Balancing Speed and Quality in Online Learning to Rank for Information Retrieval

Harrie Oosterhuis
University of Amsterdam
Amsterdam, The Netherlands
oosterhuis@uva.nl

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
derijke@uva.nl

ABSTRACT

In Online Learning to Rank (OLTR) the aim is to find an optimal ranking model by interacting with users. When learning from user behavior, systems must interact with users while simultaneously learning from those interactions. Unlike other Learning to Rank (LTR) settings, existing research in this field has been limited to linear models. This is due to the speed-quality tradeoff that arises when selecting models: complex models are more expressive and can find the best rankings but need more user interactions to do so, a requirement that risks frustrating users during training. Conversely, simpler models can be optimized on fewer interactions and thus provide a better user experience, but they will converge towards suboptimal rankings. This tradeoff creates a deadlock, since novel models will not be able to improve either the user experience or the final convergence point, without sacrificing the other.

Our contribution is twofold. First, we introduce a fast OLTR model called Sim-MGD that addresses the speed aspect of the speed-quality tradeoff. Sim-MGD ranks documents based on similarities with reference documents. It converges rapidly and, hence, gives a better user experience but it does not converge towards the optimal rankings. Second, we contribute Cascading Multileave Gradient Descent (C-MGD) for OLTR that directly addresses the speed-quality tradeoff by using a cascade that enables combinations of the best of two worlds: fast learning and high quality final convergence. C-MGD can provide the better user experience of Sim-MGD while maintaining the same convergence as the state-of-the-art MGD model. This opens the door for future work to design new models for OLTR without having to deal with the speed-quality tradeoff.

1 INTRODUCTION

The goal of Learning to Rank (LTR) in Information Retrieval (IR) is to optimize models that rank documents according to user preferences. As modern search engines may combine hundreds of ranking signals they rely on models that can combine such signals to form optimal rankings. Traditionally, this was done through Offline Learning to Rank, which relies on annotated sets of queries and documents with their relevance assessed by human raters. Over the years, the limitations of this supervised approach have become

apparent: annotated sets are expensive and time-consuming to produce [6, 22]; in some settings creating such a dataset would be a serious breach of privacy [23, 37]; and annotations are not necessarily in line with user preferences [30]. As a reaction, interest in Online Learning to Rank (OLTR), where models learn from interactions with users, has increased [7, 24, 34, 40]. While this resolves many of the issues with the offline LTR setting, it brings challenges of its own. Firstly, OLTR algorithms cannot directly observe their performance and thus have to infer from user interactions how they can improve. Secondly, they have to perform their task, i.e., decide what rankings to display, while simultaneously learning from user interactions.

In stark contrast with other work on LTR, existing work in OLTR has only considered optimizing linear models and merely focussed on improving gradient estimation. We argue that this limitation is due to a *speed-quality tradeoff* that previous work has faced. This tradeoff is a result of the dual nature of the OLTR task: algorithms are evaluated both on how they perform the task while learning and on the final ranking model they converge towards. This duality is especially important as OLTR involves human interactions: some strategies may result in an optimal ranking model but may frustrate users during learning. Consider the experiment visualized in Figure 1. Here, a Linear Model (MGD) and a simpler Similarity Model (Sim-MGD) are optimized on user interactions. The latter learns faster and fully converges in fewer than 200 impressions, while the Linear Model initially trails Sim-MGD but is more expressive, requires more impressions, and ultimately exceeds Sim-MGD in offline performance (as measured in NDCG).
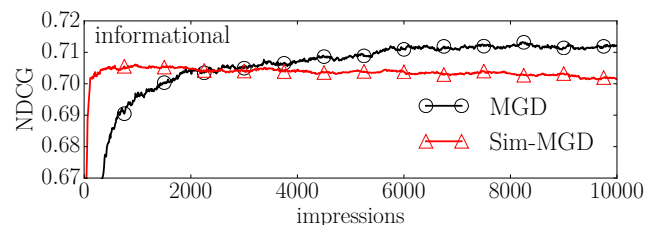

Figure 1: Offline performance (NDCG) of a Linear Model (MGD) and a Similarity Model (Sim-MGD) on the NP2003 dataset under an informational click model. (Full details of the experimental setup are provided in Section 5.)

OLTR models that are less complex, i.e., that require fewer user interactions to converge, may provide a good user experience as they adapt quickly. However, because of their limited complexity they often lack expressiveness, causing them to learn suboptimal rankings. Conversely, a more complex OLTR model may ultimately find the optimal rankings but requires more user interactions. Thus,

such models ultimately produce a better experience but risk deterring users before this level of performance is reached. As a result, a fundamental tradeoff has to be made: a good user experience during training resulting in suboptimal rankings vs. the risk of frustrating users while finding superior rankings in the end. We call this dichotomy the *speed-quality tradeoff*.

To address the speed-quality tradeoff, a method for combining the properties of multiple models is required. In this paper we meet this challenge by making two contributions. First, we introduce a novel model that uses document feature similarities (Sim-MGD) to learn more rapidly than the state-of-the-art, Multileave Gradient Descent (MGD) [24, 34]. However, Sim-MGD converges towards rankings inferior to MGD as predicted by the speed-quality tradeoff. Secondly, we propose a novel cascading OLTR approach, called C-MGD, that uses two OLTR models, a fast simple model and a slower complex model. Initially the cascade lets the faster model learn by interacting with its users. Later, when the faster learner has converged it is used to initialize the expressive model and discarded. C-MGD then continues optimization by letting the expressive model interact with the user. Consequently, the user experience is improved, both short term and long term, as users initially interact with a fast adapting model, while ultimately the better ranker using the complex model is still found. Our empirical results show that the cascade approach, i.e., C-MGD, can combine the improved user experience from Sim-MGD while still maintaining the optimal convergence of the state-of-the-art.

In this paper we address the following research questions:

RQ1 Is the user experience significantly improved when using Sim-MGD?

RQ2 Can the cascading approach, C-MGD, combine an improvement in user experience while maintaining convergence towards state-of-the-art performance levels?

To facilitate replicability and repeatability of our findings, we provide open source implementations of both Sim-MGD and C-MGD.[1]

2 RELATED WORK

We provide a brief overview of LTR and OLTR before describing methods for combining multiple models in Machine Learning.

2.1 Learning to rank

Learning to Rank (LTR) is an important part of Information Retrieval (IR) and allows modern search engines to base their rankings on hundreds of relevance signals [21]. Traditionally, a supervised approach is taken where human raters annotate whether a document is relevant to a query [6, 25]. Additionally, previous research has considered semi-supervised approaches that use unlabeled sample data next to annotated data [20, 36]. Both supervised and semi-supervised approaches are typically performed offline, meaning that training is performed after annotated data has been collected. When working with previously collected data, the speed-quality tradeoff does not arise, since users are not involved during training. Consequently, complex and expressive models have been very successful in the offline setting [5, 20].

However, in recent years several issues with training on annotated datasets have been found. Firstly, gathering annotations

is time-consuming and costly [6, 22, 25], making it infeasible for smaller organisations to collect such data. Secondly, for certain search contexts collecting data would be unethical, e.g., in the context of search within personal emails or documents [37]. Thirdly, since the datasets are static, they cannot account for future changes in what is considered relevant. Models derived from such datasets are not necessarily aligned with user satisfaction, as annotators may interpret queries differently from actual users [30].

