Beyond Relevance Ranking: A General Graph Matching Framework for Utility-oriented Learning to Rank

XINYI DAI, YUNJIA XI, and WEINAN ZHANG†, Shanghai Jiao Tong University, China
QING LIU, RUIMING TANG, and XIUQIANG HE, Noah's Ark Lab, Huawei, China
JIAWEI HOU, Shanghai Jiao Tong University, China
JUN WANG, University College London, The United Kingdom
YONG YU, Shanghai Jiao Tong University, China

Learning to rank from logged user feedback, such as clicks or purchases, is a central component of many real-world information systems. Different from human-annotated relevance labels, the user feedback is always noisy and biased. Many existing learning to rank methods infer the underlying relevance of query-item pairs based on different assumptions of examination, and still optimize a relevance based objective. Such methods rely heavily on the correct estimation of examination, which is often difficult to achieve in practice. In this work, we propose a general framework *U-rank+* for learning to rank with logged user feedback from the perspective of graph matching. We systematically analyze the biases in user feedback, including examination bias, and selection bias. Then we take both biases into consideration for unbiased utility estimation that directly based on user feedback, instead of relevance. In order to maximize the estimated utility in an efficient manner, we design two different solvers based on Sinkhorn and Lambdaloss for *U-rank+*. The former is based on a standard graph matching algorithm, and the latter is inspired by the traditional method of learning to rank. Both of the algorithms have good theoretical properties to optimize the unbiased utility objective while the latter is proved to be empirically more effective and efficient in practice. Our framework *U-rank+* can deal with a general utility function and can be used in a widespread of applications including web search, recommendation and online advertising. Semi-synthetic experiments on three benchmark learning to rank datasets demonstrate the effectiveness of *U-rank+*. Further, our proposed framework has been deployed on two different scenarios of a mainstream App store, where the online A/B testing shows that *U-rank+* achieves an average improvement of 19.2% on click-through rate and 20.8% improvement on conversion rate in recommendation scenario, and 5.12% on platform revenue in online advertising scenario over the production baselines.

CCS Concepts: • **Information systems → Learning to rank**.

Additional Key Words and Phrases: Learning to Rank, Utility Maximization, Graph Matching, Implicit Feedback, Position Bias, Examination Bias, Selection Bias

ACM Reference Format:
Xinyi Dai, Yunjia Xi, Weinan Zhang†, Qing Liu, Ruiming Tang, Xiuqiang He, Jiawei Hou, Jun Wang, and Yong Yu. 2021. Beyond Relevance Ranking: A General Graph Matching Framework for Utility-oriented Learning to

†Corresponding author.
Authors' addresses: Xinyi Dai; Yunjia Xi; Weinan Zhang†, Shanghai Jiao Tong University, China, daixinyi@sjtu.edu.cn, xiyunjia@sjtu.edu.cn, wnzhang@sjtu.edu.cn; Qing Liu; Ruiming Tang; Xiuqiang He, Noah's Ark Lab, Huawei, China, liuqing48@huawei.com, tangruiming@huawei.com, hexiuqiang1@huawei.com; Jiawei Hou, Shanghai Jiao Tong University, China, hjw99868@sjtu.edu.cn; Jun Wang, University College London, The United Kingdom, jun.wang@cs.ucl.ac.uk; Yong Yu, Shanghai Jiao Tong University, China, yyu@apex.sjtu.edu.cn.

1046-8188/2021/1-ART1 $15.00
https://doi.org/10.1145/3464303

Rank. *ACM Transactions on Information Systems* 1, 1, Article 1 (January 2021), 29 pages. https://doi.org/10.1145/3464303

1 INTRODUCTION

Ranking in Information Retrieval (IR) systems aims at providing a ranked list of items from a large item pool to maximize a certain form of utility [47], *e.g.*, the amount of clicks or purchases on an e-commerce website [51] or the user engagement on a video streaming platform [15]. To maximize this utility, a ranking metric is often needed as an evaluation measure, which also serves as a training objective of the ranking model. Many existing ranking metrics such as NDCG, describing this utility, are based on the concept of *relevance*. However, human-labeled relevance judgments are costly and time-consuming to obtain, especially in personalized IR systems such as a recommender system. Therefore, in practice, users' implicit feedback[1] (e.g., users' click-through logs) are usually utilized, as they are indicative of relevance judgments with low cost to collect, although biased [24, 25].

Existing unbiased learning to rank methods aim at optimizing a relevance-based ranking objective based on implicit feedback. They assume that the underlying relevance of all the items, which is not directly observed, can be inferred from the observed clicks. For example, click models [7, 11, 13, 16, 18, 41] make assumptions about user browsing behavior and estimate relevance via maximizing the likelihood of observed clicks. Such methods require multiple occurrences of the same query-document pair and only work well on head queries. On the basis of click models, a recent direction of counterfactual learning methods [3, 4, 17, 22, 27, 48] weight the clicks with the Inverse Propensity Weights. Such methods are theoretically proved to be consistent with the same model trained with true relevance judgements given the correct examination bias estimation. However, the correct estimation of examination is a difficult task itself, which often relies on the result of randomization [27, 48] or multiple occurrences of the same query-document pair [3, 17]. Besides, jointly optimizing bias estimation with relevance estimation [4, 22] is often biased unless the relevance estimation is highly precise.

Another view is that the clicks themselves are indicative of users' satisfaction and should be maximized [40]. This is more straightforward since clicks are directly observable. Also, the objective based on clicks is more aligned with online metrics (e.g., click-through rate, conversion rate, etc.) than that based on relevance judgements. This objective is often adopted in Online Learning to Rank (OLTR) [37, 44, 46, 52] where we can observe clicks upon changing the ranking policy. For instance, Yue and Joachims [52] used Dual Bandit Gradient Descent (DBGD) to learn ranking models based on user feedback from result interleaving. Schuth et al. [44] extended DBGD with Multileave Gradient Descent. However, such methods are often restrictive in practice since interleaving of ranking results hurts user experience.

In this work, we aim at deriving a ranking algorithm that optimizes click-based objectives, which can work in offline settings with click datasets from historical logs. In such a setting, we need to consider both intrinsic and extrinsic biases of the click datasets. The intrinsic biases are users' internal behavior biases. For example, users tend to view the items ranked on top positions and users tend to view the visually attractive items, leading to *examination bias*. The extrinsic bias refers to *selection bias* which is caused by the historical ranking function previously deployed in the system. The historical ranking function tends to place certain items at top positions, leading to Missing-Not-At-Random (MNAR) of the collected training data. All these biases together bring challenges in deriving an unbiased utility objective based on implicit feedback[2].

[1]Here implicit feedback include, but are not limited to, clicking, purchasing, viewing, add-to-cart, etc. For simplicity, in the following parts of this paper, the term "click" is used as a representative of all types of user behaviors.

[2]For a comprehensive analysis of bias issues and debias techniques in recommender systems and IR, we refer to [12].

