MergeDTS: A Method for Effective Large-Scale Online Ranker Evaluation

CHANG LI and ILYA MARKOV, University of Amsterdam
MAARTEN DE RIJKE, University of Amsterdam and Ahold Delhaize
MASROUR ZOGHI, Microsoft

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

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

CCS Concepts: • **Information systems** → **Evaluation of retrieval results**;

Additional Key Words and Phrases: Online evaluation, implicit feedback, preference learning, dueling bandits

ACM Reference format:
Chang Li, Ilya Markov, Maarten de Rijke, and Masrour Zoghi. 2020. MergeDTS: A Method for Effective Large-Scale Online Ranker Evaluation. *ACM Trans. Inf. Syst.* 38, 4, Article 40 (September 2020), 28 pages.
https://doi.org/10.1145/3411753

M. Zoghi is now at Google.
This research was partially supported by the Innovation Center for Artificial Intelligence (ICAI), as well as the Netherlands Organisation for Scientific Research (NWO) under project no. 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.
Authors' addresses: C. Li (corresponding author) and I. Markov, University of Amsterdam, Science Park 904, 1098 XH, Amsterdam, The Netherlands; emails: c.li@uva.nl, i.markov@uva.nl; M. de Rijke, University of Amsterdam, Science Park 904, 1098 XH, Amsterdam, The Netherlands, and Ahold Delhaize, Provincialeweg 11, 1506 MA, Zaandam, The Netherlands; email: m.derijke@uva.nl; M. Zoghi, Microsoft, Redmond, WA; email: masrour@zoghi.org.

1046-8188/2020/09-ART40 $15.00
https://doi.org/10.1145/3411753

1 INTRODUCTION

Online ranker evaluation concerns the task of determining the ranker with the best performance out of a finite set of rankers. It is an important challenge for information retrieval systems [26, 36, 37]. In the absence of an oracle judge who can tell the preferences between all rankers, the best ranker is usually inferred from user feedback on the result lists produced by the rankers [20]. Since user feedback is known to be noisy [17, 28, 29, 38], how to infer ranker quality and when to stop evaluating a ranker are two important challenges in online ranker evaluation.

The former challenge, i.e., how to infer the quality of a ranker, is normally addressed by *interleaving* methods [11, 13, 22–24]. Specifically, an interleaving method interleaves the result lists generated by two rankers for a given query and presents the interleaved list to the user. Then it infers the preferred ranker based on the user's click feedback. As click feedback is noisy, the interleaved comparison of two rankers has to be repeated many times so as to arrive at a reliable outcome of the comparison.

Although interleaving methods address the first challenge of online ranker evaluation (how to infer the quality of a ranker), they give rise to another challenge—for instance, which rankers to compare and when to stop the comparisons. Without enough comparisons, we may mistakingly infer the wrong ranker preferences. But with too many comparisons, we may degrade the user experience as we continue to show results from suboptimal rankers. Based on previous work [5, 54, 56], the challenge of choosing and comparing rankers can be formalized as a K-armed dueling bandit problem [48], which is an important variant of the multi-armed bandit (MAB) problem, where feedback is given in the form of pairwise preferences. In the K-armed dueling bandit problem, a ranker is defined as an arm and the best ranker is the arm that has the highest expectation to win the interleaving game against other candidates.

Several dueling bandit algorithms have been proposed (cf. Busa-Fekete et al. [8], Sui et al. [44] and Zoghi [52] for an overview. However, the study of these algorithms has mostly been limited to small-scale dueling bandit problems, with the state-of-the-art being Double Thompson Sampling (DTS) [47]. By "small scale," we mean that the number of arms being compared is small. But, in real-world online ranker evaluation problems, experiments involving hundreds or even thousands of rankers are commonplace [30]. Despite this fact, to the best of our knowledge, the only work that address this particular scalability issue is Merge Relative Upper Confidence Bound (MergeRUCB) [54]. As we demonstrate in this work, the performance of MergeRUCB can be improved upon substantially.

Here, we propose and evaluate a novel algorithm, named *Merge Double Thompson Sampling (MergeDTS)*. The main idea of MergeDTS is to combine the benefits of MergeRUCB, which is the state-of-the-art algorithm for large-scale dueling bandit problems, and the benefits of DTS, which is the state-of-the-art algorithm for small-scale problems, and attain improvements in terms of effectiveness (as measured in terms of regret) and efficiency (as measured in terms of time complexity). More specifically, what we borrow from MergeRUCB is the divide-and-conquer idea used to group rankers into small batches to avoid global comparisons. However, from DTS, we import the idea of using Thompson Sampling (TS) [45], rather than using uniform randomness as in MergeRUCB, to choose the arms to be played.

We analyze the performance of MergeDTS, and demonstrate that the soundness of MergeDTS can be guaranteed if the timestep T is known and the exploration parameter $\alpha > 0.5$ (Theorem 4.1). Finally, we conduct extensive experiments to evaluate the performance of MergeDTS in the scenario of online ranker evaluation on three widely used real-world datasets: Microsoft, Yahoo! Learning to Rank, and ISTELLA [10, 14, 35, 39]. We show that with tuned parameters MergeDTS outperforms MergeRUCB and DTS in large-scale online ranker evaluation under the

Condorcet assumption—that is, where there is a ranker preferred to all other rankers.[1] Moreover, we demonstrate the potential of using MergeDTS beyond the Condorcet assumption—that is, where there might be multiple best rankers.

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

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

The rest of the article is organized as follows. In Section 2, we detail the definition of the dueling bandit problem. We discuss prior work in Section 3. MergeDTS is proposed in Section 4. Our experimental setup is detailed in Section 5, and the results are presented in Section 6. We conclude in Section 7.

2 PROBLEM SETTING

In this section, we first describe in more precise terms the K-armed dueling bandit problem, which is a variation of the MAB problem. The latter can be described as follows: given K choices, called *arms* and denoted by $a_1, \ldots, a_K$, we are required to choose one arm at each step; choosing arm a_i generates a reward that is drawn i.i.d. from a random variable with mean, denoted by μ_i, and our goal is to maximize the expected total reward accumulated by our choices of arms over time. This objective is more commonly formulated in terms of the *cumulative regret* of the MAB algorithm, where regret at step t is the difference between the reward of the chosen arm, e.g., a_j, and the reward of the best arm, e.g., a_k, in hindsight, and the average regret of arm a_j is defined to be $\mu_k - \mu_j$: cumulative regret is defined to be the sum of the instantaneous regret over time [2, 33].

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

In a similar fashion to the MAB setting, we evaluate a dueling bandit algorithm based on its *cumulative regret*, which is the total regret incurred by choosing suboptimal arms comparing to the best arm over time [8, 52]. However, the definition of regret is less clear cut in the dueling bandit setting, because our dueling bandit problem might not contain a clear winner that is preferred to all other arms, i.e., an arm a_C, called the *Condorcet winner*, such that $p_{Cj} > 0.5$ for all $j \neq C$. There are numerous proposals in the literature for alternative notions of winners in the absence of a Condorcet winner, such as the Borda winner [27, 46], Copeland winner [32, 53], and von

[1]Our theoretical analysis is rather conservative, and the regret bound only holds for the parameter values within a certain range. This is because our bound is proven using Chernoff-Hoeffding bound [19] together with the union bound [9], both of which, in our case, introduce gaps between theory and practice. In our experiments, we show that the parameter values outside of the theoretical regime can boost the performance of MergeDTS and that of the baselines. Thus, our experimental results of MergeDTS are not restricted to the parameter values within the theoretical regime.

Neumann winner [15], with each definition having its own disadvantages and practical settings where its use is appropriate.

MergeDTS, like most of the other dueling bandits algorithms [44, 46, 54–56], relies on the existence of a Condorcet winner, in which case the Condorcet winner is the clear choice for the best arm, since it is preferred to all other arms, and with respect to which regret can be defined. We pose, as an interesting direction for future work, the task of extending the method proposed in this work to each of the other notions of winner listed earlier.

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

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

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

$$\mathcal{R}(T) = \sum_{t=1}^{T} r_t, \tag{2}$$

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

Let us translate the online ranker evaluation problem into the dueling bandit problem. The input, a finite set of arms, consists of a set of rankers, for example, based on different ranking models or based on the same model but with different parameters [30]. The Condorcet winner is the ranker that is preferred, by the majority of users, over suboptimal rankers. More specifically, a result list from the Condorcet winner is expected to receive the highest number of clicks from users when compared to a list from a suboptimal ranker. The preference matrix **P** records the users' relative preferences for all rankers. Regret measures the user frustration incurred by showing the interleaved list from suboptimal rankers instead of the Condorcet winner. In the rest of the article, we use the term *ranker* to indicate the term *arm* in K-armed dueling bandit problems since we focus on the online ranker evaluation task.

3 RELATED WORK

There are two main existing approaches for solving dueling bandit problems: (1) reducing the problem to a MAB problem, such as Sparring [1], Self-Sparring [43], and tje Relative Exponential-Weight Algorithm for Exporation and Exploitation (REX3) [16], and (2) generalizing existing MAB algorithms to the dueling bandit setting,such as Relative Upper Confidence Bound (RUCB) [55], Relative Minimum Empirical Divergence (RMED1) [31], and DTS [47]. The advantage of the latter group of algorithms is that they come equipped with theoretical guarantees, proven for a broad class of problems. The first group, however, has guarantees that either only hold for a restricted class of problems, where the dueling bandit problem is obtained by comparing the arms of an underlying MAB problem (a.k.a. utility-based dueling bandits), as in the case of Self-Sparring, REX3, and Sparring T-INF [51], or have substantially suboptimal instance-dependent regret bounds, as in the case of Sparring EXP3, which has a regret bound of the form $O(\sqrt{KT})$, as opposed to $O(K \log T)$.

Indeed, as our following experimental results demonstrate, Sparring-type algorithms can perform poorly when the dueling bandit problem does not arise from an MAB problem.

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

Self-Sparring [43] improves upon Sparring by employing a single MAB algorithm but at each step samples twice to choose rankers. More precisely, Sui et al. [43] use TS as the MAB algorithm. Self-Sparring assumes that the problem it solves arises from an MAB; it can perform poorly when there exists a cycle relation in rankers, i.e., if there are rankers a_i, a_j and a_k with $p_{ij} > 0.5, p_{jk} > 0.5$ and $p_{ki} > 0.5$. As Self-Sparring does not estimate confidence intervals of the comparison results, it does not eliminate rankers.

Another extension of Sparring is REX3 [16], which is designed for the adversarial setting. REX3 is inspired by the Exponential-Weight Algorithm for Exploration and Exploitation (EXP3) [3], an algorithm for adversarial bandits, and has a regret bound of the form $O(\sqrt{K \ln (K) T})$. Note that the regret bound grows as the square root of timesteps, but sublinearly in the number of rankers, which shows the potential for improvement in the case of large-scale problems.

RUCB [55] extends UCB to dueling bandits using a matrix of optimistic estimates of the relative preference probabilities. At each step, RUCB chooses the first ranker to be one that beats all other rankers based on the idea of *optimism in the face of uncertainty*. Then it chooses the second ranker to be the ranker that beats the first ranker with the same idea of optimism in the face of uncertainty, which translates to pessimism for the first ranker. The cumulative regret of RUCB after T steps is upper bounded by an expression of the form $O(K^2 + K \log T)$.