2.2 Online learning to rank

Online Learning to Rank (OLTR) attempts to solve the issues with offline annotations by directly learning from user interactions [38], as direct interactions with users are expected to be more representative of their preferences than offline annotations [29]. The task of OLTR algorithms is two-fold: they must choose what rankings to display to users while simultaneously learning from interactions with the presented rankings. Although the OLTR task can be modeled as a Reinforcement Learning (RL) problem [35], it differs from a typical RL setting because there is no observable reward. The main difficulties with performing both aspects of the OLTR task come in the form of *bias* and *noise*. Noise occurs when the user's interactions do not represent their true preferences, e.g., users often click on a document for unexpected reasons [30]. Bias arises in different ways, e.g., there is selection bias as interactions only involve displayed documents [37] and position bias as documents at the top of a ranking are more likely to be considered [39]. These issues complicate relevance inference, since the most clicked documents are not necessarily the most relevant.

Consequently, state-of-the-art OLTR algorithms do not attempt to predict the relevance of single documents. Instead, they approach training as a dueling bandit problem [38] which relies on methods from online evaluation to compare rankers based on user interactions [26, 28]. Interleaving methods combine rankings from two rankers to produce a single result list; from large numbers of clicks on interleavings a preference for one of the two rankers can be inferred [16, 27]. This approach has been extended to find preferences between larger sets of rankers in the form of multileaving [32, 33]. These comparison methods have recently given rise to MGD, a more sensitive OLTR algorithm that requires fewer user interactions to reach the same level of performance [34]. The improvement is achieved by comparing multiple rankers at each user impression, the results of which are then used to update the OLTR model. Initially, the number of rankers in the comparison was limited to the SERP length [32]. Probabilistic multileaving [33] allows comparisons of a virtually unlimited size, leading to even better gradient estimation [24].

In contrast to Offline LTR [5, 20], work in OLTR has only considered optimizing linear combinations of ranking features [17, 38]. Recent research has focused on improving the gradient estimation of the MGD algorithm [24, 34]. We argue this focus is a consequence of the speed-quality tradeoff; since OLTR algorithms are evaluated by the final model they produce (i.e., offline performance) and the user experience during training (i.e., online performance), improvements should not sacrifice either of these aspects. Unfortunately, every model falls on one side of the tradeoff. For instance, more complex models like regression forests or neural networks

---

Algorithm 1 Multileave Gradient Descent (MGD) [34].

1: **Input**: $n, \delta, \mathbf{w}_0^0, \eta$
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:       $\mathbf{l}_i \leftarrow generate\_list(\mathbf{w}_t^i, q_t)$ // *exploratory ranking*
9:    $\mathbf{m}_t \leftarrow multileave(\mathbf{l})$ // *multileaving*
10:   $\mathbf{c}_t \leftarrow receive\_clicks(\mathbf{m}_t)$ // *show multileaving to the user*
11:   $\mathbf{b}_t \leftarrow infer\_preferences(\mathbf{l}, \mathbf{m}_t, \mathbf{c}_t)$ // *winning rankers*
12:   $\mathbf{w}_{t+1}^0 \leftarrow \mathbf{w}_t^0 + \eta \frac{1}{|\mathbf{b}_t|} \sum_{j \in \mathbf{b}_t} \mathbf{u}_t^j$ // *winning set may be empty*

---

[5] are very prominent in offline LTR but they require much larger amounts of training data than for instance a simpler linear model. Thus initially more users will be shown inferior rankings when training such a complex model . Although such models may eventually find the optimal rankings, they sacrifice the user experience during training and thus will not beat the MGD baseline in online performance. Our solution to this tradeoff is meant to stimulate the exploration of a wider range of ranking models in OLTR.

2.3 Multileave gradient descent

We build on the Multileave Gradient Descent algorithm [34]; see Algorithm 1. Briefly, at all times the algorithm has a *current best* ranker $\mathbf{w}_0^t$ that is the estimate of the optimal ranker at timestep t. Initially, this model starts at the root $\mathbf{w}_0^0 = 0$, then after each issued query, another n rankers $\mathbf{w}_t^n$ are sampled from the unit sphere around the *current best* ranker (Line 7). These sampled rankers are candidates: slight variations of the *current best*; MGD tries to infer if these variations are an improvement and updates accordingly. The candidates produce rankings for the query, which are combined into a single multileaved result list, e.g., by using Probabilistic Multileaving [24, 33] (Line 9). The resulting result list is displayed to the user and clicks are observed (Line 10); from the clicks the rankers preferred over the *current best* are inferred (Line 11). If none of the other rankers is preferred the *current best* is kept, otherwise the model takes a η step towards the mean of the winning rankers (Line 12). After the model has been updated, the algorithm waits for the next query to repeat the process.

2.4 Combining models in machine learning

Combining models is a prevalent approach in machine learning [2]; often, this is done by averaging the predictions of a set of models [4]. Alternatively, some methods select which model to use based on the input variables [18]. A set of multiple models whose output is averaged is called a committee, a concept that can be applied in different ways. The simplest way is by *bagging*: training different models on bootstrapped datasets and taking the mean of their predictions [3]. A more powerful committee technique is boosting [11], which trains models in sequence. Each model is trained on a weighted form of the dataset where the weights of a datapoint depend on the performance of the committee thus far. Hence, training will

give more weight to points that are misclassified by the previous models. When the committee is complete their predictions are combined using a weighted voting scheme. This form of boosting is applicable to supervised classification [11] and regression [12]; it has also been used extensively in offline LTR, e.g., in LambdaMART [5]. The main difference with our approach and ensemble methods is that their aim is to reduce the final error of the committee. None of the ensemble methods are based around user interactions; hence, none deal with the speed-quality tradeoff.

On top of the related work discussed above we contribute the following: a novel OLTR method that ranks based on feature similarities with example documents. This is the first OLTR model that is not a direct linear model. Furthermore, we introduce a novel OLTR algorithm that combines multiple ranking models, unlike the model combining methods discussed before this method does not combine the output of two models. Instead, different parts of the learning process are assigned to the models that are expected to perform best during that period, i.e., a model that requires less data will perform better in the initial phase of learning. This makes it the first algorithm that uses multiple ranking models to increase the user experience during learning.

3 SIM-MGD: A FAST OLTR MODEL BASED ON DOCUMENT FEATURE SIMILARITY

In this section we introduce a novel ranking model for OLTR, by basing result lists on feature similarities with reference documents it learns more rapidly than MGD. However, as predicted by the speed-quality tradeoff, the increase in speed sacrifices some of the expressiveness of the model; Section 4 provides a method for dealing with this tradeoff.

Previous work in OLTR has only considered optimizing linear combinations of features of documents.[2] Let $\mathbf{w}$ be the set of weights that is learned and $\mathbf{d}$ the feature representation of a query document pair. Then a document is ranked according to the score of:

$$R_{MGD}(\mathbf{d}) = \sum_i w_i d_i. \tag{1}$$

There are several properties of the LTR problem that this model does not make use of. For instance, almost all LTR features are relevance signals (e.g., BM25 or PageRank), so it is very unlikely that any should be weighted negatively. However, MGD does not consider this when exploring; it may even consider a completely negative ranker.

As an alternative, we propose a ranking model based on the assumption that relevant documents have similar features. Here, a set of document-query pairs $\mathbf{D}_M = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$ is used as reference points, documents are then ranked based on their weighted similarity to those in the set:

$$R_{sim}(\mathbf{d}) = \sum_{m=1}^{M} \frac{w_m}{|\mathbf{d}_m|} \mathbf{d}^T \mathbf{d}_m \tag{2}$$

where the documents in $\mathbf{D}_M$ are L_2-normalized. Since this model consists of a linear combination, optimizing its weights $\mathbf{w}$ is straightforward with the existing MGD (Algorithm 1) or with our novel algorithm C-MGD (Algorithm 3) to be introduced below. For clarity

---

[2]Though learning the parameters for individual ranking features was researched [31].

we have displayed MGD optimizing the similarity model Sim-MGD in Algorithm 2. Unlike MGD, Sim-MGD requires a collection of document-query pairs from which the set $\mathbf{D}_m$ is sampled (Line 2). Sim-MGD is still initialized with $\mathbf{w}_0^0 = 0$ but the number of weights is now determined by the size of the reference set M. For each query that is received, a result list is created by the *current best* ranker (Line 5); here, the ranker is defined by the weights $\mathbf{w}_t^0$ and the set $\mathbf{D}_M$ according to Equation 2. Then n candidates are sampled around the *current best* ranker (Line 8) and their result lists are also created using Equation 2 (Line 9). The result lists are combined into a multileaving and presented to the user (Line 10–11); if preferences are inferred from their interactions with the displayed result list, the *current best* ranker is updated accordingly (Line 12–13).

