Multileave Gradient Descent for Fast Online Learning to Rank

Anne Schuth[†]
a.g.schuth@uva.nl

Harrie Oosterhuis[†]
harrie.oosterhuis@student.uva.nl

Shimon Whiteson[‡]
shimon.whiteson@cs.ox.ac.uk

Maarten de Rijke[†]
derijke@uva.nl

[†] University of Amsterdam, Amsterdam, The Netherlands
[‡] University of Oxford, Oxford, United Kingdom

ABSTRACT

Modern search systems are based on dozens or even hundreds of ranking features. The *dueling bandit gradient descent* (DBGD) algorithm has been shown to effectively learn combinations of these features solely from user interactions. 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. A limiting factor is that it can compare only to a single exploratory ranker.

We propose an online learning to rank algorithm called *multileave gradient descent* (MGD) that extends DBGD to learn from so-called *multileaved comparison methods* that can compare a set of rankings instead of merely a pair. 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. Hence, 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.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.3 Information Search and Retrieval

Keywords

Information retrieval; Learning to rank; Interleaved comparisons; Multileaved comparisons

1. INTRODUCTION

Modern search engines base their rankings on combinations of dozens or even hundreds of features. *Learning to rank*, i.e., finding an optimal combination of features, is an active area of research.

Traditionally, learning was done *offline* by optimizing for performance on a training set consisting of queries and relevance assessments produced by human assessors. However, such datasets are time consuming and expensive to produce. Moreover, these assessments are not always in line with actual user preferences [23]. And since data of users interacting with a search engine are often readily available, research is focussing more on learning *online* instead [10, 13, 22, 31]. 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 [22] or documents [13]. In particular, when clicks are interpreted using interleaved comparison methods, they can reliably infer preferences between a pair of rankers [4, 13, 14].

Dueling bandit gradient descent (DBGD) [31] 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 paper, we answer the following research questions.

RQ1 Can MGD learn faster from user feedback (i.e., using fewer clicks) than DBGD does?

RQ2 Does MGD find a better local optimum than DBGD?

RQ3 Which update approach, MGD-W or MGD-M, maximizes learning speed? Which finds a better local optimum?

Our contributions are:

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

WSDM 2016, February 22 - 25, 2016, San Francisco, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3716-8/16/02 ...$15.00.
DOI: http://dx.doi.org/10.1145/2835776.2835804.

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

In Section 2 we discuss related work; Section 3 includes a detailed description of interleaving and multileaving methods and of DBGD, which are three elements that play a central role in this paper. In Section 4 we introduce multileave gradient descent. In Section 5 we detail our experimental setup. Section 6 provides both results and their analysis. We conclude in Section 7.

2. RELATED WORK

Related work for this paper comes in two areas. First, there are learning to rank methods in Section 2.1. Particularly important for this paper is the *dueling bandit gradient descent* (DBGD) method, which is therefore described in more detail in Section 3.2. Second, there are evaluation methods in Section 2.2, of which the interleaving and multileaving methods play a crucial role in this paper. Therefore, these are described in more detail in Section 3.1.

2.1 Online learning to rank

Learning to rank comes in online and offline variants. Offline learning to rank methods [19] can be supervised and semi-supervised [29]. Offline learning methods suffer from drawback of requiring labeled datasets, which are expensive to produce and the models learned do not necessarily align with user satisfaction [23].

Our focus is on online learning to rank methods that learn from online evaluation methods, based on users' interactions with IR systems [10, 31], without requiring annotated datasets. One can formulate the online learning to rank for IR problem as a *reinforcement learning* (RL) problem. The problem can be modeled as a contextual bandit problem that assumes that consecutive queries are independent of each other [18, 27]. A crucial difference between typical RL problems and the application to IR is that in the IR scenario the reward cannot directly be observed. Instead, user interactions with an IR system can be interpreted as a biased and noisy preference for either rankings [22] or documents [13].

Many methods for online learning to rank have been proposed. Hofmann et al. [10] explore learning from pairwise document preference 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 2.2 and 3.1. One influential learning to rank method is *dueling bandit gradient descent* (DBGD) [31], which is extended upon in this paper and therefore explained in more detail in Section 3.2. 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. [11] such that it would reuse historical interaction data and not just live user interactions. Alternative methods are based on the k-armed bandit formulation by [32] and assume a finite space of possible rankers, the best of which needs to be found.

2.2 Evaluation for information retrieval

Evaluation has always been an integral part of information retrieval (IR) research. IR evaluation comes in three flavors: offline, in the form of users studies, and online.

Firstly, there is Cranfield-style evaluation, as introduced by Cleverdon et al. [5], a form of evaluation we refer to as *offline*. This type of evaluation is based on test collections that consist of queries, documents, and relevance assessments of these documents with respect to these queries. Test collections can be used to evaluate systems by computing offline evaluation metrics such as MAP, NDCG, ERR [23]. It forms the basis of most TREC-style evaluation benchmarks [30]. A major advantage of offline evaluation is the repeatability of experiments and the ease of evaluation of many and new rankers. However, relevance assessments are typically produced by trained human judges. This is not only time consuming and expensive, it is also not entirely clear how useful such relevance assessments really are as agreement with actual users' preferences is not necessarily high [23]. Several approaches to address the high costs of producing relevance assessments have been described [3, 24]; even further steps are taken in [1, 2], with proposals for automatically created test collections.

A second form of evaluation is through *user studies*. In this type of studies, users and their interaction behavior are studied in a lab setting [16]. This makes such evaluations even more expensive than offline evaluation, specifically because experiments are not repeatable and do not scale up.

Thirdly, *online evaluation* exploits the interactions of real users with a live search engine [17]. Online evaluation comes in several dominant forms. In *A/B testing* two rankers are shown to two distinct groups of users. For each group, click metrics are computed and the outcomes are compared to determine which ranker is better. *Interleaved comparison methods* [4, 13, 14] have been shown to be highly sensitive, much more so than A/B metrics [4, 22], meaning that less interaction data are required to detect differences between rankers. Interleaved comparison methods take as input two rankings and produce an interleaved result list. The interleaved list is shown to users and interactions with interleaved lists are interpreted so as to determine which of the two input rankers wins the comparison. Several interleaving methods exist. *Balanced interleave* (BI) [15] starts by randomly selecting a ranker to start with. Then, alternating, documents that have not been picked are picked from the two input rankings. BI can result in a preference for a ranker irrespective of where a user clicks. This bias is corrected in *team draft interleave* (TDI) [22], which we discuss in Section 3.1. More recent methods include *document constraints* (DC) [7] and *probabilistic interleave* (PI) [9]. A major advantage of PI is that it can also infer preferences between rankers that were not originally interleaved. This allows one to learn from historical interaction data [11]. However, PI risks showing users poor rankings. This shortcoming was addressed in *optimized interleave* (OI) [21] by restricting the allowed interleavings to only those that are a union of prefixes of the two input rankings.

