

[61] E. Garfield. Science Citation Index-A new dimension in indexing. *Science*, 1964. (Cited on page 16.)

[62] E. Goldberg. Statistical machine. *US Patent 1,838,389*, 1931. (Cited on page 13.)

[63] L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in WWW search. In *SIGIR '04*. ACM Press, 2004. (Cited on page 18.)

[64] F. Guo, L. Li, and C. Faloutsos. Tailoring click models to user goals. In *WSCD '09*, 2009. (Cited on pages 130 and 133.)

[65] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM '09*, 2009. (Cited on pages 25, 33, 62, 63, 96, 111, 112, 117, 130, 133, and 164.)

[66] Q. Guo and E. Agichtein. Understanding "Abandoned" Ads: Towards Personalized Commercial Intent Inference via Mouse Movement Analysis. *SIGIR-IRA*, 2008. (Cited on pages 19 and 20.)

[67] Q. Guo and E. Agichtein. Towards predicting web searcher gaze position from mouse movements. In

CHI EA '10. ACM Press, 2010. (Cited on pages 19 and 20.)

[68] D. K. Harman. Overview of the second text retrieval conference (TREC-2). *HLT '94*, 1994. (Cited on pages 17 and 165.)

[69] S. Harter. *A probabilistic approach to automatic keyword indexing*. PhD thesis, University of Chicago, 1974. (Cited on page 16.)

[70] A. Hassan, R. Jones, and K. L. Klinkner. Beyond DCG: user behavior as a predictor of a successful search. In *WSDM '10*, 2010. (Cited on page 21.)

[71] A. Hassan, X. Shi, N. Craswell, and B. Ramsey. Beyond clicks: query reformulation as a predictor of search satisfaction. In *CIKM '13*, 2013. (Cited on pages 19 and 20.)

[72] J. He, C. Zhai, and X. Li. Evaluation of methods for relative comparison of retrieval systems based on clickthroughs. In *CIKM '09*. ACM Press, 2009. (Cited on pages 21, 129, and 164.)

[73] Y. He and K. Wang. Inferring search behaviors using partially observable markov model with duration (POMD). In *WSDM '11*. ACM Press, 2011. (Cited on pages 19 and 20.)

[74] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *ICANN '99*, volume 1, 1999. (Cited on page 27.)

[75] W. Hersh, A. H. Turpin, S. Price, B. Chan, D. Kramer, L. Sacherek, and D. Olson. Do batch and user evaluations give the same results? In *SIGIR '00*. ACM Press, 2000. (Cited on pages 18, 28, and 67.)

[76] D. Hiemstra. A linguistically motivated probabilistic model of information retrieval. *Research and advanced technology for digital libraries*, 1998. (Cited on pages 16 and 164.)

[77] K. Hofmann. *Fast and Reliably Online Learning to Rank for Information Retrieval*. PhD thesis, University of Amsterdam, 2013. (Cited on pages 6, 17, 33, 34, 110, 125, 126, 128, 129, and 131.)

[78] K. Hofmann, S. Whiteson, and M. de Rijke. Contextual Bandits for Information Retrieval. *NIPS '11*, 2011.

[79] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM '11*. ACM Press, 2011. (Cited on pages 5, 21, 24, 25, 29, 39, 46, 47, 50, 51, 92, 93, 95, 125, 126, 129, 146, and 165.)

[80] K. Hofmann, F. Behr, and F. Radlinski. On Caption Bias in Interleaving Experiments. In *CIKM '12*, 2012. (Cited on pages 70 and 77.)

[81] K. Hofmann, S. Whiteson, and M. de Rijke. Estimating Interleaved Comparison Outcomes from Historical Click Data. In *CIKM '12*, 2012. (Cited on pages 95, 125, and 126.)

[82] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing Exploration and Exploitation in Listwise and Pairwise Online Learning to Rank for Information Retrieval. *Information Retrieval*, 16(1), 2012. (Cited on pages 3, 28, 92, 93, 105, and 126.)

[83] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing Historical Interaction Data for Faster Online Learning to Rank for IR. In *WSDM '13*, 2013. (Cited on pages 6, 12, 22, 28, 29, 30, 31, 32, 39, 49, 92, 93, 95, 96, 109, 126, 128, and 163.)

[84] K. Hofmann, A. Schuth, A. Bellogín, and M. de Rijke. Effects of Position Bias on Click-Based Recommender Evaluation. In *ECIR '14*, 2014. (Cited on pages 12, 31, 125, and 126.)

[85] K. Hofmann, S. Whiteson, and M. de Rijke. Fidelity, Soundness, and Efciency of Interleaved Comparison Methods. *ACM Transactions on Information Systems (TOIS)*, 31(3), 2014. (Cited on pages 43 and 56.)

[86] K. Hofmann, S. Whiteson, A. Schuth, and M. de Rijke. Learning to rank for information retrieval from user interactions. *ACM SIGWEB Newsletter*, 2014. (Cited on page 12.)

[87] H. Hollerith. The electrical tabulating machine. *Journal of the Royal Statistical Society*, 1894. (Cited on page 13.)

[88] J. Huang, T. Lin, and R. W. White. No search result left behind. In *WSDM '12*. ACM Press, 2012. (Cited on page 20.)

[89] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4), 2002. (Cited on pages 17, 34, 40, 49, 57, 97, 110, 131, 137, and 165.)

[90] S. Ji, K. Zhou, C. Liao, Z. Zheng, G.-R. Xue, O. Chapelle, G. Sun, and H. Zha. Global ranking by exploiting user clicks. In *SIGIR '09*. ACM, 2009. (Cited on page 91.)

[91] T. Joachims. Making Large-Scale SVM Learning Practical. *Advances in Kernel Methods - Support Vector Learning*, 1999. (Cited on page 132.)

[92] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*, 2002. (Cited on pages 3, 4, 7, 27, 28, 38, 91, 105, 106, 128, and 146.)

[93] T. Joachims. Evaluating Retrieval Performance using Clickthrough Data. In *Text Mining*. Physica/Springer, 2003. (Cited on pages 4, 7, 21, 24, 38, 68, 73, 106, and 146.)

[94] T. Joachims. Training linear SVMs in linear time. In *KDD '06*. ACM Press, 2006. (Cited on page 126.)

[95] T. Joachims, L. A. 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 Transactions on Information Systems (TOIS)*, 25(2), 2007. (Cited on pages 6, 19, 20, 21, 129, and 163.)

[96] S. Jung, J. L. Herlocker, and J. Webster. Click data as implicit relevance feedback in web search. *Information Processing & Management (IPM)*, 43(3), 2007. (Cited on page 91.)

[97] M. H. Kalos and P. A. Whitlock. Monte Carlo methods. Vol. 1: basics. 1986. (Cited on page 24.)

[98] J. Kamps, S. Geva, C. Peters, T. Sakai, A. Trotman, and E. M. Voorhees. Report on the SIGIR 2009 Workshop on the Future of IR Evaluation. *SIGIR Forum*, 43(2), 2009. (Cited on page 137.)

[99] E. Kanoulas, P. Clough, B. Carterette, and M. Sanderson. Session track at TREC 2010. *SIGIR Workshop on the Simulation of Interaction*, 2010. (Cited on page 137.)

[100] G. Kazai, E. Yilmaz, N. Craswell, and S. Tahaghoghi. User intent and assessor disagreement in web search evaluation. In *CIKM '13*. ACM Press, 2013. (Cited on pages 18 and 28.)

[101] D. Kelly. Methods for Evaluating Interactive Information Retrieval Systems with Users. *Foundations and Trends in Information Retrieval*, 3(1–2), 2009. (Cited on page 18.)

[102] D. Kelly and J. Teevan. Implicit Feedback for Inferring User Preference: A Bibliography. *SIGIR Forum*, 37(2), 2003. (Cited on pages 19 and 21.)

[103] D. Kelly, S. T. Dumais, and J. O. Pedersen. Evaluation Challenges and Directions for Information-Seeking Support Systems. *Computer*, 42(3), 2009. (Cited on pages 136 and 137.)

[104] L. Kelly, P. Bunbury, and G. J. F. Jones. Evaluating Personal Information Retrieval. In *ECIR '12*, 2012. (Cited on page 138.)

[105] E. Kharitonov. Improving offline and online web search evaluation by modelling the user behaviour. In *SIGIR '14*. ACM Press, 2014. (Cited on page 26.)

[106] E. Kharitonov, C. Macdonald, P. Serdyukov, I. Ounis, Ounis, Iadh, and I. Ounis. Using Historical Click Data to Increase Interleaving Sensitivity. In *CIKM '13*. ACM Press, 2013. (Cited on pages 22 and 166.)

[107] E. Kharitonov, C. Macdonald, P. Serdyukov, and I. Ounis. Generalized Team Draft Interleaving. In *CIKM '15*. ACM, 2015. (Cited on pages 22 and 164.)

[108] E. Kharitonov, C. Macdonald, P. Serdyukov, and I. Ounis. Optimised Scheduling of Online Experiments. In *SIGIR '15*. ACM Press, 2015. (Cited on pages 19 and 20.)

[109] E. Kharitonov, A. Vorobev, C. Macdonald, P. Serdyukov, and I. Ounis. Sequential Testing for Early Stopping of Online Experiments. In *SIGIR '15*. ACM Press, 2015. (Cited on pages 19 and 20.)

[110] Y. Kim, A. Hassan, R. W. White, and I. Zitouni. Modeling dwell time to predict click-level satisfaction. In *WSDM '14*, 2014. (Cited on pages 20, 70, 71, and 84.)

[111] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), 1999. (Cited on page 16.)

[112] R. Kohavi. Online Controlled Experiments. In *Proceedings of the 1st workshop on User engagement optimization*. ACM, 2013. (Cited on pages 5, 37, and 147.)

[113] R. Kohavi, R. Longbotham, D. Sommerfield, and R. Henne. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1), 2009. (Cited on pages 19, 20, 68, 74, and 136.)

[114] C. Kuhlthau. Inside the search process: Information seeking from the user's perspective. *JASIS*, 1991. (Cited on page 2.)

[115] D. Lagun, C. H. Hsieh, D. Webster, and V. Navalpakkam. Towards Better Measurement of Attention and Satisfaction in Mobile Search. In *SIGIR '14*, 2014. (Cited on pages 19 and 20.)