The intuition behind Sim-MGD is that it is easier to base a result list on good or bad examples than it is to discover how each feature should be weighed. Moreover, MGD optimizes faster in spaces with a lower dimensionality [38]; thus, a small number of reference documents M speeds up learning further. In spite of this speedup, the similarity model is less expressive than the standard linear model (Equation 1). Regardless of $\mathbf{D}_M$, the similarity model can always be rewritten to a linear model:

$$R(\mathbf{d}) = \sum_{m=1}^{M} \frac{w_m}{|\mathbf{d}_m|} \mathbf{d}^T \mathbf{d}_m = \mathbf{d}^T \sum_{m=1}^{M} \frac{w_m}{|\mathbf{d}_m|} \mathbf{d}_m. \tag{3}$$

However, not every linear model can necessarily be rewritten as a similarity model, especially if the reference set $\mathbf{D}_M$ is small. Thus the space of models is limited by $\mathbf{D}_M$, providing faster learning but potentially excluding the optimal ranker. Therefore, the similarity model falls on the speed side of the speed-quality tradeoff.

For this paper, different sampling methods for creating $\mathbf{D}_M$ (Line 2) are investigated. First, a uniform sampling, expected to cover all documents evenly, is considered. Additionally, k-means clustering is used, where $k = M$ and the centroid of each cluster is used as a reference document; this increases the chance of representing all different document types in the reference set.

Sim-MGD is expected to learn faster and provide a better initial user experience than MGD. However, it is less expressive and is thus expected to converge at an inferior optimum. Again, without the use of C-MGD the similarity model falls on the speed side of the speed-quality tradeoff.

4 C-MGD: COMBINING OLTR MODELS AS A CASCADE

We aim to combine the initial learning speed of one model and the final convergence of another. This provides the best performance and user experience in the short and long term. Our proposed algorithm makes use of a cascade: initially it optimizes the faster model by letting it interact with the users until convergence is detected. At this point, the learning speed of the faster model will no longer be of advantage as the model is oscillating around a (local) optimum. Furthermore, it is very likely that a better optimum exists in a more expressive model space, especially if the faster model is relatively simple. To make use of this likelihood, optimization is continued using a more complex model that is initialized with the first model. If this switch is made appropriately, the advantages of both models are combined: a fast initial learning speed and

---

Algorithm 2 MGD with the Similarity Model (Sim-MGD).

1: **Input**: C, M, n, δ, $\mathbf{w}_0^0$, η
2: $\mathbf{D}_M \leftarrow \{\mathbf{d}_0, \ldots, \mathbf{d}_m\} \sim sample(\mathbf{C})$ // sample reference documents
3: **for** $t \leftarrow 1..\infty$ **do**
4:   $q_t \leftarrow receive\_query(t)$ // obtain a query from a user
5:   $\mathbf{l}_0 \leftarrow generate\_list(\mathbf{w}_t^0, q_t, \mathbf{D}_M)$ // exploitive ranking (Eq. 2)
6:   **for** $i \leftarrow 1..n$ **do**
7:     $\mathbf{u}_t^i \leftarrow sample\_unit\_vector()$
8:     $\mathbf{w}_t^i \leftarrow \mathbf{w}_t^0 + \delta \mathbf{u}_t^i$ // create a candidate ranker
9:     $\mathbf{l}_i \leftarrow generate\_list(\mathbf{w}_t^i, q_t, \mathbf{D}_M)$ // exploratory ranking (Eq. 2)
10:   $\mathbf{m}_t \leftarrow multileave(\mathbf{l})$ // multileaving
11:   $\mathbf{c}_t \leftarrow receive\_clicks(\mathbf{m}_t)$ // show multileaving to the user
12:   $\mathbf{b}_t \leftarrow infer\_preferences(\mathbf{l}, \mathbf{m}_t, \mathbf{c}_t)$ // winning rankers
13:   $\mathbf{w}_{t+1}^0 \leftarrow \mathbf{w}_t^0 + \eta \frac{1}{|\mathbf{b}_t|} \sum_{j \in \mathbf{b}_t} \mathbf{u}_t^j$ // winning set may be empty

---

convergence at a better optimum. We call this algorithm *Cascading Multileave Gradient Descent* (C-MGD); before it is detailed, we discuss the main challenges of switching between models during learning.

4.1 Detecting convergence

C-MGD has to detect convergence during optimization. After sufficiently many interactions, the performance of MGD plateaus [24, 34]. However, in the online setting there is no validation set to verify this. Instead, convergence of the model itself can be measured by looking at how much it has changed over a recent period of time. If the ranker has barely changed, then either the estimated gradient is oscillating around a point in the model space, or few of the clicks prefer the candidates that MGD has proposed. Both cases are indicative of finding a (local) optimum. Correspondingly, during MGD optimization the convergence of a model $\mathbf{w}^t$ at timestep t can be assumed if it has not changed substantially during the past h iterations. C-MGD considers a change significant if the cosine similarity between the current model and the model of h iterations earlier exceeds a chosen threshold ϵ:

$$\frac{\mathbf{w}^t \cdot \mathbf{w}^{t-h}}{\|\mathbf{w}^t\| \cdot \|\mathbf{w}^{t-h}\|} < 1 - \epsilon. \tag{4}$$

The cosine similarity is appropriate here since linear combinations are unique by their direction and not their norm. Since scaling the weights of a model produces the same rankings, i.e., for a document pair $\{\mathbf{d}_i, \mathbf{d}_j\}$:

$$\mathbf{w} \cdot \mathbf{d}_i > \mathbf{w} \cdot \mathbf{d}_j \rightarrow \beta \mathbf{w} \cdot \mathbf{d}_i > \beta \mathbf{w} \cdot \mathbf{d}_j. \tag{5}$$

Therefore, a minor change in the cosine similarity indicates that the model creates rankings that are only slightly different.

4.2 Difference in confidence

C-MGD has to account for the difference in confidence when changing model space. Convergence in the simpler model space gives C-MGD confidence that an optimum was found, but some of this confidence is lost when switching model spaces since a lot of the new space has not been explored. MGD's confidence is indicated by the norm of its model's weights, which increases if a preference in the same direction is repeatedly found. Consequently, when

initializing the subsequent model C-MGD has to renormalize for the difference in confidence due to switching model spaces. This is not trivial as it affects exploration, since the norm determines how dissimilar the sampled candidates will be. If the norm is set too low it will continue by exploring a large region of model space, thus neglecting the learning done by the previous model. But if C-MGD starts with a norm that is too large it will continue with so little exploration that it may not find the new optimum in a reasonable amount of time.

Directly measuring confidence is not possible in the online setting. Instead, rescaling is estimated from the difference in dimensionality of the models:

$$\|\mathbf{w}_{complex}\| = \|\mathbf{w}_{simple}\| \cdot \frac{\sqrt{D_{simple}}}{\sqrt{D_{complex}}}, \tag{6}$$

where D_{simple} and $D_{complex}$ are the dimensionality of the simple and complex model respectively. In line with the regret bounds found by Yue and Joachims [38], the algorithm's confidence decreases when more parameters are introduced.