Recently, interleaving methods have been extended to *multileaving* methods [26] that allow for comparisons of more than two rankers at once. In particular, Schuth et al. [26] extend TDI to *team draft multileave* (TDM) and OI to *optimized multileave* (OM). TDM forms part of our motivation and is discussed in detail in Section 3.1.

Our work 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.

3. BACKGROUND

We describe in more detail two types of related work on which our work depends: interleaved and multileaved comparison methods (Section 3.1) and dueling bandit gradient descent (DBGD; Section 3.2).

3.1 Interleaving and multileaving

Interleaved comparison methods are highly sensitive online evaluation methods for information retrieval systems [4, 13, 14]. These methods combine the documents from a pair of rankings into a single document list and present this combined list to the user. The user's interaction with this result list is then used to infer a preference for one of the input rankings. It was shown that, typically, this way of comparing two rankers requires one to two orders of magnitude less data when compared to A/B testing [4]. Multileaved comparison methods [26] allow for more than two rankers to be compared using a similar methodology, reducing the amount of interaction data required even further.

Several interleaving methods have been proposed over the years, see Section 2. We build upon *team draft interleave* (TDI) [22] and an extended version called *team draft multileave* (TDM) [26].

TDI works as follows. When a user enters a query into a search engine, the query is passed to the two rankers to be compared. The rankings these rankers produce are then interleaved in a process analogous to picking teams for a friendly team-sport match. The documents are players and there are two teams representing the two rankers. Each team has a preference ordering over players, corresponding to the ranking over documents. In each round, the teams take turns picking their most preferred, still available player. Which teams chooses first is determined randomly. Selected players are appended to the interleaved list, and the team they are assigned to is recorded.[1] This interleaved list is shown to the user, who may click on some documents in this list. The clicked documents give credit to the team to which they belong. The team, or ranker, that receives the most credit wins the comparison. 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.[2] 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 teams win 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 [26]. Note that TDM reduces to TDI for $n = 1$.

3.2 Dueling bandit gradient descent

Dueling bandit gradient descent (DBGD) [31] 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 3.1). DBGD, shown in Algorithm 1, assumes that rankers can be represented by weight vectors, starting of 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 6, 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 8. Then, on line 9 this interleaving is shown to the user who issued the query and clicks are observed. The interactions of this user with the interleaving are interpreted by the interleaving method on line 10 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 adaptive system.

Algorithm 1 Dueling Bandit Gradient Descent (DBGD).

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

Algorithm 2 Multileave Gradient Descent (MGD).

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

4. MULTILEAVE GRADIENT DESCENT

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

4.1 Extending DBGD with multileaving

MGD is shown in Algorithm 2. As in DBGD, 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 5 through 8, n exploratory candidate rankers are generated along with their corresponding rankings. Unlike DBGD, which is restricted to a single candidate ranker during comparison, MGD can handle multiple candidate rankers because it uses multileaving, which on line 9 creates a single document list out of the n rankings. After observing user clicks on this ranker, on line 10, 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 15, using one of two update methods described in Section 4.2. 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 candi-

[1] For documents belonging to a prefix that the two input rankings have in common, no teams are assigned to increase sensitivity [4].

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

Algorithm 3 MGD update function: winner takes all (MGD-W).

```
Input: $\mathbf{w}, \alpha, \mathbf{b}, \mathbf{u}$        // in $\mathbf{b}$ are only winners
$\mathbf{b}^j \leftarrow pick\_random\_uniformly(\mathbf{b})$
return $\mathbf{w} + \alpha \mathbf{u}^j$
```

Algorithm 4 MGD update function: mean winner (MGD-M).

```
Input: $\mathbf{w}, \alpha, \mathbf{b}, \mathbf{u}$        // in $\mathbf{b}$ are only winners
return $\mathbf{w} + \alpha \frac{1}{|\mathbf{u}|} \sum_{\mathbf{b}^j \in \mathbf{b}} \mathbf{u}^j$
```

date 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.

4.2 Multileave approaches to gradient descent

DBGD generates each candidate ranker by sampling a unit sphere uniformly and adding the resulting unit vector to the *current best ranker*. For the MGD approaches in this paper 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 MGB. But since the dimensionality of the feature space is expected to be much greater than the number of candidates, it is most unlikely 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 Algorithm 3 and 4 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 DGBD 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) [11]. However, this method reuses historical interaction data which it requires to be generated stochastically using potentially unsafe rankings [21]. MGD is a new way of comparing candidates that does not have this drawback.

5. EXPERIMENTS

In this section, we detail our experiments which are designed to answer the research questions posed in Section 1.[3] We are interested in whether and how our newly introduced algorithm MGD (RQ1) learns faster than DBGD; (RQ2) converges to a better optimum compared to DBGD; and (RQ3) how the two variants MGD-W and MGD-M compare to each other.

Table 1: Instantiations of CCM [6] as used in our experiments.

		$P(click = 1\|R)$			$P(stop = 1\|R)$		
Relevance grade 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

All our experiments assume a stream of independent queries coming from users interacting with the system we are training. Users are presented with a results 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 datasets (Section 5.1) and the clicks from a click model (Section 5.2). In Section 5.3 we describe the experiments we run. Our evaluation measures are described in Section 5.4.

5.1 Datasets

We use nine learning to rank datasets that are distributed as LETOR 3.0 and 4.0 [20]. The datasets each consist of (1) queries, only represented by their identifier, (2) manual relevance assessments for documents with respect to these queries, and (3) documents represented as feature vectors, again, with respect to each query. Feature vectors consist of 45, 46, or 64 features encoding ranking models such as TF.IDF, BM25, Language Modeling, PageRank, and HITS on different parts of the documents. All the datasets come split into 5 folds, which we use for 5-fold cross validation. Each dataset encodes a search task. Most tasks come from TREC Web Tracks between 2003 and 2008. The only exception is the *OHSUMED* dataset with 106 queries which comes from a query log of the search engine on the MedLine abstract database. *HP2003*, *HP-2004*, *NP2003*, and *NP2004* all implement navigational tasks which are homepage finding and named-page finding, respectively; both *TD2003* and *TD2004* implement topic distillation tasks which is an informational task. *HP2003, HP2004, NP2003, NP2004, TD2003* and *TD2004* contain documents from the .GOV collection which was crawled from the .gov domain; each contains between 50 and 150 queries and about 1,000 judged documents per query. The last two datasets, *MQ2007* and *MQ2008*, are more recent and use the .GOV2 collection; they contain more queries, 1700 and 800 respectively, but far fewer assessments per query. *OHSUMED*, *MQ2007*, and *MQ2008* have graded relevance judgments from 0, not relevant, to 2, highly relevant. The other datasets have binary relevance labels with grade 0 for not relevant and grade 2 for relevant.

5.2 Simulating clicks

We use the setup described by Hofmann [8] to simulate user interactions. In their setup, clicks are produced based on a *cascade click model* (CCM) [6]. This model explains behavior of web search users as follows. Users are assumed to start examining results lists from the first document in the list and then work their way down the list. For each document they encounter, users decide whether it warrants a click, which is modeled as $P(click = 1|R)$, a click probability conditioned on the human generated relevance label R. After clicking, the user's information need may either be satisfied or they continue down the rank list. In CCM, this is modeled with $P(stop = 1|R)$, the probability of stopping conditioned on the relevance label R.