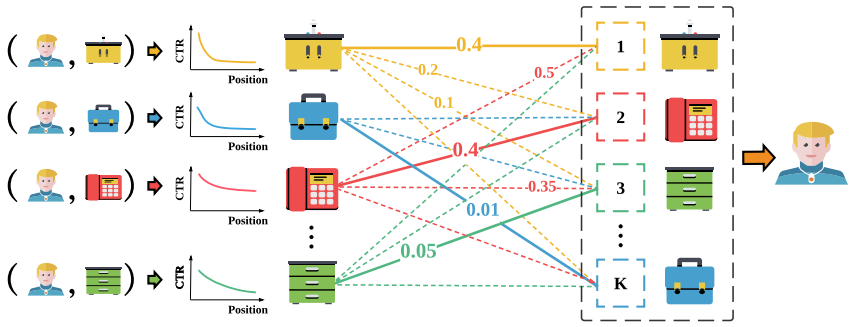

Fig. 1. The maximization of the utility can be formulated as solving a maximum-weight matching problem on the item-position bipartite graph, where the weight of an edge between an item and a position denotes the utility of placing the item at this position, e.g., the CTR or the expected revenue of the item at this position.

Based on all of these considerations, we present a general graph matching framework that optimizes a target ranking metric directly based on implicit feedback. We formulate the ranking problem as a maximum-weight matching problem on an item-position bipartite graph as denoted in Figure 1, where the weight on an edge between an item and a position denotes the utility of placing the item at this position. We define the utility as the product of the item's click-through rate (CTR) at this position and the benefit if the item is clicked, where the benefit takes different definitions in various scenarios, e.g., the bid price of each ad in sponsor search [55, 56], the watch time of each video in video recommendation [15].

Then the maximization of utility can be achieved by solving the maximum-weight matching on the item-position bipartite graph. We divide our method into two steps: firstly, we obtain the weight of each edge on the bipartite graph with a position-aware deep CTR model, which manages to capture the click-position dependency considering both item attributes and user context and thus models position bias and attention bias as a whole. Besides, we learn a propensity model in advance to correct selection bias in the logging data. Secondly, to solve the maximum-weight matching on the graph with learned weights, i.e., maximizing the expected utility, we incorporate two different types of solvers into our framework, namely *Sinkhorn* and *Lambdaloss*. Sinkhorn algorithm [2] learns a Doubly-Stochastic Matrices (DSM)-based ranking function and optimizes the expected utility in an end-to-end manner. Lambdaloss [50] solves the matching problem by learning a scoring function, which can reduce the complexity in inference stage from $O(N^3)$ to $O(N)$, where N denotes the maximum size of the ranked list. Both of the algorithms have good theoretical properties and the latter is empirically proved to be more effective in practice.

In this graph matching framework, we can deal with general utility maximization ranking problems for different application scenarios, including web search, recommendation and online advertising. We conduct semi-synthetic experiments based on three benchmark datasets for learning to rank, where we demonstrate the superiority of our proposed framework over state-of-the-art unbiased learning to rank baselines in utility based metrics like CTR. Furthermore, we deploy our method on two different application scenarios (i.e., recommendation and online advertising) of a mainstream App store, where the superiority of our proposed method is validated in industrial applications on online metrics like CTR, conversion rate, and platform revenue.

The main contributions of this paper are as follows.

- We propose a novel framework for learning to rank with implicit feedback, namely, *U-rank+*, which formulates the ranking problem as finding the maximum-weight matching on an item-position bipartite graph. Instead of optimizing a relevance-based ranking metric, we optimize for a utility metric that directly based on clicks themselves. This framework deals with a general utility function and can be used in a widespread of application scenarios including web search, recommendation and online advertising.
- To obtain an unbiased utility estimation, we estimate the utility of placing each item at each position with a position-aware deep CTR model, in which we take examination bias and selection bias into consideration. To the best of our knowledge, this is the first work that addresses the above-mentioned biases together in a unified framework for utility-oriented learning to rank. To optimize the estimated utility metric directly, we incorporate two different solvers into the framework, i.e., Sinkhorn and Lambdaloss, providing new paths for learning to rank with implicit feedback from different views.
- Semi-synthetic experiments on three real-world benchmark datasets have demonstrated the effectiveness of *U-rank+*. Further, *U-rank+* has been deployed on two different scenarios of a mainstream App store, and in our online A/B testing *U-rank+* achieves an average improvement of 19.2% on CTR and 20.8% improvement on conversion rate in recommendation scenario, and 5.12% on platform revenue in online advertising scenario over the production baselines.

2 RELATED WORK

2.1 Click Models

Implicit feedback like click data is biased according to the observations in [25, 26]. Generative click models are introduced to study user browsing behavior and extract unbiased relevance estimation from click data. For example, Position Based model (PBM) [41] assumes that a click only depends on the position and the relevance of the document. Cascade model [13] assumes that users browse a search web page sequentially from top to bottom, until a relevant document is found. Following these two classical click models, more sophisticated approaches, e.g., UBM [16], DBN [11], CCM [18] have been proposed. The most recent model, Neural Click Model [7] utilizes recurrent neural networks and vector representations of users to predict user behavior. However, click models usually require same query-document pair to appear multiple times for reliable estimation [33], thus invalid for tail queries and for personalized information systems.

2.2 Counterfactual Learning to Rank

Recently, a new line of research, referred to as counterfactual learning to rank, utilizes inverse propensity weighting (IPW) to address position bias in a learning to rank framework. Wang et al. [48] and Joachims et al. [27] proposed IPW-based methods of debiasing click data in a learning to rank framework. In both works, the propensity estimation relies on randomizing search results displayed to users, which obviously degrades users' search experience. Considering this, Agarwal et al. [3] proposed PBM to estimate propensity without Intrusive Interventions. CPBM [17], on the basis of PBM, learns a query-dependent propensity estimation. However, multiple rankers are required to learn, which makes them inconvenient to deploy in real-world applications. Besides, another category of unbiased learning to rank works jointly learn the propensity model with a relevance model, which results in a biased estimation of propensity unless the relevance estimation is very accurate. Wang et al. [49] proposed a regression-based EM algorithm to estimate the propensity by maximizing the likelihood of click data. [4] proposed a dual learning algorithm that jointly learns the propensity estimation and an unbiased ranker. Hu et al. [22] also proposed to

jointly learn the propensity estimation and an unbiased ranker where the propensity at the click and non-click positions are both estimated.

In this work, we derive a ranking objective that directly based on clicks instead of relevance. A click-based objective does not require an accurate estimation of examination bias/propensity, which is hard to achieve in practice, and is also more aligned with online metrics like CTR, CVR, etc. Besides, most of the existing works assume examination bias only depend on positions (except CPBM), we can deal with a more complicated examination bias that depends on user contexts and item contributes with position-ware deep CTR model.

Oosterhuis and de Rijke [38] focused on the item selection bias in top-k items of a ranking, which is defined as the missing feedback caused by the selection of only k items to display by the historical ranking policy. In their work, the examination bias is known beforehand. We also use the term selection bias, which is commonly used to refer to the bias caused by collecting data in a non-uniform ranking policy [21], but there is clear difference with the item selection bias in [38] which only appears in top-k rankings. In this work, selection bias refers to the over-estimation of examination bias caused by the historical ranking policy when we estimate it from the click logs.