[116] F. Lancaster and E. Fayen. Information Retrieval On-Line, Melville Publ. *Co., Los Angeles, Calif*, 1973. (Cited on pages 15 and 163.)

[117] U. Lee, Z. Liu, and J. Cho. Automatic identification of user goals in Web search. In *WWW '05*. ACM Press, 2005. (Cited on page 16.)

[118] E. L. Lehmann and J. P. Romano. *Testing statistical hypotheses*. springer, 2006. (Cited on page 74.)

[119] J. Li, S. Huffman, and A. Tokuda. Good Abandonment in Mobile and PC Internet Search. In *SIGIR '09*, 2009. (Cited on page 20.)

[120] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM '11*, 2011. (Cited on pages 25, 28, and 31.)

[121] M. Li, H. Li, and Z.-H. Zhou. Semi-supervised document retrieval. *Information Processing & Management (IPM)*, 45(3), 2009. (Cited on page 20.)

[122] W.-H. Lin and A. Hauptmann. Revisiting the effect of topic set size on retrieval error. In *SIGIR '05*. ACM Press, 2005. (Cited on page 17.)

[123] A. Lipani, M. Lupu, and A. Hanbury. Splitting Water: Precision and Anti-Precision to Reduce Pool Bias. *SIGIR '15*, 2015. (Cited on page 17.)

[124] T.-Y. Liu. *Learning to rank for information retrieval*, volume 3. Now Pub, 2009. (Cited on pages 3, 26, 27, 95, 105, 125, and 165.)

[125] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval. In *LR4IR '07*, 2007. (Cited on pages 31, 32, and 132.)

[126] H. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1957. (Cited on pages 15 and 166.)

[127] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. (Cited on pages 14, 15, 16, and 164.)

[128] I. Markov, E. Kharitonov, V. Nikulin, P. Serdyukov, M. de Rijke, and F. Crestani. Vertical-Aware Click Model-Based Effectiveness Metrics. In *CIKM '14*. ACM Press, 2014. (Cited on page 25.)

[129] M. Marx and A. Schuth. DutchParl The Parliamentary Documents in Dutch. *LREC '10*, 5(1), 2010. (Cited on page 12.)

[130] M. Marx, N. Aders, and A. Schuth. Digital sustainable publication of legacy parliamentary proceedings. *AI Magazine*, 18(4), 2010. (Cited on page 12.)

[131] N. Meinshausen. Quantile Regression Forests. *Journal of Machine Learning Research*, 7, 2006. (Cited on page 71.)

[132] S. Mizzaro. Relevance: The whole history. *JASIS*, 1997. (Cited on page 15.)

[133] A. Moffat and J. Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems (TOIS)*, 27(1), 2008. (Cited on page 137.)

[134] C. Mooers. The theory of digital handling of non-numerical information and its implications to machine economics. 1950. (Cited on pages 14 and 16.)

[135] C. Mooers. The next twenty years in information retrieval; some goals and predictions. *American Documentation*, 1960. (Cited on page 14.)

[136] R. Nallapati. Discriminative models for information retrieval. In *SIGIR '04*. ACM Press, 2004. (Cited on page 26.)

[137] A. Nusselder, H. Peetz, A. Schuth, and M. Marx. Helping people to choose for whom to vote. a web information system for the 2009 European elections. 2009. (Cited on page 12.)

[138] H. Oosterhuis, A. Schuth, and M. de Rijke. Probabilistic Multileave Gradient Descent. In *ECIR '16*. Springer, 2016. (Cited on pages 12, 31, 121, 125, and 126.)

[139] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing Order to the Web. *World Wide Web Internet And Web Information Systems*, 54(1999-66), 1998. (Cited on pages 3 and 16.)

[140] J. Ponte and W. B. Croft. A language modeling approach to information retrieval. *SIGIR '98*, 1998. (Cited on pages 16 and 164.)

[141] 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), 2010. (Cited on pages 31, 32, 94, 96, 97, 98, and 100.)

[142] F. Radlinski and N. Craswell. Comparing the sensitivity of information retrieval metrics. In *SIGIR '10*, 2010. (Cited on pages 17 and 22.)

[143] F. Radlinski and N. Craswell. Optimized interleaving for online retrieval evaluation. In *WSDM '13*. ACM Press, 2013. (Cited on pages 4, 22, 38, 41, 42, 43, 44, 45, 50, 60, 70, 107, 109, 129, 146, and 165.)

[144] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM '08*. ACM Press, 2008. (Cited on pages 3, 4, 6, 7, 21, 23, 28, 29, 38, 41, 50, 68, 70, 72, 91, 95, 105, 106, 107, 126, 129, 141, 146, and 166.)

[145] S. Robertson. The probability ranking principle in IR. *Journal of documentation*, 1977. (Cited on page 15.)

[146] S. Robertson and S. Walker. Okapi at TREC-3. *NIST SPECIAL PUBLICATION SP*, 1995. (Cited on pages 3, 6, 15, 91, 92, 94, and 163.)

[147] S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3(4), 2009. (Cited on pages 6, 93, and 147.)

[148] J. Rocchio. Relevance feedback in information retrieval. 1971. (Cited on page 19.)

[149] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review*, 2003. (Cited on page 19.)

[150] T. Sakai and Z. Dou. Summaries, Ranked Retrieval and Sessions: A Unified Framework for Information Access Evaluation. In *SIGIR '13*, 2013. (Cited on page 137.)

[151] G. Salton. Automatic information organization and retrieval. 1968. (Cited on page 14.)

[152] G. Salton. Historical note: The past thirty years in information retrieval. *American Documentation*, 11 (3), 1987. (Cited on page 16.)

[153] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 1975. (Cited on pages 15 and 166.)

[154] G. Salton, E. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 1983. (Cited on page 15.)

[155] M. Sanderson. Test Collection Based Evaluation of Information Retrieval Systems. *Foundations and Trends in Information Retrieval*, 4(4), 2010. (Cited on pages 1, 2, 3, 4, 7, 17, 18, 26, 28, 38, 67, and 105.)

[156] M. Sanderson and W. B. Croft. The History of Information Retrieval Research. *Proceedings of the IEEE*, 100(Special Centennial Issue), 2012. (Cited on pages 13 and 16.)

[157] M. Sanderson and H. Joho. Forming test collections with no system pooling. In *SIGIR '04*. ACM, 2004.

(Cited on page 18.)

[158] T. Schnabel, A. Swaminathan, and T. Joachims. Unbiased Ranking Evaluation on a Budget. *WWW '15*, 2015. (Cited on page 25.)

[159] A. Schuth and M. Marx. Evaluation methods for rankings of facetvalues for faceted search. In *CLEF '11*, volume 6941. Springer Berlin Heidelberg, 2011. (Cited on pages 12 and 17.)

[160] A. Schuth, M. Marx, and M. de Rijke. Extracting the discussion structure in comments on news-articles. In *WIDM '07*, 2007. (Cited on page 12.)

[161] A. Schuth, K. Hofmann, S. Whiteson, and M. de Rijke. Lerot: an Online Learning to Rank Framework. In *LivingLab '13*. ACM Press, 2013. (Cited on pages 11, 12, 31, 32, 49, 96, and 125.)

[162] A. Schuth, F. Sietsma, S. Whiteson, and M. de Rijke. Optimizing Base Rankers Using Clicks: A Case Study using BM25. In *ECIR '14*, 2014. (Cited on pages 12, 31, 91, 125, and 126.)

[163] A. Schuth, F. Sietsma, S. Whiteson, D. Lefortier, and M. de Rijke. Multileaved Comparisons for Fast Online Evaluation. In *CIKM '14*. ACM Press, 2014. (Cited on pages 11, 31, 37, 63, 107, 125, 126, 130, and 141.)

[164] A. Schuth, K. Balog, and L. Kelly. Extended Overview of the Living Labs for Information Retrieval Evaluation (LL4IR) CLEF Lab 2015. In *CLEF 2015 Online Working Notes*, 2015. (Cited on pages 12, 135, 136, 142, and 164.)

[165] A. Schuth, R.-J. R. Bruintjes, F. Büttner, J. van Doorn, C. Groenland, H. Oosterhuis, C.-N. Tran, B. Veeling, J. van der Velde, R. Wechsler, D. Woudenberg, and M. de Rijke. Probabilistic Multileave for Online Retrieval Evaluation. In *SIGIR '15*, 2015. (Cited on pages 12, 31, 37, 125, 126, 130, and 141.)

[166] A. Schuth, K. Hofmann, and F. Radlinski. Predicting Search Satisfaction Metrics with Interleaved Comparisons. In *SIGIR '15*, 2015. (Cited on pages 12, 21, 67, and 141.)

[167] A. Schuth, H. Oosterhuis, S. Whiteson, and M. de Rijke. Multileave Gradient Descent for Fast Online Learning to Rank. In *WSDM '16*, 2016. (Cited on pages 11, 12, 31, 105, 125, and 126.)

[168] D. Sculley. Large scale learning to rank. In *NIPS 2009 Workshop on Advances in Ranking*, 2009. (Cited on pages 27, 126, and 128.)

[169] M. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *CIKM '07*. ACM Press, 2007. (Cited on page 34.)

[170] Y. Song, X. Shi, R. W. White, and A. Hassan. Context-Aware Web Search Abandonment Prediction. In *SIGIR '14*, 2014. (Cited on page 20.)

[171] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972. (Cited on pages 15, 92, 164, and 166.)

[172] K. Sparck Jones. *Information retrieval experiment*. Butterworth-Heinemann, 1981. (Cited on page 14.)

[173] K. Sparck Jones. *Readings in information retrieval*. Morgan Kaufmann, 1997. (Cited on page 16.)

[174] K. Sparck Jones, S. Walker, and S. Robertson. A probabilistic model of information retrieval: Development and comparative experiments. *Information Processing & Management (IPM)*, 36, 2000. (Cited on page 93.)

[175] A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the Web: The Public and Their Queries. *J. Am. Soc. Inf. Sci. Technol.*, 52(3), 2001. (Cited on page 140.)

[176] A. Strehl, J. Langford, L. Li, and S. M. Kakade. Learning from Logged Implicit Exploration Data. In *NIPS '10*, 2010. (Cited on page 25.)