In Table 1, we first list the three instantiations of CCM that we use in all our experiments. These three instantiations model a *perfect*

[3] All our experimental code is open source and available at https://bitbucket.org/ilps/lerot [25].

user that clicks on all highly relevant and only on relevant documents. Then, a *navigational* instantiation encoding a navigational task where a user is mostly looking for a single highly relevant document. And lastly, an *informational* instantiation that models a user who would typically click on several documents, less dependent on their relevance. These three models have increasing levels of noise, as less and less is determined by the relevance labels of documents. Then, we use the *almost random* instantiation only for some of our experiments to test what happens when feedback becomes extremely noisy. Note that the datasets that only have binary relevance use instantiations for the lowest and highest relevance labels (0 and 2) in Table 1.

5.3 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 were 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 with $n = 1$ is included to verify wether this setting has no significant difference with the baseline. The bulk of our experiments consist of 1,000 iterations (i.e., simulated user impressions) each and is run 25 times on each fold resulting in 125 runs over each dataset. One experiment is 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 [31]. 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 6.4.2.

5.4 Evaluation

To assess performance, NDCG [12] is computed on held-out data. We use the top $\kappa = 10$ for simulating clicks and computing the NDCG:

$$NDCG = \sum_{i=1}^{\kappa} \frac{2^{rel(\mathbf{r}[i])-1}}{log_2(i+1)} iNDCG^{-1}.$$

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. 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. For online performance, a discount factor of $\gamma = 0.995$ is used [8, 28]. This factor ensures that impressions beyond a horizon of 1,000 impressions have an impact of less than 1%. To verify whether differences are statistically significantly different, a two tailed Student's t-test is used.

6. RESULTS AND ANALYSIS

In this section we present the results of our experiments and answer the research questions posed in Section 1. Furthermore, in Section 6.4, we investigate the effect of n, the number of candidates, and α, the learning rate.

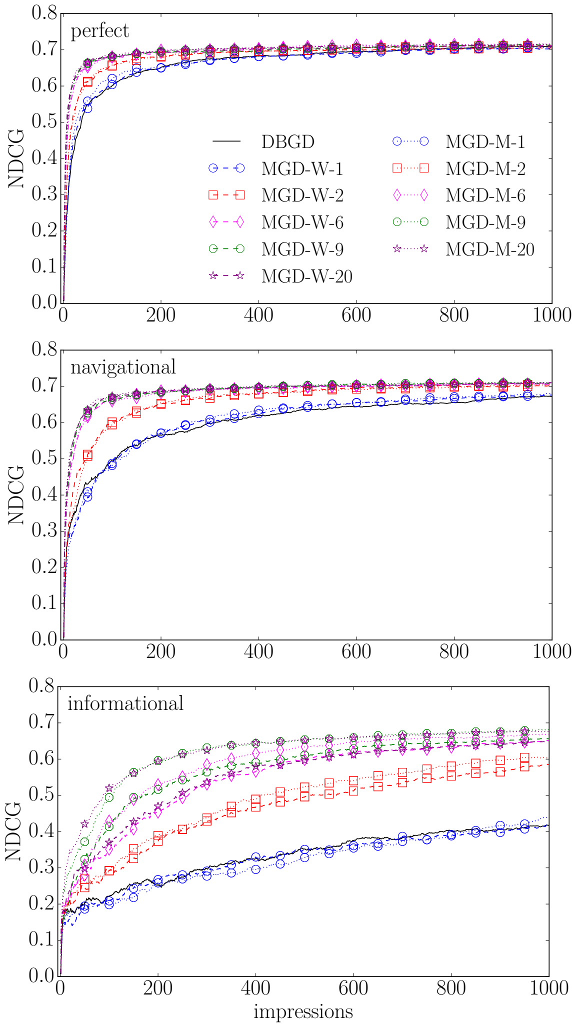

Figure 1: Offline performance (NDCG) on MGD-W and MGD-M with varying number of candidates compared to DBGD on *NP2003* dataset for the *perfect*, *navigational* and *informational* click model.

6.1 Learning speed

We start by answering RQ1: whether MGD learns faster than DBGD. The plots in Figure 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 datasets. 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 2 we see the same effect for all datasets: generally, under *perfect* feedback converged performance does not change much; however, if the feedback is noisier, then the more candidates are added, the

Table 2: Offline score (NDCG) after 1,000 query impressions of each of the algorithms for the 3 instantiations of the CCM (see Table 1). Bold values indicate maximum performance per dataset and click model. Statistically significant improvements (losses) over the DBGD baseline are indicated by $^{\vartriangle}$ ($p < 0.05$) and $^{\blacktriangle}$ ($p < 0.01$) ($^{\triangledown}$ and $^{\blacktriangledown}$). We show the standard deviation between brackets.