4.3 A walkthrough of C-MGD

Finally, C-MGD is formulated in Algorithm 3. As input, C-MGD takes two ranking models R_{simple} and $R_{complex}$ with dimensionalities D_{simple} and $D_{complex}$. C-MGD will optimize its *current best* weights $\mathbf{w}_t^0$ for its current model R_*. Initially, R_* is set to the fast learner: R_{simple} (Line 2). Then, for each incoming query (Line 4) the ranking of the current model $(R_*, \mathbf{w}_t^0)$ is generated (Line 5). Subsequently, n candidates are sampled from the unit sphere around the current weights and the ranking of each candidate is generated (Line 7–9). All of the rankings are then combined into a single multileaving [33] and displayed to the user (Line 10–11). Based on the clicks of the user, a preference between the candidates and the *current best* can be inferred (Line 12). If some candidates are preferred over the *current best*, an update is performed to take an η step towards them (Line 13). Otherwise, the *current best* weights will be carried over to the next iteration. At this point C-MGD will check for convergence by comparing the cosine similarity between the *current best* and the weights from h iterations before: w_{t-h}^0 (Line 14). If convergence is detected, C-MGD switches to the complex model (Line 15) and the *current best* weights are converted to the new model space (Line 16). The weights now have to be renormalized to account for the change in model space and rescaled for the difference in confidence (Line 17). Optimization now continues without the check for convergence.

The result is an algorithm that optimizes a cascade of two models, combining the advantages of both. For this study we only considered a cascade of two models, extending this approach to a larger number is straightforward.

5 EXPERIMENTS

This section describes the experiments we run to answer the research questions posed in Section 1. Firstly (**RQ1**), we are interested in whether Sim-MGD provides a better user experience, i.e., online performance, than MGD. Secondly (**RQ2**), we wish to know if C-MGD is capable of dealing with the speed-quality tradeoff,

---

Algorithm 3 Cascading Multileave Gradient Descent (C-MGD).

1: **Input:** $n, \delta, \mathbf{w}_0^0, h, \epsilon, R_{simple}, R_{complex}, D_{simple}, D_{complex}$
2: $R_* \leftarrow R_{simple}$
3: **for** $t \leftarrow 1..\infty$ **do**
4:     $q_t \leftarrow receive\_query(t)$ // obtain a query from a user
5:     $\mathbf{l}_0 \leftarrow generate\_list(R_*, \mathbf{w}_t^0, q_t)$ // ranking of current best
6:     **for** $i \leftarrow 1...n$ **do**
7:         $\mathbf{u}_t^i \leftarrow sample\_unit\_vector()$
8:         $\mathbf{w}_t^i \leftarrow \mathbf{w}_t^0 + \delta\mathbf{u}_t^i$ // create a candidate ranker
9:         $\mathbf{l}_i \leftarrow generate\_list(\mathbf{w}_t^i, q_t, R_*)$ // exploratory ranking
10:     $\mathbf{m}_t, \mathbf{t}_t \leftarrow multileave(\mathbf{l})$ // multileaving and teams
11:     $\mathbf{c}_t \leftarrow receive\_clicks(\mathbf{m}_t)$ // show multileaving to the user
12:     $\mathbf{b}_t \leftarrow infer\_preferences(\mathbf{t}_t, \mathbf{c}_t)$ // winning candidates
13:     $\mathbf{w}_{t+1}^0 \leftarrow \mathbf{w}_t^0 + \eta \frac{1}{|\mathbf{b}_t|} \sum_{j \in \mathbf{b}_t} \mathbf{u}_t^j$ // winning set may be empty
14:     **if** $t \geq h \wedge R_* = R_{simple} \wedge 1 - \cos(\mathbf{w}_{t+1}^0, \mathbf{w}_{t-h}^0) < \epsilon$ **then**
15:         $R_* \leftarrow R_{complex}$
16:         $\mathbf{w}' \leftarrow convert_{R_{simple} \rightarrow R_{complex}}(\mathbf{w}_{t+1})$ // new model space
17:         $\mathbf{w}_{t+1} \leftarrow \mathbf{w}' \cdot \frac{\|\mathbf{w}_{t+1}\|}{\|\mathbf{w}'\|} \cdot \frac{\sqrt{D_{simple}}}{\sqrt{D_{complex}}}$

---

that is, whether C-MGD can provide the improved user experience of Sim-MGD (online performance) while also having the optimal convergence of MGD (offline performance).

Every experiment below is based around a stream of independent queries coming from users. The system responds to a query by presenting a list of documents to the user in an impression. The user may or may not interact with the list by clicking on one or more documents. The queries and documents come from static datasets (Section 5.1), users are simulated using click models (Section 5.2). Our experiments are described in Section 5.3 and our metrics in Section 5.4.

5.1 Datasets

Our experiments are performed over eleven publicly available OLTR datasets with varying sizes and representing different search tasks. Each dataset consists of a set of queries and a set of corresponding documents for every query. While queries are represented only by their identifiers, feature representations and relevance labels are available for every document-query pair. Relevance labels are graded differently by the datasets depending on the task they model; for instance, the navigational datasets have binary labels for not relevant (0) and relevant (1), whereas most informational tasks have labels ranging from not relevant (0) to perfect relevancy (5). Every dataset is divided in training, validation and test partitions.

The first publicly available *Learning to Rank* datasets are distributed as LETOR 3.0 and 4.0 [22]; they use representations of 45, 46, or 64 features, respectively, that encode ranking models such as TF.IDF, BM25, Language Modelling, PageRank, and HITS on different parts of the documents. The datasets in LETOR are divided by their tasks, most of which come from the TREC Web Tracks between 2003 and 2008 [9, 10]: *HP2003*, *HP2004*, *NP2003*, and *NP2004* are based on navigational tasks; both *TD2003* and *TD-2004* implement the informational task of topic distillation. *HP2003, HP2004, NP2003, NP2004, TD2003* and *TD2004* each contain between 50 and 150 queries and 1,000 judged documents per query. The

Table 1: Instantiations of Cascading Click Models [13] as used for simulating user behavior in experiments.

R	$P(click = 1 \mid R)$					$P(stop = 1 \mid R)$				
	0	1	2	3	4	0	1	2	3	4
perf	0.0	0.2	0.4	0.8	1.0	0.0	0.0	0.0	0.0	0.0
nav	0.05	0.3	0.5	0.7	0.95	0.2	0.3	0.5	0.7	0.9
inf	0.4	0.6	0.7	0.8	0.9	0.1	0.2	0.3	0.4	0.5

OHSUMED dataset is based on a query log of the search engine on the MedLine abstract database, and contains 106 queries. Lastly, the two most recent datasets *MQ2007* and *MQ2008* were based on the Million Query Track [1] and consist of 1700 and 800 queries, respectively, but have far fewer assessed documents per query.

In 2010 Microsoft released the *MSLR-WEB30k* and *MLSR-WEB10K* [25], the former consists of 30,000 queries obtained from a retired labelling set of a commercial web search engine (Bing), the latter is a subsampling of 10,000 queries from the former dataset. The datasets uses 136 features to represent its documents, each query has around 125 assessed documents. For practical reasons only *MLSR-WEB10K* was used for this paper.

Lastly, also in 2010 Yahoo! organised a public Learning to Rank Challenge [6] with an accompanying dataset. This set consist of 709,877 documents encoded in 700 features and sampled from query logs of the Yahoo! search engine spanning 29,921 queries.