RMED1 [31] extends an asymptotically optimal MAB algorithm, called *Deterministic Minimum Empirical divergence* (DMED) [11], by first proving an asymptotic lower bound on the cumulative regret of all dueling bandit algorithms, which has the order of $\Omega(K \log T)$, and pulling each pair of rankers the minimum number of times prescribed by the lower bound. RMED1 outperforms RUCB and Sparring.

DTS [47] improves upon RUCB by using TS to break ties when choosing the first ranker. Specifically, it uses one TS to choose the first ranker from a set of candidates that are pre-chosen by UCB. Then it uses another TS to choose the second ranker that performs the best compared to the first one. The cumulative regret of DTS is upper bounded by $O(K \log T + K^2 \log \log T)$. Note that the bound of DTS is higher than that of RUCB. We hypothesize that this is because the bound of DTS is rather loose. DTS outperforms other dueling bandits algorithms empirically and is the state of the art in the case of small-scale dueling bandit problems [43, 47]. As discussed in Section 5, for computational reasons, DTS is not suitable for large-scale problems.

The work that is the closest to ours is by Zoghi et al. [54]. They propose MergeRUCB, which is the state of the art for large-scale dueling bandit problems. MergeRUCB partitions rankers into small batches and compares rankers within each batch. A ranker is eliminated from a batch once we realize that even according to the most optimistic estimate of the preference probabilities it loses to another ranker in the batch. Once enough rankers have been eliminated, MergeRUCB repartitions the remaining rankers and continues as before. Importantly, MergeRUCB does not require global pairwise comparisons between all pairs of rankers, so it reduces the computational complexity and increases the time efficiency, as shown in Section 6.2. The cumulative regret of MergeRUCB can be upper bounded by $O(K \log T)$ [54], i.e., with no quadratic dependence on the number of rankers. This upper bound has the same order as the lower bound proposed by Komiyama et al. [31] in terms of $K \log T$, but it is not optimal in the sense that it has large constant coefficients. As we demonstrate in our experiments, MergeRUCB can be improved by making use of TS to reduce the amount of randomness in the choice of rankers. More precisely, the cumulative regret of MergeRUCB is almost twice as large as that of MergeDTS in the large-scale setup shown in Section 6.

A recent extension of dueling bandits is called *multi-dueling bandits* [5, 40, 43], where more than two rankers can be compared at each step. The Multi-Dueling Bandit (MDB) algorithm [5] is the first proposed algorithm in this setting, which is specifically designed for online ranker evaluation. It maintains two UCB estimators for each pair of rankers, a looser confidence bound and a tighter one. At each step, if there is more than one ranker that is valid for the tighter UCB estimators, MDB compares all of the rankers that are valid for the looser UCB estimators. MDB is outperformed by Self-Sparring, the state-of-the-art *multi-dueling bandit* algorithm, significantly [43]. In this article, we do not focus on the *multi-dueling bandit* setup. The reasons are twofold. First, to the best of our knowledge, there are no theoretical results in the multi-dueling setting that allow for the presence of cyclical preference relationships among the rankers. Second, Saha and Gopalan [40] state that "(perhaps surprisingly) [...] the flexibility of playing size-k subsets does not really help to gather information faster than the corresponding dueling case ($k = 2$), at least for the current subset-wise feedback choice model." This statement demonstrates that there is no clear advantage to using multi-dueling comparisons over pairwise dueling comparisons at this moment.

4 MERGE DOUBLE THOMPSON SAMPLING

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

4.1 The MergeDTS Algorithm

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

The pseudo-code for MergeDTS is provided in Algorithms 1 through 3, with the notation summarized in Table 1 for the reader's convenience. The input parameters are the exploration parameter α, the size of a batch M, and the failure probability $\epsilon \in (0, 1)$. The algorithm records the outcomes of the past comparisons in matrix $\mathbf{W}$, whose element w_{ij} is the number of times ranker a_i has beaten ranker a_j so far. MergeDTS stops when only one ranker remains and then returns that ranker, which it claims to be the Condorcet winner.[2]

MergeDTS begins by grouping rankers into small batches (line 4). At each timestep, MergeDTS checks whether there is more than one ranker remaining (line 7). If so, MergeDTS returns that ranker, the potential Condorcet winner. If not, MergeDTS considers one batch B_m and, using optimistic estimates of the preference probabilities (line 10), it purges any ranker that loses to another ranker even with an optimistic boost in favor of the former (line 11).

If, as a result of the preceding purge, B_m becomes a single-element batch, it is merged with the next batch B_{m+1} (line 13). Here, $m + 1$ is interpreted as modulo b_s, where b_s is the number of batches in the current stage. This is done to avoid comparing a suboptimal ranker against itself, since if there is more than one batch, the best ranker in any given batch is unlikely to be the Condorcet winner of the whole dueling bandit problem. As we will see again in the following, MergeDTS takes great care to avoid comparing suboptimal rankers against themselves because it results in added regret, but it yields no extra information, since we know that each ranker is tied with itself.

After the preceding elimination step, the algorithm proceeds in four phases: choosing the first ranker (Phase I), choosing the second ranker based on the first ranker (Phase II), comparing the two rankers and updating the statistics (Phase III), and repartitioning the rankers at the end of each stage (Phase IV). Of the four phases, Phase I and Phase II are the major reasons that lead to

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

Table 1. Notations Used in This Article

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

a boost in effectiveness of MergeDTS when compared to MergeRUCB. We will elaborate on both phases in the remainder of this section.

In Phase I, the method *SampleTournament* (Algorithm 2) chooses the first candidate ranker: MergeDTS samples preference probabilities θ_{ij} from the posterior distributions to estimate the true preference probabilities p_{ij} for all pairs of rankers in the batch B_m (lines 1–4, the first TS). Based on these sampled probabilities, MergeDTS chooses the first candidate a_c so that it beats most of the other rankers according to the sampled preferences (line 6).

In Phase II, the method *RelativeTournament* (Algorithm 3) chooses the second candidate ranker: MergeDTS samples another set of preference probabilities ϕ_j from the posteriors of p_{jc} for all rankers a_j in $B_m \setminus \{a_c\}$ (lines 1–3, the second TS). Moreover, we set ϕ_c to be 1 (line 4). This is done to avoid self-comparisons between suboptimal rankers for the reasons described earlier.

Once the probabilities ϕ_j have been sampled, we choose the ranker a_d that is going to be compared against a_c, using the following strategy. The worst ranker according to the sampled probabilities ϕ_j is chosen as the second candidate a_d (line 5). The rationale for this discrepancy is that we would like to eliminate rankers as quickly as possible, so rather than using the upper confidence bounds to explore when choosing a_d, we use the lower confidence bounds to knock the weakest link out of the batch as quickly as possible.

In Phase III (line 17) of Algorithm 1, MergeDTS plays a_c and a_d and updates the comparison matrix $\mathbf{W}$ based on the observed feedback.

Finally, in Phase IV (lines 18–22), if the number of remaining rankers in the current stage is half of the rankers of the previous stage (line 18), MergeDTS enters the next stage, before which it repartitions the rankers. Following the design of MergeRUCB, this is done by merging batches of

ALGORITHM 1: MergeDTS

```
Input: $K$ rankers $a_1, a_2, \ldots, a_K$; partition size $M$; exploration parameter $\alpha > 0.5$; running time steps $T$; probability of failure $\epsilon = 1/T$.
Output: The Condorcet winner.
1: $\mathbf{W} \leftarrow \mathbf{0}_{K,K}$                                   // The comparison matrix
2: $C(\epsilon) = \left(\frac{(4\alpha-1)K^2}{(2\alpha-1)\epsilon}\right)^{\frac{1}{2\alpha-1}}$
3: $s = 1$                                                          // The stage of the algorithm
4: $\mathcal{B}_s = \left\{\underbrace{[a_1, \ldots, a_M]}_{B_1}, \ldots, \underbrace{[a_{(b_1-1)M+1}, \ldots, a_K]}_{B_{b_1}}\right\}$   // Disjoint batches of rankers, with $b_1 = \lceil \frac{K}{M} \rceil$
5: for $t = 1, 2, \ldots T$ do
6:     $m = t \mod b_s$                                             // Index of the batches
7:     if $b_s = 1$ and $|B_m| = 1$ then                              // One ranker left
8:         Return the remaining ranker $a \in B_m$.
9:     end if
10:    $\mathbf{U} = \frac{\mathbf{W}}{\mathbf{W}+\mathbf{W}^T} + \sqrt{\left(\frac{\alpha \log(t+C(\epsilon))}{\mathbf{W}+\mathbf{W}^T}\right)}$   // UCB estimators: operations are element-wise and $\frac{x}{0} := 1$
11:    Remove $a_i$ from $B_m$ if $u_{ij} < 0.5$ for any $a_j \in B_m$.
12:    if $b_s > 1$ and $|B_m| = 1$ then
13:        Merge $B_m$ with the next batch and decrement $b_s$.
14:    end if
   // Phase I: Choose the first candidate $a_c$
15:    $a_c$ = SampleTournament($\mathbf{W}$, $B_m$)                       // See Algorithm 2
   // Phase II: Choose the second candidate $a_d$
16:    $a_d$ = RelativeTournament($\mathbf{W}$, $B_m$, $a_c$)                 // See Algorithm 3
   // Phase III: Compare candidates and update batches
17:    Compare pair $(a_c, a_d)$ and increment $w_{cd}$ if $a_c$ wins otherwise increment $w_{dc}$.
    // Phase IV: Update batch set
18:    if $\sum_m |B_m| \leq \frac{K}{2^s}$ then
19:        Pair the larger size batches with the smaller ones, making sure the size of every batch is in $[0.5M, 1.5M]$.
20:        $s = s + 1$
21:        Update $\mathcal{B}_s$, $b_s = |\mathcal{B}_s|$.
22:    end if
23: end for
```

ALGORITHM 2: SampleTournament

```
Input: The comparison matrix $\mathbf{W}$ and the current batch $B_m$.
Output: The first candidate $a_c$.
1: for $a_i, a_j \in B_m$ and $i < j$ do
2:     Sample $\theta_{ij} \sim Beta(w_{ij} + 1, w_{ji} + 1)$
3:     $\theta_{ji} = 1 - \theta_{ij}$
4: end for
5: $\kappa_i = \frac{1}{|B_m|-1} \sum_{a_j \in B_m, j \neq i} \mathbb{1}(\theta_{ij} > 0.5)$
6: $a_c = \arg\max_{a_i \in B_m} \kappa_i$; breaking ties randomly                 // First candidate
```

ALGORITHM 3: RelativeTournament

```
Input: The comparison matrix $\mathbf{W}$, the current batch $B_m$, and the first candidate $a_c$.
Output: The second candidate $a_d$.
1: for $a_j \in B_m$ and $j \neq c$ do
2:     Sample $\phi_j \sim Beta(w_{jc} + 1, w_{cj} + 1)$
3: end for
4: $\phi_c = 1$                                          // Avoid self-comparison
5: $a_d = \arg\min_{a_j \in B_m} \phi_j$; breaking ties randomly        // Second candidate
```

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

4.2 Theoretical Guarantees

In this section, we state and prove a high probability upper bound on the regret accumulated by MergeDTS after T steps, under the assumption that the dueling bandit problem contains a Condorcet winner. Since the theoretical analysis of MergeDTS is based on that of MergeRUCB, we start by listing two assumptions that we borrow from MergeRUCB in the work of Zoghi et al. [54, Section 7]:

Assumption 1: There is no repetition in rankers. All ranker pairs (a_i, a_j) with $i \neq j$ are distinguishable, i.e., $p_{ij} \neq 0.5$, unless both of them are "uninformative" rankers that provide random ranked lists and cannot beat any other rankers.

