BubbleRank: Safe Online Learning to Re-Rank via Implicit Click Feedback

Chang Li[1] **Branislav Kveton**[2] **Tor Lattimore**[3] **Ilya Markov**[1]
Maarten de Rijke[1] **Csaba Szepesvári**[3,4] **Masrour Zoghi**[2]
[1]University of Amsterdam [2]Google Research [3]DeepMind [4]University of Alberta
c.li@uva.nl, bkveton@google.com, tor.lattimore@gmail.com, i.markov@uva.nl,
derijke@uva.nl, szepesva@cs.ualberta.ca, masrour@zoghi.org

Abstract

In this paper, we study the problem of safe online learning to re-rank, where user feedback is used to improve the quality of displayed lists. Learning to rank has traditionally been studied in two settings. In the offline setting, rankers are typically learned from relevance labels created by judges. This approach has generally become standard in industrial applications of ranking, such as search. However, this approach lacks exploration and thus is limited by the information content of the offline training data. In the online setting, an algorithm can experiment with lists and learn from feedback on them in a sequential fashion. Bandit algorithms are well-suited for this setting but they tend to learn user preferences from scratch, which results in a high initial cost of exploration. This poses an additional challenge of *safe* exploration in ranked lists. We propose `BubbleRank`, a bandit algorithm for safe re-ranking that combines the strengths of both the offline and online settings. The algorithm starts with an initial base list and improves it online by gradually exchanging higher-ranked less attractive items for lower-ranked more attractive items. We prove an upper bound on the n-step regret of `BubbleRank` that degrades gracefully with the quality of the initial base list. Our theoretical findings are supported by extensive experiments on a large-scale real-world click dataset.

1 INTRODUCTION

Learning to rank (LTR) is an important problem in many application domains, such as information retrieval, ad placement, and recommender systems [23]. More generally, LTR arises in any situation where multiple items, such as web pages, are presented to users. It is particularly relevant when the diversity of users makes it hard to decide which item should be presented to a specific user [28, 37].

A traditional approach to LTR is offline learning of rankers from either relevance labels created by judges [26] or user interactions [13, 24]. Recent experimental results [38] shows that such rankers, even in a highly-optimized search engine, can be improved by online LTR with exploration. Exploration is the key component in multi-armed bandit algorithms [3]. Many such algorithms have been proposed recently for online LTR in specific user-behavior models [15, 17, 19], the so-called *click models* [6]. Compared to earlier online LTR algorithms [28], these click model-based algorithms gain in statistical efficiency while giving up on generality. Empirical results indicate that click model-based algorithms are likely to be beneficial in practice.

Yet, existing algorithms for online LTR in click models are impractical for at least three reasons. First, an actual model of user behavior is typically unknown. This problem was initially addressed by Zoghi et al. [39]. They showed that the list of items in the descending order of relevance is optimal in several click models and proposed `BatchRank` for learning it. Then Lattimore et al. [20] built upon this work and proposed `TopRank`, which is the state-of-the-art online LTR algorithm. Second, these algorithms lack safety constraints and explore aggressively by placing potentially irrelevant items at high positions, which may significantly degrade user experience [34]. A third and related problem is that the algorithms are not well suited for so-called *warm start* scnearios [33], where the offline-trained production ranker already generates a good list, which only needs to be safely improved. Warm-starting an online LTR algorithm is challenging since existing posterior sampling algorithms, such as Thompson sampling [32], require item-level priors while only list-level priors are available practically.

We make the following contributions. First, motivated by the exploration scheme of Radlinski and Joachims [27], we propose a bandit algorithm for online LTR that addresses all three issues mentioned above. The proposed algorithm gradually improves upon an initial base list by exchanging higher-ranked less attractive items for lower-ranked more attractive items. The algorithm resembles bubble sort [8], and therefore we call it `BubbleRank`. Second, we prove an upper bound on the n-step regret of `BubbleRank`. The bound reflects the behavior of `BubbleRank`: worse initial base lists lead to a higher regret. Third, we define our safety constraint, which is based on incorrectly-ordered item pairs in the ranked list, and prove that `BubbleRank` never violates this constraint with a high probability. Finally, we evaluate `BubbleRank` extensively on a large-scale real-world click dataset.

2 BACKGROUND

This section introduces our online learning problem. We first review click models [6] and then introduce a stochastic click bandit [39], a learning to rank framework for multiple click models.

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

2.1 CLICK MODELS

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

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

In the *cascade model* (CM) [9], the user scans list $\mathcal{R}$ from the first item $\mathcal{R}(1)$ to the last $\mathcal{R}(K)$. If item $\mathcal{R}(k)$ is *attractive*, the user *clicks* on it and does not examine the remaining items. If item $\mathcal{R}(k)$ is not attractive, the user *examines* item $\mathcal{R}(k+1)$. The first item $\mathcal{R}(1)$ is examined with probability one. Therefore, the expected number of clicks is equal to the probability of clicking on any item, and is $r(\mathcal{R}) = \sum_{k=1}^{K} \chi(\mathcal{R}, k)\alpha(\mathcal{R}(k))$, where $\chi(\mathcal{R}, k) = \prod_{i=1}^{k-1}(1 - \alpha(\mathcal{R}(i)))$ is the examination probability of position k in list $\mathcal{R}$.