5.2 Simulating user behavior

Users are simulated using the standard setup for OLTR simulations [14, 24, 34]. First, a user issues a query simulated by uniformly sampling a query from the static dataset. Subsequently, the algorithm decides the result list of documents to display. The behavior of the user after it receives this result list is simulated using a *cascade click model* [8, 13]. This model assumes a user to examine the documents of the result list in their displayed order. For each document that is considered the user decides whether it warrants a click. This is modelled as the conditional probability $P(click = 1 \mid R)$, where R is the relevance label provided by the dataset. Accordingly, *cascade click model* instantiations increase the probability of a click with the degree of the relevance label. After the user has clicked on a document, their information need may be satisfied, otherwise they will continue by considering the remaining documents. The probability of the user not examining more documents after clicking is modeled as $P(stop = 1 \mid R)$, where it is more likely that the user is satisfied from a very relevant document. For this paper $\kappa = 10$ documents are displayed to the user at each impression.

Table 1 lists the three instantiations of cascade click models that were used for this paper. The first models a *perfect* user that considers every document and clicks on all relevant documents and nothing else. Secondly, the *navigational* instantiation models a user performing a navigational task and is mostly looking for a single highly relevant document. Finally, the *informational* instantiation models a user without a very specific information need that typically clicks on multiple documents. These three models have increasing levels of noise, as the behavior of each depends less on the relevance labels of the displayed documents.

5.3 Experimental runs

As a baseline, Probabilistic-MGD [24] is used. Based on previous work this study uses $n = 19$ candidates per iteration sampled from the unit sphere with $\delta = 1$; updates are performed with $\eta = 0.01$ and weights are intialized as $\mathbf{w}_0^0 = \mathbf{0}$ [15, 24, 34, 38]. All runs are run over 10,000 impressions. Probabilistic Multileaving inferences are computed using a sample-based method [33], where the number of document assignments sampled for every inference is 10,000 [24].

Sim-MGD uses $M = 50$ reference documents that are selected from the training set at the start of each run. The choice for $M = 50$ was based on preliminary results on the evaluation sets. Two selection methods are investigated: uniform sampling and k-means clustering. The clustering method uses $k = M$, i.e., producing a reference document for every cluster it finds. The expectation is that Sim-MGD has a higher learning speed but is less expressive than MGD, thus, we expect to see a substantial increase in online performance but a decrease in offline performance compared to MGD. Clustering is expected to provide reference documents that cover all kinds of documents better, potentially resulting in a further increase of online performance and a lower standard deviation compared to uniform sampling.

Finally, to evaluate whether C-MGD can successfully combine speed and quality of two models, C-MGD is run with Sim-MGD as R_{simple} (Equation 2) and the linear model as $R_{complex}$ (Equation 1). If the cascade can successfully swap models then we expect to see no significant decrease in offline performance but a substantial increase in online performance compared to MGD. When comparing to Sim-MGD we expect a significant increase in offline performance due to C-MGD's ability to switch models. However, it is very likely that a slight decrease in online performance is observed, since the change of model space introduces more exploration. Lastly, the reference document selection methods are expected to have the same effects on C-MGD as they have on Sim-MGD.

5.4 Metrics and tests

The task in OLTR consists of two parts: a ranker has to be optimized and users have to be attended during optimization. Accordingly, both aspects are evaluated separately.

Offline performance considers the quality of the learned model by taking the average NDCG score of the *current best ranker* over a held-out set. Performance is assessed using the NDCG [19] metric:

$$NDCG = \sum_{i=1}^{\kappa} \frac{2^{rel(\mathbf{r}[i])} - 1}{\log_2(i + 1)} iDCG^{-1}. \tag{7}$$

This metric calculates the Discounted Cumulative Gain (DCG) over the relevance labels $rel(\mathbf{r}[i])$ for each document in the top κ of a ranking. Subsequently, this is normalized by the maximal DCG possible for a query: the ideal DCG (iDCG). This results in Normalized DCG (NDCG) which measures the quality of a single ranked list of documents. Offline performance is averaged over a held-out set after 10,000 impressions to give an indication at what performance the algorithms converge.

Conversely, the user experience during training is essential as well, since deterring users during training would compromise the purpose of the system. *Online performance* is assessed by computing the cumulative NDCG of the rankings shown to the users [14, 35].

For T successive queries this is the discounted sum:

$$Online\_Performance = \sum_{t=1}^{T} NDCG(\mathbf{m}_t) \cdot \gamma^{(t-1)} \qquad (8)$$

where $\mathbf{m}_t$ is the ranking displayed to the user at timestep t. This metric is common in *online learning* and can be interpreted as the expected reward with γ as the probability that another query will be issued. For *online performance* a discount factor of $\gamma = 0.9995$ was chosen so that queries beyond the horizon of 10,000 queries have a less than 1% impact [24].

Finally, all runs are repeated 125 times, spread evenly over the dataset's folds; results for each run are averaged and a two tailed Student's t-test is used to verify whether differences are statistically significant [41]. In total, our experiments are based on over 200 million user impressions.

6 RESULTS AND ANALYSIS

This section presents the results of our experiments and answers the research questions posed in Section 1.

6.1 Improving the user experience with Sim-MGD

First we consider **RQ1**: whether Sim-MGD improves the user experience compared to MGD.

6.1.1 Online performance. Table 2 (Columns 2–4) displays the online performance of Sim-MGD and MGD. In the large majority of cases Sim-MGD provides a significant increase in online performance over MGD, both with the uniform and k-means document selection strategies. E.g., under the perfect user model, 7 out of 11 datasets for uniform and 8 out of 11 for k-means. Significant decreases in online performance are found for *HP2003*, *TD2003*, *TD2004* and *OHSUMED* for uniform and for *TD2003*, *TD2004* and *OHSUMED* for k-means. Interestingly, all of these datasets model informational tasks, which suggests that it is more difficult to create an appropriate reference set in these cases. Furthermore, the differences between Sim-MGD and MGD are consistent over the different click-models. Therefore, we conclude that Sim-MGD is as robust to noise as MGD.

Finally, Table 2 (Columns 3–4) allows us to contrast the online performance of different document selection strategies: k-means beats uniform on the majority of datasets under all user models, and the noisier the user model is, the bigger the majority is. Therefore, it seems that clustering results in a faster learning speed of Sim-MGD; this could be because k-means will provide more dissimilar reference documents. Hence, the parameters in Sim-MGD will be less correlated making learning faster than for uniform sampling.

In conclusion, Sim-MGD improves the user experience most of the time, but is not reliable as it may provide a significantly worse experience depending on the dataset.

6.1.2 Offline performance. Table 3 (Columns 2–4) displays the offline performance of Sim-MGD and MGD. As predicted by the speed-quality tradeoff, we see that the convergence of Sim-MGD after 10,000 impressions is substantially worse than MGD. This suggests that the optimum found by MGD can generally not be
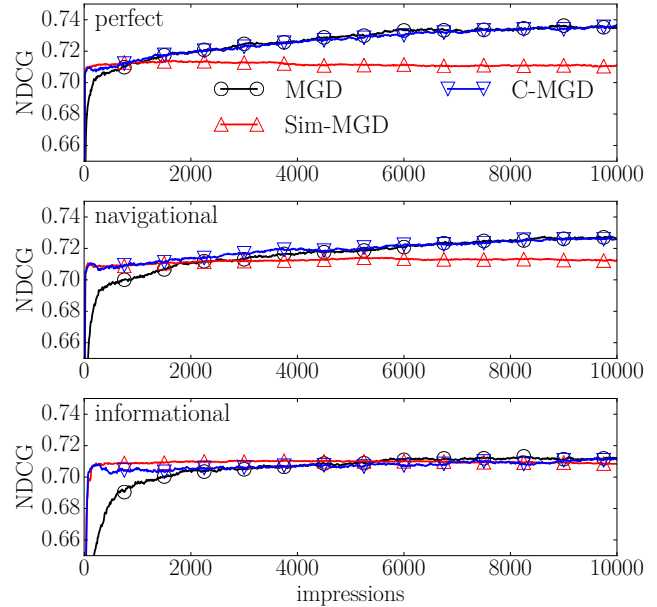

Figure 2: Offline performance (NDCG) of MGD, the Sim-MGD and C-MGD (k-means initialization) on the NP2003 dataset under three click models.

expressed by the similarity model in Sim-MGD, i.e., it is not a linear combination of document features.