Assumption 2: The uninformative rankers are at most one-third of the full set of rankers.

These assumptions arise from the Yahoo! Learning to Rank Challenge dataset, where there are 181 out of 700 rankers that always provide random ranked lists. Assumption 1 ensures that each informative ranker is distinguishable from other rankers. Assumption 2 restricts the maximal percentage of uninformative rankers and thus ensures that the probability of triggering the merge condition (line 18 in Algorithm 1) is larger than 0.[3] Moreover, we emphasize that Assumption 1 and Assumption 2 are milder than the assumptions made in Self-Sparring and DTS, where indistinguishability is simply not allowed.

We now state our main theoretical result.

Theorem 4.1. *With the known timestep T, applying MergeDTS with $\alpha > 0.5$, $M \geq 4$, and $\epsilon = 1/T$ to a K-armed Condorcet dueling bandit problem under Assumption 1 and Assumption 2, with probability $1 - \epsilon$ the cumulative regret $\mathcal{R}(T)$ after T steps is bounded by*

$$\mathcal{R}(T) < \frac{8\alpha MK \ln(T + C(\epsilon))}{\Delta_{\min}^2}, \tag{3}$$

where

$$\Delta_{\min} := \min_{\Delta_{ij} > 0} \Delta_{ij} \tag{4}$$

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

[3]In practice, MergeDTS works without Assumption 2 because the Condorcet winner eliminates all other arms eventually with $O(K^2 \log T)$ comparisons. We keep Assumption 2 to ensure that MergeDTS also works in cases where we have the $O(K \log(T))$ guarantee. We refer readers to the work of Zoghi et al. [54] for a detailed discussion.

The upper bound on the T-step cumulative regret of MergeDTS is $O(K \ln(T)/\Delta^2_{\min})$. In other words, the cumulative regret grows linearly with the number of rankers, K. This is the most important advantage of MergeDTS, which states the potential of applying it to the large-scale online evaluation. We emphasize that for most of the K-armed dueling bandit algorithms in the literature, the upper bounds contain a K^2 term, which renders them unsuitable for large-scale online ranker evaluation. By the definition of $\Delta_{\min}$ in Equation (4) we have $\Delta_{\min} > 0$, and thus our bound is well defined. However, the performance of MergeDTS may degrade severely when $\Delta_{\min}$ is small. α is a common parameter in UCB-type algorithms, called the *exploration parameter.* α controls the trade-off between exploitation and exploration: larger α results in more exploration, whereas smaller α makes the algorithm more exploitative. Theoretically, α should be larger than 0.5. However, as shown in our experiments, using some values of α that are outside the theoretical regime can lead to a boost in the effectiveness of MergeDTS.

Theorem 4.1 provides a finite-horizon high probability bound. From a practical point of view, this type of bound is of great utility. In practice, bandit algorithms are always deployed and evaluated within limited user iterations [34, 49]. Here, each timestep is one user interaction. As the number of interactions is provided, we can choose a reasonable step T to make sure the high probability bound holds. We can also get an expected regret bound of MergeDTS at step T by setting $\epsilon = 1/T$ and adding 1 to the right-hand side of (3): this is because $\mathbb{E}[\mathcal{R}(T)]$ can be bounded by

$$\frac{1}{T} \cdot T + \frac{T-1}{T} \cdot \frac{8\alpha MK \ln(T + C(\epsilon))}{\Delta^2_{\min}} \leq 1 + \frac{8\alpha MK \ln(T + C(\epsilon))}{\Delta^2_{\min}}. \tag{5}$$

We note that the preceding expected bound holds only at timestep T, and thus the horizonless version of MergeDTS does not possess an expected regret bound.

The proof of Theorem 4.1 relies on Lemma 3 of Zoghi et al. [54]. We repeat it here for the reader's convenience.

Lemma 4.2 (Lemma 3 of Zoghi et al. [54]). *Given any pair of distinguishable rankers $a_i, a_j \in B$ and $\epsilon \in [0, 1]$, with the probability of $1 - \epsilon$, the maximum number of comparisons that could have been carried out between these two rankers in the first T timesteps before a merger between B and another batch occurs is bounded by*

$$\frac{4\alpha \ln(T + C(\epsilon))}{\Delta^2_{B,min}}, \tag{6}$$

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

Proof of Theorem 4.1. Lemma 4.2 states that with probability $1 - \epsilon$ the number of comparisons between a pair of distinguishable rankers $(i, j) \in B$ is bounded by

$$\frac{4\alpha \ln(T + C(\epsilon))}{\Delta^2_{B,min}}, \tag{7}$$

regardless of the way the rankers are selected, as long as the same criterion as MergeRUCB is used for eliminating rankers. Since the elimination criterion for MergeDTS is the same as that of MergeRUCB, we can apply the same argument used to prove Theorem 1 of Zoghi et al. [54] to get a bound of

$$\frac{8\alpha MK \ln(T + C(\epsilon))}{\Delta^2_{\min}} \tag{8}$$

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

4.3 Discussion

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

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

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

The second item is borrowed from the design of MergeRUCB.

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

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

5 EXPERIMENTAL SETUP

5.1 Research Questions

In this article, we investigate the application of dueling bandits to the large-scale online ranker evaluation setup. Our experiments are designed to answer the following research questions.

RQ1. Does MergeDTS outperform the state-of-the-art large-scale algorithm MergeRUCB as well as the more recently proposed Self-Sparring in terms of cumulative regret, i.e., effectiveness?

In the bandit literature [7, 8, 44], regret is a measure of the rate of convergence to the Condorcet winner in hindsight. Mapping this to the online ranker evaluation setting, RQ1 asks whether MergeDTS hurts the user experience less than baselines while it is being used for large-scale online ranker evaluation.

RQ2. How do MergeDTS and the baselines scale computationally?

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

RQ3. How do different levels of noise in the feedback signal affect cumulative regret of MergeDTS and the baselines?

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

RQ4. How do MergeDTS and the baselines perform when the Condorcet dueling bandit problem contains cycles?

Previous work has found that cyclical preference relations between rankers are abundant in online ranker comparisons [54, 56]. Can MergeDTS and the baselines find the Condorcet winner when the experimental setup features a large number of cyclical relations between rankers?

RQ5. How does MergeDTS perform when the dueling bandit problem violates the Condorcet assumption?

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

RQ6. Which approach finds the best ranker faster: MergeDTS together with Probabilistic Interleaving (PI) or Sample-Only Scored Multileave (SOSM)?

There are two general approaches to evaluate rankers online: (1) using dueling bandits together with interleaving [54, 56] and (2) directly using multileaving methods [4]. The former approach can output the best ranker with high confidence, whereas the latter approach can compare multiple rankers simultaneously, which may shorten the comparison process.

RQ7. What is the parameter sensitivity of MergeDTS?

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

5.2 Datasets

To answer our research questions, we use two types of dataset: three real-world datasets and a synthetic dataset.

First, to answer RQ1 through RQ3 and RQ6, we run experiments on three large-scale datasets: the Microsoft Learning to Rank (MSLR) WEB30K dataset [39], the Yahoo! Learning to Rank Challenge Set 1 (Yahoo) [10], and the ISTELLA dataset [35].[4] These datasets contain a large number of features based on unsupervised ranking functions, such as BM25 and TF-IDF. In our experiments, we take the widely used setup in which each individual feature is regarded as a ranker [22, 54]. This is different from a real-world setup, where a search system normally ranks documents using a well-trained learning to rank algorithm that combines multiple features. However, the difficulty of a dueling bandit problem comes from the relative quality of pairs of rankers and not from their absolute quality. In other words, evaluating rankers with similar and possibly low performance is as hard as evaluating state-of-the-art rankers, e.g., LambdaMART [6]. Therefore, we stick to the standard setup of Hofmann et al. [22] and Zoghi et al. [54], treating each feature as a ranker and each ranker as an arm in the K-armed dueling bandit problem. We leave experiments aimed at comparing different well-trained learning to rank algorithms as future work. As a summary, the MSLR dataset contains 136 rankers, the Yahoo dataset contains 700 rankers, and the ISTELLA

[4]We omit the Yahoo Set 2 dataset because it contains far fewer queries than the Yahoo Set 1 dataset.

dataset contains 220 rankers. Compared to the typical K-armed dueling bandit setups, where K is generally substantially smaller than 100 [1, 43, 47, 50], these are large numbers of rankers.

Second, to answer RQ4, we use a synthetic dataset, generated by Zoghi et al. [54], which contains cycles (called the *Cycle dataset* in the rest of the article). The Cycle dataset has 20 rankers with one Condorcet winner, a_1, and 19 suboptimal rankers, $a_2, \ldots, a_{20}$. The Condorcet winner beats the other 19 suboptimal rankers. And those 19 rankers have a cyclical preference relationship between them. More precisely, following Zoghi et al. [54], the estimated probability p_{1j} of a_1 beating a_j ($j = 2, \ldots, 20$ is set to $p_{1j} = 0.51$, and the preference relationships between the suboptimal rankers are described as follows: visualize the 19 rankers $a_2, \ldots, a_{20}$ sitting at a round table, then each ranker beats every ranker to its left with probability 1 and loses to every ranker to its right with probability 1. In this way, we obtain the Cycle dataset.

Note that this is a difficult setup for Self-Sparring, because Self-Sparring chooses rankers based on their Borda scores, and the Borda scores ($\sum_{j=1}^{K} p_{ij}$ for each ranker a_i [46]) are close to each other in the Cycle dataset. For example, the Borda score of the Condorcet winner is 10.19, whereas the Borda score of a suboptimal ranker is 9.99. This makes it hard for Self-Sparring to distinguish between rankers. To be able to conduct a fair comparison, we generate the *Cycle2 dataset*, where each suboptimal ranker beats every ranker to its left with a probability of 0.51 and the Condorcet winner beats all others with probability 0.6. Now, in the Cycle2 dataset, the Borda score of the Condorcet winner is 11.90, whereas the Borda score of a suboptimal ranker is 9.9. Thus, it is an easier setup for Self-Sparring.

Furthermore, to answer RQ5, we use the MSLR-non-Condorcet dataset from [47], which is a subset of the MSLR dataset that does not contain a Condorcet winner. This dataset has 32 rankers with two Copeland winners (instead of one), each of which beats the other 30 rankers. A Copeland winner is a ranker that beats the largest number of other rankers; every dueling bandit dataset contains at least one Copeland winner [53].

Finally, we use the MSLR dataset with the navigational configuration (described in Section 5.4) to assess the parameter sensitivity of MergeDTS (RQ7).