		HP2003	NP2003	TD2003	HP2004	NP2004	TD2004	MQ2007	MQ2008	OHSUMED
perfect	**DBGD**	0.766 (0.06)	0.710 (0.05)	0.299 (0.09)	0.730 (0.07)	0.715 (0.08)	0.303 (0.03)	0.381 (0.03)	0.476 (0.04)	0.443 (0.05)
	MGD-W-2	0.771 (0.06)	0.705 (0.05)	0.314 (0.08)	0.731 (0.06)	0.726 (0.07)	0.306 (0.03)	0.392 (0.02) $^{\blacktriangle}$	0.480 (0.04)	0.445 (0.05)
	MGD-W-4	0.771 (0.06)	0.712 (0.05)	0.318 (0.08)	0.742 (0.06)	**0.732** (0.07)	0.310 (0.04)	0.396 (0.02) $^{\blacktriangle}$	0.481 (0.04)	**0.447** (0.05)
	MGD-W-6	0.778 (0.06)	0.712 (0.05)	0.314 (0.08)	0.745 (0.06)	0.725 (0.07)	0.308 (0.04)	0.398 (0.02) $^{\blacktriangle}$	0.479 (0.04)	0.444 (0.05)
	MGD-W-9	0.774 (0.06)	0.713 (0.05)	0.314 (0.07)	0.744 (0.06)	0.725 (0.07)	0.311 (0.04)	0.400 (0.02) $^{\blacktriangle}$	0.481 (0.04)	0.430 (0.04) $^{\triangledown}$
	MGD-W-20	0.776 (0.06)	0.710 (0.05)	0.314 (0.07)	**0.749** (0.06) $^{\vartriangle}$	0.726 (0.07)	0.308 (0.04)	0.396 (0.02) $^{\blacktriangle}$	0.480 (0.04)	0.438 (0.05)
	MGD-M-2	0.771 (0.07)	0.712 (0.05)	0.312 (0.08)	0.743 (0.06)	0.730 (0.08)	0.311 (0.04)	0.392 (0.02) $^{\blacktriangle}$	0.480 (0.04)	0.443 (0.05)
	MGD-M-4	0.777 (0.07)	0.711 (0.05)	0.317 (0.07)	0.742 (0.07)	0.729 (0.07)	0.315 (0.04) $^{\blacktriangle}$	0.400 (0.02) $^{\blacktriangle}$	0.482 (0.04)	**0.447** (0.05)
	MGD-M-6	0.779 (0.06)	**0.716** (0.04)	0.320 (0.07)	0.747 (0.06) $^{\vartriangle}$	0.725 (0.07)	0.312 (0.04) $^{\vartriangle}$	0.402 (0.02) $^{\blacktriangle}$	0.481 (0.04)	**0.447** (0.05)
	MGD-M-9	**0.780** (0.06)	0.714 (0.05)	**0.322** (0.07) $^{\vartriangle}$	0.747 (0.06) $^{\vartriangle}$	0.726 (0.07)	0.311 (0.04)	0.406 (0.02) $^{\blacktriangle}$	**0.484** (0.04)	0.437 (0.04)
	MGD-M-20	0.777 (0.06)	0.714 (0.05)	0.321 (0.07) $^{\vartriangle}$	0.747 (0.06) $^{\vartriangle}$	0.724 (0.08)	**0.316** (0.04) $^{\blacktriangle}$	**0.408** (0.02) $^{\blacktriangle}$	**0.484** (0.04)	0.446 (0.04)
navigational	**DBGD**	0.725 (0.07)	0.672 (0.06)	0.281 (0.09)	0.676 (0.08)	0.693 (0.08)	0.281 (0.03)	0.370 (0.03)	0.460 (0.04)	0.433 (0.06)
	MGD-W-2	0.766 (0.06) $^{\blacktriangle}$	0.702 (0.05) $^{\blacktriangle}$	0.306 (0.09) $^{\vartriangle}$	0.732 (0.06) $^{\blacktriangle}$	0.715 (0.08) $^{\vartriangle}$	0.303 (0.03) $^{\blacktriangle}$	0.372 (0.03)	0.466 (0.04)	**0.438** (0.05)
	MGD-W-4	0.769 (0.06) $^{\blacktriangle}$	0.708 (0.05) $^{\blacktriangle}$	0.314 (0.08) $^{\blacktriangle}$	0.735 (0.06) $^{\blacktriangle}$	0.720 (0.08) $^{\blacktriangle}$	**0.307** (0.04) $^{\blacktriangle}$	0.380 (0.02) $^{\blacktriangle}$	0.469 (0.04)	0.437 (0.04)
	MGD-W-6	**0.772** (0.06) $^{\blacktriangle}$	0.705 (0.05) $^{\blacktriangle}$	0.312 (0.08) $^{\blacktriangle}$	0.738 (0.06) $^{\blacktriangle}$	0.721 (0.08) $^{\blacktriangle}$	0.304 (0.04) $^{\blacktriangle}$	0.382 (0.02) $^{\blacktriangle}$	0.468 (0.04)	0.431 (0.05)
	MGD-W-9	0.771 (0.06) $^{\blacktriangle}$	0.708 (0.05) $^{\blacktriangle}$	0.304 (0.07) $^{\vartriangle}$	0.738 (0.06) $^{\blacktriangle}$	**0.725** (0.08) $^{\blacktriangle}$	0.304 (0.04) $^{\blacktriangle}$	0.388 (0.02) $^{\blacktriangle}$	0.470 (0.04) $^{\vartriangle}$	0.431 (0.05)
	MGD-W-20	0.771 (0.06) $^{\blacktriangle}$	0.710 (0.05) $^{\blacktriangle}$	0.314 (0.07) $^{\blacktriangle}$	0.738 (0.06) $^{\blacktriangle}$	0.721 (0.07) $^{\blacktriangle}$	0.304 (0.04) $^{\blacktriangle}$	0.386 (0.03) $^{\blacktriangle}$	0.470 (0.04)	0.432 (0.05)
	MGD-M-2	0.766 (0.06) $^{\blacktriangle}$	0.703 (0.06) $^{\blacktriangle}$	0.302 (0.08)	0.726 (0.06) $^{\blacktriangle}$	0.717 (0.08) $^{\vartriangle}$	0.301 (0.04) $^{\blacktriangle}$	0.376 (0.03)	0.467 (0.04)	0.435 (0.05)
	MGD-M-4	0.768 (0.06) $^{\blacktriangle}$	0.705 (0.05) $^{\blacktriangle}$	0.312 (0.08) $^{\blacktriangle}$	0.738 (0.06) $^{\blacktriangle}$	0.721 (0.07) $^{\blacktriangle}$	0.305 (0.04) $^{\blacktriangle}$	0.385 (0.02) $^{\blacktriangle}$	0.468 (0.04)	0.435 (0.04)