Figure 2 shows the offline performance of MGD and Sim-MGD on the *NP2003* dataset for the three click models. Here, the improved learning speed is visible as Sim-MGD outperforms MGD in the initial phase of learning, under more click-noise MGD requires more impressions to reach the same performance. For the *informational* click model over 2000 impressions are required for MGD to reach the performance Sim-MGD had in fewer than 200. However, it is clear that Sim-MGD has an inferior point of convergence, as it is eventually overtaken by MGD under all click models.

Lastly, Table 3 (Columns 3–4) shows the scores for Sim-MGD with different reference document selection methods. The k-means selection method provides a higher online performance and a slightly better point of convergence. Therefore, it seems that clustering helps in selecting reference documents but has a limited effect.

To answer **RQ1**, Sim-MGD improves the user experience in most cases, i.e., on most datasets and under all user models, with a consistent benefit for the k-means document selection strategy. As predicted by the speed-quality tradeoff, Sim-MGD converges towards inferior rankings than MGD, due to its less expressive model.

6.2 Resolving the speed-quality tradeoff with C-MGD

Next, we address the speed-quality tradeoff with **RQ2**: whether C-MGD is capable of improving the user experience while maintaining the state-of-the-art convergence point.

6.2.1 Learning speed. To evaluate the user experience, the online performance of C-MGD and MGD can be examined in Table 2

Table 2: Online performance (Discounted Cumulative NDCG, Section 5.4) for different instantiations of CCM (Table 1). The standard deviation is shown in brackets, bold values indicate the highest performance per dataset and click model, significant improvements and losses over the MGD baseline are indicated by $\triangle$ (p < 0.05) and $\blacktriangle$ (p < 0.01) and by $\triangledown$ and $\blacktriangledown$, respectively.

	MGD	Sim-MGD		C-MGD	
		uniform	k-means	uniform	k-means
perfect					
HP2003	764.4 (16.7)	750.8 (36.2) $\blacktriangledown$	771.1 (23.6) $\triangle$	770.5 (17.9) $\blacktriangle$	**773.0** (15.9) $\blacktriangle$
NP2003	699.5 (19.5)	789.1 (17.8) $\blacktriangle$	**794.2** (18.7) $\blacktriangle$	724.6 (16.7) $\blacktriangle$	723.9 (18.6) $\blacktriangle$
TD2003	**312.2** (20.0)	280.1 (29.4) $\blacktriangledown$	279.1 (23.0) $\blacktriangledown$	306.0 (21.9) $\triangledown$	307.8 (21.7)
HP2004	732.3 (19.0)	766.8 (28.0) $\blacktriangle$	**777.7** (21.6) $\blacktriangle$	748.1 (20.4) $\blacktriangle$	746.9 (20.1) $\blacktriangle$
NP2004	719.9 (17.8)	769.8 (23.9) $\blacktriangle$	**781.8** (20.2) $\blacktriangle$	737.8 (17.6) $\blacktriangle$	740.5 (15.9) $\blacktriangle$
TD2004	**298.9** (12.5)	268.1 (19.8) $\blacktriangledown$	267.6 (11.2) $\blacktriangledown$	295.2 (11.3) $\triangledown$	296.4 (11.5)
MQ2007	412.5 (10.4)	**448.4** (10.2) $\blacktriangle$	443.1 (10.7) $\blacktriangle$	423.4 (10.8) $\blacktriangle$	421.1 (9.9) $\blacktriangle$
MQ2008	523.2 (15.8)	**547.3** (16.2) $\blacktriangle$	543.1 (16.5) $\blacktriangle$	531.3 (15.1) $\blacktriangle$	527.1 (15.2) $\triangle$
MSLR-WEB10k	336.6 (6.3)	347.9 (6.6) $\blacktriangle$	**351.2** (6.5) $\blacktriangle$	340.7 (6.3) $\blacktriangle$	342.1 (5.9) $\blacktriangle$
OHSUMED	494.8 (15.8)	483.5 (16.4) $\blacktriangledown$	483.2 (17.3) $\blacktriangledown$	494.4 (17.2)	**495.4** (16.6)
Yahoo	732.1 (10.9)	773.7 (12.5) $\blacktriangle$	**778.6** (9.9) $\blacktriangle$	741.5 (10.0) $\blacktriangle$	742.7 (11.7) $\blacktriangle$
navigational					
HP2003	701.2 (19.7)	717.3 (37.5) $\blacktriangle$	**734.6** (21.6) $\blacktriangle$	715.6 (18.3) $\blacktriangle$	718.0 (18.3) $\blacktriangle$
NP2003	637.6 (23.0)	765.0 (19.4) $\blacktriangle$	**772.4** (19.0) $\blacktriangle$	684.8 (18.0) $\blacktriangle$	686.8 (19.3) $\blacktriangle$
TD2003	**272.5** (19.9)	253.9 (25.2) $\blacktriangledown$	256.4 (22.3) $\blacktriangledown$	265.6 (20.1) $\blacktriangledown$	269.8 (20.8)
HP2004	663.0 (20.9)	724.5 (30.5) $\blacktriangle$	**742.9** (24.4) $\blacktriangle$	693.4 (24.1) $\blacktriangle$	693.7 (20.7) $\blacktriangle$
NP2004	653.2 (20.3)	743.9 (24.3) $\blacktriangle$	**756.1** (21.7) $\blacktriangle$	686.6 (19.5) $\blacktriangle$	692.2 (17.3) $\blacktriangle$
TD2004	**263.3** (12.6)	242.5 (16.3) $\blacktriangledown$	243.8 (10.4) $\blacktriangledown$	260.7 (11.9)	261.8 (10.4)
MQ2007	385.9 (11.9)	**430.5** (12.3) $\blacktriangle$	424.3 (11.7) $\blacktriangle$	402.4 (9.7) $\blacktriangle$	402.0 (11.2) $\blacktriangle$
MQ2008	501.5 (16.3)	**534.6** (14.1) $\blacktriangle$	529.5 (14.4) $\blacktriangle$	513.4 (15.2) $\blacktriangle$	511.8 (15.3) $\blacktriangle$
MSLR-WEB10k	323.2 (7.2)	335.0 (8.1) $\blacktriangle$	**338.2** (7.8) $\blacktriangle$	327.7 (6.7) $\blacktriangle$	330.0 (6.1) $\blacktriangle$
OHSUMED	**482.6** (15.9)	464.2 (19.4) $\blacktriangledown$	465.0 (17.3) $\blacktriangledown$	478.8 (16.6)	480.1 (15.2)
Yahoo	721.1 (14.4)	758.2 (27.1) $\blacktriangle$	**767.1** (22.2) $\blacktriangle$	732.2 (17.3) $\blacktriangle$	733.9 (18.4) $\blacktriangle$
informational					
HP2003	650.9 (22.6)	680.5 (33.0) $\blacktriangle$	**703.7** (22.3) $\blacktriangle$	673.2 (21.5) $\blacktriangle$	675.6 (20.5) $\blacktriangle$
NP2003	603.0 (26.1)	750.8 (19.4) $\blacktriangle$	**757.0** (20.6) $\blacktriangle$	655.7 (21.9) $\blacktriangle$	657.5 (17.3) $\blacktriangle$
TD2003	251.6 (20.4)	247.3 (22.0)	248.8 (20.6)	**257.0** (22.3) $\triangle$	255.3 (21.1)
HP2004	616.1 (25.4)	697.9 (33.9) $\blacktriangle$	**718.7** (23.1) $\blacktriangle$	652.6 (28.9) $\blacktriangle$	651.8 (23.6) $\blacktriangle$
NP2004	617.8 (23.4)	719.0 (26.8) $\blacktriangle$	**736.2** (23.7) $\blacktriangle$	661.9 (21.9) $\blacktriangle$	663.3 (20.1) $\blacktriangle$
TD2004	245.0 (15.4)	232.8 (15.9) $\blacktriangledown$	237.0 (12.0) $\blacktriangledown$	**250.2** (13.1) $\blacktriangle$	247.9 (11.5)
MQ2007	377.2 (15.1)	**415.4** (39.6) $\blacktriangle$	404.5 (46.7) $\blacktriangle$	386.5 (35.7) $\blacktriangle$	387.6 (34.2) $\blacktriangle$
MQ2008	496.3 (22.0)	508.1 (60.6) $\triangle$	**512.9** (50.7) $\blacktriangle$	489.5 (52.8)	491.2 (51.2)
MSLR-WEB10k	321.4 (8.5)	329.9 (22.9) $\blacktriangle$	**331.3** (25.4) $\blacktriangle$	321.8 (23.9)	324.1 (24.2)
OHSUMED	474.3 (15.1)	457.5 (19.6) $\blacktriangledown$	460.6 (18.5) $\blacktriangledown$	473.1 (16.5)	**474.4** (17.5)
Yahoo	707.3 (19.4)	728.8 (55.4) $\blacktriangle$	**734.3** (52.7) $\blacktriangle$	716.9 (28.6) $\blacktriangle$	714.5 (31.7) $\triangle$