In the *position-based model* (PBM) [29], the probability of clicking on an item depends on both its identity and position. Therefore, in addition to item-dependent attraction probabilities α, the PBM is parameterized by K *position-dependent examination probabilities* $\chi \in [0,1]^K$, where $\chi(k)$ is the examination probability of position k. The user interacts with list $\mathcal{R}$ as follows. The user *examines* position $k \in [K]$ with probability $\chi(k)$ and then *clicks* on item $\mathcal{R}(k)$ at that position with probability $\alpha(\mathcal{R}(k))$. Therefore, the expected number of clicks on list $\mathcal{R}$ is $r(\mathcal{R}) = \sum_{k=1}^{K} \chi(k)\alpha(\mathcal{R}(k))$.

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

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

when $\alpha(1) \geq \dots \geq \alpha(L)$. Therefore, perhaps not surprisingly, the problem of learning the optimal list in both models can be viewed as the same problem, a stochastic click bandit [39].

2.2 STOCHASTIC CLICK BANDIT

An instance of a *stochastic click bandit* [39] is a tuple (K, L, P_α, P_χ), where $K \leq L$ is the number of positions, L is the number of items, P_α is a distribution over binary attraction vectors $\{0,1\}^L$, and P_χ is a distribution over binary examination matrices $\{0,1\}^{\Pi_K(\mathcal{D}) \times K}$.

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

$$\boldsymbol{c}_t(k) = \boldsymbol{X}_t(\boldsymbol{\mathcal{R}}_t, k)\boldsymbol{A}_t(\boldsymbol{\mathcal{R}}_t(k)), \tag{2}$$

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

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

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

$$\mathbb{E}\left[\boldsymbol{c}_t(k) \,|\, \boldsymbol{\mathcal{R}}_t = \mathcal{R}\right] = \chi(\mathcal{R}, k)\alpha(\mathcal{R}(k)), \qquad (3)$$

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

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

$$R(n) = \sum_{t=1}^{n} \mathbb{E}\left[\max_{\mathcal{R} \in \Pi_K(\mathcal{D})} r(\mathcal{R}, \alpha, \chi) - r(\boldsymbol{\mathcal{R}}_t, \alpha, \chi)\right]. \quad (4)$$

3 ONLINE LEARNING TO RE-RANK

Multi-stage ranking is widely used in production ranking systems [5, 14, 22], with the re-ranking stage at the very end [5]. In the re-ranking stage, a relatively small number of items, typically 10–20, are re-ranked. One reason for re-ranking is that offline rankers are typically trained to minimize the average loss across a large number of queries. Therefore, they perform well on very frequent queries and poorly on infrequent queries. On moderately frequent queries, the so-called *torso queries*, their performance varies. As torso queries are sufficiently frequent, an online algorithm can be used to re-rank so as to optimize their value, such as the number of clicks [38].

We propose an online algorithm that addresses the above problem and adaptively re-ranks a list of items generated by a production ranker with the goal of placing more attractive items at higher positions. We study a non-contextual variant of the problem, where we re-rank a small number of items in a single query. Generalization across queries and items is an interesting direction for future work. We follow the setting in Section 2.2, except that $\mathcal{D} = [K]$. Despite these simplifying assumptions, our learning problem remains a challenge. In particular, the attraction of items is only observed through clicks in (2), which are affected by other items in the list.

3.1 ALGORITHM

Our algorithm is presented in Algorithm 1. The algorithm gradually improves upon an *initial base list* $\mathcal{R}_0$ by "bubbling up" more attractive items. Therefore, we refer to it as `BubbleRank`. `BubbleRank` determines more attractive items by randomly exchanging neighboring items. If the lower-ranked item is found to be more attractive, the items are permanently exchanged and never randomly exchanged again. If the lower-ranked item is found to be less attractive, the items are never randomly exchanged again. We describe `BubbleRank` in detail below.

Algorithm 1: `BubbleRank`

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

`BubbleRank` maintains a *base list* $\bar{\boldsymbol{\mathcal{R}}}_t$ at each time t. From the viewpoint of `BubbleRank`, this is the best list at time t. The list is initialized by the initial base list $\mathcal{R}_0$ (line 3). At time t, `BubbleRank` permutes $\bar{\boldsymbol{\mathcal{R}}}_t$ into a *displayed list* $\boldsymbol{\mathcal{R}}_t$ (lines 6–10). Two kinds of permutations are employed. If t is odd and so $h = 0$, the items at positions 1 and 2, 3 and 4, and so on, are randomly exchanged. If t is even and so $h = 1$, the items at positions 2 and 3, 4 and 5, and so on are randomly exchanged. The items are exchanged only if `BubbleRank` is uncertain regarding which item is more attractive (line 9).

The list $\boldsymbol{\mathcal{R}}_t$ is displayed and `BubbleRank` gets feedback (line 11). Then it updates its statistics (lines 12–19). For any exchanged items i and j, if item i is clicked and item j is not, the belief that i is more attractive than j, $\boldsymbol{s}_t(i,j)$, increases; and the belief that j is more attractive than i, $\boldsymbol{s}_t(j,i)$, decreases. The number of observations, $\boldsymbol{n}_t(i,j)$ and $\boldsymbol{n}_t(j,i)$, increases. These statistics are updated only if one of the items is clicked (line 15), not both.