	MGD-M-6	**0.772** (0.06) $^{\blacktriangle}$	0.707 (0.05) $^{\blacktriangle}$	0.309 (0.07) $^{\blacktriangle}$	0.736 (0.07) $^{\blacktriangle}$	0.723 (0.08) $^{\blacktriangle}$	0.305 (0.04) $^{\blacktriangle}$	0.387 (0.03) $^{\blacktriangle}$	**0.473** (0.04) $^{\blacktriangle}$	0.437 (0.05)
	MGD-M-9	0.771 (0.06) $^{\blacktriangle}$	0.710 (0.05) $^{\blacktriangle}$	0.317 (0.08) $^{\blacktriangle}$	**0.741** (0.06) $^{\blacktriangle}$	0.724 (0.07) $^{\blacktriangle}$	0.302 (0.04) $^{\blacktriangle}$	**0.391** (0.02) $^{\blacktriangle}$	0.472 (0.04) $^{\vartriangle}$	0.436 (0.04)
	MGD-M-20	0.769 (0.06) $^{\blacktriangle}$	**0.711** (0.04) $^{\blacktriangle}$	**0.318** (0.08) $^{\blacktriangle}$	**0.741** (0.06) $^{\blacktriangle}$	0.720 (0.07) $^{\blacktriangle}$	0.306 (0.04) $^{\blacktriangle}$	0.390 (0.02) $^{\blacktriangle}$	0.472 (0.04) $^{\vartriangle}$	0.437 (0.05)
informational	**DBGD**	0.460 (0.22)	0.418 (0.19)	0.167 (0.10)	0.401 (0.21)	0.489 (0.19)	0.197 (0.08)	0.323 (0.05)	0.419 (0.05)	0.407 (0.05)
	MGD-W-2	0.677 (0.11) $^{\blacktriangle}$	0.585 (0.13) $^{\blacktriangle}$	0.223 (0.10) $^{\blacktriangle}$	0.625 (0.11) $^{\blacktriangle}$	0.629 (0.11) $^{\blacktriangle}$	0.233 (0.07) $^{\blacktriangle}$	0.338 (0.04) $^{\vartriangle}$	0.427 (0.05)	0.418 (0.05)
	MGD-W-4	0.722 (0.07) $^{\blacktriangle}$	0.636 (0.09) $^{\blacktriangle}$	0.253 (0.09) $^{\blacktriangle}$	0.667 (0.10) $^{\blacktriangle}$	0.662 (0.09) $^{\blacktriangle}$	0.258 (0.05) $^{\blacktriangle}$	0.343 (0.04) $^{\blacktriangle}$	0.440 (0.04) $^{\blacktriangle}$	0.422 (0.05) $^{\vartriangle}$
	MGD-W-6	0.727 (0.06) $^{\blacktriangle}$	0.652 (0.07) $^{\blacktriangle}$	0.249 (0.09) $^{\blacktriangle}$	0.674 (0.09) $^{\blacktriangle}$	0.669 (0.10) $^{\blacktriangle}$	0.272 (0.04) $^{\blacktriangle}$	0.344 (0.04) $^{\blacktriangle}$	0.441 (0.04) $^{\blacktriangle}$	0.423 (0.05) $^{\vartriangle}$
	MGD-W-9	0.727 (0.06) $^{\blacktriangle}$	0.656 (0.07) $^{\blacktriangle}$	0.264 (0.10) $^{\blacktriangle}$	0.679 (0.07) $^{\blacktriangle}$	0.670 (0.09) $^{\blacktriangle}$	0.259 (0.05) $^{\blacktriangle}$	0.339 (0.04) $^{\blacktriangle}$	0.434 (0.04) $^{\vartriangle}$	0.418 (0.06)
	MGD-W-20	0.729 (0.06) $^{\blacktriangle}$	0.649 (0.06) $^{\blacktriangle}$	0.252 (0.09) $^{\blacktriangle}$	0.675 (0.09) $^{\blacktriangle}$	0.664 (0.10) $^{\blacktriangle}$	0.260 (0.05) $^{\blacktriangle}$	0.341 (0.04) $^{\blacktriangle}$	0.434 (0.05) $^{\vartriangle}$	0.415 (0.05)
	MGD-M-2	0.696 (0.09) $^{\blacktriangle}$	0.606 (0.10) $^{\blacktriangle}$	0.241 (0.09) $^{\blacktriangle}$	0.634 (0.12) $^{\blacktriangle}$	0.645 (0.12) $^{\blacktriangle}$	0.246 (0.07) $^{\blacktriangle}$	0.333 (0.04)	0.430 (0.05)	0.421 (0.05) $^{\vartriangle}$
	MGD-M-4	0.736 (0.06) $^{\blacktriangle}$	0.659 (0.07) $^{\blacktriangle}$	0.276 (0.09) $^{\blacktriangle}$	0.685 (0.08) $^{\blacktriangle}$	0.682 (0.09) $^{\blacktriangle}$	0.271 (0.04) $^{\blacktriangle}$	0.350 (0.03) $^{\blacktriangle}$	0.443 (0.04) $^{\blacktriangle}$	0.427 (0.05) $^{\blacktriangle}$
	MGD-M-6	0.742 (0.06) $^{\blacktriangle}$	0.667 (0.06) $^{\blacktriangle}$	0.275 (0.09) $^{\blacktriangle}$	0.692 (0.07) $^{\blacktriangle}$	0.687 (0.09) $^{\blacktriangle}$	0.278 (0.04) $^{\blacktriangle}$	0.351 (0.04) $^{\blacktriangle}$	0.448 (0.04) $^{\blacktriangle}$	0.427 (0.05) $^{\blacktriangle}$
	MGD-M-9	0.745 (0.06) $^{\blacktriangle}$	**0.681** (0.06) $^{\blacktriangle}$	0.283 (0.08) $^{\blacktriangle}$	**0.710** (0.08) $^{\blacktriangle}$	0.698 (0.08) $^{\blacktriangle}$	0.284 (0.04) $^{\blacktriangle}$	**0.361** (0.03) $^{\blacktriangle}$	**0.454** (0.04) $^{\blacktriangle}$	0.425 (0.05) $^{\blacktriangle}$
	MGD-M-20	**0.752** (0.06) $^{\blacktriangle}$	0.677 (0.07) $^{\blacktriangle}$	**0.295** (0.08) $^{\blacktriangle}$	0.703 (0.07) $^{\blacktriangle}$	**0.703** (0.08) $^{\blacktriangle}$	**0.291** (0.04) $^{\blacktriangle}$	0.356 (0.03) $^{\blacktriangle}$	0.452 (0.04) $^{\blacktriangle}$	**0.430** (0.05) $^{\blacktriangle}$