5.3 Evaluation Methodology

To evaluate dueling bandit algorithms, we follow the proxy approach of Zoghi et al. [54]. It first uses an interleaving algorithm to obtain a preference matrix, i.e., a matrix that for each pair of rankers contains the probability that one ranker beats the other. More precisely, for each pair of rankers a_i and a_j, p_{ij} is the estimation that a_i beats a_j in the simulated interleaved comparisons. Then, this obtained preference matrix is used to evaluate dueling bandit algorithms: for two rankers a_i and a_j chosen by a dueling bandit algorithm, we compare them by drawing a sample from a Bernoulli distribution with mean p_{ij}, i.e., 1 means that a_i beats a_j and vice versa. This is a standard approach to evaluating dueling bandit algorithms [47, 50, 56]. Moreover, the proxy approach has been shown to have the same quality as interleaving in terms of evaluating dueling bandit algorithms [54].

In this work, we adopt the procedure described by Zoghi et al. [54], who use PI [22] to obtain a preference matrix for the MSLR dataset, and obtain a preference matrix for the Yahoo datasets.[5] In the case of the MSLR dataset, the number of comparisons for every pair of rankers is 400,000, and the case of the Yahoo dataset, we use 60,000 comparisons per pair of rankers. The reason for this discrepancy is pragmatic: the latter dataset has roughly 27 times as many pairs of rankers to be compared.

[5]We use the implementation of PI in the LEROT software package [42].

5.4 Click Simulation

Since the interleaved comparisons mentioned earlier are carried out using click feedback, we follow Hofmann et al. [24] and simulate clicks using three configurations of a click model [12], namely *perfect*, *navigational*, and *informational*. The perfect configuration simulates a user who checks every document and clicks on a document with a probability proportional to the query-document relevance. This configuration is the easiest one for dueling bandit algorithms to find the best ranker, because it contains very little noise. The navigational configuration mimics a user who seeks specific information, i.e., who may be searching for the link of a website, and is likely to stop browsing results after finding a relevant document. The navigational configuration contains more noise than the perfect configuration and is harder for dueling bandit algorithms to find the best ranker. Finally, the informational configuration represents a user who wants to gather all available information for a query and may click on documents that are not relevant with high probability. In the informational configuration, the feedback contains more noise than in the perfect and navigational configurations, which makes it the most difficult configuration for dueling bandit algorithms to determine the best ranker, which, in turn, may result in the highest cumulative regret among the three configurations.

To answer the research questions that concern large-scale dueling bandit problems, namely RQ1, RQ2, RQ6, and RQ7, we use the navigational configuration, which represents a reasonable middle ground between the perfect and informational configurations [22]. The corresponding experimental setups are called *MSLR-Navigational*, *Yahoo-Navigational*, and *ISTELLA-Navigational*. To answer RQ3 regarding the effect of feedback with different levels of noise, we use all three configurations on the MSLR, Yahoo, and ISTELLA datasets, namely MSLR-Perfect, MSLR-Navigational, and MSLR-Informational; Yahoo-Perfect, Yahoo-Navigational, and Yahoo-Informational; and ISTELLA-Perfect, ISTELLA-Navigational, and ISTELLA-Informational. Thus, we have nine large-scale setups in total.

5.5 Baselines

We compare MergeDTS to five state-of-the-art dueling bandit algorithms: MergeRUCB [54], DTS [47], RMED1 [31], Self-Sparring [43], and REX3 [16]. Among these algorithms, MergeRUCB is designed for large-scale online ranker evaluation and is the state-of-the-art large-scale dueling bandit algorithm. DTS is the state-of-the-art small-scale dueling bandit algorithm. RMED1 is motivated by the lower bound of the Condorcet dueling bandit problem and matches the lower bound up to a factor of $O(K^2)$, which indicates that RMED1 has low regret in small-scale problems but may have large regret when the number of rankers is large. Self-Sparring is a more recently proposed dueling bandit algorithm that is the state-of-the-art algorithm in the multi-dueling setup, with which multiple rankers can be compared in each step. REX3 is proposed for the adversarial dueling bandit problem but also performs well for the large-scale stochastic dueling bandit problem [16]. We do not include RUCB [55] and Sparring [1] in our experiments since they have been outperformed by more than one of our baselines [16, 43, 54].

5.6 Parameters

Recall that Theorem 4.1 is based on Lemma 3 of Zoghi et al. [54]. The latter provides a high probability guarantee that the confidence intervals will not mislead the algorithm into eliminating the Condorcet winner by mistake. However, this result is proven using the Chernoff-Hoeffding [19] bound together with an application of the union bound [9], both of which introduce certain gaps between theory and practice. In other words, the analysis of regret mainly considers the worst-case scenario rather than the average-case scenario, which makes regret bounds much looser than they could have been. We conjecture that the expression for $C(\epsilon)$, which derives its form from Lemma 3 of Zoghi et al. [54], is excessively conservative. Put differently, Theorem 4.1 specifies a sufficient condition for the proper functioning of MergeDTS, not a necessary one. Therefore, a natural question that arises is the following: to what extent can restrictions imposed by our theoretical results be violated without the algorithm failing in practice? In short, what is the gap between theory and practice, and what is the parameter sensitivity of MergeDTS?

To address these questions and answer RQ7, we conduct extensive parameter sensitivity analyses in the MSLR-Navigational setup with the following parameters: $\alpha \in \{0.8^0, 0.8^1, \ldots, 0.8^9\}$, $C \in \{4 \times 10^2, 4 \times 10^3, \ldots, 4 \times 10^6, 4,726,908\}$, and $M \in \{2, 4, 8, 16\}$. C is short for $C(\epsilon)$, where $C(\epsilon)$ is the exploration bonus added to the confidence intervals. According to Table 1, $C(\epsilon)$ is a function of α and ϵ. However, to simplify our experimental setup, we consider C as an individual parameter rather than a function parameterized by α and ϵ, and study the impact of C directly. The details about the choice of the values are explained in the following paragraph.

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

The sensitivity of parameters is analyzed by following the order of their importance to Theorem 4.1, i.e., α and M have a linear relation to the cumulative regret, and C has a logarithmic relation to the cumulative regret. We first evaluate the sensitivity of α with the default values of M and C. Then we use the best value of α to test a range of values of M (with default C). Finally, we analyze the impact of C using the best values of α and M.

We discover the practically optimal parameters for MergeDTS to be $\alpha = 0.8^6$, $M = 16$, and $C = 4,000,000$, in Section 6.7. We repeat the procedure for MergeRUCB and DTS, and use their optimal parameter values in our experiments, which are $\alpha = 0.8^6$, $M = 8$, and $C = 400,000$ for MergeRUCB, and $\alpha = 0.8^7$ for DTS. Then, we use these values to answer RQ1 through RQ5. Self-Sparring does not have any parameters, so further analysis and tuning are not needed here.

The shrewd readers may notice that the parameters are somewhat overtuned in the MSLR-Navigational setup, and MergeDTS with the tuned parameters does not enjoy the theoretical guarantees in Theorem 4.1. However, because of the existence of the gap between theory and practice, we want to answer the question whether we can improve the performance of MergeDTS and that of the baselines by tuning the parameters outside of their theoretical limits. We also want to emphasize that the parameters of MergeDTS and baselines are only tuned in the MSLR-Navigational setup, but MergeDTS is compared to baselines in nine setups. If, with the tuned parameters, MergeDTS outperforms baselines in the other eight setups, we can also show the potential of improving MergeDTS in an empirical way.

5.7 Metrics

In our experiments, we assess the efficiency (time complexity) and effectiveness (cumulative regret) of MergeDTS and baselines. The metric for efficiency is the running time in days. We compute the running time from the start of the first step to the end of the T-th step, where $T = 10^8$ in our experiments. A commercial search engine may serve more than 1 billion search requests per day [18], and each search request can be used to conduct one dueling bandit comparison. The total number of timesteps, i.e., T, considered in our experiments is about 1% of the 1-week traffic of a commercial search engine.

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

$$\mathcal{R}(T) = \sum_{t=1}^{T} r(t) = \sum_{t=1}^{T} \frac{1}{2}(\Delta_{1,c_t} + \Delta_{1,d_t}), \tag{9}$$

where $r(t)$ is the regret at step t; c_t and d_t are the indices of rankers chosen at step t; and without loss of generality, we assume a_1 to be the Condorcet winner. The regret $r(t)$ arises from the comparisons between the two suboptimal rankers at t step. It is the average suboptimality of comparing two rankers a_i and a_j with respect to the Condorcet winner a_1, i.e., $\frac{p_{1i}+p_{1j}}{2} - 0.5$. In a real-world scenario, we have a fixed time period to conduct our online ranker evaluation, and thus the number of steps T can be estimated beforehand. In our online ranker evaluation task, the T-step cumulative regret is related to the drop in user satisfaction during the evaluation process, i.e., higher regret means larger degradation of user satisfaction, because the preference p_{1i} can be interpreted as the probability of the Condorcet winner being preferred to ranker i.

Unless stated differently, the results in all experiments are averaged over 50 and 100 independent runs on large- and small-scale datasets, respectively, where both numbers are either equal to or larger than the choices in previous studies [43, 47, 54]. In the effectiveness experiments, we also report the standard error of the average cumulative regret, which measures the differences between the average of samples and the expectation of the sample population.

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

6 RESULTS

In this section, we analyze the results of our experiments. In Section 6.1, we compare the effectiveness (cumulative regret) of MergeDTS and the baselines in three large-scale online evaluation setups. In Section 6.2, we compare and analyze the efficiency (time complexity) of MergeDTS and the baselines. In Section 6.3, we study the impact of different levels of noise in the click feedback on the algorithms. In Section 6.4 and Section 6.5, we evaluate MergeDTS and the baselines in two alternative setups: the cyclic case and the non-Condorcet case, respectively. In Section 6.6, we compare MergeDTS to multileaving methods. In Section 6.7, we analyze the parameter sensitivity of MergeDTS.