At the end of time t, the base list $\bar{\boldsymbol{\mathcal{R}}}_t$ is improved (lines 20–24). More specifically, if any lower-ranked item j is found to be more attractive than its higher-ranked neighbor i (line 23), the items are permanently exchanged in the next base list $\bar{\boldsymbol{\mathcal{R}}}_{t+1}$.

A notable property of `BubbleRank` is that it explores safely, since any item in the displayed list $\boldsymbol{\mathcal{R}}_t$ is at most one position away from its position in the base list $\bar{\boldsymbol{\mathcal{R}}}_t$. Moreover, any base list improves upon the initial base list $\mathcal{R}_0$, because it is obtained by bubbling up more attractive items with a high confidence. We make this notion of safety more precise in Section 4.2.

4 THEORETICAL ANALYSIS

In this section, we provide theoretical guarantees on the performance of `BubbleRank`, by bounding the n-step regret in (4).

The content is organized as follows. In Section 4.1, we present our upper bound on the n-step regret of `BubbleRank`, together with our assumptions. In Section 4.2, we prove that `BubbleRank` is safe. In Section 4.3, we discuss our theoretical results. The regret bound is proved in Section 4.4. Our technical lemmas are stated and proved in Appendix A.

4.1 REGRET BOUND

Before we present our result, we introduce our assumptions[1] and complexity metrics.

Assumption 1. *For any lists $\mathcal{R}, \mathcal{R}' \in \Pi_K(\mathcal{D})$ and positions $k, \ell \in [K]$ such that $k < \ell$:*

A1. $r(\mathcal{R}, \alpha, \chi) \leq r(\mathcal{R}^*, \alpha, \chi)$*, where $\mathcal{R}^*$ is defined in* (1)*;*

A2. $\{\mathcal{R}(1), \dots, \mathcal{R}(k-1)\} = \{\mathcal{R}'(1), \dots, \mathcal{R}'(k-1)\} \implies \chi(\mathcal{R}, k) = \chi(\mathcal{R}', k)$*;*

A3. $\chi(\mathcal{R}, k) \geq \chi(\mathcal{R}, \ell)$*;*

A4. If $\mathcal{R}$ and $\mathcal{R}'$ differ only in that the items at positions k and ℓ are exchanged, then $\alpha(\mathcal{R}(k)) \leq \alpha(\mathcal{R}(\ell)) \iff \chi(\mathcal{R}, \ell) \geq \chi(\mathcal{R}', \ell)$; and

A5. $\chi(\mathcal{R}, k) \geq \chi(\mathcal{R}^*, k)$.

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

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

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

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

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

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

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

4.2 SAFETY

Let

$$\mathcal{V}(\mathcal{R}) = \left\{(i,j) \in [K]^2 : i < j, \mathcal{R}^{-1}(i) > \mathcal{R}^{-1}(j)\right\} \quad (5)$$

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

Lemma 2. *Let*

$$\mathcal{V}_0 = \mathcal{V}(\mathcal{R}_0) \quad (6)$$

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

[1] Our assumptions are slightly weaker than those of Zoghi et al. [39]. For instance, Assumption A2 is on the probability of examination. Zoghi et al. [39] make this assumption on the realization of examination.

Proof. Our claim follows from two observations. First, by the design of `BubbleRank`, any displayed list $\boldsymbol{\mathcal{R}}_t$ contains at most $K/2$ item pairs that are ordered differently from its base list $\bar{\boldsymbol{\mathcal{R}}}_t$. Second, no base list $\bar{\boldsymbol{\mathcal{R}}}_t$ contains more incorrectly-ordered item pairs than $\mathcal{R}_0$ with a high probability. In particular, under event $\mathcal{E}$ in Lemma 9, any change in the base list (line 24 of `BubbleRank`) reduces the number of incorrectly-order item pairs by one. In Lemma 9, we prove that $\mathbb{P}(\mathcal{E}) \geq 1 - \delta^{\frac{1}{2}} K^2 n$. □

4.3 DISCUSSION

Our upper bound on the n-step regret of `BubbleRank` (Theorem 1) is $O(\Delta_{\min}^{-1} \log n)$ for $\delta = n^{-4}$. This dependence is considered to be optimal in gap-dependent bounds. Our gap $\Delta_{\min}$ is the minimum difference in the attraction probabilities of items, and reflects the hardness of sorting the items by their attraction probabilities. This sorting problem is equivalent to the problem of learning $\mathcal{R}^*$. So, a gap like $\Delta_{\min}$ is expected, and is the same as that in Zoghi et al. [39].

Our regret bound is notable because it reflects two key characteristics of `BubbleRank`. First, the bound is linear in the number of incorrectly-ordered item pairs in the initial base list $\mathcal{R}_0$. This suggests that `BubbleRank` should have lower regret when initialized with a better list of items. We validate this dependence empirically in Section 5. In many domains, such lists exist and are produced by existing ranking policies. They only need to be safely improved.