(Column 2 vs. 5 and 6). The online performance of C-MGD is predominantly a significant improvement over that of MGD. Moreover, when k-means document selection is used, no significant decreases are measured on any dataset or click model. Even on datasets where Sim-MGD performs significantly worse than MGD in terms of online performance, no significant decrease is observed for C-MGD. Thus, C-MGD deals with the inferior performance of its starting model by effectively switching to a more expressive model space.

6.2.2 Quality convergence. Furthermore, the quality side of the tradeoff is examined by considering the offline performance after 10,000 impressions, displayed in Table 3 (Column 2 vs. 5 and 6). In the vast majority of cases C-MGD shows no significant change in offline performance compared to MGD. For C-MGD with uniform selection only four instances of significant decreases in offline performance w.r.t. MGD are found scattered over different datasets and user models; this number is further reduced when k-mean selection is used. Only for MQ2008 under the informational user model this difference is greater than 0.1 NDCG. In all other cases, the offline performance of MGD is maintained by C-MGD or slightly improved. Conclusively, C-MGD converges towards rankings of the same quality as MGD.

Table 3: Offline performance (NDCG) after 10,000 impressions, notation is identical to Table 2.

	MGD	Sim-MGD		C-MGD	
		uniform	k-means	uniform	k-means
perfect					
HP2003	**0.782** (0.06)	0.709 (0.06) ▼	0.720 (0.06) ▼	0.781 (0.07)	**0.782** (0.07)
NP2003	0.719 (0.04)	0.708 (0.04) ▽	0.713 (0.04)	**0.720** (0.04)	0.719 (0.04)
TD2003	**0.327** (0.08)	0.253 (0.09) ▼	0.243 (0.08) ▼	0.322 (0.08)	0.325 (0.08)
HP2004	**0.751** (0.07)	0.709 (0.09) ▼	0.714 (0.08) ▼	0.750 (0.07)	0.747 (0.07)
NP2004	0.719 (0.04)	0.693 (0.07) ▼	0.698 (0.08) ▼	0.719 (0.04)	**0.720** (0.04)
TD2004	**0.333** (0.05)	0.254 (0.03) ▼	0.254 (0.02) ▼	0.328 (0.05)	0.329 (0.05)
MQ2007	0.406 (0.02)	0.401 (0.02) ▽	0.395 (0.02) ▼	**0.408** (0.02)	0.407 (0.02)
MQ2008	**0.493** (0.04)	0.485 (0.04)	0.479 (0.04) ▼	0.491 (0.04)	0.490 (0.04)
MSLR-WEB10k	**0.312** (0.00)	0.301 (0.00) ▼	0.303 (0.00) ▼	**0.312** (0.00)	**0.312** (0.00)
OHSUMED	**0.456** (0.05)	0.439 (0.05) ▼	0.442 (0.05) ▽	0.455 (0.05)	0.455 (0.05)
Yahoo	**0.675** (0.01)	0.656 (0.01) ▼	0.657 (0.00) ▼	0.672 (0.01) ▼	0.672 (0.01) ▽
navigational					
HP2003	0.764 (0.06)	0.683 (0.07) ▼	0.690 (0.06) ▼	0.765 (0.06)	**0.767** (0.06)
NP2003	0.711 (0.04)	0.707 (0.04)	0.711 (0.04)	0.712 (0.04)	**0.714** (0.04)
TD2003	**0.315** (0.09)	0.237 (0.08) ▼	0.237 (0.07) ▼	0.299 (0.09)	0.306 (0.10)
HP2004	0.740 (0.07)	0.694 (0.08) ▼	0.700 (0.08) ▼	**0.743** (0.07)	0.740 (0.07)
NP2004	0.717 (0.04)	0.686 (0.08) ▼	0.691 (0.07) ▼	0.720 (0.05)	**0.722** (0.05)
TD2004	**0.314** (0.05)	0.230 (0.03) ▼	0.225 (0.03) ▼	0.308 (0.04)	0.307 (0.04)
MQ2007	0.356 (0.02)	**0.387** (0.02) ▲	0.377 (0.02) ▲	0.364 (0.02) ▲	0.363 (0.02) ▲
MQ2008	0.468 (0.03)	**0.478** (0.04) △	0.470 (0.03)	0.470 (0.03)	0.467 (0.03)
MSLR-WEB10k	**0.307** (0.00)	0.300 (0.00) ▼	0.302 (0.00) ▼	**0.307** (0.00)	**0.307** (0.00)
OHSUMED	**0.439** (0.05)	0.405 (0.06) ▼	0.404 (0.06) ▼	0.430 (0.06)	0.430 (0.06)
Yahoo	**0.660** (0.03)	0.637 (0.04) ▼	0.643 (0.03) ▼	0.655 (0.03)	0.657 (0.04)
informational					
HP2003	0.759 (0.06)	0.649 (0.07) ▼	0.667 (0.06) ▼	**0.764** (0.07)	0.763 (0.07)
NP2003	0.704 (0.05)	0.705 (0.04)	**0.710** (0.04)	0.706 (0.04)	0.705 (0.05)
TD2003	**0.286** (0.10)	0.232 (0.08) ▼	0.241 (0.07) ▼	0.275 (0.09)	0.267 (0.09)
HP2004	0.732 (0.07)	0.681 (0.07) ▼	0.688 (0.08) ▼	0.735 (0.07)	**0.738** (0.07)
NP2004	0.711 (0.05)	0.683 (0.07) ▼	0.688 (0.07) ▼	0.714 (0.05)	**0.717** (0.06)
TD2004	**0.299** (0.04)	0.221 (0.03) ▼	0.220 (0.03) ▼	0.289 (0.03) ▽	0.292 (0.03)
MQ2007	0.340 (0.02)	**0.370** (0.06) ▲	0.352 (0.07)	0.336 (0.05)	0.339 (0.05)
MQ2008	**0.456** (0.05)	0.440 (0.10)	0.441 (0.08)	0.424 (0.10) ▼	0.426 (0.09) ▼
MSLR-WEB10k	**0.301** (0.01)	0.292 (0.04) ▼	0.293 (0.04) ▽	0.292 (0.04) ▼	0.293 (0.04) ▽
OHSUMED	**0.433** (0.05)	0.402 (0.06) ▼	0.404 (0.06) ▼	0.424 (0.06)	0.426 (0.06)
Yahoo	**0.618** (0.05)	0.590 (0.08) ▼	0.600 (0.08) ▽	0.616 (0.07)	0.610 (0.07)