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 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 list 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 5.4). Figure 2 displays the online performance for all systems, again on a single dataset. Just like with offline performance, MGD outperforms DBGD more when the noise in the feedback increases. In fact, Table 3 shows that under *perfect* feedback, online performance for one dataset 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 outperforms DBGD. Also for online performance, the standard deviation of MGD is much lower than for DBGD, irrespective of the noise.

Our answer to RQ1 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 realistic levels of noise.

6.2 Convergence

In this section, we answer RQ2: whether MGD converges to a better optimum than DBGD. To do so, we investigate converged offline performance only. Table 2 shows converged performance after 1,000 query impressions for all the datasets that we consider. Note that for *NP2003* these values correspond to the points on the

Table 3: Online score (discounted cumulative NDCG, see Section 5.4) of each of the algorithms for the 3 instantiations of the CCM (see Table 1). Bold values indicate maximum performance per dataset and click model. Statistically significant improvements (losses) over the DBGD baseline are indicated by △ ($p < 0.05$) and ▲ ($p < 0.01$) (▽ and ▼). We show the standard deviation between brackets.

		HP2003	NP2003	TD2003	HP2004	NP2004	TD2004	MQ2007	MQ2008	OHSUMED
perfect	**DBGD**	95.88 (23.04)	97.79 (7.19)	36.28 (16.18)	97.93 (19.02)	102.92 (8.81)	**42.92** (14.73)	60.11 (4.49)	78.17 (4.54)	70.43 (3.85)
	MGD-W-2	110.77 (5.91) ▲	101.71 (5.87) ▲	41.19 (4.84) ▲	100.92 (6.67)	106.69 (6.17) ▲	38.15 (3.32) ▼	60.49 (3.23)	78.79 (3.97)	72.58 (3.58) ▲
	MGD-W-4	112.96 (5.14) ▲	103.42 (5.94) ▲	42.36 (3.91) ▲	104.44 (5.27) ▲	108.94 (5.48) ▲	38.50 (2.66) ▼	61.33 (3.47) △	78.52 (4.96)	72.73 (3.14) ▲
	MGD-W-6	113.37 (5.13) ▲	104.13 (5.42) ▲	43.00 (3.94) ▲	104.79 (5.46) ▲	110.02 (5.39) ▲	38.13 (2.43) ▼	61.22 (3.30) △	78.76 (4.01)	72.73 (3.48) ▲
	MGD-W-9	114.66 (4.57) ▲	**105.79** (5.88) ▲	43.53 (3.63) ▲	**107.22** (4.80) ▲	110.27 (5.69) ▲	38.39 (2.35) ▼	60.62 (3.42)	78.12 (3.93)	70.32 (3.11)
	MGD-W-20	**116.25** (4.21) ▲	104.96 (5.23) ▲	44.43 (4.19) ▲	106.23 (5.22) ▲	109.94 (5.91) ▲	39.79 (2.42) ▽	60.28 (3.35)	78.07 (3.87)	72.42 (3.58) ▲
	MGD-M-2	111.23 (5.59) ▲	101.42 (6.54) ▲	41.26 (4.37) ▲	101.67 (7.35) △	108.24 (6.08) ▲	37.51 (2.86) ▼	61.43 (3.54) △	79.08 (4.29)	72.74 (4.00) ▲
	MGD-M-4	113.91 (4.86) ▲	103.48 (5.61) ▲	42.64 (4.39) ▲	103.58 (5.62) ▲	109.70 (5.79) ▲	38.39 (2.43) ▼	**61.85** (3.10) ▲	**79.11** (4.41)	**72.84** (3.63) ▲
	MGD-M-6	113.46 (4.66) ▲	104.25 (5.02) ▲	43.22 (3.88) ▲	105.31 (5.16) ▲	109.80 (5.66) ▲	38.68 (2.47) ▼	61.00 (3.06)	78.97 (3.91)	72.78 (3.16) ▲
	MGD-M-9	115.81 (4.21) ▲	105.09 (4.71) ▲	44.02 (4.06) ▲	106.79 (5.36) ▲	**110.88** (5.94) ▲	38.38 (2.11) ▼	60.64 (2.75)	77.88 (3.86)	70.97 (3.12)
	MGD-M-20	115.67 (4.99) ▲	104.75 (4.79) ▲	**44.50** (3.71) ▲	107.05 (4.71) ▲	110.51 (5.69) ▲	40.18 (2.31) ▽	61.57 (3.10) ▲	78.69 (3.73)	72.51 (3.24) ▲
navigational	**DBGD**	78.79 (24.72)	85.83 (13.70)	32.21 (15.15)	80.61 (20.58)	90.92 (13.27)	37.46 (13.46)	58.07 (4.96)	76.04 (5.21)	66.99 (5.71)
	MGD-W-2	105.53 (8.45) ▲	95.55 (8.07) ▲	38.54 (5.71) ▲	92.59 (10.96) ▲	100.96 (9.07) ▲	35.72 (4.74)	59.24 (4.34) △	77.18 (4.95)	71.34 (4.56) ▲
	MGD-W-4	110.07 (6.55) ▲	100.22 (6.74) ▲	41.58 (4.81) ▲	100.37 (7.05) ▲	105.81 (6.72) ▲	37.90 (2.68)	59.57 (4.11) △	78.18 (4.52) ▲	72.00 (3.47) ▲
	MGD-W-6	109.97 (5.47) ▲	101.36 (5.14) ▲	41.66 (4.39) ▲	101.00 (6.46) ▲	107.07 (5.95) ▲	38.21 (3.00)	60.18 (3.28) ▲	77.88 (4.35) ▲	72.62 (3.67) ▲
	MGD-W-9	**113.17** (5.33) ▲	101.71 (5.19) ▲	43.03 (4.53) ▲	102.54 (5.67) ▲	106.63 (6.02) ▲	39.04 (2.86)	**60.71** (3.54) ▲	77.91 (4.49) ▲	72.69 (3.84) ▲
	MGD-W-20	112.18 (4.70) ▲	101.85 (5.68) ▲	42.24 (4.60) ▲	102.82 (5.80) ▲	107.02 (6.20) ▲	39.28 (2.91)	60.55 (3.14) ▲	77.77 (4.34) ▲	72.39 (3.37) ▲
	MGD-M-2	106.27 (8.78) ▲	94.34 (8.44) ▲	39.21 (5.48) ▲	94.70 (9.55) ▲	103.34 (8.49) ▲	36.40 (4.00)	59.65 (4.32) ▲	77.72 (4.73) ▲	71.04 (4.52) ▲
	MGD-M-4	109.44 (6.21) ▲	99.22 (6.87) ▲	41.02 (4.33) ▲	99.01 (7.37) ▲	105.82 (6.92) ▲	38.09 (3.21)	60.25 (3.59) ▲	78.16 (4.53) ▲	72.49 (3.81) ▲
	MGD-M-6	110.70 (5.77) ▲	100.56 (5.54) ▲	42.45 (4.57) ▲	102.04 (6.37) ▲	106.04 (6.51) ▲	38.28 (3.48)	60.31 (3.30) ▲	77.84 (4.89) ▲	**72.79** (3.57) ▲
	MGD-M-9	112.30 (5.35) ▲	**102.95** (5.62) ▲	42.74 (4.55) ▲	**103.96** (6.55) ▲	**107.41** (5.94) ▲	**39.55** (3.08)	60.36 (3.29) ▲	77.89 (3.94) ▲	72.71 (3.97) ▲
	MGD-M-20	111.38 (5.37) ▲	102.23 (5.45) ▲	**43.10** (4.34) ▲	102.88 (5.79) ▲	106.78 (5.16) ▲	38.79 (2.76)	60.21 (3.56) ▲	**78.23** (4.13) ▲	72.53 (3.11) ▲
informational	**DBGD**	48.23 (27.66)	50.42 (19.83)	22.46 (11.82)	43.36 (21.88)	59.58 (22.88)	27.76 (11.89)	55.60 (6.00)	71.94 (5.56)	62.99 (7.22)
	MGD-W-2	72.76 (18.10) ▲	64.09 (18.01) ▲	25.94 (7.55) ▲	62.61 (18.11) ▲	69.48 (16.80) ▲	26.52 (6.26)	55.83 (4.98)	73.33 (5.09) △	66.39 (5.93) ▲
	MGD-W-4	78.49 (16.93) ▲	73.02 (14.67) ▲	29.34 (6.15) ▲	70.73 (14.81) ▲	78.97 (13.16) ▲	28.21 (5.60)	56.04 (4.11)	74.31 (4.93) ▲	67.70 (4.44) ▲
	MGD-W-6	81.96 (14.73) ▲	73.66 (12.21) ▲	30.84 (6.17) ▲	70.90 (12.56) ▲	81.44 (11.32) ▲	29.12 (4.57)	56.24 (4.18)	**74.86** (4.79) ▲	67.75 (4.55) ▲
	MGD-W-9	85.14 (10.63) ▲	77.57 (10.24) ▲	30.26 (5.61) ▲	73.60 (11.66) ▲	82.86 (11.52) ▲	28.45 (4.36)	56.09 (3.61)	73.64 (4.71) △	68.48 (4.52) ▲
	MGD-W-20	84.44 (11.41) ▲	74.65 (9.53) ▲	29.91 (4.96) ▲	69.06 (13.01) ▲	81.78 (10.73) ▲	28.56 (4.28)	56.18 (3.46)	73.09 (4.87)	67.51 (4.63) ▲
	MGD-M-2	70.70 (18.56) ▲	66.03 (18.38) ▲	27.33 (6.76) ▲	61.78 (20.42) ▲	71.30 (18.17) ▲	27.29 (6.20)	56.28 (4.55)	73.35 (5.68) △	67.15 (5.63) ▲
	MGD-M-4	84.38 (14.30) ▲	76.41 (12.63) ▲	28.98 (5.88) ▲	72.82 (14.05) ▲	82.12 (11.01) ▲	28.44 (4.77)	56.58 (4.35)	74.45 (4.87) ▲	**68.75** (4.49) ▲
	MGD-M-6	85.51 (11.36) ▲	76.37 (11.58) ▲	31.55 (6.33) ▲	73.52 (13.17) ▲	83.05 (10.23) ▲	28.59 (4.16)	**56.88** (4.11)	74.26 (5.07) ▲	67.97 (4.47) ▲
	MGD-M-9	**87.84** (8.50) ▲	**81.04** (8.74) ▲	**32.45** (4.90) ▲	75.81 (10.21) ▲	**86.05** (7.79) ▲	**30.21** (3.74) △	55.91 (3.67)	74.50 (4.44) ▲	68.65 (4.51) ▲
	MGD-M-20	86.73 (7.24) ▲	80.99 (7.72) ▲	32.07 (4.23) ▲	**76.72** (9.64) ▲	82.92 (8.21) ▲	29.68 (3.32)	56.57 (3.48)	73.99 (4.61) ▲	68.50 (3.49) ▲