Second, the bound is $O(\chi_{\max}\chi_{\min}^{-1})$, where $\chi_{\max}$ and $\chi_{\min}$ are the maximum and minimum examination probabilities, respectively. In Section 5.4, we show that this dependence can be observed in problems where most attractive items are placed at infrequently examined positions. This limitation is intrinsic to `BubbleRank`, because attractive lower-ranked items cannot be placed at higher positions unless they are observed to be attractive at lower, potentially infrequently examined, positions.

The safety constraint of `BubbleRank` is stated in Lemma 2. For $\delta = n^{-4}$, as discussed above, `BubbleRank` becomes a rather safe algorithm, and is unlikely to display any list with more than $K/2$ incorrectly-ordered item pairs than the initial base list $\mathcal{R}_0$. More precisely, $|\mathcal{V}(\boldsymbol{\mathcal{R}}_t)| \leq |\mathcal{V}_0| + K/2$ holds uniformly over time with probability of at least $1 - K^2/n$. This safety feature of `BubbleRank` is confirmed by our experiments in Section 5.3.

The above discussion assumes that the time horizon n is known. However, in practice, this is not always possible. We can extend `BubbleRank` to the setting of an unknown time horizon by using the so-called *doubling trick* [4, Section 2.3]. Let n be the estimated horizon. Then at time $n+1$, $\bar{\mathcal{R}}_{n+1}$ is set to $\mathcal{R}_0$ and n is doubled. The statistics do not need to be reset.

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

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

4.4 PROOF OF THEOREM 1

In Lemma 9, we establish that there exists a favorable event $\mathcal{E}$ that holds with probability $1 - \delta^{\frac{1}{2}} K^2 n$, when all beliefs $\boldsymbol{s}_t(i,j)$ are at most $2\sqrt{\boldsymbol{n}_t(i,j)\log(1/\delta)}$ from their respective means, uniformly for $i < j$ and $t \in [n]$. Since the maximum n-step regret is Kn, we get that

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

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

$$\begin{aligned}\boldsymbol{\mathcal{P}}_t = \Big\{(i,j) \in [K]^2 : i < j, \left|\bar{\boldsymbol{\mathcal{R}}}_t^{-1}(i) - \bar{\boldsymbol{\mathcal{R}}}_t^{-1}(j)\right| = 1, \\ \boldsymbol{s}_{t-1}(i,j) \leq 2\sqrt{\boldsymbol{n}_{t-1}(i,j)\log(1/\delta)}\Big\}\end{aligned}$$

be the set of potentially randomized item pairs at time t. Then, by Lemma 5 on event $\mathcal{E}$, which bounds the regret of list $\boldsymbol{\mathcal{R}}_t$ with the difference in the attraction probabilities of items $(i,j) \in \boldsymbol{\mathcal{P}}_t$, we have that

$$\begin{aligned}&\hat{R}(n) \leq \\ &\quad 3K\chi_{\max}\sum_{i=1}^{K}\sum_{j=i+1}^{K}\sum_{t=1}^{n}(\alpha(i) - \alpha(j))\mathbb{1}\{(i,j) \in \boldsymbol{\mathcal{P}}_t\} \,.\end{aligned}$$

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

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

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

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

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

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

The first inequality is by Lemma 7, which establishes that the maximum difference in clicks of any randomized pair of items is bounded. After that, the better item is found and the pair of items is never randomized again. The last inequality is by $\alpha(i)+\alpha(j) \leq 2$ and $\alpha(i)-\alpha(j) \geq \Delta_{\min}$. Now we chain the above two inequalities and get that

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

Finally, let $\boldsymbol{\mathcal{P}} = \bigcup_{t\in[n]} \boldsymbol{\mathcal{P}}_t$. Then, on event $\mathcal{E}$, $|\boldsymbol{\mathcal{P}}| \leq K - 1 + 2|\mathcal{V}_0|$. This follows from the design of `BubbleRank` (Lemma 6) and completes the proof. □

5 EXPERIMENTAL RESULTS

We conduct four experiments to evaluate `BubbleRank`. In Section 5.1, we describe our experimental setup. In Section 5.2, we report the regret of compared algorithms, which measures the rate of convergence to the optimal list in hindsight. In Section 5.3, we validate the safety of `BubbleRank`. In Section 5.4, we validate the tightness of the regret bound in Theorem 1. Due to space limitations, we report the *Normalized Discounted Cumulative Gain* (NDCG) of compared algorithms, which measures the quality of displayed lists, in Appendix B.

5.1 EXPERIMENTAL SETUP

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

We preprocess the dataset as in [39]. In particular, we randomly select 100 frequent search queries, and then learn the parameters of three click models using the PyClick[3] package: CM and PBM, described in Section 2.1, as well as the *dependent click model (DCM)* [10].