6.1 Large-Scale Experiments

To answer RQ1, we compare MergeDTS to the large-scale state-of-the-art baseline, MergeRUCB, as well as the more recently proposed Self-Sparring, in three large-scale online evaluation setups, namely MSLR-Navigational, Yahoo-Navigational, and ISTELLA-Navigational. The results are reported in Figure 1, which depicts the cumulative regret of each algorithm, averaged over 50 independent runs. As mentioned in Section 5, cumulative regret measures the rate of convergence to the Condorcet winner in hindsight, and thus lower regret curves are better. DTS and RMED1 are not considered in the Yahoo-Navigational setup, and the cumulative regret of DTS is reported with in 10^7 steps on the ISTELLA dataset, because of the computational issues, which are further discussed in Section 6.2. Figure 1 shows that MergeDTS outperforms the large-scale

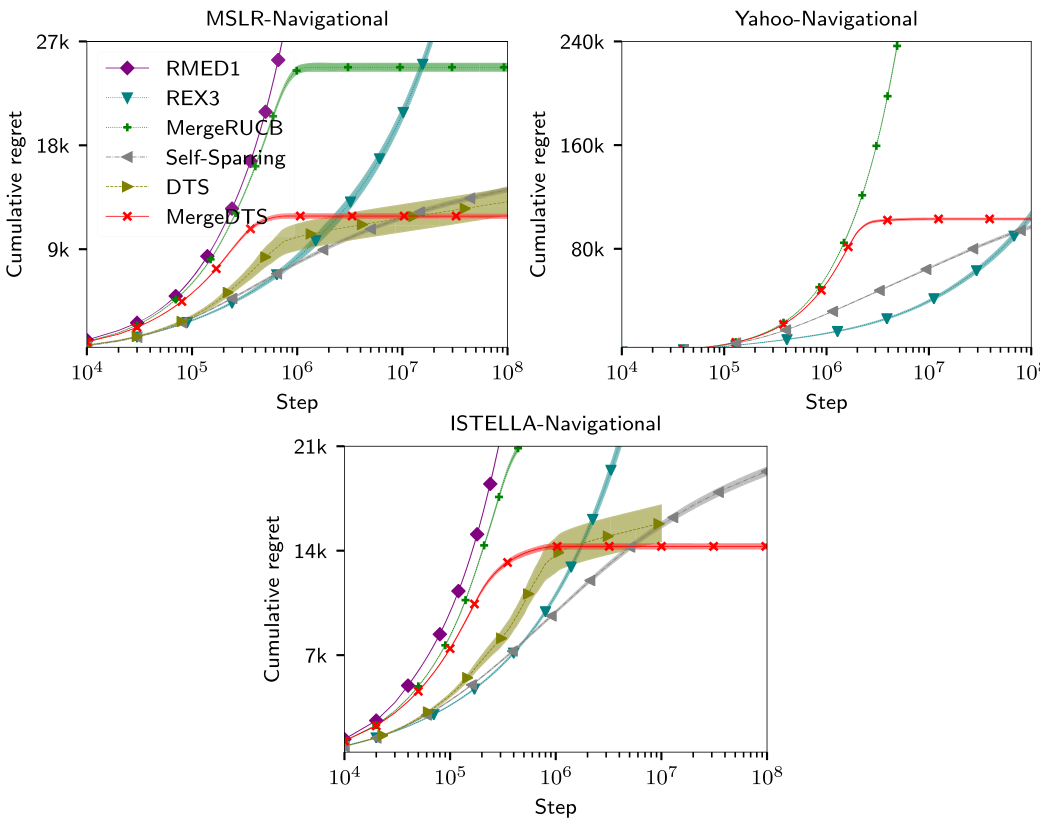

Fig. 1. Cumulative regret on large-scale online ranker evaluation: lower is better. Note that the scales of the y-axes are different. The shaded areas are ± standard error. The results are averaged over 50 independent runs.

state-of-the-art MergeRUCB with large gaps. Regarding the comparison with Self-Sparring and DTS, we would like to point out the following facts: (1) MergeDTS outperforms DTS and Self-Sparring in MSLR-Navigational and ISTELLA-Navigational setups, and (2) in the Yahoo-Navigational setup, MergeDTS has slightly higher cumulative regret than Self-Sparring, but MergeDTS converges to the Condorcet winner after 3 million steps while the cumulative regret of Self-Sparring is still growing after 100 million steps. Translating these facts into real-world scenarios, MergeDTS has higher regret compared to DTS and Self-Sparring in the early steps, but MergeDTS eventually outperforms DTS and Self-Sparring with longer experiments. As for REX3, we see that it has a higher order of regret than other algorithms since REX3 is designed for the adversarial dueling bandits and the regret of REX3 is $O(\sqrt{T})$. In this work, we consider the stochastic dueling bandits, and the regret of the other algorithms is $\log(T)$. MergeDTS outperforms the baselines in most setups, but we need to emphasize that the performance of MergeDTS here cannot be guaranteed by Theorem 4.1. This is because we use the parameter setup outside of the theoretical regime, as discussed in Section 5.6.

6.2 Computational Scalability

To address RQ2, we report in Table 2 the average running time (in days) of each algorithm in three large-scale dueling bandit setups, namely MSLR-Navigational, ISTELLA and Yahoo-Navigational.

Table 2. Average Running Time in Days of Each Algorithm on Large-Scale Problems for 10^8 Steps Averaged over 50 Independent Runs

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

The italics font indicates the estimated running time. The running time of DTS in the ISTELLA-Navigational setup is estimated based on the running time with 10^7 steps multiplied by 10. The running time of DTS and RMED1 in the Yahoo-Navigational setup is estimated by multiplying the average running time at 10^5 steps by 10^3. The experiments on the ISTELLA dataset are conducted on different computer clusters from those on the MSLR and Yahoo datasets. The speed of the former ones is about one time faster than the latter ones. Therefore the numbers may not be directly compared.

As before, each algorithm is run for 10^8 steps. An individual run of DTS and RMED1 in the Yahoo-Navigational setup takes around 100.27 and 18.39 days, respectively, which is simply impractical for our experiments; therefore, the running time of DTS and RMED1 in this setup is estimated by multiplying the average running time at 10^5 steps by 10^3. For a similar reason, we estimate the running time of DTS on the ISTELLA-Navigational setup by multiplying the average running time at 10^7 by 10.

Table 2 shows that MergeDTS and MergeRUCB have very low running times. This is because they perform computations inside batches and their computational complexity is $O(TM^2)$, where T is the number of steps and M is the size of batches. Moreover, MergeDTS is considerably faster in the MSLR-Navigational setup, because there it finds the best ranker with fewer steps, as can be seen in Figure 1. After finding this best ranker, the size of batches M becomes 1, and from that moment on, MergeDTS does not perform any extra computations. The running time of Self-Sparring is also low but grows with the number of rankers, roughly linearly. This is because at each step, Self-Sparring draws a sample from the posterior distribution of each ranker and its running time is $\Omega(TK)$, where K is the number of rankers. DTS is orders of magnitude slower than other algorithms and its running time grows roughly quadratically, because DTS requires a sample for each pair of rankers at each step and its running time is $\Omega(TK^2)$.

Large-scale commercial search systems process more than a billion queries a day [18] and run hundreds of different experiments [30] concurrently, in each of which two rankers are compared. The running time for DTS and RMED1 that appears in Table 2 is far beyond the realm of what might be considered reasonable to process on even 20% of 1 day's traffic: note that one query in the search setting corresponds to one step for a dueling bandit algorithm, since each query could be used to compare two rankers by performing an interleaved comparison. Given the estimated running times listed in Table 2, we exclude DTS and RMED1 from our experiments on the large-scale datasets for practical reasons.

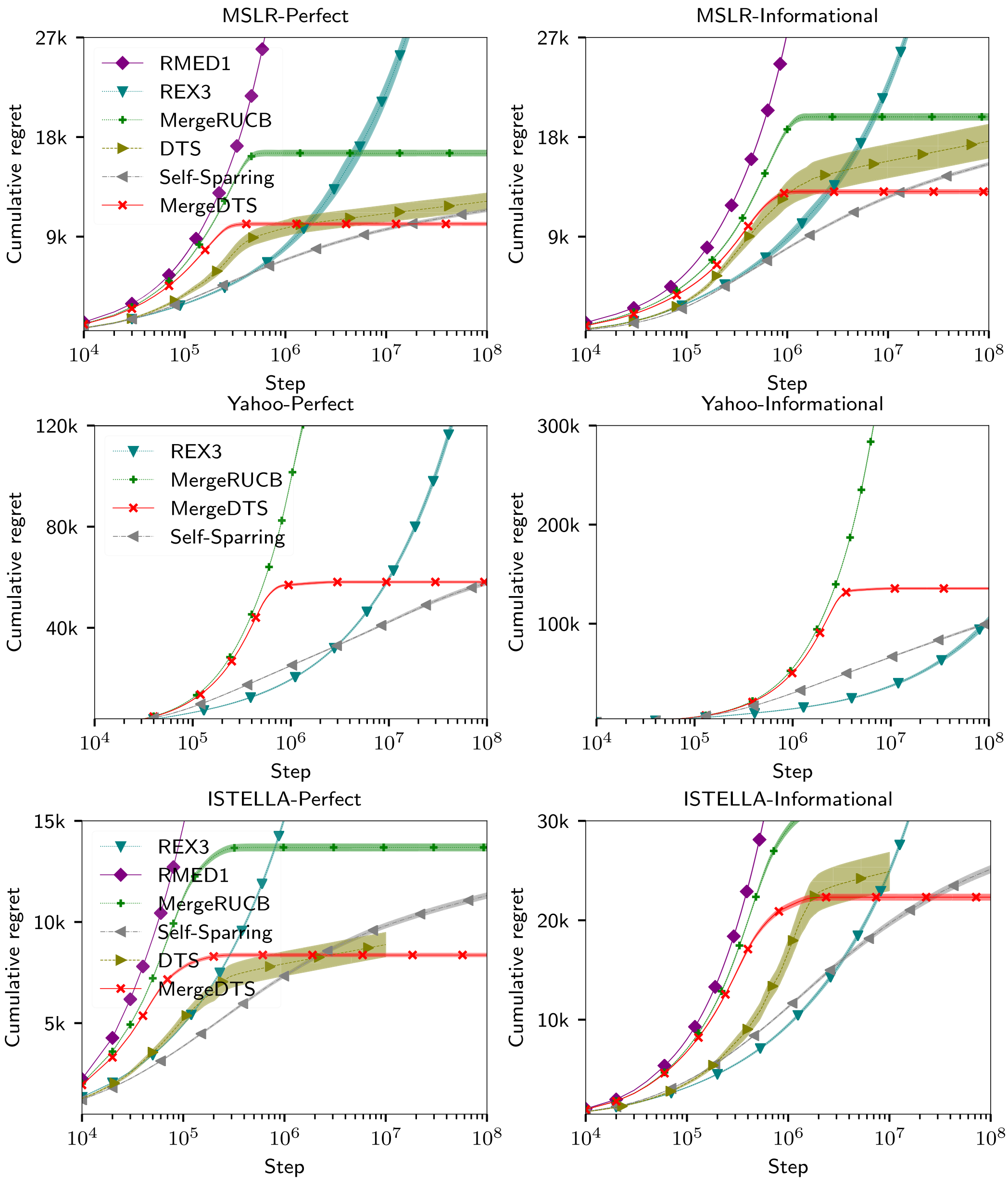

Fig. 2. Effect of the level of noise on cumulative regret in click feedback (left: perfect configuration; right: informational configuration). The results are averaged over 50 independent runs. (Contrast with figures in Figure 1.)

6.3 Impact of Noise

To address RQ3, we run MergeDTS in the perfect, navigational, and informational configurations (see Section 5.4). As discussed in Section 5.4, the perfect configuration is the easiest one for dueling bandit algorithms to reveal the best ranker, whereas the informational configuration is the hardest.

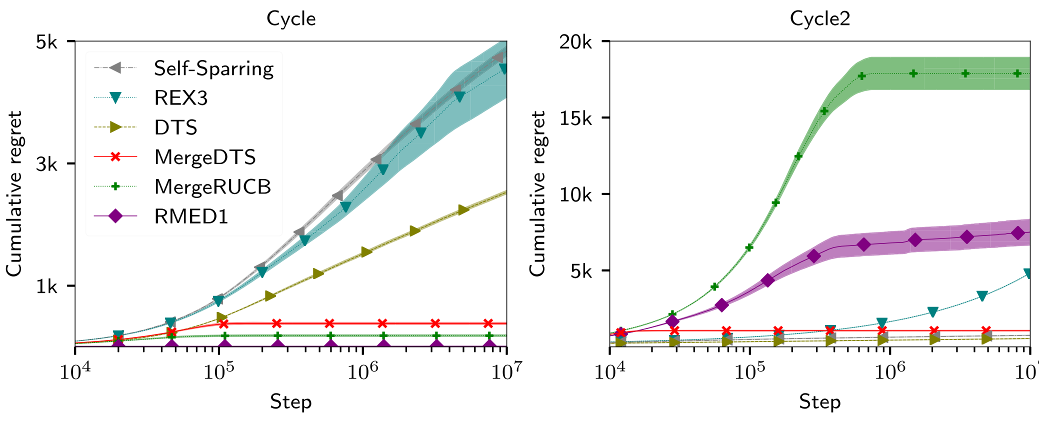

Fig. 3. Cumulative regret in the cycle setup. The results are averaged over 100 independent runs.