2.3 Online Learning to Rank

The central idea of Online Learning to Rank (OLTR) is to optimize the ranking model interactively from user clicks. Dueling Bandit Gradient Descent (DBGD) based algorithms are commonly used in OLTR. The original DBGD algorithm proposed by Yue and Joachims [52] randomly perturbs parameters and updates models towards perturbed parameters that produces ranked lists with more clicks. Extensive researches extend DBGD with different exploration strategies [37, 44, 46] and variance reduction techniques [45]. Schuth et al. [44] propose Mutileave Gradient Descent (MGD) based on DBGD by sampling multiple perturbed parameters each time to accelerate convergence. Null Space Gradient Descent (NSGD) proposed by Wang et al. [46] reduces the exploration space to the null space of recent poorly-performed gradients. Oosterhuis and de Rijke [37] propose Pairwise Differentiable Gradient Descent (PDGD) which estimates unbiased gradients based on pairwise preference from user clicks instead of interleaving or multileaving experiments. Wang et al. [45] reduce the variance of PDGD by projecting the estimated gradients into a document space spanned by the feature vectors of examined documents under current query. OLTR methods can not be directly used in offline setting since they require the ranked list to be dynamically sampled according to the ranking model while the ranked list in offline data is fixed. Instead, our proposed method can work under pure offline settings and does not require a direct interaction with users, which might damage their user experience.

3 PROBLEM FORMULATION AND PRELIMINARIES

3.1 Problem Formulation

When a user issues a new query q, the system delivers a ranked list $(f_i, b_i)_{i=1}^{n_q}$ of n_q items to the user according to a ranking model over all the candidate items. The feature vector f_i of each item i consists of item features, context features, and user/query features[3]. The scalar b_i denotes the benefit brought to the system if item i is clicked, *e.g.*, the watch time of each video in video recommendation, or the bid price of each ad in sponsored search. Let $\mathcal{F}$, $\mathcal{B}$ and $\mathcal{K}$ be the random variables for item feature, benefit and position, respectively. Let C and O denote the random variable for click and examination, respectively. We use k_i to denote the position that item i is assigned to by the current ranking model. To represent the observed clicks of item i at different positions, for each

[3]We treat context features and user/query features as "special" item features for simplicity, such that the same items for different queries/users are considered to be different.

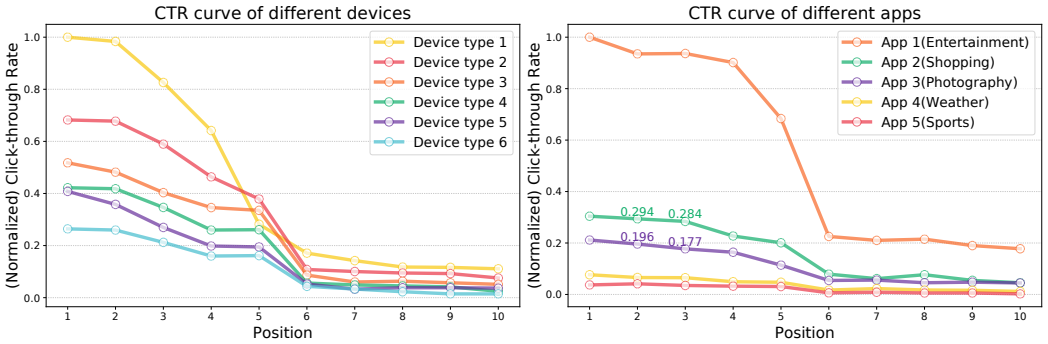

Fig. 2. The CTR analysis w.r.t. query/item features. The data is collected through 120-days' click logs on random recommendation traffic in a mainstream App store.

query q, we introduce a square matrix $\mathbf{C} \in \{0,1\}^{n_q \times n_q}$, where $\mathbf{C}_{i,k} = 1/0$ denotes click/non-click of item i at position k.

The users' click-through log is a set $S = \{(f_i, k_i^h, b_i, c_i)_{i=1}^{n_q}\}_{q \in Q}$. Besides the item feature f_i and benefit b_i as mentioned above, k_i^h is the position of item i assigned by the historical ranking model, and c_i is the users' click on item i when displaying it at position k_i^h in the logging data, *i.e.*, $c_i = 1$ for click and $c_i = 0$ for non-click. We do not abuse different notations to represent position and click, as different notations are used to distinguish them under current ranking policy from those under historical ranking policy. More specifically, k_i and $\mathbf{C}$ represent the position and click under current ranking policy, while k_i^h and c_i notate the corresponding information under the historical ranking policy.

The ultimate goal of this ranking system is to find the best permutation of candidate items for each query q to maximize the utility. The utility is defined as the expectation of weighted sum of clicks on the items in the ranking list, as shown in Eq. (1).

$$U_q = \mathbb{E}\Big[\sum_{i=1}^{n_q} \mathbf{C}_{i,k_i} \cdot b_i\Big] = \sum_{i=1}^{n_q} P(C = 1 | \mathcal{F} = f_i, \mathcal{K} = k_i) \cdot b_i \,, \tag{1}$$

where $P(C = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)$ is the probability of the item i being clicked if displayed at position k_i. Maximizing utility U_q is equivalent to solving a maximum-weight matching problem on the item-position bipartite graph, where $P(C = 1|\mathcal{F} = f_i, \mathcal{K} = k_i) \cdot b_i$ is the weight of the edge between the item i and the position k_i in the graph, as shown in Figure 1. In order to estimate $P(C = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)$ accurately, we need to consider several biases in the click-through log.

3.2 Examination Bias

To visualize the examination bias, we show CTR at different positions of different device types and of different Apps respectively in Figure 2. The data is collected from the random recommendation traffic of a mainstream App store. From Figure 2, we can draw two observations: 1) the CTR decreases as the presented position goes from top to bottom; 2) the magnitude of the decreasing differs among different items and different device types.

As supported by eye-tracking studies [30, 31], the decrease of CTR w.r.t position is the effect of the decrease of user's attention. Most existing works only consider position bias, which is considered to be decorrelated with specific ranked items, i.e., making the same effect on all items. This assumption is generally correct in the traditional *ten blue links* scenario. Under such an assumption, ranking by relevance in descending order leads to the highest expected utility. However, in real world

applications, as indicated in the second observation, the users' examination does not only depend on positions but also on item attributes and user context. All of them will affect the accurate estimation of users' examination biases.

In web search, an example of item-specific examination bias is vertical bias, commonly observed when the web page contains multiple vertical search results (such as images, videos, maps, etc.). Metrikov et al. [35] found that an image in search results can raise CTR and flatten the CTR curve at the same time. This is consistent with our observation in the App recommendation scenario. A visually attractive content, such as an App with a fancy thumbnail, can attract users' attention even when it is placed at a lower position, which results in a flatter CTR curve. In other words, such visually attractive items are less sensitive to the position change.

What is more, the item-specific examination biases lead to a different solution with the traditional relevance-based ranking. Consider a non-personalized case for simplicity that we recommend App 1, App 2, and App 3 in Figure 2 to one user. If the Apps are sorted by Probabilistic Ranking Principle (PRP) [42], the ranked list will be App 1, App 2, and App 3 by their relevance in terms of click-through rate (CTR). However, the optimal ranked list with the maximum utility should be App 1, App 3, and App 2, since the utility gain of promoting App 3 from the 3rd to the 2nd position is 0.196-0.177=0.019, which is larger than the utility loss 0.294-0.284=0.010 of dragging App 2 from the 2nd to the 3rd position. For two items with close relevance estimation, putting the item which is more sensitive to position change at a higher position will bring a higher expected utility, even if it is slightly less relevant.