The DCM is an extension of the CM [9] where each position k is associated with an abandonment probability $v(k)$. When the user clicks on an item at position k, the user stops scanning the list with probability $v(k)$. Therefore, the DCM can model multiple clicks. Following the work in [15], we incorporate abandonment into our definition of reward for DCM and define it as the number of abandonment clicks. The *abandonment click* is a click after which a user stops browsing the list, and each time step contains at most one abandonment click. So, the expected reward for DCM equals the probability of abandonment clicks, which is computed as follows:

$$\begin{aligned}r(\mathcal{R},\alpha,\chi) = \sum_{k=1}^{K}\chi(\mathcal{R},k)v(k)\alpha(\mathcal{R}(k)), \\ \chi(\mathcal{R},k) = \prod_{i=1}^{k-1}(1-v(i))\alpha(\mathcal{R}(i))\end{aligned}$$

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

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

For each query, we choose 10 items. The number of positions is equal to the number of items, $K = L = 10$. The objective of our re-ranking problem is to place 5 most attractive items in the descending order of their attractiveness at the 5 highest positions, as in [39]. The performance of `BubbleRank` and our baselines is also measured only at the top 5 positions.

`BubbleRank` is compared to three baselines `CascadeKL-UCB` [17], `BatchRank` [39], and `TopRank` [20]. The former is near optimal in the CM [17], but can have linear

[2] https://academy.yandex.ru/events/data_analysis/relpred2011

[3] https://github.com/markovi/PyClick

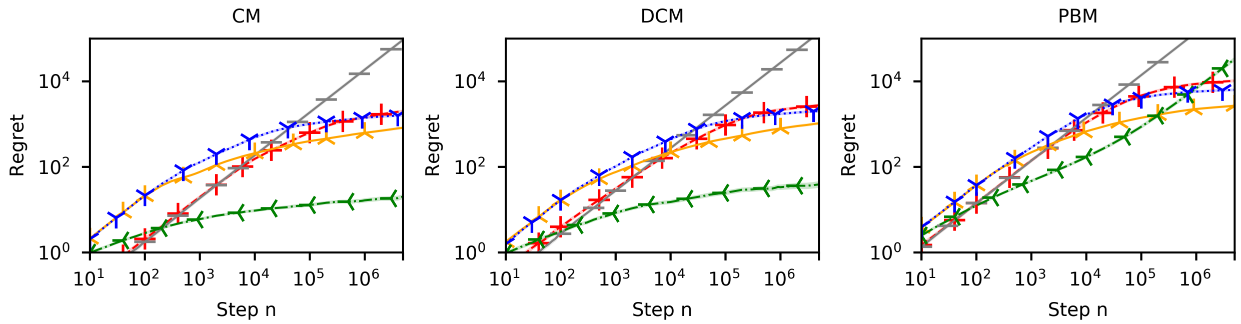

Figure 1: The n-step regret of `BubbleRank` (red), `CascadeKL-UCB` (green), `BatchRank` (blue), `TopRank` (orange), and `Baseline` (grey) in the CM, DCM, and PBM in up to 5 million steps. Lower is better. The results are averaged over all 100 queries and 10 runs per query. The shaded regions represent standard errors of our estimates.

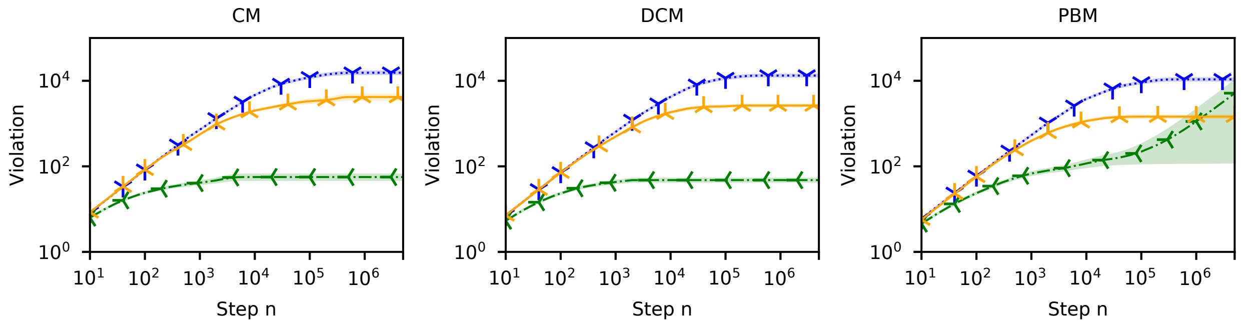

Figure 2: The n-step violation of the safety constraint of `BubbleRank` by `CascadeKL-UCB` (green), `BatchRank` (blue), and `TopRank` (orange) in the CM, DCM, and PBM in up to 5 million steps. Lower is better. The shaded regions represent standard errors of our estimates.

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

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

5.2 RESULTS WITH REGRET

In the first experiment, we compare `BubbleRank` to `CascadeKL-UCB`, `BatchRank`, and `TopRank` in the CM, DCM, and PBM of all 100 queries. Among them, `TopRank` is the state-of-the-art online LTR algorithm in multiple click models. We evaluate these algorithms by their cumulative regret, which is defined in (4), at the top 5 positions. The regret, a measure of convergence, is a widely-used metric in the bandit literature [3, 15, 17, 39]. In the CM and PBM, the regret is the cumulative loss in clicks when a sequence of learned lists is compared to the optimal list in hindsight. In the DCM, the regret is the cumulative loss in abandonment clicks. We also report the regret of `Baseline`.