right vertical axis of Figure 1. These graphs are also illustrative for all datasets 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 datasets 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 2 shows, this effect is significant and substantial as soon as more than one candidate is used.

For many datasets, performance of MGD after 1,000 query impressions is almost on par with DBGD trained without noise. However, as Figure 1 and Table 2 show, MGD with enough candidates *always* outperforms DBGD after 1,000 queries.

The graphs in Figure 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 3 shows the results with the same setup as in Figure 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 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 RQ2 as follows: MGD converges to an optimum which is at least as good as the optimum DBGD finds. However, MGD does so much faster, as shown in Section 6.1.

6.3 Comparing outcome interpretations

In this section we answer RQ3: how MGD-W and MGD-M compare to each other. Figure 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 datasets we considered, see Table 2. 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 2 and Table 3. Again, the effect is

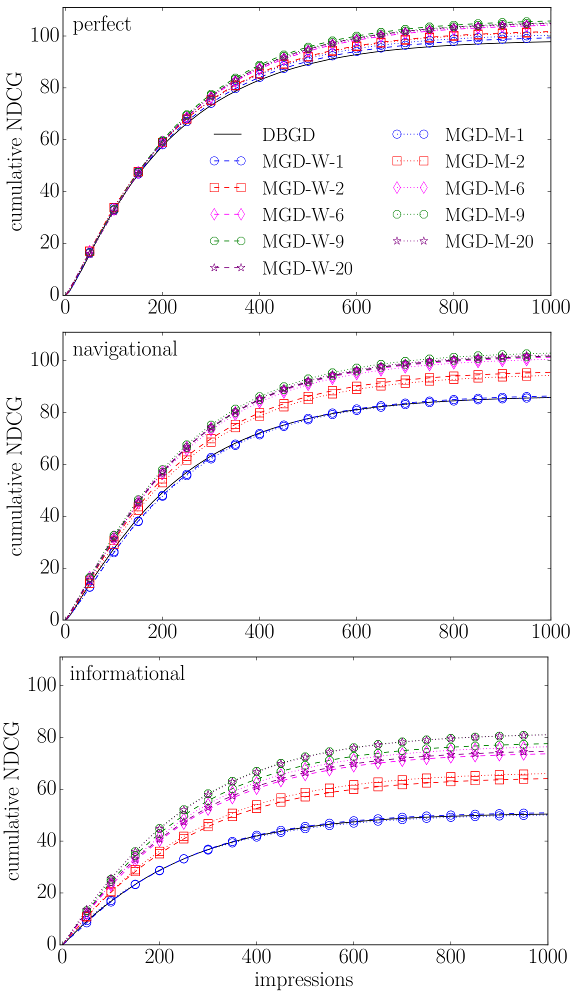

Figure 2: Online performance (discounted cumulative NDCG) on MGD-W and MGD-M with varying number of candidates compared to DBGD on *NP2003* dataset for *perfect*, *navigational* and *informational* click model instantiations.

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 1). 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 have 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, so Algorithm 4 was effectively changed such that $|w_t^0 - w_{t+1}^0| = \alpha \cdot \delta$. The result is depicted in Table 4, 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, so 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.

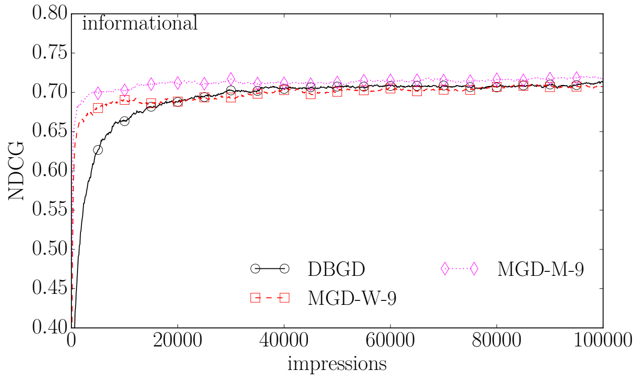

Figure 3: Offline performance (NDCG) on MGD-W and MGD-M with 9 candidates compared to DBGD on *NP2003* dataset an *informational* click model.

Table 4: Performance of MGD-M, MGD-W and normalized MGD-M each with 9 candidates for the three instantiations of the CCM (see Table 1). Run on the *NP2003* dataset; 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)

To answer RQ3, 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.

6.4 Number of candidates and learning rate

In this section, we investigate some remaining questions.

6.4.1 *Number of candidates*

In Section 6.1 we have already discussed the interplay between the amount of noise in the feedback and the optimal number of candidates in MGD. Figure 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.[4] We see in Figure 4, Table 2 and Table 3 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 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.