We report the results of the perfect and informational configurations in Figure 2. For comparison, we refer readers to plots in Figure 1 for the results of navigational configuration.

On the MSLR and ISTELLA datasets, in all three configurations, MergeDTS with the chosen parameters outperforms the baselines, and the gaps get larger as click feedback gets noisier. The results also show that Self-Sparring is severely affected by the level of noise. This is because Self-Sparring estimates the Borda score [46] of a ranker, and in our experiments, the noisier click feedback is, the closer the Borda scores are to each other, making it harder for Self-Sparring to identify the winner.

Results on the Yahoo dataset disagree with results on the MSLR dataset. On the Yahoo dataset, MergeDTS is affected more severely by the level of noise than Self-Sparring. This is because of the existence of uninformative rankers as stated in Assumption 1. In noisier configurations, the gaps between uninformative and informative rankers are smaller, which results in the long time of comparisons for MergeDTS to eliminate the uninformative rankers. Comparing those uninformative rankers leads to high regret.

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

6.4 Cycle Experiment

We address RQ4 by running the algorithms that we consider on the Cycle and Cycle2 datasets introduced in Section 5.2. In particular, we have already observed that Self-Sparring performs well in some cases (see the preceding experiments and results), but we argue that Self-Sparring may perform poorly when a dueling bandit problem contains cyclic preference relationships. This has been identified as a point of grave concern in online evaluation [56]. Therefore, here we assess how dueling bandit algorithms behave when a dueling bandit problem contains cycles.

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

For the Cycle dataset (left plot in Figure 3), the cumulative regret of Self-Sparring is an order of magnitude higher than that of MergeDTS, although it performs well in some cases (see the preceding experiments). As discussed in Section 5.2, Self-Sparring chooses rankers based on their Borda scores, and when the Borda scores of different arms become close to each other as in the Cycle dataset, Self-Sparring may perform poorly. In addition, we notice that when the gaps in

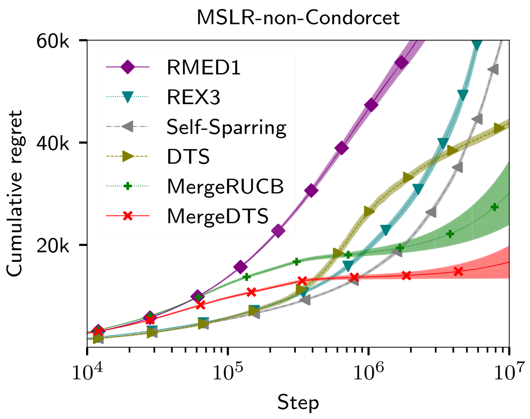

Fig. 4. Cumulative regret in the non-Condorcet setup. The results are averaged over 100 independent runs.

Borda scores of the Condorcet winner and other rankers are large, Self-Sparring performs well, as shown in the right plot in Figure 3.

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

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

6.5 Beyond the Condorcet Assumption

To answer RQ5, we evaluate MergeDTS on the MSLR-non-Condorcet dataset that does not contain a Condorcet winner. Instead, the dataset contains two Copeland winners, and this dueling bandit setup is called the *Copeland dueling bandit* [47, 53]. The Copeland winner is selected by the Copeland score $\zeta_i = \frac{1}{K-1} \sum_{k \neq i} \mathbb{1}(p_{ik} > 1/2)$ that measures the number of rankers beaten by ranker a_i. The Copeland winner is defined as $\zeta^* = \max_{1 \leq i \leq K} \zeta_i$. In the MSLR-non-Condorcet dataset, each Copeland winner beats 30 other rankers. Specifically, one of the Copeland winners beats the other one but is beaten by a suboptimal ranker. In the Copeland dueling bandit setup, regret is computed differently from the Condorcet dueling bandit setup. Given a pair of rankers (a_i, a_j), regret at step t is computed as

$$r_t = \zeta^* - 0.5(\zeta_i + \zeta_j). \tag{10}$$

Among the considered algorithms, only DTS can solve the Copeland dueling bandit problem and is the state-of-the-art Copeland dueling bandit algorithm. We conduct the experiment for 10 million steps with which DTS converges to the Copeland winners. We run each algorithm 100 times independently. The results are shown in Figure 4.

MergeDTS has the lowest cumulative regret. However, in our experiments, we find that MergeDTS eliminates the two Copeland winners one time out of 100 individual repeats. In the other 99 repeats, we find that MergeDTS eliminates one of the two existing winners, which may not be ideal in practice. Note that we evaluate MergeDTS in a relatively easy setup, where only two Copeland winners are considered. For more complicated setups, where more than two Copeland winners are considered or the Copeland winners are beaten by several suboptimal rankers, we speculate that MergeDTS can fail more frequently. In our experiments, we do not evaluate MergeDTS in the more complicated setups, because MergeDTS is designed for the Condorcet dueling bandits and is only guaranteed to work under the Condorcet assumption. The answer to RQ5 is that MergeDTS may perform well for some easy setups that go beyond the Condorcet assumption without any guarantees.

6.6 Comparison to Multileaving

To answer RQ6, we compare MergeDTS together with PI [23] to SOSM [4],[6] a multileaving method that is also designed for large-scale online ranker evaluation. Multileaving methods are designed to infer the preferences of rankers from multileaving comparisons. At each step, a multileaving method generates a ranked list from the lists produced by multiple rankers and infers the preferences of rankers based on the click feedback. Since the click feedback is noisy, the inferred preferences are also noisy. To get the best ranker, we need to run the multileaving method a large number of steps. However, there is no multileaving method that estimates the confidence of the inferred preferences, and thus we do not know when to stop the multileaving comparison.

In this experiment, we use the idea of UCB to choose the rankers. Specifically, we run SOSM as follows: and (1) at each step, we compute UCB estimators, defined in Table 1, of the relative preferences of every pair of rankers; (2) we consider rankers that have not been beaten by any other rankers based on the UCB estimators; (3) we use SOSM to multileave the results of all considered rankers; (4) the multileaved list is shown to the user and click feedback is received; (5) with the click feedback, we use SOSM to infer the relative preferences of considered rankers and update the relative preferences to the matrix **W**, defined in Table 1. This approach is similar to MergeDTS with only one batch. However, now we do not eliminate rankers. In Step 1, to compute UCB estimators, we choose $\alpha = 0.51$. This is a conservative setup, because we may multileave the results of hundreds of rankers and the preferences inferred from a single multileaving comparison with hundreds of rankers can be rather noisy [4]. For a fair comparison, in this experiment we also set $\alpha = 0.51$ for MergeDTS. Step 2 is from the Condorcet assumption that the Condorcet winner beats all others.

To conduct the multileaving experiments, we use the LEROT [42] online simulation setup instead of the proxy setup [54] used in other sections. We choose the MSLR, Yahoo, and ISTELLA datasets with the navigational configuration to simulate clicks and consider the top 10 positions. These are standard setups in online ranker evaluation [4, 21, 24, 41, 54, 55]. Since we use simulated clicks in the experiments, we use the average number of clicks and the NDCG@10 of interleaved or multileaved results as metrics. We compute $NDCG@10$ of the ranked list R as follows:

$$NDCG@10(R) = \frac{DCG@10(R)}{DCG@10(R^*)}, \qquad DCG@10(R) = \sum_{i=1}^{10} \frac{2^{rel(R_i)} - 1}{\log_2{(i+1)}}, \tag{11}$$

where R^* is the optimal ranking, R_i is the i-th item in the ranked list R, and $rel(R_i)$ is the relevance of the item R_i. Both metrics measure the quality of the displayed ranked lists. Higher values

[6]The implementations of PI and SOSM are from this repository: https://github.com/HarrieO/PairwisePreferenceMultileave.

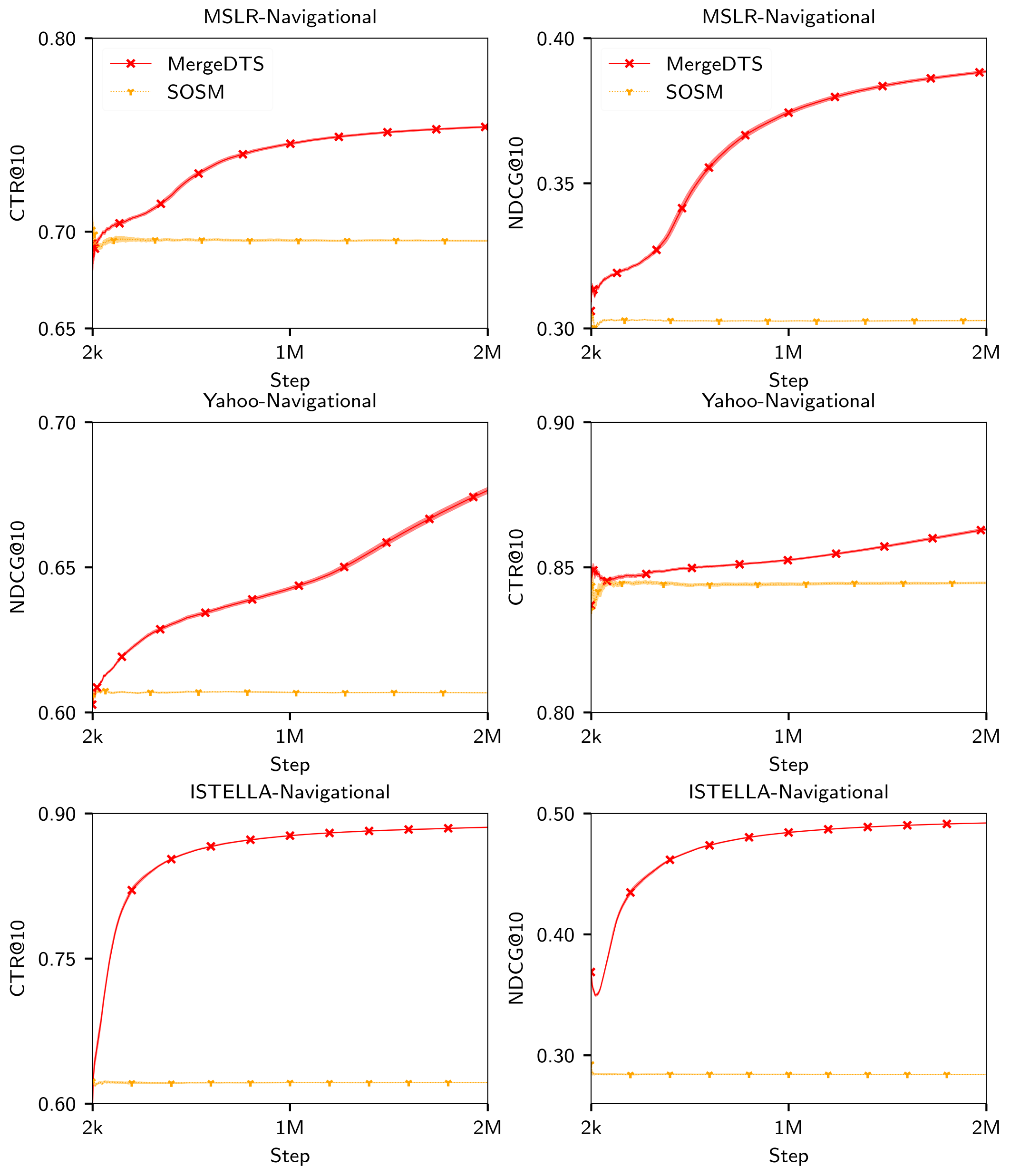

Fig. 5. Left: The average number of clicks received at the top 10 positions (higher is better). Right: The average NDCG@10 of the interleaved or multileaved ranked lists (higher is better). The results are averaged over five independent runs. The shaded areas are ± standard error.

mean less loss of the user experience during online evaluation. The experiments are conducted for 2 million steps, with which MergeDTS converges to the Condorcet winner, as shown in the preceding results. Since LETOR-based simulations are much slower than the proxy method [54], the experiments are repeated (and averaged over) five times.