Based on the above considerations, a position-aware modeling of clicks, which takes item attributes and user context into consideration, is indispensable for the correct estimation of expected utility.

3.3 Selection Bias

Although we can attend to examination bias with the position-aware click modeling, we might tend to overestimate the severity of examination bias due to the non-uniform historical ranking policy. In other words, we might overestimate the click probability of items placed at top positions and underestimate the click probability of items placed at bottom positions.

The bias caused by collecting data from a non-uniform logging policy is often referred to as selection bias [21] or missing-not-at-random [29]. In the left panel of Figure 3, we show a simple example to illustrate the selection bias in a position-aware click model. Assume that we recommend two items to two individual users in the form of ranked lists (of two items). User 1 favors item 2 and user 2 favors item 1. The CTR of a user on a liked item at 1st position and 2nd position are 0.7 and 0.6, respectively, while the CTR of a user on a disliked item at two positions are 0.2 and 0.1, respectively. In a real-world recommender system, a well-trained historical ranking policy will rank the liked item before the disliked one with a high probability. Assume that to user 1 for 90 times we present the ranked list to her in the order of (item 2, item 1) and only for 10 times we present her with (item 1, item 2). To user 2, we do just the opposite. We will find that in the biased click data the estimated CTR of the two items at position 1 and position 2 are $\frac{0.7\times90+0.2\times10}{100}$=0.65 and $\frac{0.1\times90+0.6\times10}{100}$=0.15, respectively, while the true CTR of these two items at position 1 and position 2 are $\frac{0.7\times50+0.2\times50}{100}$=0.45 and $\frac{0.1\times50+0.6\times50}{100}$=0.35, respectively.

The example shows that in a biased logging data, we tend to overestimate the click probability at top positions and underestimate the click probability at bottom positions. This is consistent with the real-world data analysis in the App recommendation scenario where we collect click data from the biased and random traffic, respectively. Here, the biased traffic is the normal traffic served by the production model while the random traffic is a small part of live traffic that we perform uniform

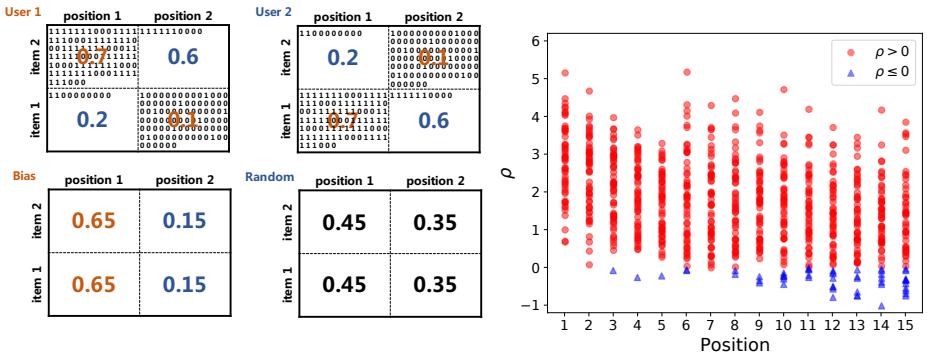

Fig. 3. An analysis of selection bias by comparing the estimated CTR from biased and unbiased click datasets. See the definition of ρ in Eq. (2).

ranking policy on. We estimate the CTR for item i at position k as $CTR_{i,k}$. In order to present the difference of $CTR_{i,k}$ estimated from biased data and random traffic data, we compute $\rho_{i,k}$ as in Eq. (2).

$$\rho_{i,k} = \log \frac{CTR_{i,k} \text{ observed from biased traffic}}{CTR_{i,k} \text{ observed from random traffic}} \tag{2}$$

Then, we plot points $(k, \rho_{i,k})$ in the right panel of Figure 3. We can find that with the increase of position k, the CTR estimation of items gradually change from overestimation ($\rho > 0$) to underestimation ($\rho \leq 0$). We only show the results of top 15 positions, since at lower positions we collect less data, which makes the CTR estimation unreliable.

The analysis of selection bias shows that we need to address the exaggerated position effect caused by non-uniform training data in position-aware click modeling.

4 METHOD

In this section, we present a general ranking framework to maximize the utility U_q in Eq. (1) directly. In Section 4.1, we estimate utility unbiasedly from click through logs. The examination bias and selection bias are addressed by a position-aware click model and a position propensity model as described in Section 4.1.1 and 4.1.2. In Section 4.2, we design two different solvers, namely Sinkhorn (Section 4.2.1) and Lambdaloss (Section 4.2.2) to optimize this metric. The overall procedure of our ranking method is illustrated in Algorithm 1.

4.1 Unbiased Estimation of the Ranking Metric

4.1.1 Position-aware Click Estimation. The main difficulty of learning to rank via implicit feedback lies in the estimation of examination, since we do not observe it directly from the data. With the new learning objective utility, we do not need to infer relevance or examination explicitly. Instead, we need to deal with the mismatch between the CTR of the historically presented position and that of the finally presented position. For example, if an item is ranked on the first position in the click logs but presented at the 10th position in the final ranking, then its utility is overestimated. To correct this bias, we need an accurate model of user's CTR on different positions.

The estimation of the position-aware click probability $P(C = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)$ refers to one of the most well-studied tasks in recommender systems, i.e., the CTR prediction [39, 53]. Some recent works [5, 20] pointed out that position is a very important feature in CTR prediction. However, in inference stage of the ranking model, the position feature is often vacant. Therefore, we use a

Algorithm 1: Graph Matching for Learning to Rank

Input: Click through logs S = $\{(f_i, b_i, c_i, k_i^h)_i^{n_q}\}_{q \in Q}$, solver ∈ {Sinkhorn, Lambdaloss}
Output: A family of functions Ψ, or a scoring function Φ

Step 1 → Unbiased position-aware click probability estimation;
for N_0 *iterations* **do**
  Choose a batch of (f_i, k_i^h) samples from S;
  Update ω according to $\mathcal{L}_p(\omega)$ in Eq. (5);
end
for N_1 *iterations* **do**
  Choose a batch of (f_i, k_i^h, c_i) samples from S;
  Update θ according to $\mathcal{L}_c(\theta)$ in Eq. (8) and $h_\omega(\cdot, \cdot)$;
end
Step 2 → Unbiased utility maximization;
if *solver* ***is*** *Sinkhorn* **then**
  for N_s *iterations* **do**
    Choose a set $(f_i, b_i)_{i=1}^{n_q}$ from S;
    Compute the Pre-Sinkhorn matrix A_{n_q} according to Eq. (9).;
    Compute the DSM W_{n_q} via Sinkhorn layers according to Eq. (11).;
    Update the family of functions Ψ according to $\mathcal{L}_s(\Psi, q)$ in Eq. (12) and $g_\theta(\cdot, \cdot)$;
  end
end
else
  `// solver is Lambdaloss`
  for N_r *iterations* **do**
    Choose a batch of paired samples $(f_i, k_i^h, c_i, f_j, k_j^h, c_j)$ from S;
    Compute $\Delta Util(i, j)$ of each pair according to Eq. (18).;
    Update the scoring function Φ according to $\mathcal{L}_r'(\Phi, q)$ in Eq. (19) and $g_\theta(\cdot, \cdot)$;
  end
end