6.2.3 Switching models. Lastly, we consider whether C-MGD is able to effectively switch between model spaces. As discussed in the previous paragraphs, Table 2 and 3 show that C-MGD improves the user experience of MGD while maintaining the final performance at convergence. This switching of models can also be observed in Figure 2, where the offline performance of Sim-MGD, MGD and Sim-MGD on the *NP2003* dataset for the three click models is displayed. As expected, we see that initially Sim-MGD learns very fast and converges in less than 300 impressions; C-MGD has the same performance during this period. When convergence of Sim-MGD is approached C-MGD switches to the linear model. A small drop in NDCG is visible when this happens under the informational click model. However, from this point on C-MGD uses the same model

as MGD and eventually reaches a higher performance than it had before the switch was made. This indicates that the switch was made effectively but had some minor short-term costs, which can be accounted to the change in confidence: after switching, C-MGD will perform more exploration in the new model space. As a result, C-MGD may explore inferior parts of the model space before oscillating towards the optimum. Despite these costs, when switching C-MGD is able to provide a reliable improvement in user experience over MGD while Sim-MGD cannot (Table 2). Thus we conclude that the switching of models is done effectively by C-MGD as evident by the reliable improvement of online performance over MGD while also having the same final offline performance.

In conclusion, we answer **RQ2** positively: our results show that in spite of the speed-quality tradeoff, C-MGD improves the user experience of MGD while still converging towards the same quality rankings. These findings were made across eleven datasets and varying levels of noise in the user models employed.

7 CONCLUSION

In this paper we have addressed the speed-quality tradeoff that has been facing the field of OLTR. Expressive models are capable of learning the most optimal rankings but require more user interactions and as a result frustrate more users during training. To put it bluntly, users may be frustrated in the initial phase of learning; models that converge at the best rankings frustrate the users for the longest period.

As a solution we have introduced two methods. The first method is a ranking model that ranks by feature similarities with reference documents (Sim-MGD). Sim-MGD learns faster and consequently provides a much better initial user experience. As predicted by the speed-quality tradeoff it converges towards rankings inferior to MGD. The second is a cascading approach, C-MGD, that deals with the speed-quality tradeoff by using a cascade of models. Initially the simplest model in the cascade interacts with the users until convergence is detected; at this point a more expressive model continues the learning process. By doing so the cascade combines the best of both models: fast initial learning speed and optimal convergence.

The introduction of C-MGD opens an array of possibilities. A natural extension is to consider expressive models that have been successful in Offline-LTR and place them in C-MGD as the short-term user experience can be addressed by C-MGD. E.g., an OLTR version of LambdaMart [5] could be appended to a cascade that starts with the Sim-MGD model, then switches to MGD and finally switches to a novel OLTR regression forest. Currently, there is no OLTR method of gradient estimation for non-linear structures like regression trees: the introduction of C-MGD removes an important hurdle for research into such methods. Additionally, an initialization method has to be introduced to enable the switch between such models. Ideally, the cascading approach should be extended to predict whether switching model space will have a positive effect; multileaving may be adapted to infer such differences.

REFERENCES
[1] J. Allan, B. Carterette, J. A. Aslam, V. Pavlu, B. Dachev, and E. Kanoulas. Million query track 2007 overview. In *TREC*. NIST, 2007.
[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
[5] C. J. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research, 2010.
[6] O. Chapelle and Y. Chang. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research*, 14:1–24, 2011.
[7] S. Chaudhuri and A. Tewari. Online learning to rank with feedback at the top. *arXiv preprint arXiv:1603.01855*, 2016.
[8] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Morgan & Claypool Publishers, 2015.
[9] C. L. Clarke, N. Craswell, and I. Soboroff. Overview of the TREC 2009 Web Track. In *TREC*. NIST, 2009.
[10] N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. Overview of the TREC 2003 Web Track. In *TREC*. NIST, 2003.
[11] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156. Morgan Kaufmann, 1996.
[12] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.
[13] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM*, pages 124–131. ACM, 2009.
[14] K. Hofmann. *Fast and Reliably Online Learning to Rank for Information Retrieval*. PhD thesis, University of Amsterdam, 2013.
[15] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in learning to rank online. In *ECIR*, pages 251–263. Springer, 2011.
[16] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM*, pages 249–258. ACM, 2011.
[17] 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):63–90, 2012.
[18] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.
[19] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.
[20] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142. ACM, 2002.
[21] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.
[22] 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.
[23] M. Najork. Using machine learning to improve the email experience. In *CIKM*, pages 891–891. ACM, 2016.
[24] H. Oosterhuis, A. Schuth, and M. de Rijke. Probabilistic multileave gradient descent. In *ECIR*, pages 661–668. Springer, 2016.
[25] T. Qin and T. Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013. URL http://arxiv.org/abs/1306.2597.
[26] F. Radlinski and N. Craswell. Comparing the sensitivity of information retrieval metrics. In *SIGIR*, pages 667–674. ACM, 2010.
[27] F. Radlinski and N. Craswell. Optimized interleaving for online retrieval evaluation. In *WSDM*, pages 245–254. ACM, 2013.
[28] F. Radlinski and K. Hofmann. Practical online retrieval evaluation. In *ECIR*, pages 878–881. Springer, 2013.
[29] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM*, pages 43–52. ACM, 2008.
[30] M. Sanderson. Test collection based evaluation of information retrieval systems. *Foundations and Trends in Information Retrieval*, 4(4):247–375, 2010.
[31] A. Schuth, F. Sietsma, S. Whiteson, and M. de Rijke. Optimizing base rankers using clicks. In *ECIR*, pages 75–87. Springer, 2014.
[32] A. Schuth, F. Sietsma, S. Whiteson, D. Lefortier, and M. de Rijke. Multileaved comparisons for fast online evaluation. In *CIKM*, pages 71–80. ACM, 2014.
[33] A. Schuth, R.-J. Bruintjes, F. Büttner, J. van Doorn, et al. Probabilistic multileave for online retrieval evaluation. In *SIGIR*, pages 955–958. ACM, 2015.
[34] A. Schuth, H. Oosterhuis, S. Whiteson, and M. de Rijke. Multileave gradient descent for fast online learning to rank. In *WSDM*, pages 457–466. ACM, 2016.
[35] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[36] M. Szummer and E. Yilmaz. Semi-supervised learning to rank with preference regularization. In *CIKM*, pages 269–278. ACM, 2011.
[37] X. Wang, M. Bendersky, D. Metzler, and M. Najork. Learning to rank with selection bias in personal search. In *SIGIR*, pages 115–124. ACM, 2016.
[38] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML '09*, 2009.
[39] Y. Yue, R. Patel, and H. Roehrig. Beyond position bias: Examining result attractiveness as a source of presentation bias in clickthrough data. In *WWW*, pages 1011–1018. ACM, 2010.
[40] T. Zhao and I. King. Constructing reliable gradient exploration for online learning to rank. In *CIKM*, pages 1643–1652. ACM, 2016.
[41] D. W. Zimmerman. Comparative power of student t test and mann-whitney u test for unequal sample sizes and variances. *The Journal of Experimental Education*, 55(3):171–174, 1987.