Our results are reported in Figure 1. We observe that the regret of `Baseline` grows linearly with time n, which means that it is not optimal on average. `CascadeKL-UCB` learns $\mathcal{R}^*$ quickly in both the CM and DCM, but has linear regret in the PBM. This is expected since `CascadeKL-UCB` is designed for the CM, and the DCM is an extension of the CM. As for the PBM, which is beyond the modeling assumptions of `CascadeKL-UCB`, there is no guarantee on the performance of `CascadeKL-UCB`. `BubbleRank`, `BatchRank`, and `TopRank` can learn in all three click models. Compared to `BatchRank` and `TopRank`, `BubbleRank` has a higher regret in 5 million steps. However, in earlier steps, `BubbleRank` has a lower regret than `BatchRank` and `TopRank`, as it takes advantage of the initial base list $\mathcal{R}_0$. In general, these results show that BubbleRank converges to the optimal list slower than BatchRank and TopRank. This is expected because BubbleRank is designed to be a safe algorithm, and only learns better lists by exchanging neighboring items in the base list.

5.3 SAFETY RESULTS

In the previous experiment, we show that BubbleRank does not learn as fast as CascadeKL-UCB, BatchRank, and TopRank because it operates under the additional safety constraint in Lemma 2. The constraint is that BubbleRank is unlikely to display any list with more than $K/2$ incorrectly-ordered item pairs than the initial base list $\mathcal{R}_0$. More precisely, $|\mathcal{V}(\boldsymbol{\mathcal{R}}_t)| \leq |\mathcal{V}(\mathcal{R}_0)| + K/2$ holds uniformly over time with probability of at least $1 - K^2/n$ for $\delta = n^{-4}$ (Section 4.3), where $\mathcal{V}$ is defined in (5). In the second experiment, we answer the question how often do BubbleRank, CascadeKL-UCB, BatchRank, and TopRank violate this constraint empirically. We define the *safety constraint violation in n steps* as

$$V(n) = \sum_{t=1}^{n} \mathbb{1}\{|\mathcal{V}(\boldsymbol{\mathcal{R}}_t)| > |\mathcal{V}(\mathcal{R}_0)| + K/2\}\,, \qquad (7)$$

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

We report the n-step safety constraint violation of CascadeKL-UCB, BatchRank, and TopRank in Figure 2. We do not include results of BubbleRank since BubbleRank never violates the constraint in our experiments. We observe that the safety constraint violations of CascadeKL-UCB in the first 100 steps are 24.12 ± 0.76, 23.33 ± 0.90, and 23.63 ± 0.96 in the CM, DCM, and PBM, respectively. Translating this to a search scenario, CascadeKL-UCB may show unsafe results, which are significantly worse than the initial base list $\mathcal{R}_0$, and may hurt user experience, more than 20% of search sessions in the first 100 steps. Even worse, the violations of CascadeKL-UCB grow linearly with time in the PBM. The safety issues of BatchRank, and TopRank are more severe than that of CascadeKL-UCB. More precisely, the violations of BatchRank in the first 100 steps are 83.01 ± 0.56, 71.89 ± 0.92, and 59.63 ± 0.98 in the CM, DCM, and PBM, respectively. And the violations of TopRank are 83.56 ± 0.70, 71.47 ± 1.07, and 57.00 ± 1.24 in the CM, DCM, and PBM, respectively. Note that the performance of TopRank is close to that of BatchRank in the first 100 steps since they both require the ranked lists to be randomly shuffled during the initial stages. Thus, BatchRank, and TopRank would frequently hurt the user experience during the early stages of learning.

To conclude, BubbleRank learns without violating its safety constraint, while CascadeKL-UCB, BatchRank, and TopRank violate the constraint frequently. Together with results in Section 5.2, BubbleRank is a safe algorithm but, to satisfy the safety constraint, it compromises the performance and learns slower than BatchRank and TopRank. In Appendix B, we compare BubbleRank to baselines in NDCG and show that BubbleRank converges to the optimal lists in hindsight.

5.4 SANITY CHECK ON THE REGRET BOUND

We prove an upper bound on the n-step regret of BubbleRank in Theorem 1. In comparison to the upper bounds of CascadeKL-UCB [17] and BatchRank [39], we have two new problem-specific constants: $|\mathcal{V}_0|$ and $1/\chi_{\min}$. In this section, we show that these constants are intrinsic to the behavior of BubbleRank.

We first study how the number of incorrectly-ordered item pairs in the initial base list $\mathcal{R}_0$, $|\mathcal{V}_0|$, impacts the regret of BubbleRank. We choose 10 random initial base lists $\mathcal{V}_0$ in each of our 100 queries and plot the regret of BubbleRank as a function of $|\mathcal{V}_0|$. Our results are shown in Figure 3. We observe that the regret of BubbleRank is linear in $|\mathcal{V}_0|$ in the CM, DCM, and PBM. This is the same dependence as in our regret bound (Theorem 1).