position-aware click model to maintain an unbiased utility estimation instead of directly using it as a ranking model. Assume that the probability function of item i displayed at position k_i, i.e., $P(\mathcal{C} = 1 | \mathcal{F} = f_i, \mathcal{K} = k_i)$ is a function of item feature f_i and position k_i and we use $g_\theta(f_i, k_i)$ to approximate it. $g_\theta(\cdot, \cdot)$ is a neural network with a sigmoid activation as output activation to ensure the output value in the range of [0, 1]. Then we can estimate the parameter θ via maximum likelihood estimation by minimizing the following loss:

$$\mathcal{L}_c(\theta) = \sum_{q \in Q} \sum_{i=1}^{n_q} l\left(c_i, g_\theta\left(f_i, k_i^h\right)\right), \tag{3}$$

where $l(p, q) = -p \log q - (1 - p) \log(1 - q)$ is the cross entropy loss.

4.1.2 Positional Propensity Estimation. This position-aware click model that we described above suffers from selection bias. From the perspective of domain adaptation [6], selection bias is caused by the distributional mismatch between source distribution $P_s(\mathcal{F}, \mathcal{K})$, i.e., the distribution of $(\mathcal{F}, \mathcal{K})$ pair in the biased logging data and the target distribution $P_t(\mathcal{F}, \mathcal{K})$, i.e., the distribution of $(\mathcal{F}, \mathcal{K})$

pair of an unbiased click data. Consider the density function of $(\mathcal{F}, \mathcal{K})$,

$$P(\mathcal{F}, \mathcal{K}) = P(\mathcal{F})P(\mathcal{K}|\mathcal{F}) \tag{4}$$

where $P(\mathcal{F})$ denotes the distribution of item feature and $P(\mathcal{K}|\mathcal{F})$ denotes the distribution of displayed positions of an item in the logging data conditioned on the item feature. We notice that $P(\mathcal{F})$ is consistent in source distribution and target distribution. However, $P(\mathcal{K}|\mathcal{F})$, which is influenced by the historical ranking policy, shows inconsistency between source distribution and target distribution. $P_t(\mathcal{K}|\mathcal{F})$ is a uniform distribution, which means each item can be placed at each position with equal probability. However, $P_s(\mathcal{K}|\mathcal{F})$ is often influenced by the historical ranking policy, which results in a non-uniform distribution.

We address the distribution mismatch with inverse propensity weighting. Firstly, we use a conditional propensity model $h_\omega(f_i)$, to estimate the position propensity[4] given item feature, i.e., $P_s(\mathcal{K} = k_i|\mathcal{F} = f_i)$, where $h_\omega(f_i)$ is a neural network with a softmax layer as the output layer. It takes the item feature f_i as input and outputs probabilities of this item to appear on different positions. The k_i-th output of $h_\omega(f_i)$ is denoted as $h_\omega(f_i, k_i)$, and the loss function of $h_\omega(f_i, k_i)$ is defined as

$$\mathcal{L}_p(\omega) = -\sum_{q\in Q}\sum_{i=1}^{n_q} \log(h_\omega(f_i, k_i^h)). \tag{5}$$

Now we rewrite the loss function to estimate the click probability of target distribution with the data from source distribution as

$$\begin{aligned}
\mathcal{L}_T(\theta) &= \int \sum_{\mathcal{K}} l\left(C_{i,k_i}, g_\theta\left(f_i, k_i\right)\right) P_t(\mathcal{F} = f_i, \mathcal{K} = k_i) d\mathcal{F} \\
&= \int \sum_{\mathcal{K}} l\left(C_{i,k_i}, g_\theta\left(f_i, k_i\right)\right) P_t(\mathcal{F} = f_i) P_t(\mathcal{K} = k_i|\mathcal{F} = f_i) d\mathcal{F} \\
&= \int \sum_{\mathcal{K}} l\left(C_{i,k_i}, g_\theta\left(f_i, k_i\right)\right) P_t(\mathcal{F} = f_i) \frac{1}{n_q} d\mathcal{F} \\
&= \int \sum_{\mathcal{K}} \frac{1}{n_q P_s(\mathcal{K} = k_i|\mathcal{F} = f_i)} l\left(C_{i,k_i}, g_\theta\left(f_i, k_i\right)\right) P_s(\mathcal{F} = f_i) P_s(\mathcal{K} = k_i|\mathcal{F} = f_i) d\mathcal{F}
\end{aligned} \tag{6}$$

where l is the cross entropy as in Eq. (3). This loss can be estimated with sample average as

$$\hat{\mathcal{L}}_T(\theta) = \sum_{q\in Q}\sum_{i=1}^{n_q} \frac{1}{n_q P_s(\mathcal{K} = k_i^h|\mathcal{F} = f_i)} l\left(c_i, g_\theta\left(f_i, k_i^h\right)\right). \tag{7}$$

Finally we can remove the selection bias and obtain the unbiased estimation of position-aware click model $g_\theta(f_i, k_i)$, the unbiased loss function of which is defined as

$$\mathcal{L}_{uc}(\theta) = \hat{\mathcal{L}}_T(\theta) = \sum_{q\in Q}\sum_{i=1}^{n_q} \frac{1}{n_q h_\omega(f_i, k_i^h)} l\left(c_i, g_\theta\left(f_i, k_i^h\right)\right). \tag{8}$$

[4]In this work we assume that we do not know the logging policy or there is multiple ranking policy, which is quite common in a real-world platform. If we know the logging policy exactly, then we can use the true position propensities instead of learning it from click logs.

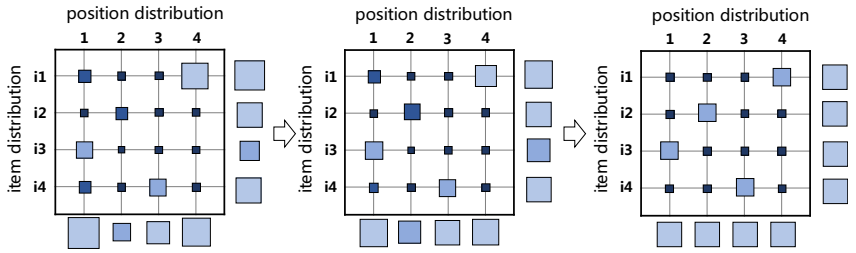

Fig. 4. Sinkhorn Normalization of two layers. The square inside the box denotes the value of each element in the matrix. The square at the end of each column/row outside the box denotes sum of the elements in the column/row. In each Sinkhorn layer, row and column normalization are performed to the matrix. After several layers, the row-wise and column-wise sums are almost indistinguishable.

4.2 Learning to Optimize the Ranking Metric

4.2.1 Sinkhorn. The Sinkhorn Algorithm [2] is an elegant method to solve the graph matching problem in an end-to-end differentiable way. One successful application for Sinkhorn algorithm is [43], which adopted it to solve image matching problem in a deep learning framework. According to the analysis in [34], Sinkhorn algorithm is an approximate and differential version of the Hungarian algorithm. Thus, it is a direct solution to the graph matching problem defined in Section 3.1.