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

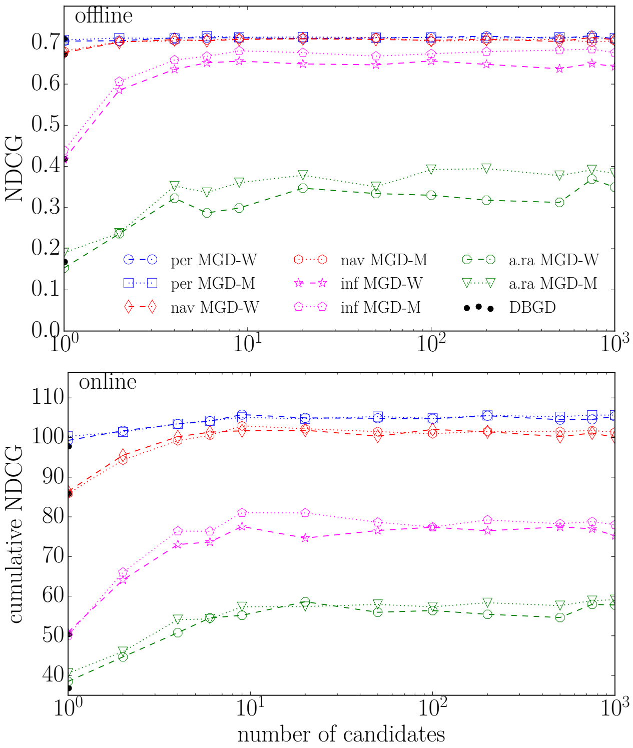

Figure 4: Sweep over the number of candidates in terms of offline and online performance for MGD-M and MGD-W after 1,000 impressions. Performed on *NP2003* using *all four* instantiations of the click model. DBGD is displayed by the black dots on the left axis. Note the log scale on the horizontal axis.

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.

6.4.2 *Learning rate*

Our MGD algorithms are sufficiently different from DBGD to warrant a new investigation of the learning rate α. The results in Section 6.3 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 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 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 other 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.

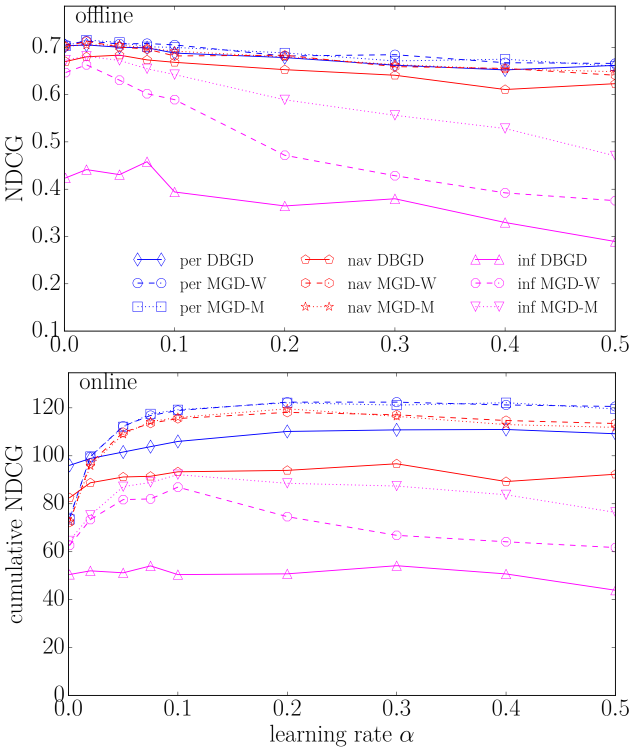

Figure 5: Sweep over learning rate values in terms of offline and online performance after 1,000 impressions for MGD-M and MGD-W with 9 candidates and DBGD. Performed on *NP2003* using three different click models with varying degrees of noise.

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 learning rate is chosen appropriately.

7. 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 datasets 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 obtains, 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.

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.

Acknowledgements. This research was partially supported by Amsterdam Data Science, the Dutch national program COMMIT, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the ESF Research Network Program ELIAS, the HPC Fund, the Royal Dutch Academy of Sciences (KNAW) under the Elite Network Shifts project, the Microsoft Research Ph.D. program, the Netherlands eScience Center under project number 027.012.105, the Netherlands Institute for Sound and Vision, the Netherlands Organisation for Scientific Research (NWO) under project nrs 727.011.005, 612.001.116, HOR-11-10, 640.006.013, 612.066.930, CI-14-25, SH-322-15, the Yahoo! Faculty Research and Engagement Program, and Yandex.

REFERENCES

[1] 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.

[2] R. Berendsen, M. Tsagkias, W. Weerkamp, and M. de Rijke. Pseudo test collections for training and tuning microblog rankers. In *SIGIR '13*. ACM, 2013.

[3] B. Carterette and J. Allan. Incremental test collections. In *CIKM*. ACM, 2005.

[4] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale validation and analysis of interleaved search evaluation. *ACM Trans. Inf. Syst.*, 30(1), 2012.

[5] C. W. Cleverdon, J. Mills, and M. Keen. Factors determining the performance of indexing systems. Aslib cranfield project, Cranfield: College of Aeronautics, 1966.

[6] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM '09*. ACM, 2009.

[7] J. He, C. Zhai, and X. Li. Evaluation of methods for relative comparison of retrieval systems based on clickthroughs. In *CIKM '09*. ACM, 2009.

[8] K. Hofmann. *Fast and Reliably Online Learning to Rank for Information Retrieval*. PhD thesis, U. Amsterdam, 2013.

[9] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM '11*. ACM, 2011.

[10] 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.

[11] 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*. ACM, 2013.

[12] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4), 2002.

[13] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*. ACM, 2002.

[14] T. Joachims. Evaluating retrieval performance using clickthrough data. In *Text Mining*. Physica/Springer, 2003.

[15] 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 Trans. Inf. Syst.*, 25(2), 2007.

[16] D. Kelly. Methods for evaluating interactive information retrieval systems with users. *Found. & Tr. Inform. Retr.*, 3 (1–2):1–224, 2009.

[17] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: survey and practical guide. *Data Min. & Knowl. Disc.*, 18(1), 2009.

[18] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM '11*, 2011.

[19] T.-Y. Liu. Learning to rank for information retrieval. *Found. & Tr. Inform. Retr.*, 3(3):225–331, 2009.

[20] 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.

[21] F. Radlinski and N. Craswell. Optimized interleaving for online retrieval evaluation. In *WSDM '13*. ACM, 2013.

[22] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM '08*. ACM, 2008.

[23] M. Sanderson. Test collection based evaluation of information retrieval systems. *Found. & Tr. Inform. Retr.*, 4(4):247–375, 2010.

[24] M. Sanderson and H. Joho. Forming test collections with no system pooling. In *SIGIR '04*. ACM, 2004.

[25] A. Schuth, K. Hofmann, S. Whiteson, and M. de Rijke. Lerot: An online learning to rank framework. In *LivingLab '13*. ACM, 2013.

[26] A. Schuth, F. Sietsma, S. Whiteson, D. Lefortier, and M. de Rijke. Multileaved comparisons for fast online evaluation. In *CIKM '14*, pages 71–80. ACM, Nov. 2014.

[27] A. L. Strehl, C. Mesterharm, M. L. Littman, and H. Hirsh. Experience-efficient learning in associative bandit problems. In *ICML '06*, 2006.

[28] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[29] M. Szummer and E. Yilmaz. Semi-supervised learning to rank with preference regularization. In *CIKM '11*. ACM, 2011.

[30] E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.

[31] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML '09*, 2009.

[32] Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The K-armed dueling bandits problem. *J. Comp. and System Sciences*, 78(5), 2009.