[177] A. L. Strehl, C. Mesterharm, M. L. Littman, and H. Hirsh. Experience-efficient learning in associative bandit problems. In *ICML '06*, 2006. (Cited on page 28.)

[178] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998. (Cited on pages 28, 34, 115, and 166.)

[179] K. M. Svore and C. J. Burges. A machine learning approach for improved BM25 retrieval. In *CIKM '09*.

ACM Press, 2009. (Cited on pages 93, 97, and 110.)

[180] A. Swaminathan and T. Joachims. Counterfactual Risk Minimization: Learning from Logged Bandit Feedback. *arXiv:1502.02362*, 2015. (Cited on page 25.)

[181] A. Swaminathan and T. Joachims. Counterfactual Risk Minimization. *WWW '15*, 2015. (Cited on page 25.)

[182] M. Szummer and E. Yilmaz. Semi-supervised learning to rank with preference regularization. In *CIKM '11*. ACM, 2011. (Cited on pages 26 and 28.)

[183] J. M. Tague, M. J. Nelson, and H. Wu. Problems in the simulation of bibliographic retrieval systems. In *SIGIR '80*, 1980. (Cited on page 18.)

[184] H. Tavani. Search Engines and Ethics. In *The Stanford Encyclopedia of Philosophy*. 2014. (Cited on page 1.)

[185] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. J. Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM '06*. ACM Press, 2006. (Cited on pages 92, 93, 97, and 110.)

[186] R. Taylor. The process of asking questions. *American documentation*, 1962. (Cited on page 14.)

[187] J. Teevan, S. T. Dumais, and E. Horvitz. The potential value of personalizing search. In *SIGIR '07*, 2007. (Cited on pages 1 and 67.)

[188] J. Teevan, S. T. Dumais, and E. Horvitz. Potential for personalization. *TOCHI '10*, 17(1), 2010. (Cited on page 16.)

[189] P. Thomas and D. Hawking. Evaluation by comparing result sets in context. In *CIKM '06*. ACM, 2006. (Cited on page 19.)

[190] W. Thomson. Electrical units of measurement. *Popular lectures and addresses*, 1883. (Cited on pages 1 and 3.)

[191] A. Tombros, B. Larsen, and S. Malik. The interactive track at INEX 2004. *Advances in XML Information Retrieval*, 2005. (Cited on page 137.)

[192] A. H. Turpin and W. Hersh. Why batch and user evaluations do not give the same results. In *SIGIR '01*. ACM Press, 2001. (Cited on pages 18, 28, and 67.)

[193] A. H. Turpin and F. Scholar. User performance versus precision measures for simple search tasks. In *SIGIR '06*, 2006. (Cited on pages 1 and 67.)

[194] T. Urvoy, F. Clerot, R. Féraud, and S. Naamane. Generic Exploration and K-armed Voting Bandits. In *ICML '13*, 2013. (Cited on pages 24 and 64.)

[195] B. Vickery. The structure of information retrieval systems. *Proceedings of the International Conference on Scientific Information*, 1959. (Cited on page 15.)

[196] E. M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing & Management (IPM)*, 36(5), 2000. (Cited on pages 2 and 82.)

[197] E. M. Voorhees. The philosophy of information retrieval evaluation. *Evaluation of cross-language information retrieval systems*, 2002. (Cited on pages 17, 135, and 166.)

[198] E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval*. Digital Libraries and Electronic Publishing. MIT Press, 2005. (Cited on page 17.)

[199] H. Wang, Y. Song, M. W. Chang, X. He, A. Hassan, and R. W. White. Modeling Action-level Satisfaction for Search Task Satisfaction Prediction. In *SIGIR '14*, 2014. (Cited on page 21.)

[200] K. Wang, T. Walker, and Z. Zheng. PSkip: Estimating Relevance Ranking Quality from Web Search Clickthrough Data. In *KDD '09*, 2009. (Cited on pages 20 and 68.)

[201] K. Wang, N. Gloy, and X. Li. Inferring search behaviors using partially observable Markov (POM) model. In *WSDM '10*. ACM Press, 2010. (Cited on pages 19 and 20.)

[202] R. W. White, J. M. Jose, C. J. V. Rijsbergen, and I. Ruthven. A simulated study of implicit feedback

models. In *ECIR '04*, 2004. (Cited on page 137.)

[203] R. W. White, P. N. Bennett, and S. T. Dumais. Predicting short-term interests using activity-based search context. In *CIKM '10*. ACM Press, 2010. (Cited on page 16.)

[204] Wikipedia. List of countries by number of Internet users, 2015. URL `https://en.wikipedia.org/wiki/List_of_countries_by_number_of_Internet_users`. Accessed 17-October-2015. (Cited on pages 1 and 18.)

[205] Z. Xu, R. Akella, and Y. Zhang. Incorporating Diversity and Density in Active Learning for Relevance Feedback. In *ECIR '07*, 2007. (Cited on page 28.)

[206] E. Yilmaz, M. Verma, N. Craswell, F. Radlinski, and P. Bailey. Relevance and Effort: An Analysis of Document Utility. In *CIKM '14*. ACM, 2014. (Cited on pages 18, 19, 20, 28, 67, 71, and 166.)

[207] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML '09*, 2009. (Cited on pages 3, 6, 7, 17, 24, 28, 92, 93, 95, 105, 106, 110, 126, 128, 129, 133, 147, 148, and 163.)

[208] Y. Yue and T. Joachims. Beat the mean bandit. In *ICML '11*, 2011. (Cited on pages 24 and 163.)

[209] Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The K-armed dueling bandits problem. *Journal of Computer and System Sciences*, 78(5), 2009. (Cited on pages 24, 28, and 65.)

[210] Y. Yue, Y. Gao, O. Chapelle, Y. Zhang, and T. Joachims. Learning more powerful test statistics for click-based retrieval evaluation. In *SIGIR '10*, 2010. (Cited on pages 70 and 77.)

[211] Y. Yue, R. Patel, and H. Roehrig. Beyond position bias: examining result attractiveness as a source of presentation bias in clickthrough data. In *WWW '10*. ACM Press, 2010. (Cited on pages 70 and 77.)

[212] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML '04*. ACM, 2004. (Cited on page 128.)

[213] M. Zoghi, S. Whiteson, M. de Rijke, and R. Munos. Using Confidence Bounds for Efficient On-Line Ranker Evaluation. In *WSDM '14*. ACM, 2014. (Cited on page 24.)

[214] M. Zoghi, S. Whiteson, R. Munos, and M. de Rijke. Relative Upper Confidence Bound for the K-Armed Dueling Bandit Problem. In *ICML '14*, 2014. (Cited on pages 24, 64, and 166.)

[215] M. Zoghi, Z. S. Karnin, S. Whiteson, and M. de Rijke. Copeland Dueling Bandits. In *NIPS '15*, 2015. (Cited on page 24.)

[216] M. Zoghi, S. Whiteson, and M. de Rijke. MergeRUCB: A method for large-scale online ranker evaluation. In *WSDM '15*. ACM Press, 2015. (Cited on pages 24, 31, 64, 125, 126, and 165.)

List of Terms

A/B metric Absolute user interaction metric used in A/B testing. Computed for system A and B, after which a t-test is used to select a winner. Examples are CTR, time to click, and abandonment rate. See Section 2.3.2. (Used on pages 5, 8, 9, 20–22, 67–74, 76–79, 81–88, 147.)

A/B test Controlled experiment conducted on users of a running system. A random sample of users is exposed to the treatment system, a second sample is exposed to the control system. Any differences in online performance measured on the two samples can be attributed to differences between treatment and control system. See Section 2.3.2. (Used on pages 4, 5, 8–10, 19–21, 24, 38, 67–71, 73, 87, 88, 129, 136, 141, 145–147, 149, 150, 163.)

BI *balanced interleave*. The first interleaved comparison method by Joachims et al. [95]. This method was shown to be biased which was corrected in TDI. See Section 2.3.3. (Used on pages 21, 73, 129.)

BM *boolean model*. Retrieval model by Lancaster and Fayen [116] that uses Boolean logic, described in Section 2.1.3. (Used on pages 15, 16.)

BM25 *best match 25*. Widely used retrieval model by Robertson and Walker [146], a weighted variant of TF.IDF, described in Section 2.1.3. See also Chapter 6. (Used on pages 6, 7, 9–11, 15, 16, 26, 32, 49, 91–102, 147, 148, 150.)

BTM *beat the mean*. Algorithm by Yue and Joachims [208] for the *K-armed dueling bandits problem*, see Section 2.3.4. (Used on pages 24.)

CCM *cascade click model*. Click model by Craswell et al. [47] that assumes users scan a result list from top to bottom and click on a result as soon as it seems to satisfy their information need, see Section 2.3.6. (Used on pages 25, 110, 164.)

CLEF *conference and labs of the evaluation forum*. The CLEF Initiative is a self-organized body whose main mission is to promote research, innovation, and development of information access systems with an emphasis on multilingual and multimodal information with various levels of structure. (Used on pages 137, 138, 143, 149, 152, 164.)

CPS *candidate preselection*. Extension of DBGD by Hofmann et al. [83] that uses PI to select a promising candidate using historical interaction data. See Section 2.5.2. (Used on pages 29, 95, 96, 126, 128.)

CTR *click through rate*. A common A/B metric: the number of result lists with a click over the total number of impressions. (Used on pages 20, 68, 71, 163.)

DBGD *dueling bandit gradient descent*. Online learning to rank method by Yue and Joachims [207], see Section 2.5.1 for details. We extend DBGD to MGD in Chapter 7. (Used on pages 7, 24, 28–30, 65, 95, 99, 101, 105–121, 126–129, 133, 147, 148, 151, 163, 165.)

DBN *dynamic bayesian network model*. Click model by Chapelle and Zhang [32] that extends CCM, see Section 2.3.6. (Used on pages 25.)

DCI *document constraints interleave*. Interleaving method by He et al. [72] which infers constraints on documents pairs based on the rank of the documents and on user interactions. See Section 2.3.3. (Used on pages 21, 129.)

DCM *dependent click model*. This cascading click model by Guo et al. [65] is used throughout this thesis to simulate clicks. See Section 2.3.6 for details and see Table 3.2 for instantiations. (Used on pages 25, 32–34, 96, 111, 112, 117, 130–133.)