We then study the impact of the minimum examination probability $\chi_{\min}$ on the regret of BubbleRank. We experiment with a synthetic PBM with 10 items, which is parameterized by $\alpha = (0.9, 0.5, \ldots, 0.5)$ and $\chi = (0.9, \ldots, 0.9, 0.5^i, 0.5^i)$ for $i \geq 1$. The most attractive item is placed at the last position in $\mathcal{R}_0$, $\mathcal{R}_0 = (2, \ldots, K-1, 1)$. Since this position is examined with probability 0.5^i, we expect the regret to double when i increases by one. We experiment with $i \in [5]$ in Figure 3 and observe this trend in 1 million steps. This confirms that the dependence on $1/\chi_{\min}$ in Theorem 1 is generally unavoidable.

6 RELATED WORK

Online LTR via click feedback has been mainly studied in two approaches: under specific click models [7, 15, 17, 18, 19, 40]; or without a particular assumption on click models [20, 28, 30, 39]. Algorithms from the first group efficiently learn optimal rankings in the their considered click models but do not have guarantees beyond their specific click models. Algorithms from the second group, on the other hand, learn the optimal rankings in a broader class of click models. TopRank [20] is the state-of-the-art of the second group, which has the regret of $O(K^2 \log(n))$ in our re-ranking setup, that is $L = K$. BubbleRank also belongs to the second group and the regret of BubbleRank is comparable to that of TopRank given a good initial list, when $|\mathcal{V}_0| = O(K)$. However,

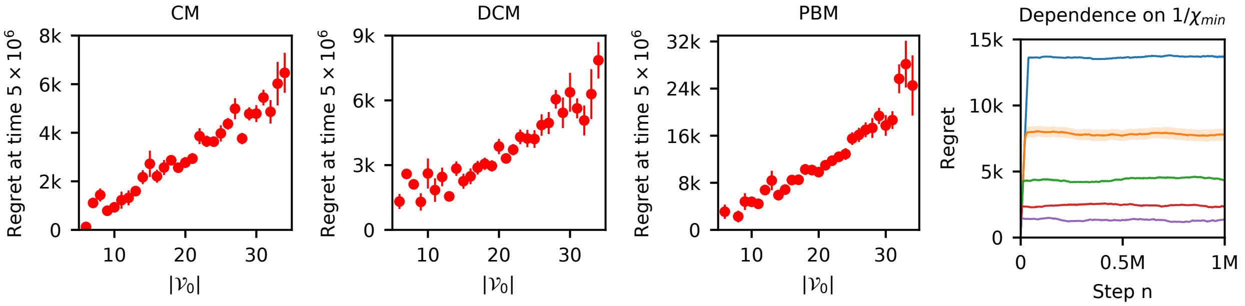

Figure 3: Regret of BubbleRank as a function of the number of incorrectly-ordered item pairs $|\mathcal{V}_0|$, and the minimal examination probability $\chi_{\min}$. In the right plot, the purple, red, green, orange, and blue colors represent $\chi_{\min}$ equals 0.5, 0.5^2, 0.5^3, 0.5^4, and 0.5^5,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

unlike BubbleRank, TopRank and all the previous algorithms do not consider safety. They explore aggressively in the initial steps and may display irrelevant items at high positions, which may then hurt user experiences [34].

Our safety problem is related to the warm start problem [31]. Contextual bandits [1, 21, 25] deal with a broader class of models than we do and are used to address the warm start problem. But they are limited to small action sets, and thus unsuitable for the ranking setup that we consider in this paper.

The warm start LTR has been studied in multiple papers [11, 33, 36], where the goal is to use an online algorithm to fine tune the results generated by an offline-trained ranker. In these papers, different methods for learning prior distributions of Thompson sampling based online LTR algorithms from offline datasets have been proposed. However, these methods have the following drawbacks. First, the offline data may not well align with user preferences [38], which may result in a biased prior assumption. Second, grid search with online A/B tests may alleviate this and find a proper prior assumption [33], but the online A/B test requires additional costs in terms of user experience. Third, there is no safety constraint in these methods. Even with carefully picked priors, they may recommend irrelevant items to users, e.g., new items with little prior knowledge. In contrast, BubbleRank starts from the production ranked list and learns under the safety constraint. Thus, BubbleRank gets rid of these drawbacks.

Another related line of work are conservative bandits [16, 35]. In conservative bandits, the learned policy is safe in the sense that its expected *cumulative* reward is at least $1 - \alpha$ fraction of that of the baseline policy with high probability. This notion of safety is less stringent than that in our work (Section 4.2). In particular, our notion of safety is *per-step*, in the sense that any displayed list is only slightly worse than the initial base list with a high probability. We do not compare to conservative bandits in our experiments because existing algorithms for conservative bandits require the action space to be small. The actions in our problem are ranked lists, and their number is exponential in K.

7 CONCLUSIONS

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

We leave open several questions of interest. For instance, our paper studies BubbleRank in the setting of re-ranking. Although we explain an approach of extending BubbleRank to the general ranking setup in Section 4.3, we expect further experiments to validate this approach. Our general topic of interest are exploration schemes that are more conservative than those of existing online LTR methods. Existing methods are not very practical because they can explore highly irrelevant items at frequently examined positions.

ACKNOWLEDGMENTS

We thank our reviewers for helpful feedback and suggestions. This research was supported by Ahold Delhaize, the Association of Universities in the Netherlands (VSNU), the Innovation Center for Artificial Intelligence (ICAI), and the Netherlands Organisation for Scientific Research (NWO) under project nr. 612.001.551. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

References

[1] A. Agarwal, D. Hsu, S. Kale, J. Langford, L. Li, and R. Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *ICML*, pages II–1638–II–1646, 2014.