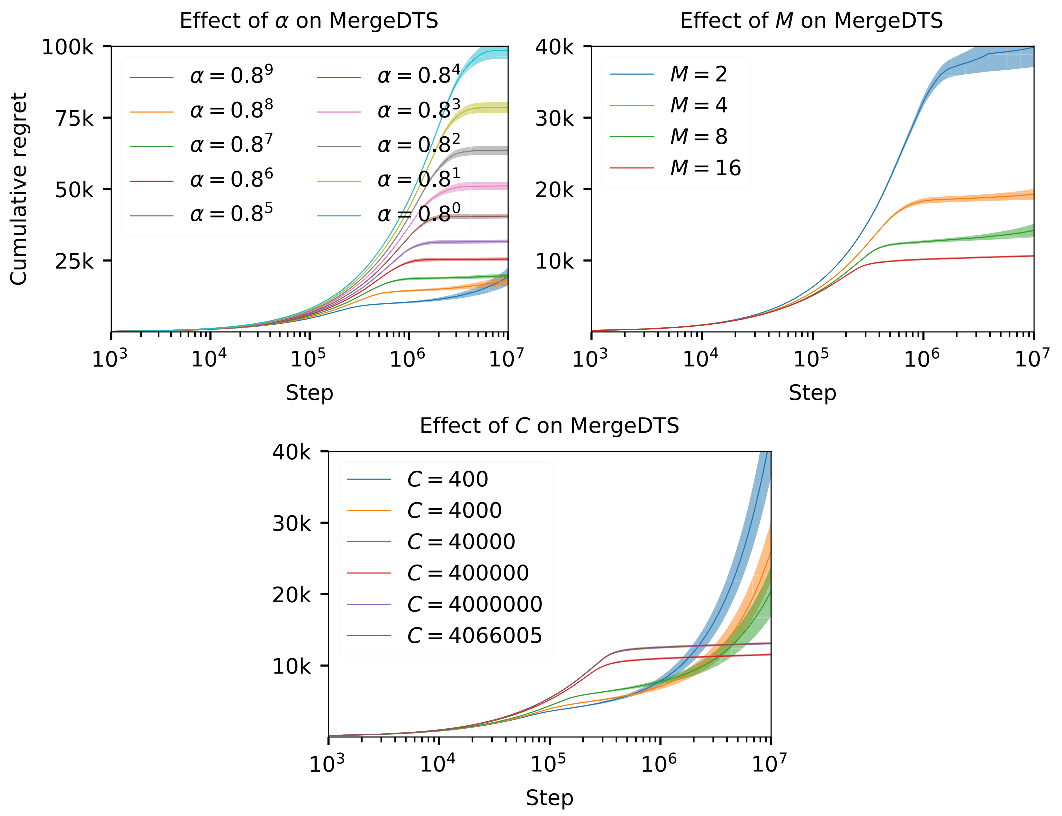

Fig. 6. Effect of the parameters α, M, and C on the performance of MergeDTS in the MSLR-Navigational setup. The results are averaged over 100 independent runs. The shaded areas are ± standard error.

We report the number of clicks as CTR@10 and NDCG@10 in Figure 5. MergeDTS outperforms SOSM on both datasets. Particularly, the quality of the ranked lists identified by MergeDTS increases quickly, which means that MergeDTS can quickly eliminate suboptimal rankers. In the experiments, MergeDTS finds the best ranker in the MSLR and ISTELLA datasets in less than 2 million steps. In the Yahoo dataset, MergeDTS finds two rankers in each repeat, and in a total of five repeats, MergeDTS finds four rankers that contain the best ranker based on the NDCG@10. The curves of SOSM are almost flat, which means that SOSM always compares a large number of rankers. Specifically, in our experiments, SOSM compares all rankers in each step. Summarizing, the answer to RQ6 is that MergeDTS with PI finds the best ranker faster than SOSM.

6.7 Parameter Sensitivity

We answer RQ7 and analyze the parameter sensitivity of MergeDTS using the setup described in Section 5.6. Since MergeDTS converges to the Condorcet winner within 10 million steps, we conduct the experiments with 10 million steps and repeat 100 times. Recall that we conduct the experiments in the MSLR-Navigational setup. The results are reported in Figure 6. We also report the standard errors in the plots.

The top-left plot in Figure 6 shows the effect of the exploration parameter α on the performance of MergeDTS. First, lowering α can significantly increase the performance, e.g., the cumulative regret for $\alpha = 0.8^4$ is about one-third of the reward for $\alpha = 1.0$ (which is close to the theoretically optimal value $\alpha = 1.01$). Second, as we decrease α, the number of failures increases, where a failure is an event that MergeDTS eliminates the Condorcet winner: with $\alpha = \{0.8^9, 0.8^8, 0.8^7\}$, we observe 10, 4, and 1 failures, respectively, and thus the cumulative regret increases linearly w.r.t. T. Since in practice we do not want to eliminate the best ranker, we choose $\alpha = 0.8^6 \approx 0.2621$ in our experiments.

The top-right plot in Figure 6 shows the effect of the batch size M. The larger the batch size, the lower the regret. This can be explained as follows. The DTS-based strategy uses the full local knowledge in a batch to choose the best ranker. A larger batch size M provides more knowledge to MergeDTS to make decisions, which leads to a better choice of rankers. But the time complexity of MergeDTS is $O(TM^2)$, i.e., quadratic in the batch size. Thus, for realistic scenarios, we cannot increase M indefinitely. We choose $M = 16$ as a tradeoff between effectiveness (cumulative regret) and efficiency (running time).

The bottom plot in Figure 6 shows the dependency of MergeDTS on C. Similarly to the effect of α, lower values of C lead to lower regret, but also to a larger number of failures. $C = 4{,}000{,}000$ is the lowest value that does not lead to any failures, so we choose it in our experiments.

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

7 CONCLUSION

In this article, we have studied the large-scale online ranker evaluation problem under the Condorcet assumption, which can be formalized as a K-armed dueling bandit problem. We have proposed a scalable version of the state-of-the-art DTS algorithm, which we call *MergeDTS*.

Our experiments have shown that by choosing the parameter values outside of the theoretical regime, MergeDTS is considerably more efficient than DTS in terms of computational complexity, and that it significantly outperforms the large-scale state-of-the-art algorithm MergeRUCB. Furthermore, we have demonstrated the robustness of MergeDTS when dealing with difficult dueling bandit problems containing cycles among the arms. We have also demonstrated that MergeDTS can be applied to some of the dueling bandit tasks that do not contain a Condorcet winner. Last, we have shown that the performance of MergeDTS is guaranteed if the parameter values fall within the theoretical regime.

Several interesting directions for future work arise from this study. First in our experiments, we have shown that there is a large gap between theory and practice. It will be interesting to study this gap and provide a tighter theoretical bound. Second, we only study dueling bandits in this article. We believe that it is interesting to study a generalization of MergeDTS, as well as the theoretical analysis presented here, to the case of online ranker evaluation tasks with a multi-dueling setup. Third, since multi-dueling bandits also compare multiple rankers at each step based on relative feedback, it is an interesting direction to compare dueling bandits to multi-dueling bandits in the large-scale setup. Fourth, we suspect that the UCB-based elimination utilized in MergeDTS is too conservative, and it might be that more recent minimum empirical divergence–based techniques [32] may be leveraged to speed up the elimination of the rankers. Fifth the feature rankers are chosen as arms in our experiments. A more interesting and realistic way of choosing arms is to use well-trained learning to rank algorithms.

CODE AND DATA

To facilitate reproducibility of the results in this article, we are sharing the code and the data we used to run the experiments at https://github.com/chang-li/MergeDTS.

ACKNOWLEDGMENTS

We thank Artem Grotov, Rolf Jagerman, Harrie Oosterhuis, Christophe Van Gysel, and Nikos Voskarides for their helpful comments and technical support. We also thank our editor and the anonymous reviewers for extensive comments and suggestions that helped us improve the article.

REFERENCES

[1] Nir Ailon, Zohar Karnin, and Thorsten Joachims. 2014. Reducing dueling bandits to cardinal bandits. In *Proceedings of the 31st International Conference on on Machine Learning—Volume 32 (ICML '14)*. II-856–II-864. http://dl.acm.org/citation.cfm?id=3044805.3044988

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47, 2-3 (May 2002), 235–256. DOI: https://doi.org/10.1023/A:1013689704352

[3] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. 2003. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing* 32, 1 (Jan. 2003), 48–77. DOI: https://doi.org/10.1137/S0097539701398375

[4] Brian Brost, Ingemar J. Cox, Yevgeny Seldin, and Christina Lioma. 2016. An improved multileaving algorithm for online ranker evaluation. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'16)*. ACM, New York, NY, 745–748. DOI: https://doi.org/10.1145/2911451.2914706

[5] Brian Brost, Yevgeny Seldin, Ingemar J. Cox, and Christina Lioma. 2016. Multi-dueling bandits and their application to online ranker evaluation. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM'16)*. ACM, New York, NY, 2161–2166. DOI: https://doi.org/10.1145/2983323.2983659

[6] Chris J. C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Technical Report MSR-TR-2010-82. Microsoft Research. Available at https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/.

[7] Róbert Busa-Fekete and Eyke Hüllermeier. 2014. A survey of preference-based online learning with bandit algorithms. In *Proceedings of the 25th International Conference on Algorithmic Learning Theory (ALT'14)*. 18–39. DOI: https://doi.org/10.1007/978-3-319-11662-4_3

[8] Robert Busa-Fekete, Eyke Hüllermeier, and Adil El Mesaoudi-Paul. 2018. Preference-based online learning with dueling bandits: A survey. arXiv:1807.11398

[9] George Casella and Roger L. Berger. 2002. *Statistical Inference*. Duxbury, Pacific Grove, CA.