The Sinkhorn Algorithm shows that each non-negative square matrix can be converted to a Doubly Stochastic matrix (DSM) [34], which is a differentiable relaxation of permutation matrix [2]. Then the matching objective can be computed with the DSM and thus be optimized in an end-to-end manner.

The target for the Sinkhorn solver is to learn a family of functions, $\Psi = \{\psi_{n_q} : q \in \mathcal{Q}\}$. Each function in this family takes the features of items $(f_i, b_i)_{i=1}^{n_q}$ as input and outputs a DSM $\mathbf{W}$. Each column and each row in a DSM sums to one. We define each function in the family as $\psi_{n_q} : (\mathcal{F}, \mathcal{B})_{n_q} \rightarrow \mathcal{W}_{n_q}$ where $(\mathcal{F}, \mathcal{B})_{n_q}$ denotes the set of features associate with each of the n_q documents, and $\mathcal{W}_{n_q} \in [0, 1]^{n_q \times n_q}$ refers to the space of $n_q \times n_q$ doubly stochastic matrices.

The process of the sinkhorn solver is shown from line 11 to line 17 in Algorithm 1. Before we apply sinkhorn layers, we need to construct a non-negative square matrix $\mathbf{A} \in \mathbb{R}_+^{n_q \times n_q}$ from the item features $(f_i, b_i)_{n_q}$ (see line 14 in Algorithm 1). We call this non-negative matrix as *Pre-Sinkhorn* matrix. To obtain the Pre-Sinkhorn matrix, we use another function $\varphi : (\mathcal{F}, \mathcal{B}, \mathcal{K}) \rightarrow \mathbb{R}_+$. Specifically, $\varphi(f_i, b_i, k_i)$ is a neural network which outputs a single value. Note that the activation function of last layer should be carefully chosen to ensure the non-negativity. Each element of the Pre-Sinkhorn matrix $\mathbf{A}$ is computed by

$$\mathbf{A}_{i,k} = \varphi(f_i, b_i, k) \ . \tag{9}$$

After that, the DSM $\mathbf{W}$ is obtained from the Pre-Sinkhorn matrix $\mathbf{A}$ by repeatedly normalizing rows and columns (see line 15 in Algorithm 1). Specifically, we defined the row and column normalization functions as

$$\mathcal{T}_{\mathrm{R}}(\mathbf{A}) = \mathbf{A} \oslash \left(\mathbf{A}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right) \ , \quad \mathcal{T}_{\mathrm{C}}(\mathbf{A}) = \mathbf{A} \oslash \left(\mathbf{1}\mathbf{1}^{\mathrm{T}}\mathbf{A}\right) \ , \tag{10}$$

where $\oslash$ is the elementwise division and $\mathbf{1}$ is a vector of ones. Then, the Sinkhorn layers are defined recursively as

$$\mathrm{Sinkhorn}^{(l)}(\mathbf{A}) = \begin{cases} \mathbf{A} & \text{if } l = 0 \\ \mathcal{T}_{\mathrm{R}}\left(\mathcal{T}_{\mathrm{C}}\left(\mathrm{Sinkhorn}^{(l-1)}(\mathbf{A})\right)\right) & \text{otherwise} \end{cases} \tag{11}$$

where l denotes number of layers. As l increases, Sinkhorn$^{(l)}(\mathbf{A})$ converges to DSM $\mathbf{W}$. Each element in the DSM, for example, $\mathbf{W}_{i,k}$, denotes the probability of placing item i at position k according to the family of functions Ψ. To maximize the expected utility, we can define the objective function w.r.t Ψ as,

$$\mathcal{L}_s(\Psi, q) = -\sum_{i=1}^{n_q}\sum_{k=1}^{n_q} g_\theta\left(f_i, k\right)\cdot b_i \cdot \mathbf{W}_{i,k}\ . \tag{12}$$

We prove that minimizing the loss defined in Eq. (12) is equivalent to maximizing the expected utility in Section 4.3.1. We minimize this objective to update the parameters in Ψ (see line 16 in Algorithm 1). Then repeat the process until convergence and we obtain the optimal Ψ. The most exciting part about Sinkhorn algorithm is that all of the operations in Eq. (9), Eq. (10) and Eq. (11) are differentiable and can be optimized in an end-to-end manner.

During testing, we have to apply Hungarian algorithm with cubic complexity [34] to the DSM $\mathbf{W}$ to obtain the permutation matrix $\mathbf{S}$. Each permutation matrix corresponds to a ranking result with each row representing an item, each column representing a position, where each element in permutation matrix $\mathbf{S}$ is defined as

$$\mathbf{S}_{i,k} = \begin{cases} 1 & k_i = k \\ 0 & k_i \neq k \end{cases}\ . \tag{13}$$

However, the cubic complexity becomes a bottleneck in a real-world ranking system. Thus in practice, according to [2], we can compute a global ordering according to the expected rank of each document as

$$\mathbb{E}_W[k_i] = \sum_{k=1}^{n_q} W_{i,k}\cdot k\ , \tag{14}$$

such that Hungarian algorithm can be performed on top $P < n_q$ items instead of on all n_q items to improve the inference efficiency. Thus the complexity in inference stage is reduced to $O(N^2 + P^3)$, where N is the maximum number of n_q.

4.2.2 LambdaLoss. Although we reduce the complexity of sinkhorn as in Eq. (14) from $O(N^3)$ to $O(N^2 + P^3)$, it is still too consuming for a real-world production system. In this section, we aim to learn a parameterized scoring function $\Phi(\cdot)$ to approximate the maximum-weight graph matching procedure on each query, still aiming at the maximization of the utility, so that the complexity in the inference stage can be reduced to $O(N)$.

The scoring function $\Phi(\cdot)$ takes item feature and bid as inputs and outputs a ranking score s_i: $\Phi : (\mathcal{F}, \mathcal{B}) \rightarrow \mathbb{R}$. The scoring function $\Phi(\cdot)$ can be a neural network or a tree model. (Both of the implementations are proposed in the experiment section.) For each query q, we compute the score s_i of each item i and the result list is generated by sorting their scores in descending order.

To elicit the objective for the scoring function, based on users' click through logs, we derive an unbiased metric of utility as

$$U'_q = \sum_{i=1}^{n_q} u(i, k_i) \tag{15}$$

where the utility $u(i, k_i)$ of displaying item i at position k_i is defined as

$$u(i, k_i) = \frac{P(C=1|\mathcal{F}=f_i, \mathcal{K}=k_i)}{P(C=1|\mathcal{F}=f_i, \mathcal{K}=k_i^h)}\cdot c_i \cdot b_i = \frac{g_\theta\left(f_i, k_i\right)}{g_\theta\left(f_i, k_i^h\right)}\cdot c_i \cdot b_i\ . \tag{16}$$

The reason that we choose U'_q instead of U_q is its resemblance to existing ranking metrics. We will leave the detailed discussion of such resemblance in Section 4.4, in which we also show the

unbiasedness of U'_q. With k_i^* being the optimal position assigned to item i, we define the regret of the utility as:

$$\mathcal{L}_r(\Phi, q) = \sum_{i=1}^{n_q} u(i, k_i^*) - \sum_{i=1}^{n_q} u(i, k_i) \tag{17}$$

which defines the objective of a ranking model. However, minimizing the regret of the utility $\mathcal{L}_r(\Phi, q)$ directly w.r.t the positions is infeasible since positions are discrete values. Therefore, we adapt the LambdaLoss framework [50] to learn a ranking model towards the optimal ranking by optimizing our proposed loss function (which will be presented in Eq. (19)) with iterative pairwise permutation. Like in LambdaLoss we follow an EM procedure where in E step we obtain the ranked list based on current scoring function $\Phi^{(t)}$ and in M step we re-estimate the scoring function $\Phi^{(t+1)}$ to minimize our loss function. The learning procedure of our learning algorithm is as follows (see line 19 to line 25 in Algorithm 1).