[2] J. Allan, D. Harman, E. Kanoulas, D. Li, C. Van Gysel, and E. Voorhees. TREC 2017 common core track overview. TREC, 2017.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[4] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[5] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. Efficient cost-aware cascade ranking in multi-stage retrieval. In *SIGIR*, pages 445–454, 2017.

[6] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Morgan & Claypool, 2015.

[7] R. Combes, S. Magureanu, A. Proutiere, and C. Laroche. Learning to rank: Regret lower bounds and efficient algorithms. In *SIGMETRICS*, pages 231–244, 2015.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[9] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94, 2008.

[10] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM*, pages 124–131, 2009.

[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*, 2013.

[12] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[13] T. Joachims. Optimizing search engines using clickthrough data. *SIGKDD*, pages 133–142, 2002.

[14] S. K. Karmaker Santu, P. Sondhi, and C. Zhai. On application of learning to rank for e-commerce search. In *SIGIR*, pages 475–484, New York, NY, USA, 2017.

[15] S. Katariya, B. Kveton, C. Szepesvari, and Z. Wen. DCM bandits: Learning to rank with multiple clicks. In *ICML*, pages 1215–1224, 2016.

[16] A. Kazerouni, M. Ghavamzadeh, Y. Abbasi, and B. Van Roy. Conservative contextual linear bandits. In *NIPS*, pages 3913–3922, 2017.

[17] B. Kveton, C. Szepesvari, Z. Wen, and A. Ashkan. Cascading bandits: Learning to rank in the cascade model. In *ICML-15*, pages 767–776, 2015.

[18] B. Kveton, Z. Wen, A. Ashkan, and C. Szepesvari. Combinatorial cascading bandits. In *NIPS*, pages 1450–1458, 2015.

[19] P. Lagrée, C. Vernade, and O. Cappe. Multiple-play bandits in the position-based model. In *NIPS*, pages 1605–1613, 2016.

[20] T. Lattimore, B. Kveton, S. Li, and C. Szepesvari. Toprank: A practical algorithm for online stochastic ranking. In *NIPS*, pages 3945–3954, 2018.

[21] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, pages 661–670, 2010.

[22] S. Liu, F. Xiao, W. Ou, and L. Si. Cascade ranking for operational e-commerce search. *arXiv preprint arXiv:1706.02093*, 2017.

[23] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3 (3):225–331, 2009.

[24] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *SIGKDD*, pages 1222–1230, 2013.

[25] T. Moon, L. Li, W. Chu, C. Liao, Z. Zheng, and Y. Chang. Online learning for recency search ranking using real-time user feedback. In *CIKM*, pages 1501–1504, 2010.

[26] T. Qin, T.-Y. Liu, J. Xu, and H. Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4): 346–374, 2010.

[27] F. Radlinski and T. Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *AAAI*, pages 1406–1412, 2006.

[28] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML*, pages 784–791, 2008.

[29] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: Estimating the click-through rate for new ads. In *WWW*, pages 521–530, 2007.

[30] A. Slivkins, F. Radlinski, and S. Gollapudi. Ranked bandits in metric spaces: Learning diverse rankings

over large document collections. *Journal of Machine Learning Research*, 14(1):399–436, 2013.

[31] A. Strehl, J. Langford, L. Li, and S. M. Kakade. Learning from logged implicit exploration data. In *NIPS*, pages 2217–2225, 2010.

[32] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285–294, 1933.

[33] A. Vorobev, D. Lefortier, G. Gusev, and P. Serdyukov. Gathering additional feedback on search results by multi-armed bandits with respect to production ranking. In *WWW*, pages 1177–1187, 2015.

[34] X. Wang, N. Golbandi, M. Bendersky, D. Metzler, and M. Najork. Position bias estimation for unbiased learning to rank in personal search. In *WSDM*, pages 610–618, 2018.

[35] Y. Wu, R. Shariff, T. Lattimore, and C. Szepesvári. Conservative bandits. In *ICML*, pages 1254–1262, 2016.

[36] Y. Yan, Z. Liu, M. Zhao, W. Guo, W. P. Yan, and Y. Bao. A practical deep online ranking system in e-commerce recommendation. In *ECML PKDD*, pages 186–201, 2018.

[37] Y. Yue and C. Guestrin. Linear submodular bandits and their application to diversified retrieval. In *NIPS*, pages 2483–2491, 2011.

[38] M. Zoghi, T. Tunys, L. Li, D. Jose, J. Chen, C. M. Chin, and M. de Rijke. Click-based hot fixes for underperforming torso queries. In *ACM SIGIR*, pages 195–204. ACM, 2016.

[39] M. Zoghi, T. Tunys, M. Ghavamzadeh, B. Kveton, C. Szepesvari, and Z. Wen. Online learning to rank in stochastic click models. In *ICML*, pages 4199–4208, 2017.

[40] S. Zong, H. Ni, K. Sung, N. R. Ke, Z. Wen, and B. Kveton. Cascading bandits for large-scale recommendation problems. In *UAI*, pages 835–844, 2016.