DFR *divergence from randomness*. Retrieval model by Amati [8] based on the 2-Poisson indexing model, described in Section 2.1.3. (Used on pages 6, 16, 92.)

ERR *expected reciprocal rank*. Offline evaluation metric by Chapelle et al. [33], see Section 2.2.1. (Used on pages 17, 38.)

FCM *federated click model*. Click model by [35] that takes attraction of vertical documents into account, see Section 2.3.6 and Section 8.2.3. (Used on pages 131.)

GTDI *generalized team draft interleave*. Interleaving method by Kharitonov et al. [107] which jointly optimizes for credit assignment and an interleaving policy. See Section 2.3.3. (Used on pages 22.)

IDF *inverse document frequency*. "The specificity of a term can be quantified as an inverse function of the number of documents in which it occurs" by Sparck Jones [171], described in Section 2.1.3. (Used on pages 15, 93, 166.)

IR *information retrieval*. "Information retrieval is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)." by Manning et al. [127]. See Section 2.1. (Used on pages 1, 6–9, 11, 13–17, 21, 26–28, 67–69, 125, 126, 135–138, 145, 147–150, 152, 165.)

Lerot *learning and evaluating rankers online toolkit*. Implementation of the online evaluation framework introduced in Chapter 8. (Used on pages 8, 10, 12, 125–135, 145, 148, 149, 151, 152.)

LL4IR *living labs for IR evaluation lab at CLEF*. LL4IR is the CLEF lab [164] that uses the OpenSearch methodology introduced in Chapter 9 of this thesis. (Used on pages 135–137, 142, 143, 149.)

LM *language model*. Ranking model in which documents and queries are modeled as sequences drawn from a probability distribution over sequences. Documents are then ranked by the probability that document and query come from the same underlying distribution [76, 140]. See Section 2.1.3. (Used on pages 16, 26, 32.)

LTR *learning to rank*. LTR is about finding an optimal combination of ranking features using machine learning techniques [124]. See Section 2.4. (Used on pages 3, 26–28, 125, 135.)

MAP *mean average precisions*. Offline evaluation metric by Harman [68], see Section 2.2.1. (Used on pages 17, 38.)

MGD *multileave gradient descent*. Online learning to rank extension of DBGD that learns from multileaving methods such as TDM, introduced in Chapter 7 of this thesis. (Used on pages 7, 9, 105–110, 115–121, 126–128, 148, 151, 163, 165.)

MGD-M *MGD mean winner*. Variant of MGD that updates towards the mean of all winners, introduced in Chapter 7 of this thesis. (Used on pages 7, 10, 106, 109, 110, 113–121, 148, 151.)

MGD-W *MGD winner takes all*. Variant of MGD that randomly selects a winner, introduced in Chapter 7 of this thesis. (Used on pages 7, 10, 106, 109, 110, 113–121, 148, 151.)

MRUCB *multileave RUCB*. Variant of RUCB by Zoghi et al. [216], an algorithm for the K*-armed dueling bandits problem*, see Section 2.3.4. (Used on pages 64.)

nDCG *normalized discounted cumulative gain*. Offline evaluation metric by Järvelin and Kekäläinen [89] that is used throughout this thesis, see Section 2.2.1 and Section 3.4. (Used on pages 17, 20, 34, 38, 40, 49, 50, 57–59, 97–99, 101, 110–117, 131.)

OI *optimized interleave*. Recent interleaving method by Radlinski and Craswell [143] that optimizes for sensitivity. See Section 2.3.3. We extended OI to OM. (Used on pages 4, 9, 10, 22, 37, 38, 41, 50, 51, 53, 54, 56–61, 65, 107, 129, 146, 165.)

OM *optimized multileave*. Multileaving method introduced in Chapter 4 of this thesis. OM extends OI. (Used on pages 4, 5, 8–10, 37–39, 41, 43, 45, 46, 50–61, 65, 107, 130, 146, 165.)

OpenSearch OpenSearch is a new evaluation paradigm for IR. The experimentation platform is an existing search engine. Researchers have the opportunity to replace components of this search engine and evaluate these components using interactions with real, unsuspecting users of this search engine. See Chapter 9. (Used on pages 8, 9, 11, 12, 121, 135–140, 142, 143, 149, 151, 152, 164.)

PI *probabilistic interleave*. An probabilistic generalization of TDI by Hofmann et al. [79]. This interleaving method has a non-zero probability of any interleaving occurring as documents are sampled from a softmax distribution over the ranking instead of a deterministic ranking. See Section 2.3.3. We extended PI to PM. (Used on pages 5, 8, 21, 22, 24, 29, 37, 39, 46–50, 61–63, 65, 129, 130, 146, 147, 163, 165.)

PM *probabilistic multileave*. PM is a multileaving method that extends PI and is introduced in Chapter 4 of this thesis. (Used on pages 5, 8, 10, 37, 39–41, 46–48, 50, 51, 61–63, 65, 121, 130, 146, 147, 165.)

RCM *random click model*. A click model that mimics a user not having any preference, this click model is used to validate the absence of bias in evaluation methods, see Section 2.3.6 and Section 8.2.3. (Used on pages 131.)

RL *reinforcement learning*. Machine learning from interactions with users, see the book by Sutton and Barto [178]. (Used on pages 28, 34.)

RUCB *relative upper confidence bound*. Algorithm by Zoghi et al. [214] for the *K-armed dueling bandits problem*, see Section 2.3.4. (Used on pages 24, 64, 165.)

SAT *satisfied*. Click that satisfied the user, typically with dwell time above a fixed cutoff of 30 seconds [206]. See Section 5.3.1. (Used on pages 71, 78, 83.)

TDI *team draft interleave*. Popular interleaving method by Radlinski et al. [144]. See Section 2.3.3. We extended TDI to TDM. (Used on pages 4, 9, 10, 21–23, 28, 29, 37, 38, 41, 46, 47, 50, 51, 53, 54, 56–58, 65, 72, 73, 76–79, 81–84, 86, 87, 107, 129, 141, 146, 163, 165, 166.)

TDI-VA *vertical aware team draft interleave*. Extension of TDI by [36] that can interleave result lists with verticals. (Used on pages 129, 131.)

TDM *team draft multileave*. TDM is a multileaving method that extends TDI and is introduced in Chapter 4 of this thesis. (Used on pages 4, 5, 7–10, 37–39, 41, 42, 46, 47, 49–59, 61–63, 65, 107, 108, 118, 121, 130, 146, 151, 165, 166.)

TF *term frequency*. The number of occurrences of a term in a document, by Luhn [126]. Can be used as a retrieval model, see Section 2.1.3. (Used on pages 15, 93, 166.)

TF.IDF *term frequency times inverse document frequency*. Retrieval model by Sparck Jones [171] that multiplies the TF with the IDF. (Used on pages 6, 15, 16, 49, 92, 163.)

TREC *text retrieval conference*. The text retrieval conference has been organized as a yearly conference for over 25 years [197]. (Used on pages 17, 32, 96, 135–137, 143, 149, 152.)

UBI *upper bound interleave*. Interleaving method by Kharitonov et al. [106] that uses historical click data to increase the sensitivity of interleaving. See Section 2.3.3. (Used on pages 22.)

UBM *user browsing model*. Click model by Dupret and Piwowarski [53] that assumes that the examination probability of a document depends on previous clicks, see Section 2.3.6. (Used on pages 25.)

VSM *vector space model*. The vector space model by Salton et al. [153] represents both documents and queries in a space of terms. Documents can then be ranked by how close they are to the query in this space. See Section 2.1.3. (Used on pages 15, 16.)

Samenvatting

Ruim de helft van de wereldbevolking gebruikt tegenwoordig webzoekmachines. Dagelijks worden meer dan een half miljard zoekvragen gesteld. Zoekmachines zoals Baidu, Bing, Google, en Yandex zijn voor veel mensen het eerste waar ze zich toe wenden als ze een vraag hebben. Sterker nog, zoekmachines zijn voor veel mensen de meest betrouwbare route naar informatie geworden, betrouwbaarder zelfs dan traditionele media zoals kranten, nieuwswebsites en het nieuws op televisie. Mensen worden sterk beïnvloed door hetgeen zoekmachines ze voorschotelen. Het beïnvloedt hun gedachten, meningen, beslissingen en acties. Met dit in gedachten, en vanuit het perspectief van het vakgebied information retrieval (IR), zijn twee zaken belangrijk. Ten eerste is het belangrijk om te begrijpen hoe goed zoekmachines presteren. Ten tweede moeten we ons inzicht hierin gebruiken om zoekmachines te verbeteren. Dit proefschrift gaat over deze twee onderwerpen: de evaluatie van zoekmachines en lerende zoekmachines.

In het eerste deel van dit proefschrift onderzoeken we hoe interacties van zoekmachinegebruikers ingezet kunnen worden om zoekmachines te evalueren. We introduceren een nieuw online evaluatieparadigma dat we *multileaving* noemen. Multileaving is een uitbreiding op interleaving: in plaats van slechts twee rankers kunnen met multileaving meerdere rankers tegelijkertijd met elkaar vergeleken worden. Multileaving combineert de resultaatlijsten van al deze rankers tot een enkele lijst. De interacties van gebruikers met die lijst worden geïnterpreteerd om te bepalen hoe die rankers zich tot elkaar verhouden. Daarnaast bestuderen we de relatie tussen A/B testen en interleaving methoden. Beide studies leiden tot veel hogere gevoeligheid van de evaluatiemethoden. Dit betekent dat met onze methoden veel minder gebruikers-interacties nodig zijn om tot betrouwbare conclusies te komen. Een belangrijke implicatie hiervan is dat veel minder gebruikers worden blootgesteld aan inferieure zoekmachines.

In het tweede deel richten we ons op lerende zoekmachines. We leren van de evaluatiemethoden die we in het eerste deel hebben geintroduceerd. We leren de parameters van basisrankers door naar gebruikerinteracties te kijken. Verder gebruiken we de multileaving methoden uit het eerste deel als feedbackmechanisme in onze leermethode. Dit leidt tot veel snellere convergentie dan bij bestaande methoden. Wederom is een belangrijke implicatie dat minder gebruikers blootgesteld worden aan mogelijk inferieure zoekmachines doordat deze zich nu sneller aanpassen aan de voorkeuren van gebruikers.