[10] Olivier Chapelle and Yi Chang. 2010. Yahoo! Learning to rank challenge overview. In *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge—Volume 14 (YLRC'10)*. 1–24. http://dl.acm.org/citation.cfm?id=3045754.3045756

[11] Olivier Chapelle, Thorsten Joachims, Filip Radlinski, and Yisong Yue. 2012. Large-scale validation and analysis of interleaved search evaluation. *ACM Transactions on Information Systems* 30, 1 (March 2012), Article 6, 41 pages. DOI: https://doi.org/10.1145/2094072.2094078

[12] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search*. Morgan & Claypool.DOI: https://doi.org/10.2200/S00654ED1V01Y201507ICR043

[13] Aleksandr Chuklin, Anne Schuth, Ke Zhou, and Maarten de Rijke. 2015. A comparative analysis of interleaving methods for aggregated search. *ACM Transactions on Information Systems* 33, 2 (Feb. 2015), Article 5, 38 pages. DOI: https://doi.org/10.1145/2668120

[14] Romain Deveaud, Josiane Mothe, Md Zia Ullah, and Jian-Yun Nie. 2018. Learning to adaptively rank document retrieval system configurations. *ACM Transactions on Information Systems* 37, 1 (Oct. 2018), Article 3, 41 pages. DOI: https://doi.org/10.1145/3231937

[15] Miroslav Dudík, Katja Hofmann, Robert E. Schapire, Aleksandrs Slivkins, and Masrour Zoghi. 2015. Contextual dueling bandits. In *Proceedings of the 28th Conference on Learning Theory—Volume 40 (COLT'15)*. 563–587. http://jmlr.org/proceedings/papers/v40/Dudik15.html.

[16] Pratik Gajane, Tanguy Urvoy, and Fabrice Clérot. 2015. A relative exponential weighing algorithm for adversarial utility-based dueling bandits. In *Proceedings of the 32nd International Conference on Machine Learning—Volume 37 (ICML'15)*. 218–227. http://dl.acm.org/citation.cfm?id=3045118.3045143

[17] David Goldberg, Andrew Trotman, Xiao Wang, Wei Min, and Zongru Wan. 2018. Further insights on drawing sound conclusions from noisy judgments. *ACM Transaction Information Systems* 36, 4 (April 2018), Article 36, 31 pages. DOI: https://doi.org/10.1145/3186195

[18] Google. 2010. Google Instant, Behind the Scenes. Retrieved August 17, 2020 from https://googleblog.blogspot.no/2010/09/google-instant-behind-scenes.html.

[19] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58, 301 (1963), 13–30. https://www.jstor.org/stable/2282952?seq=1

[20] Katja Hofmann, Lihong Li, and Filip Radlinski. 2016. Online evaluation for information retrieval. *Foundations and Trends in Information Retrieval* 10, 1 (June 2016), 1–117. DOI: https://doi.org/10.1561/1500000051

[21] Katja Hofmann, Anne Schuth, Shimon Whiteson, and Maarten de Rijke. 2013. Reusing historical interaction data for faster online learning to rank for information retrieval. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM'13)*. ACM, New York, NY, 183–192. DOI: https://doi.org/10.1145/2433396.2433419

[22] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2011. A probabilistic method for inferring preferences from clicks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11)*. ACM, New York, NY, 249–258. DOI: https://doi.org/10.1145/2063576.2063618

[23] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2012. Estimating interleaved comparison outcomes from historical click data. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*. ACM, New York, NY, 1779–1783. DOI: https://doi.org/10.1145/2396761.2398516

[24] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2013. Fidelity, soundness, and efficiency of interleaved comparison methods. *ACM Transactions on Information Systems* 31, 4 (Nov. 2013), Article 17, 43 pages. DOI: https://doi.org/10.1145/2536736.2536737

[25] Junya Honda and Akimichi Takemura. 2011. An asymptotically optimal policy for finite support models in the multiarmed bandit problem. *Machine Learning* 85, 3 (2011), 361–391. DOI: https://doi.org/10.1007/s10994-011-5257-4

[26] Muhammad Ibrahim and Mark Carman. 2016. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. *ACM Transactions on Information Systems* 34, 4 (Aug. 2016), Article 20, 38 pages. DOI: https://doi.org/10.1145/2866571

[27] Kevin Jamieson, Sumeet Katariya, Atul Deshpande, and Robert Nowak. 2015. Sparse dueling bandits. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics—Volume 38 (AISTATS'15)*. 416–424. http://jmlr.org/proceedings/papers/v38/jamieson15.html.

[28] Thorsten Joachims. 2003. Evaluating retrieval performance using clickthrough data. In *Text Mining*. Cornell University, Ithaca, NY, 79–96. Available at https://www.cs.cornell.edu/people/tj/publications/joachims_02b.pdf.

[29] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems* 25, 2 (April 2007), Article 7. DOI: https://doi.org/10.1145/1229179.1229181

[30] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. 2013. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. ACM, New York, NY, 1168–1176. DOI: https://doi.org/10.1145/2487575.2488217

[31] Junpei Komiyama, Junya Honda, Hisashi Kashima, and Hiroshi Nakagawa. 2015. Regret lower bound and optimal algorithm in dueling bandit problem. In *Proceedings of the 28th Conference on Learning Theory—Volume 40 (COLT'15)*. 1141–1154. http://jmlr.org/proceedings/papers/v40/Komiyama15.html.

[32] Junpei Komiyama, Junya Honda, and Hiroshi Nakagawa. 2016. Copeland dueling bandit problem: Regret lower bound, optimal algorithm, and computationally efficient algorithm. In *Proceedings of the 33rd International Conference on Machine Learning—Volume 48 (ICML'16)*. 1235–1244. http://dl.acm.org/citation.cfm?id=3045390.3045521

[33] Chang Li, Branislav Kveton, Tor Lattimore, Ilya Markov, Maarten de Rijke, Csaba Szepesvari, and Masrour Zoghi. 2019. BubbleRank: Safe online learning to re-rank via implicit click feedback. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'2019)*. http://auai.org/uai2019/proceedings/papers/47.pdf.

[34] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the Conference on World Wide Web*(WWW'10). 661–670.

[35] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. 2016. Post-learning optimization of tree ensembles for efficient ranking. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'16)*. 949–952. DOI: https://doi.org/10.1145/2911451.2914763

[36] Ilya Markov and Maarten de Rijke. 2019. What should we teach in information retrieval? *SIGIR Forum* 52, 2 (Jan. 2019), 19–39. DOI: https://doi.org/10.1145/3308774.3308780

[37] Alistair Moffat, Peter Bailey, Falk Scholer, and Paul Thomas. 2017. Incorporating user expectations and behavior into the measurement of search effectiveness. *ACM Transactions on Information Systems* 35, 3 (June 2017), Article 24, 38 pages. DOI: https://doi.org/10.1145/3052768

[38] Klaas Nelissen, Monique Snoeck, Seppe Vanden Broucke, and Bart Baesens. 2018. Swipe and tell: Using implicit feedback to predict user engagement on tablets. *ACM Transactions on Information Systems* 36, 4 (June 2018), Article 35, 36 pages. DOI: https://doi.org/10.1145/3185153

[39] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. arXiv:1306.2597

[40] Aadirupa Saha and Aditya Gopalan. 2018. Battle of bandits. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI'18)*. http://auai.org/uai2018/proceedings/papers/290.pdf.

[41] Anne Schuth, Robert-Jan Bruintjes, Fritjof Büttner, Joost van Doorn, Carla Groenland, Harrie Oosterhuis, Cong-Nguyen Tran, et al. 2015. Probabilistic multileave for online retrieval evaluation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'15)*. ACM, New York, NY, 955–958. DOI: https://doi.org/10.1145/2766462.2767838

[42] Anne Schuth, Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2013. Lerot: An online learning to rank framework. In *Proceedings of the 2013 Workshop on Living Labs for Information Retrieval Evaluation (LivingLab'13)*. ACM, New York, NY, 23–26. DOI: https://doi.org/10.1145/2513150.2513162

[43] Yanan Sui, Vincent Zhuang, Joel W. Burdick, and Yisong Yue. 2017. Multi-dueling bandits with dependent arms. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI'17)*. http://auai.org/uai2017/proceedings/papers/155.pdf.

[44] Yanan Sui, Masrour Zoghi, Katja Hofmann, and Yisong Yue. 2018. Advancements in dueling bandits. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. 5502–5510. http://dl.acm.org/citation.cfm?id=3304652.3304790

[45] William R. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25 (1933), 285–294. https://www.jstor.org/stable/pdf/2332286.pdf?seq=1

[46] Tanguy Urvoy, Fabrice Clerot, Raphael Féraud, and Sami Naamane. 2013. Generic exploration and K-armed voting bandits. In *Proceedings of the 30th International Conference on Machine Learning—Volume 28 (ICML'13)*. II-91–II-99. http://dl.acm.org/citation.cfm?id=3042817.3042904

[47] Huasen Wu and Xin Liu. 2016. Double Thompson sampling for dueling bandits. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. 649–657. http://dl.acm.org/citation.cfm?id=3157096.3157169

[48] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. 2012. The K-armed dueling bandits problem. *Journal of Computer and System Sciences* 78, 5 (Sept. 2012), 1538–1556. DOI: https://doi.org/10.1016/j.jcss.2011.12.028

[49] Yisong Yue and Thorsten Joachims. 2009. Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*. ACM, New York, NY, 1201–1208. DOI: https://doi.org/10.1145/1553374.1553527

[50] Yisong Yue and Thorsten Joachims. 2011. Beat the mean bandit. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*. 241–248. http://dl.acm.org/citation.cfm?id=3104482.3104513

[51] Julian Zimmert and Yevgeny Seldin. 2019. An optimal algorithm for stochastic and adversarial bandits. In *Proceedings of Machine Learning Research*, K. Chaudhuri and M. Sugiyama (Eds.), Vol. 89. PMLR, 467–475.

[52] Masrour Zoghi. 2017. *Dueling Bandits for Online Ranker Evaluation*. Ph.D. Dissertation. University of Twente. https://research.utwente.nl/en/publications/dueling-bandits-for-online-ranker-evaluation.

[53] Masrour Zoghi, Zohar Karnin, Shimon Whiteson, and Maarten de Rijke. 2015. Copeland dueling bandits. In *Proceedings of the 28th International Conference on Neural Information Processing Systems—Volume 1 (NIPS'15)*. 307–315. http://dl.acm.org/citation.cfm?id=2969239.2969274

[54] Masrour Zoghi, Shimon Whiteson, and Maarten de Rijke. 2015. MergeRUCB: A method for large-scale online ranker evaluation. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining (WSDM'15)*. ACM, New York, NY, 17–26. DOI: https://doi.org/10.1145/2684822.2685290

[55] Masrour Zoghi, Shimon Whiteson, Remi Munos, and Maarten de Rijke. 2014. Relative upper confidence bound for the K-armed dueling bandit problem. In *Proceedings of the 31st International Conference on Machine Learning—Volume 32 (ICML'14)*. II-10–II-18. http://dl.acm.org/citation.cfm?id=3044805.3044894

[56] Masrour Zoghi, Shimon A. Whiteson, Maarten de Rijke, and Remi Munos. 2014. Relative confidence sampling for efficient on-line ranker evaluation. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM'14)*. ACM, New York, NY, 73–82. DOI: https://doi.org/10.1145/2556195.2556256

Received November 2018; revised July 2020; accepted July 2020