We first initialize the ranking model with a random initialization of $\Phi^{(0)}$. Inspired by the re-weighting technique used in LambdaRank [10], we compute the difference between the unbiased utility $\Delta Util(i, j)$ when the positions of two items i and j are flipped (see line 22 in Algorithm 1), as

$$\Delta Util(i, j) = u(i, k_j) + u(j, k_i) - u(i, k_i) - u(j, k_j)\ . \tag{18}$$

Then this difference value is used as the weight in the pairwise loss for each pair of items. Following [8, 10], we design our loss function in the form of logistic loss, as

$$\mathcal{L}'_r(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j:k_j<k_i} \Delta Util(i, j) \log_2 \left(1 + e^{-\sigma(s_i - s_j)}\right), \tag{19}$$

where k_i and k_j denote the position assigned to item i and j by ranking model at the last step (by the scoring function $\Phi^{(t)}$). This loss is minimized, so that we get a new scoring function $\Phi^{(t+1)}$ (see line 23 in Algorithm 1). Then we repeat the process until convergence.

Notice that in a standard LambdaLoss framework, the LambdaLoss is defined as

$$\mathcal{L}_\lambda(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j:y_i>y_j} |\Delta NDCG(i, j)| \log_2 \left(1 + e^{-\sigma(s_i - s_j)}\right)\ . \tag{20}$$

Note that the differences between our objective (19) and the LambdaLoss objective (20) lie in (1) the subscript of the summation symbol and (2) the absolute value symbol of the difference term Δ. In LambdaLoss framework, the absolute value of the difference term is used and the subscript of the summation symbol is $y_i > y_j$. The pairwise label of each item pair (i, j) is determined. The optimal ranking order is known to us by ranking all the items according to relevance label or click label, (denoted by y_i for item i), in descending order. However, in our framework, we cannot obtain an explicit label y_i for item i . An item is treated as the positive item if it is placed at a lower position by scoring function $\Phi^{(t)}$ and the exchange brings utility gain or it is placed at a higher position and the exchange introduces utility drop. We do not know the optimal ranking order in each query beforehand, where the optimal order is achieved through iterative pairwise permutation.

4.3 Theoretical Analysis

4.3.1 Sinkhorn. In this section, we theoretically prove that Sinkhorn algorithm maximizes the utility defined in Eq. (1), by relaxing permutation matrix to DSM.

Firstly, we define a family of functions $\Psi^* = \{\psi^*_{n_q} : q \in \mathcal{Q}\}$. Each function in this family takes the features of items $(f_i, b_i)_{i=1}^{n_q}$ as input and outputs a permutation matrix $\mathbf{S}$. It is denoted as

$\Psi^*_{n_q} : (\mathcal{F}, \mathcal{B})_{n_q} \rightarrow \mathcal{S}_{n_q}$, where $\mathcal{S}_{n_q}$ is the space of permutation matrices of dimension $n_q \times n_q$. To maximize the utility defined in Eq. (1), the loss function w.r.t Ψ^* is defined as

$$\begin{aligned}\mathcal{L}_s(\Psi^*, q) = -U_q &= -\sum_{i=1}^{n_q} P(C = 1|\mathcal{F} = f_i, \mathcal{K} = k_i) \cdot b_i \\ &= -\sum_{i=1}^{n_q}\sum_{k=1}^{n_q} \mathbf{S}_{i,k} P(C = 1|\mathcal{F} = f_i, \mathcal{K} = k) \cdot b_i \\ &= -\sum_{i=1}^{n_q}\sum_{k=1}^{n_q} \mathbf{S}_{i,k} \cdot g_\theta\left(f_i, k\right) \cdot b_i\end{aligned} \tag{21}$$

The loss $\mathcal{L}_s(\Psi^*, q)$ in Eq. (21) is not differentiable w.r.t. $\mathbf{S}_{i,k}$. Thus, Sinkhorn considers the expectation of this loss, which is computed by the marginal probability of $\mathbf{S}_{i,k} = 1$, as follows,

$$\mathbb{E}_{\Psi^*}[\mathcal{L}_s(q)] = -\sum_{i=1}^{n_q}\sum_{k=1}^{n_q} g_\theta\left(f_i, k\right) \cdot b_i \cdot \mathbb{E}_{\Psi^*}[\mathbf{S}_{i,k}] = -\sum_{i=1}^{n_q}\sum_{k=1}^{n_q} g_\theta\left(f_i, k\right) \cdot b_i \cdot \mathbf{W}_{i,k} \tag{22}$$

which means optimizing the loss function for Ψ defined in Eq. (12) is equivalent to optimizing the loss function for Ψ^* defined in Eq. (21), which is also equivalent to maximizing U_q.

4.3.2 Lambdaloss. In this section, we theoretically prove that the training objective $\mathcal{L}'_r(\Phi, q)$ is an upper bound of the utility regret $\mathcal{L}_r(\Phi, q)$. To make the proof easier to understand, we construct a function:

$$\mathcal{L}''_r(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j:s_i<s_j} |u(i, k_j) - u(i, k_i)|. \tag{23}$$

We start with several lemmas which will be used in our proof.

Lemma 4.1. *Given an indicator function $f(x) = \mathbb{I}(x \leq 0)$ and a function $g(x) = \log_2(1 + e^{-\sigma x})$ where σ is a constant in $\mathbb{R}$, it holds that $f(x) \leq g(x)$ for all $x \in \mathbb{R}$.*

Lemma 4.2. *Given an indicator function $f(x) = \mathbb{I}(x \geq 0)$ and a function $g(x) = \max\{\log(1 + \exp(\sigma C)), 2\} - \log_2(1 + e^{-\sigma x})$ where σ is a constant in $\mathbb{R}$, it holds that $f(x) \leq g(x)$ for $x \in [-C, C]$.*

Lemma 4.3. *Given a sum function $f(x) = \sum_i x_i$ and a max function $g(x) = \max_i x_i$, it holds that $f(x) \geq g(x)$ for $x_i \geq 0, \forall i$.*

Now we are ready to derive the main theoretical result.