Het laatste deel van dit proefschrift is van heel andere aard. In tegenstelling tot de eerste twee delen bestuderen we niet langer algoritmen. Vooruitgang in IR werd altijd al gedreven door een combinatie van algoritmen, gedeelde hulpmiddelen en evaluatie. In het laatste deel besteden we aandacht aan de laatste twee onderwerpen. We introduceren een nieuw gedeeld hulpmiddel en een nieuw evaluatieparadigma. Ten eerste introduceren we Lerot, een online evaluatieraamwerk dat het mogelijk maakt om gebruikers-interactie met zoekmachines te simuleren. Lerot is als opensource-software uitgebracht en wordt door onderzoekers over de hele wereld gebruikt. Ten tweede introduceren we OpenSearch, een nieuw evaluatieparadigma dat gebruikmaakt van *echte* gebruikers van *echte* zoekmachines. We beschrijven een implementatie van dit paradigma dat inmiddels door de onderzoeksgemeenschap gebruikt wordt bij zowel CLEF als TREC.

SIKS Dissertation Series

1998

1 Johan van den Akker (CWI) *DEGAS: An Active, Temporal Database of Autonomous Objects*
2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*
3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations*
4 Dennis Breuker (UM) *Memory versus Search in Games*
5 E. W. Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

1999

1 Mark Sloof (VUA) *Physiology of Quality Change Modelling: Automated modelling of*
2 Rob Potharst (EUR) *Classification using decision trees and neural nets*
3 Don Beal (UM) *The Nature of Minimax Search*
4 Jacques Penders (UM) *The practical Art of Moving Physical Objects*
5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven*
6 Niek J. E. Wijngaards (VUA) *Re-design of compositional systems*
7 David Spelt (UT) *Verification support for object database design*
8 Jacques H. J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism*

2000

1 Frank Niessink (VUA) *Perspectives on Improving Software Maintenance*
2 Koen Holtman (TUe) *Prototyping of CMS Storage Management*
3 Carolien M. T. Metselaar (UvA) *Sociaal-organisatorische gevolgen van kennistechnologie*
4 Geert de Haan (VUA) *ETAG, A Formal Model of Competence Knowledge for User Interface*
5 Ruud van der Pol (UM) *Knowledge-based Query Formulation in Information Retrieval*
6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*
7 Niels Peek (UU) *Decision-theoretic Planning of Clinical Patient Management*
8 Veerle Coupé (EUR) *Sensitivity Analyis of Decision-Theoretic Networks*
9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*
10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*
11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

2001

1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*
2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*
3 Maarten van Someren (UvA) *Learning as problem solving*
4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*
5 Jacco van Ossenbruggen (VUA) *Processing Structured Hypermedia: A Matter of Style*
6 Martijn van Welie (VUA) *Task-based User Interface Design*
7 Bastiaan Schonhage (VUA) *Diva: Architectural Perspectives on Information Visualization*
8 Pascal van Eck (VUA) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*
9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models*
10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice*
11 Tom M. van Engers (VUA) *Knowledge Management*

2002

1 Nico Lassing (VUA) *Architecture-Level Modifiability Analysis*
2 Roelof van Zwol (UT) *Modelling and searching web-based document collections*
3 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*
4 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*
5 Radu Serban (VUA) *The Private Cyberspace Modeling Electronic*
6 Laurens Mommers (UL) *Applied legal epistemology: Building a knowledge-based ontology of*
7 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive*
8 Jaap Gordijn (VUA) *Value Based Requirements Engineering: Exploring Innovative*
9 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy*
10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*
11 Wouter C. A. Wijngaards (VUA) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*
12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*

13 Hongjing Wu (TUe) *A Reference Architecture for Adaptive Hypermedia Applications*
14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*
15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*
16 Pieter van Langen (VUA) *The Anatomy of Design: Foundations, Models and Applications*
17 Stefan Manegold (UvA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003

1 Heiner Stuckenschmidt (VUA) *Ontology-Based Information Sharing in Weakly Structured Environments*
2 Jan Broersen (VUA) *Modal Action Logics for Reasoning About Reactive Systems*
3 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*
4 Milan Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*
5 Jos Lehmann (UvA) *Causation in Artificial Intelligence and Law: A modelling approach*
6 Boris van Schooten (UT) *Development and specification of virtual environments*
7 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*
8 Yongping Ran (UM) *Repair Based Scheduling*
9 Rens Kortmann (UM) *The resolution of visually guided behaviour*
10 Andreas Lincke (UvT) *Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture*
11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*
12 Roeland Ordelman (UT) *Dutch speech recognition in multimedia information retrieval*
13 Jeroen Donkers (UM) *Nosce Hostem: Searching with Opponent Models*
14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*
15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*
16 Menzo Windhouwer (CWI) *Feature Grammar Systems: Incremental Maintenance of Indexes to Digital Media Warehouses*
17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*
18 Levente Kocsis (UM) *Learning Search Decisions*

2004

1 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*
2 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*
3 Perry Groot (VUA) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*
4 Chris van Aart (UvA) *Organizational Principles for Multi-Agent Architectures*
5 Viara Popova (EUR) *Knowledge discovery and monotonicity*
6 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*
7 Elise Boltjes (UM) *Voorbeeldig onderwijs: voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes*
8 Joop Verbeek (UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise*
9 Martin Caminada (VUA) *For the Sake of the Argument: explorations into argument-based reasoning*
10 Suzanne Kabel (UvA) *Knowledge-rich indexing of learning-objects*
11 Michel Klein (VUA) *Change Management for Distributed Ontologies*
12 The Duy Bui (UT) *Creating emotions and facial expressions for embodied agents*
13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*
14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*
15 Arno Knobbe (UU) *Multi-Relational Data Mining*
16 Federico Divina (VUA) *Hybrid Genetic Relational Search for Inductive Learning*
17 Mark Winands (UM) *Informed Search in Complex Games*
18 Vania Bessa Machado (UvA) *Supporting the Construction of Qualitative Knowledge Models*
19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*
20 Madelon Evers (Nyenrode) *Learning from Design: facilitating multidisciplinary design teams*

2005

1 Floor Verdenius (UvA) *Methodological Aspects of Designing Induction-Based Applications*
2 Erik van der Werf (UM) *AI techniques for the game of Go*
3 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*
4 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*
5 Gabriel Infante-Lopez (UvA) *Two-Level Probabilistic Grammars for Natural Language Parsing*

6 Pieter Spronck (UM) *Adaptive Game AI*
7 Flavius Frasincar (TUe) *Hypermedia Presentation Generation for Semantic Web Information Systems*
8 Richard Vdovjak (TUe) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*
9 Jeen Broekstra (VUA) *Storage, Querying and Inferencing for Semantic Web Languages*
10 Anders Bouwer (UvA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*
11 Elth Ogston (VUA) *Agent Based Matchmaking and Clustering: A Decentralized Approach to Search*
12 Csaba Boer (EUR) *Distributed Simulation in Industry*
13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*
14 Borys Omelayenko (VUA) *Web-Service configuration on the Semantic Web: Exploring how semantics meets pragmatics*
15 Tibor Bosse (VUA) *Analysis of the Dynamics of Cognitive Processes*
16 Joris Graaumans (UU) *Usability of XML Query Languages*
17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*
18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*
19 Michel van Dartel (UM) *Situated Representation*
20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*
21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006

1 Samuil Angelov (TUe) *Foundations of B2B Electronic Contracting*
2 Cristina Chisalita (VUA) *Contextual issues in the design and use of information technology in organizations*
3 Noor Christoph (UvA) *The role of metacognitive skills in learning to solve problems*
4 Marta Sabou (VUA) *Building Web Service Ontologies*
5 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*
6 Ziv Baida (VUA) *Software-aided Service Bundling: Intelligent Methods & Tools for Graphical Service Modeling*
7 Marko Smiljanic (UT) *XML schema matching: balancing efficiency and effectiveness by means of clustering*
8 Eelco Herder (UT) *Forward, Back and Home Again: Analyzing User Behavior on the Web*
9 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*
10 Ronny Siebes (VUA) *Semantic Routing in Peer-to-Peer Systems*
11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*
12 Bert Bongers (VUA) *Interactivation: Towards an e-cology of people, our technological environment, and the arts*
13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*
14 Johan Hoorn (VUA) *Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change*
15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*
16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*
17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*
18 Valentin Zhizhkun (UvA) *Graph transformation for Natural Language Processing*
19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*
20 Marina Velikova (UvT) *Monotone models for prediction in data mining*
21 Bas van Gils (RUN) *Aptness on the Web*
22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*
23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*
24 Laura Hollink (VUA) *Semantic Annotation for Retrieval of Visual Resources*
25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*
26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*
27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*
28 Borkur Sigurbjornsson (UvA) *Focused Information Access using XML Element Retrieval*

2007

1 Kees Leune (UvT) *Access Control and Service-Oriented Architectures*
2 Wouter Teepe (RUG) *Reconciling Information Exchange and Confidentiality: A Formal Approach*
3 Peter Mika (VUA) *Social Networks and the Semantic Web*
4 Jurriaan van Diggelen (UU) *Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach*
5 Bart Schermer (UL) *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*

6 Gilad Mishne (UvA) *Applied Text Analytics for Blogs*
7 Natasa Jovanovic' (UT) *To Whom It May Concern: Addressee Identification in Face-to-Face Meetings*
8 Mark Hoogendoorn (VUA) *Modeling of Change in Multi-Agent Organizations*
9 David Mobach (VUA) *Agent-Based Mediated Service Negotiation*
10 Huib Aldewereld (UU) *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*
11 Natalia Stash (TUe) *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*
12 Marcel van Gerven (RUN) *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*
13 Rutger Rienks (UT) *Meetings in Smart Environments: Implications of Progressing Technology*
14 Niek Bergboer (UM) *Context-Based Image Analysis*
15 Joyca Lacroix (UM) *NIM: a Situated Computational Memory Model*
16 Davide Grossi (UU) *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems*
17 Theodore Charitos (UU) *Reasoning with Dynamic Networks in Practice*
18 Bart Orriens (UvT) *On the development an management of adaptive business collaborations*
19 David Levy (UM) *Intimate relationships with artificial partners*
20 Slinger Jansen (UU) *Customer Configuration Updating in a Software Supply Network*
21 Karianne Vermaas (UU) *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*
22 Zlatko Zlatev (UT) *Goal-oriented design of value and process models from patterns*
23 Peter Barna (TUe) *Specification of Application Logic in Web Information Systems*
24 Georgina Ramírez Camps (CWI) *Structural Features in XML Retrieval*
25 Joost Schalken (VUA) *Empirical Investigations in Software Process Improvement*

2008

1 Katalin Boer-Sorbán (EUR) *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*
2 Alexei Sharpanskykh (VUA) *On Computer-Aided Methods for Modeling and Analysis of Organizations*
3 Vera Hollink (UvA) *Optimizing hierarchical menus: a usage-based approach*
4 Ander de Keijzer (UT) *Management of Uncertain Data: towards unattended integration*
5 Bela Mutschler (UT) *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*
6 Arjen Hommersom (RUN) *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective*
7 Peter van Rosmalen (OU) *Supporting the tutor in the design and support of adaptive e-learning*
8 Janneke Bolt (UU) *Bayesian Networks: Aspects of Approximate Inference*
9 Christof van Nimwegen (UU) *The paradox of the guided user: assistance can be counter-effective*
10 Wauter Bosma (UT) *Discourse oriented summarization*
11 Vera Kartseva (VUA) *Designing Controls for Network Organizations: A Value-Based Approach*
12 Jozsef Farkas (RUN) *A Semiotically Oriented Cognitive Model of Knowledge Representation*
13 Caterina Carraciolo (UvA) *Topic Driven Access to Scientific Handbooks*
14 Arthur van Bunningen (UT) *Context-Aware Querying: Better Answers with Less Effort*
15 Martijn van Otterlo (UT) *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains*
16 Henriette van Vugt (VUA) *Embodied agents from a user's perspective*
17 Martin Op 't Land (TUD) *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*
18 Guido de Croon (UM) *Adaptive Active Vision*
19 Henning Rode (UT) *From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search*
20 Rex Arendsen (UvA) *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven*
21 Krisztian Balog (UvA) *People Search in the Enterprise*
22 Henk Koning (UU) *Communication of IT-Architecture*
23 Stefan Visscher (UU) *Bayesian network models for the management of ventilator-associated pneumonia*
24 Zharko Aleksovski (VUA) *Using background knowledge in ontology matching*
25 Geert Jonker (UU) *Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency*
26 Marijn Huijbregts (UT) *Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled*

27 Hubert Vogten (OU) *Design and Implementation Strategies for IMS Learning Design*
28 Ildiko Flesch (RUN) *On the Use of Independence Relations in Bayesian Networks*
29 Dennis Reidsma (UT) *Annotations and Subjective Machines: Of Annotators, Embodied Agents, Users, and Other Humans*
30 Wouter van Atteveldt (VUA) *Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content*
31 Loes Braun (UM) *Pro-Active Medical Information Retrieval*
32 Trung H. Bui (UT) *Toward Affective Dialogue Management using Partially Observable Markov Decision Processes*
33 Frank Terpstra (UvA) *Scientific Workflow Design: theoretical and practical issues*
34 Jeroen de Knijf (UU) *Studies in Frequent Tree Mining*
35 Ben Torben Nielsen (UvT) *Dendritic morphologies: function shapes structure*

2009

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*
2 Willem Robert van Hage (VUA) *Evaluating Ontology-Alignment Techniques*
3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*
4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*
5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks: Based on Knowledge, Cognition, and Quality*
6 Muhammad Subianto (UU) *Understanding Classification*
7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*
8 Volker Nannen (VUA) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*
9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*
10 Jan Wielemaker (UvA) *Logic programming for knowledge-intensive interactive applications*
11 Alexander Boer (UvA) *Legal Theory, Sources of Law & the Semantic Web*
12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin) *Operating Guidelines for Services*
13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*
14 Maksym Korotkiy (VUA) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*
15 Rinke Hoekstra (UvA) *Ontology Representation: Design Patterns and Ontologies that Make Sense*
16 Fritz Reul (UvT) *New Architectures in Computer Chess*
17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*
18 Fabian Groffen (CWI) *Armada, An Evolving Database System*
19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*
20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*
21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*
22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*
23 Peter Hofgesang (VUA) *Modelling Web Usage in a Changing Environment*
24 Annerieke Heuvelink (VUA) *Cognitive Models for Training Simulations*
25 Alex van Ballegooij (CWI) *RAM: Array Database Management through Relational Mapping*
26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*
27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*
28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*
29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*
30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*
31 Sofiya Katrenko (UvA) *A Closer Look at Learning Relations from Text*
32 Rik Farenhorst (VUA) *Architectural Knowledge Management: Supporting Architects and Auditors*
33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*
34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*
35 Wouter Koelewijn (UL) *Privacy en Politiegegevens: Over geautomatiseerde normatieve informatie-uitwisseling*
36 Marco Kalz (OUN) *Placement Support for Learners in Learning Networks*
37 Hendrik Drachsler (OUN) *Navigation Support for Learners in Informal Learning Networks*
38 Riina Vuorikari (OU) *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context*
39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) *Service Substitution: A Behavioral Approach Based on Petri Nets*
40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*
41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*

42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*
43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*
44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*
45 Jilles Vreeken (UU) *Making Pattern Mining Useful*
46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

2010

1 Matthijs van Leeuwen (UU) *Patterns that Matter*
2 Ingo Wassink (UT) *Work flows in Life Science*
3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*
4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*
5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*
6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*
7 Wim Fikkert (UT) *Gesture interaction at a Distance*
8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*
9 Hugo Kielman (UL) *A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging*
10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*
11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*
12 Susan van den Braak (UU) *Sensemaking software for crime analysis*
13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*
14 Sander van Splunter (VUA) *Automated Web Service Reconfiguration*
15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*
16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*
17 Spyros Kotoulas (VUA) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*
18 Charlotte Gerritsen (VUA) *Caught in the Act: Investigating Crime by Agent-Based Simulation*
19 Henriette Cramer (UvA) *People's Responses to Autonomous and Adaptive Systems*
20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*
21 Harold van Heerde (UT) *Privacy-aware data management by means of data degradation*
22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*
23 Bas Steunebrink (UU) *The Logical Structure of Emotions*
24 Zulfiqar Ali Memon (VUA) *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*
25 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*
26 Marten Voulon (UL) *Automatisch contracteren*
27 Arne Koopman (UU) *Characteristic Relational Patterns*
28 Stratos Idreos (CWI) *Database Cracking: Towards Auto-tuning Database Kernels*
29 Marieke van Erp (UvT) *Accessing Natural History: Discoveries in data cleaning, structuring, and retrieval*
30 Victor de Boer (UvA) *Ontology Enrichment from Heterogeneous Sources on the Web*
31 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*
32 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*
33 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*
34 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*
35 Jose Janssen (OU) *Paving the Way for Lifelong Learning: Facilitating competence development through a learning path specification*
36 Niels Lohmann (TUe) *Correctness of services and their composition*
37 Dirk Fahland (TUe) *From Scenarios to components*
38 Ghazanfar Farooq Siddiqui (VUA) *Integrative modeling of emotions in virtual agents*
39 Mark van Assem (VUA) *Converting and Integrating Vocabularies for the Semantic Web*
40 Guillaume Chaslot (UM) *Monte-Carlo Tree Search*
41 Sybren de Kinderen (VUA) *Needs-driven service bundling in a multi-supplier setting: the computational e3-service approach*
42 Peter van Kranenburg (UU) *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*
43 Pieter Bellekens (TUe) *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*
44 Vasilios Andrikopoulos (UvT) *A theory and model for the evolution of software services*

45 Vincent Pijpers (VUA) *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*
46 Chen Li (UT) *Mining Process Model Variants: Challenges, Techniques, Examples*
47 Jahn-Takeshi Saito (UM) *Solving difficult game positions*
48 Bouke Huurnink (UvA) *Search in Audiovisual Broadcast Archives*
49 Alia Khairia Amin (CWI) *Understanding and supporting information seeking tasks in multiple sources*
50 Peter-Paul van Maanen (VUA) *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention*
51 Edgar Meij (UvA) *Combining Concepts and Language Models for Information Access*

2011

1 Botond Cseke (RUN) *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*
2 Nick Tinnemeier (UU) *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language*
3 Jan Martijn van der Werf (TUe) *Compositional Design and Verification of Component-Based Information Systems*
4 Hado van Hasselt (UU) *Insights in Reinforcement Learning: Formal analysis and empirical evaluation of temporal-difference*
5 Base van der Raadt (VUA) *Enterprise Architecture Coming of Age: Increasing the Performance of an Emerging Discipline*
6 Yiwen Wang (TUe) *Semantically-Enhanced Recommendations in Cultural Heritage*
7 Yujia Cao (UT) *Multimodal Information Presentation for High Load Human Computer Interaction*
8 Nieske Vergunst (UU) *BDI-based Generation of Robust Task-Oriented Dialogues*
9 Tim de Jong (OU) *Contextualised Mobile Media for Learning*
10 Bart Bogaert (UvT) *Cloud Content Contention*
11 Dhaval Vyas (UT) *Designing for Awareness: An Experience-focused HCI Perspective*
12 Carmen Bratosin (TUe) *Grid Architecture for Distributed Process Mining*
13 Xiaoyu Mao (UvT) *Airport under Control. Multi-agent Scheduling for Airport Ground Handling*
14 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*
15 Marijn Koolen (UvA) *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*
16 Maarten Schadd (UM) *Selective Search in Games of Different Complexity*
17 Jiyin He (UvA) *Exploring Topic Structure: Coherence, Diversity and Relatedness*
18 Mark Ponsen (UM) *Strategic Decision-Making in complex games*
19 Ellen Rusman (OU) *The Mind ' s Eye on Personal Profiles*
20 Qing Gu (VUA) *Guiding service-oriented software engineering: A view-based approach*
21 Linda Terlouw (TUD) *Modularization and Specification of Service-Oriented Systems*
22 Junte Zhang (UvA) *System Evaluation of Archival Description and Access*
23 Wouter Weerkamp (UvA) *Finding People and their Utterances in Social Media*
24 Herwin van Welbergen (UT) *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior*
25 Syed Waqar ul Qounain Jaffry (VUA) *Analysis and Validation of Models for Trust Dynamics*
26 Matthijs Aart Pontier (VUA) *Virtual Agents for Human Communication: Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots*
27 Aniel Bhulai (VUA) *Dynamic website optimization through autonomous management of design patterns*
28 Rianne Kaptein (UvA) *Effective Focused Retrieval by Exploiting Query Context and Document Structure*
29 Faisal Kamiran (TUe) *Discrimination-aware Classification*
30 Egon van den Broek (UT) *Affective Signal Processing (ASP): Unraveling the mystery of emotions*
31 Ludo Waltman (EUR) *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*
32 Nees-Jan van Eck (EUR) *Methodological Advances in Bibliometric Mapping of Science*
33 Tom van der Weide (UU) *Arguing to Motivate Decisions*
34 Paolo Turrini (UU) *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations*
35 Maaike Harbers (UU) *Explaining Agent Behavior in Virtual Training*
36 Erik van der Spek (UU) *Experiments in serious game design: a cognitive approach*
37 Adriana Burlutiu (RUN) *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference*
38 Nyree Lemmens (UM) *Bee-inspired Distributed Optimization*
39 Joost Westra (UU) *Organizing Adaptation using Agents in Serious Games*
40 Viktor Clerc (VUA) *Architectural Knowledge Management in Global Software Development*
41 Luan Ibraimi (UT) *Cryptographically Enforced Distributed Data Access Control*
42 Michal Sindlar (UU) *Explaining Behavior through Mental State Attribution*

43 Henk van der Schuur (UU) *Process Improvement through Software Operation Knowledge*
44 Boris Reuderink (UT) *Robust Brain-Computer Interfaces*
45 Herman Stehouwer (UvT) *Statistical Language Models for Alternative Sequence Selection*
46 Beibei Hu (TUD) *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work*
47 Azizi Bin Ab Aziz (VUA) *Exploring Computational Models for Intelligent Support of Persons with Depression*
48 Mark Ter Maat (UT) *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent*
49 Andreea Niculescu (UT) *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality*

2012

1 Terry Kakeeto (UvT) *Relationship Marketing for SMEs in Uganda*
2 Muhammad Umair (VUA) *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models*
3 Adam Vanya (VUA) *Supporting Architecture Evolution by Mining Software Repositories*
4 Jurriaan Souer (UU) *Development of Content Management System-based Web Applications*
5 Marijn Plomp (UU) *Maturing Interorganisational Information Systems*
6 Wolfgang Reinhardt (OU) *Awareness Support for Knowledge Workers in Research Networks*
7 Rianne van Lambalgen (VUA) *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions*
8 Gerben de Vries (UvA) *Kernel Methods for Vessel Trajectories*
9 Ricardo Neisse (UT) *Trust and Privacy Management Support for Context-Aware Service Platforms*
10 David Smits (TUe) *Towards a Generic Distributed Adaptive Hypermedia Environment*
11 J. C. B. Rantham Prabhakara (TUe) *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*
12 Kees van der Sluijs (TUe) *Model Driven Design and Data Integration in Semantic Web Information Systems*
13 Suleman Shahid (UvT) *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions*
14 Evgeny Knutov (TUe) *Generic Adaptation Framework for Unifying Adaptive Web-based Systems*
15 Natalie van der Wal (VUA) *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes*
16 Fiemke Both (VUA) *Helping people by understanding them: Ambient Agents supporting task execution and depression treatment*
17 Amal Elgammal (UvT) *Towards a Comprehensive Framework for Business Process Compliance*
18 Eltjo Poort (VUA) *Improving Solution Architecting Practices*
19 Helen Schonenberg (TUe) *What's Next? Operational Support for Business Process Execution*
20 Ali Bahramisharif (RUN) *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing*
21 Roberto Cornacchia (TUD) *Querying Sparse Matrices for Information Retrieval*
22 Thijs Vis (UvT) *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?*
23 Christian Muehl (UT) *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction*
24 Laurens van der Werff (UT) *Evaluation of Noisy Transcripts for Spoken Document Retrieval*
25 Silja Eckartz (UT) *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application*
26 Emile de Maat (UvA) *Making Sense of Legal Text*
27 Hayrettin Gurkok (UT) *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games*
28 Nancy Pascall (UvT) *Engendering Technology Empowering Women*
29 Almer Tigelaar (UT) *Peer-to-Peer Information Retrieval*
30 Alina Pommeranz (TUD) *Designing Human-Centered Systems for Reflective Decision Making*
31 Emily Bagarukayo (RUN) *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure*
32 Wietske Visser (TUD) *Qualitative multi-criteria preference representation and reasoning*
33 Rory Sie (OUN) *Coalitions in Cooperation Networks (COCOON)*
34 Pavol Jancura (RUN) *Evolutionary analysis in PPI networks and applications*
35 Evert Haasdijk (VUA) *Never Too Old To Learn: On-line Evolution of Controllers in Swarm- and Modular Robotics*
36 Denis Ssebugwawo (RUN) *Analysis and Evaluation of Collaborative Modeling Processes*
37 Agnes Nakakawa (RUN) *A Collaboration Process for Enterprise Architecture Creation*
38 Selmar Smit (VUA) *Parameter Tuning and Scientific Testing in Evolutionary Algorithms*
39 Hassan Fatemi (UT) *Risk-aware design of value and coordination networks*
40 Agus Gunawan (UvT) *Information Access for SMEs in Indonesia*

41 Sebastian Kelle (OU) *Game Design Patterns for Learning*
42 Dominique Verpoorten (OU) *Reflection Amplifiers in self-regulated Learning*
43 Anna Tordai (VUA) *On Combining Alignment Techniques*
44 Benedikt Kratz (UvT) *A Model and Language for Business-aware Transactions*
45 Simon Carter (UvA) *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation*
46 Manos Tsagkias (UvA) *Mining Social Media: Tracking Content and Predicting Behavior*
47 Jorn Bakker (TUe) *Handling Abrupt Changes in Evolving Time-series Data*
48 Michael Kaisers (UM) *Learning against Learning: Evolutionary dynamics of reinforcement learning algorithms in strategic interactions*
49 Steven van Kervel (TUD) *Ontologogy driven Enterprise Information Systems Engineering*
50 Jeroen de Jong (TUD) *Heuristics in Dynamic Sceduling: a practical framework with a case study in elevator dispatching*

2013

1 Viorel Milea (EUR) *News Analytics for Financial Decision Support*
2 Erietta Liarou (CWI) *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing*
3 Szymon Klarman (VUA) *Reasoning with Contexts in Description Logics*
4 Chetan Yadati (TUD) *Coordinating autonomous planning and scheduling*
5 Dulce Pumareja (UT) *Groupware Requirements Evolutions Patterns*
6 Romulo Goncalves (CWI) *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience*
7 Giel van Lankveld (UvT) *Quantifying Individual Player Differences*
8 Robbert-Jan Merk (VUA) *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators*
9 Fabio Gori (RUN) *Metagenomic Data Analysis: Computational Methods and Applications*
10 Jeewanie Jayasinghe Arachchige (UvT) *A Unified Modeling Framework for Service Design*
11 Evangelos Pournaras (TUD) *Multi-level Reconfigurable Self-organization in Overlay Services*
12 Marian Razavian (VUA) *Knowledge-driven Migration to Services*
13 Mohammad Safiri (UT) *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly*
14 Jafar Tanha (UvA) *Ensemble Approaches to Semi-Supervised Learning Learning*
15 Daniel Hennes (UM) *Multiagent Learning: Dynamic Games and Applications*
16 Eric Kok (UU) *Exploring the practical benefits of argumentation in multi-agent deliberation*
17 Koen Kok (VUA) *The PowerMatcher: Smart Coordination for the Smart Electricity Grid*
18 Jeroen Janssens (UvT) *Outlier Selection and One-Class Classification*
19 Renze Steenhuizen (TUD) *Coordinated Multi-Agent Planning and Scheduling*
20 Katja Hofmann (UvA) *Fast and Reliable Online Learning to Rank for Information Retrieval*
21 Sander Wubben (UvT) *Text-to-text generation by monolingual machine translation*
22 Tom Claassen (RUN) *Causal Discovery and Logic*
23 Patricio de Alencar Silva (UvT) *Value Activity Monitoring*
24 Haitham Bou Ammar (UM) *Automated Transfer in Reinforcement Learning*
25 Agnieszka Anna Latoszek-Berendsen (UM) *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System*
26 Alireza Zarghami (UT) *Architectural Support for Dynamic Homecare Service Provisioning*
27 Mohammad Huq (UT) *Inference-based Framework Managing Data Provenance*
28 Frans van der Sluis (UT) *When Complexity becomes Interesting: An Inquiry into the Information eXperience*
29 Iwan de Kok (UT) *Listening Heads*
30 Joyce Nakatumba (TUe) *Resource-Aware Business Process Management: Analysis and Support*
31 Dinh Khoa Nguyen (UvT) *Blueprint Model and Language for Engineering Cloud Applications*
32 Kamakshi Rajagopal (OUN) *Networking For Learning: The role of Networking in a Lifelong Learner's Professional Development*
33 Qi Gao (TUD) *User Modeling and Personalization in the Microblogging Sphere*
34 Kien Tjin-Kam-Jet (UT) *Distributed Deep Web Search*
35 Abdallah El Ali (UvA) *Minimal Mobile Human Computer Interaction*
36 Than Lam Hoang (TUe) *Pattern Mining in Data Streams*
37 Dirk Börner (OUN) *Ambient Learning Displays*
38 Eelco den Heijer (VUA) *Autonomous Evolutionary Art*
39 Joop de Jong (TUD) *A Method for Enterprise Ontology based Design of Enterprise Information Systems*
40 Pim Nijssen (UM) *Monte-Carlo Tree Search for Multi-Player Games*
41 Jochem Liem (UvA) *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning*

42 Léon Planken (TUD) *Algorithms for Simple Temporal Reasoning*
43 Marc Bron (UvA) *Exploration and Contextualization through Interaction and Concepts*

2014

1 Nicola Barile (UU) *Studies in Learning Monotone Models from Data*
2 Fiona Tuliyano (RUN) *Combining System Dynamics with a Domain Modeling Method*
3 Sergio Raul Duarte Torres (UT) *Information Retrieval for Children: Search Behavior and Solutions*
4 Hanna Jochmann-Mannak (UT) *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation*
5 Jurriaan van Reijsen (UU) *Knowledge Perspectives on Advancing Dynamic Capability*
6 Damian Tamburri (VUA) *Supporting Networked Software Development*
7 Arya Adriansyah (TUe) *Aligning Observed and Modeled Behavior*
8 Samur Araujo (TUD) *Data Integration over Distributed and Heterogeneous Data Endpoints*
9 Philip Jackson (UvT) *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language*
10 Ivan Salvador Razo Zapata (VUA) *Service Value Networks*
11 Janneke van der Zwaan (TUD) *An Empathic Virtual Buddy for Social Support*
12 Willem van Willigen (VUA) *Look Ma, No Hands: Aspects of Autonomous Vehicle Control*
13 Arlette van Wissen (VUA) *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains*
14 Yangyang Shi (TUD) *Language Models With Meta-information*
15 Natalya Mogles (VUA) *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare*
16 Krystyna Milian (VUA) *Supporting trial recruitment and design by automatically interpreting eligibility criteria*
17 Kathrin Dentler (VUA) *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability*
18 Mattijs Ghijsen (UvA) *Methods and Models for the Design and Study of Dynamic Agent Organizations*
19 Vinicius Ramos (TUe) *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support*
20 Mena Habib (UT) *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link*
21 Kassidy Clark (TUD) *Negotiation and Monitoring in Open Environments*
22 Marieke Peeters (UU) *Personalized Educational Games: Developing agent-supported scenario-based training*
23 Eleftherios Sidirourgos (UvA/CWI) *Space Efficient Indexes for the Big Data Era*
24 Davide Ceolin (VUA) *Trusting Semi-structured Web Data*
25 Martijn Lappenschaar (RUN) *New network models for the analysis of disease interaction*
26 Tim Baarslag (TUD) *What to Bid and When to Stop*
27 Rui Jorge Almeida (EUR) *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty*
28 Anna Chmielowiec (VUA) *Decentralized k-Clique Matching*
29 Jaap Kabbedijk (UU) *Variability in Multi-Tenant Enterprise Software*
30 Peter de Cock (UvT) *Anticipating Criminal Behaviour*
31 Leo van Moergestel (UU) *Agent Technology in Agile Multiparallel Manufacturing and Product Support*
32 Naser Ayat (UvA) *On Entity Resolution in Probabilistic Data*
33 Tesfa Tegegne (RUN) *Service Discovery in eHealth*
34 Christina Manteli (VUA) *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems*
35 Joost van Ooijen (UU) *Cognitive Agents in Virtual Worlds: A Middleware Design Approach*
36 Joos Buijs (TUe) *Flexible Evolutionary Algorithms for Mining Structured Process Models*
37 Maral Dadvar (UT) *Experts and Machines United Against Cyberbullying*
38 Danny Plass-Oude Bos (UT) *Making brain-computer interfaces better: improving usability through post-processing*
39 Jasmina Maric (UvT) *Web Communities, Immigration, and Social Capital*
40 Walter Omona (RUN) *A Framework for Knowledge Management Using ICT in Higher Education*
41 Frederic Hogenboom (EUR) *Automated Detection of Financial Events in News Text*
42 Carsten Eijckhof (CWI/TUD) *Contextual Multidimensional Relevance Models*
43 Kevin Vlaanderen (UU) *Supporting Process Improvement using Method Increments*
44 Paulien Meesters (UvT) *Intelligent Blauw: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden*

45 Birgit Schmitz (OUN) *Mobile Games for Learning: A Pattern-Based Approach*
46 Ke Tao (TUD) *Social Web Data Analytics: Relevance, Redundancy, Diversity*
47 Shangsong Liang (UvA) *Fusion and Diversification in Information Retrieval*

2015

1 Niels Netten (UvA) *Machine Learning for Relevance of Information in Crisis Response*
2 Faiza Bukhsh (UvT) *Smart auditing: Innovative Compliance Checking in Customs Controls*
3 Twan van Laarhoven (RUN) *Machine learning for network data*
4 Howard Spoelstra (OUN) *Collaborations in Open Learning Environments*
5 Christoph Bösch (UT) *Cryptographically Enforced Search Pattern Hiding*
6 Farideh Heidari (TUD) *Business Process Quality Computation: Computing Non-Functional Requirements to Improve Business Processes*
7 Maria-Hendrike Peetz (UvA) *Time-Aware Online Reputation Analysis*
8 Jie Jiang (TUD) *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions*
9 Randy Klaassen (UT) *HCI Perspectives on Behavior Change Support Systems*
10 Henry Hermans (OUN) *OpenU: design of an integrated system to support lifelong learning*
11 Yongming Luo (TUe) *Designing algorithms for big graph datasets: A study of computing bisimulation and joins*
12 Julie M. Birkholz (VUA) *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks*
13 Giuseppe Procaccianti (VUA) *Energy-Efficient Software*
14 Bart van Straalen (UT) *A cognitive approach to modeling bad news conversations*
15 Klaas Andries de Graaf (VUA) *Ontology-based Software Architecture Documentation*
16 Changyun Wei (UT) *Cognitive Coordination for Cooperative Multi-Robot Teamwork*
17 André van Cleeff (UT) *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs*
18 Holger Pirk (CWI) *Waste Not, Want Not!: Managing Relational Data in Asymmetric Memories*
19 Bernardo Tabuenca (OUN) *Ubiquitous Technology for Lifelong Learners*
20 Loïs Vanhée (UU) *Using Culture and Values to Support Flexible Coordination*
21 Sibren Fetter (OUN) *Using Peer-Support to Expand and Stabilize Online Learning*
22 Zhemin Zhu (UT) *Co-occurrence Rate Networks*
23 Luit Gazendam (VUA) *Cataloguer Support in Cultural Heritage*
24 Richard Berendsen (UvA) *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation*
25 Steven Woudenberg (UU) *Bayesian Tools for Early Disease Detection*
26 Alexander Hogenboom (EUR) *Sentiment Analysis of Text Guided by Semantics and Structure*
27 Sándor Héman (CWI) *Updating compressed column-stores*
28 Janet Bagorogoza (TiU) *Knowledge Management and High Performance: The Uganda Financial Institutions Model for HPO*
29 Hendrik Baier (UM) *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains*
30 Kiavash Bahreini (OUN) *Real-time Multimodal Emotion Recognition in E-Learning*
31 Yakup Koç (TUD) *On Robustness of Power Grids*
32 Jerome Gard (UL) *Corporate Venture Management in SMEs*
33 Frederik Schadd (UM) *Ontology Mapping with Auxiliary Resources*
34 Victor de Graaff (UT) *Geosocial Recommender Systems*
35 Junchao Xu (TUD) *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction*

2016

1 Syed Saiden Abbas (RUN) *Recognition of Shapes by Humans and Machines*
2 Michiel Christiaan Meulendijk (UU) *Optimizing medication reviews through decision support: prescribing a better pill to swallow*
3 Maya Sappelli (RUN) *Knowledge Work in Context: User Centered Knowledge Worker Support*
4 Laurens Rietveld (VUA) *Publishing and Consuming Linked Data*
5 Evgeny Sherkhonov (UvA) *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers*
6 Michel Wilson (TUD) *Robust scheduling in an uncertain environment*
7 Jeroen de Man (VUA) *Measuring and modeling negative emotions for virtual training*
8 Matje van de Camp (TiU) *A Link to the Past: Constructing Historical Social Networks from Unstructured Data*
9 Archana Nottamkandath (VUA) *Trusting Crowdsourced Information on Cultural Artefacts*
10 George Karafotias (VUA) *Parameter Control for Evolutionary Algorithms*
11 Anne Schuth (UvA) *Search Engines that Learn from Their Users*

More than half the world's population uses web search engines, resulting in over half a billion queries every single day. For many people, web search engines such as Baidu, Bing, Google, and Yandex are among the first resources they go to when a question arises. Moreover, for many search engines have become the most trusted route to information, more so even than traditional media such as newspapers, news websites or news channels on television. What web search engines present people with greatly influences what they believe to be true and consequently it influences their thoughts, opinions, decisions, and the actions they take. With this in mind two things are important, from an information retrieval research perspective. First, it is important to understand how well search engines (rankers) perform and secondly this knowledge should be used to improve them. This thesis is about these two topics: *evaluation of search engines* and *learning search engines*.

In the first part of this thesis we investigate how user interactions with search engines can be used to evaluate search engines. In particular, we introduce a new online evaluation paradigm called *multileaving* that extends upon interleaving. With multileaving, many rankers can be compared at once by combining document lists from these rankers into a single result list and attributing user interactions with this list to the rankers. Then we investigate the relation between A/B testing and interleaved comparison methods. Both studies lead to much higher sensitivity of the evaluation methods, meaning that fewer user interactions are required to arrive at reliable conclusions. This has the important implication that fewer users need to be exposed to the results from possibly inferior search engines.

In the second part of this thesis we turn to online learning to rank. We learn from the evaluation methods introduced and extended upon in the first part. We learn the parameters of base rankers based on user interactions. Then we use the multileaving methods as feedback in our learning method, leading to much faster convergence than existing methods. Again, the important implication is that fewer users need to be exposed to possibly inferior search engines as they adapt more quickly to changes in user preferences.

The last part of this thesis is of a different nature than the earlier two parts. As opposed to the earlier chapters, we no longer study algorithms. Progress in information retrieval research has always been driven by a combination of algorithms, shared resources, and evaluation. In the last part we focus on the latter two. We introduce a new shared resource and a new evaluation paradigm. Firstly, we propose Lerot. Lerot is an online evaluation framework that allows us to simulate users interacting with a search engine. Our implementation has been released as open source software and is currently being used by researchers around the world. Secondly we introduce OpenSearch, a new evaluation paradigm involving real users of real search engines. We describe an implementation of this paradigm that has already been widely adopted by the research community through challenges at CLEF and TREC.

ISBN 978-94-6182-674-9