Theorem 4.4. *Assume the utility function $u(i, k_i)$ is a monotonic decreasing function w.r.t k_i and the ranking score s_i is bounded in the range of [-C,C]. Let $C_1 = max\{\log(1 + \exp(2\sigma C)), 2\}$ and $C_2 = C1 \cdot \sum_{j=1}^{n_q} \sum_{i:k_i>k_j} (u(j, k_j) - u(j, k_i))$. Then we have $\mathcal{L}''_r(\Phi, q) \leq \mathcal{L}'_r(\Phi, q) + C_2$.*

Proof.

$$\begin{aligned}
\mathcal{L}_r''(\Phi, q) &= \sum_{i=1}^{n_q}\sum_{j=1}^{n_q} |u(i,k_j) - u(i,k_i)|\mathbb{I}(s_i < s_j) \\
&= \sum_{i=1}^{n_q}\sum_{j:k_j<k_i} (u(i,k_j) - u(i,k_i))\mathbb{I}(s_i < s_j) + \sum_{i=1}^{n_q}\sum_{j:k_j>k_i} (u(i,k_i) - u(i,k_j))\mathbb{I}(s_i < s_j) \\
&= \sum_{i=1}^{n_q}\sum_{j:k_j<k_i} (u(i,k_j) - u(i,k_i))\mathbb{I}(s_i < s_j) + \sum_{j=1}^{n_q}\sum_{i:k_i>k_j} (u(j,k_j) - u(j,k_i))\mathbb{I}(s_j < s_i) \\
&\leq \sum_{i=1}^{n_q}\sum_{j:k_j<k_i} (u(i,k_j) - u(i,k_i)) \log_2\left(1 + e^{-\sigma(s_i - s_j)}\right) \\
&\quad - \sum_{i=1}^{n_q}\sum_{j:k_j<k_i} (u(j,k_j) - u(j,k_i))[\log_2\left(1 + e^{-\sigma(s_i - s_j)}\right) + C_1] \\
&= \sum_{i=1}^{n_q}\sum_{j:k_j<k_i} (u(i,k_j) - u(i,k_i) - u(j,k_j) + u(j,k_i)) \log_2(1 + e^{-\sigma(s_i - s_j)}) + C_2 \\
&= \sum_{i=1}^{n_q}\sum_{j:k_j<k_i} \Delta Util(i,j) \log_2\left(1 + e^{-\sigma(s_i - s_j)}\right) + C_2 \qquad (24) \\
&= \mathcal{L}_r'(\Phi, q) + C_2 \qquad (25)
\end{aligned}$$

where the inequality holds due to Lemma 4.1 and Lemma 4.2. □

Theorem 4.4 states that $\mathcal{L}_r''(\Phi, q)$ is upper bounded by our objective $\mathcal{L}_r'(\Phi, q)$ plus C_2. C_2 is a constant since in the M step C_2 only depends on the current scoring function $\Phi^{(t)}$. Notice that the assumptions in the theorem are not restrictive in practice. As illustrated in Figure 2, the real utility basically satisfies the monotonic decreasing assumption. Moreover, the ranking score is often clipped in implementation to avoid explosion in exponential function.

Theorem 4.5. *Assume the utility $u(i,k_i)$ is a monotonic decreasing function w.r.t k_i. Then $\mathcal{L}_r(\Phi, q)$ is upper bounded by $\mathcal{L}_r''(\Phi, q)$.*

Proof.

$$\begin{aligned}
\mathcal{L}_r''(\Phi, q) &= \sum_{i=1}^{n_q}\sum_{j:s_i<s_j} |u(i,k_j) - u(i,k_i)| = \sum_{i=1}^{n_q}\sum_{k=1}^{k_i-1} |u(i,k) - u(i,k_i)| \\
&= \sum_{i=1}^{n_q}\sum_{k=1}^{k_i-1} (u(i,k) - u(i,k_i)) \geq \sum_{i=1}^{n_q} u(i,1) - \sum_{i=1}^{n_q} u(i,k_i) \\
&\geq \sum_{i=1}^{n_q} u(i,k_i^*) - \sum_{i=1}^{n_q} u(i,k_i) = \mathcal{L}_r(\Phi, q) \qquad (26)
\end{aligned}$$

where the first inequality holds due to Lemma 4.3. □

Theorem 4.6. *Under the assumption of Theorem 4.4 and Theorem 4.5, we have that $\mathcal{L}_r(\Phi, q) \leq \mathcal{L}_r'(\Phi, q) + C_2$.*

The proof of Theorem 4.6 is trivial due to Theorem 4.4 and Theorem 4.5. Theorem 4.6 demonstrates that the utility regret $\mathcal{L}_r(\Phi, q)$ is bounded by our proposed objective $\mathcal{L}_r'(\Phi, q)$ plus a constant. It

Table 1. Comparison of learning metrics in previous works.

	Discount factor $\mathcal{D}_i$	Propensity $\mathcal{Q}_i$	Benefit b_i
SVMRank [23]	$-k_i$	1	1
PropSVMRank [27]	$-k_i$	$\theta_{k_i^h}$	1
PBM [49]	$-k_i$	$\theta_{k_i^h}$	1
CPBM [17]	$-k_i$	$f(k_i^h, q)$	1
LambdaRank (NDCG) [10]	$1/\log(k_i+1)$	1	1
Sponsored search [55, 56]	1	1	b_i
U-rank+	$f(k_i, q, i)$	$f(k_i^h, q, i)$	b_i

implies that optimizing our proposed objective is actually minimizing the upper bound of the utility regret, which guarantees the effectiveness of our ranking algorithm theoretically.

4.4 Relations to Previous Works

Many existing relevance based ranking metrics [54] approximate the utility in Eq. (1) by the inner product of a relevance vector $\mathcal{J}$ and a rank discount vector $\mathcal{D}$. For simplicity, a binary relevance label, $r_i \in \{0, 1\}$ is typically considered. Then, the mainstream relevance based metric/objective function is in the following form

$$\text{Metric} = \sum_{i=1}^{n_q} \mathcal{D}_i \cdot r_i \,, \tag{27}$$

where the discount factor $\mathcal{D}_i$ is normally a decreasing function of position k_i (note that no item features are considered), indicating users' decreasing attention from top to bottom positions of a list. For example, in DCG, $\mathcal{D}_i = 1/log(k_i + 1)$ and in Prec@K, $\mathcal{D}_i = \mathbb{I}\{k_i \leq K\}/K$. The discount factor $\mathcal{D}_i$ is often interpreted as the examination probability at the displayed position, *i.e.*, $P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)$ where $\mathcal{O}$ is a binary variable denoting whether the item i is examined at position k_i. The underlying assumption of these metrics is the examination hypothesis [41] that a user clicks an item only when it is examined and relevant, *i.e.*,

$$P(\mathcal{C} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i) = P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i) \cdot P(r_i = 1) \,. \tag{28}$$

Moreover, in order to facilitate learning from users' click through logs, counterfactual learning to rank methods [22, 27, 48, 49] address the mismatch between the binary relevance r_i and click feedback c_i in the historical data with inverse propensity weighting. The propensity $\mathcal{Q}_i$ used in counterfactual learning to rank methods also refers to an examination probability. Different from the discount factor, here the examination probability refers to that of item i displayed at position k_i^h in the logging data, *i.e.*, $P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)$.

According to the general setting of counterfactual learning, an estimate of Metric is defined as

$$\text{Metric}_{IPS} = \sum_{i=1}^{n_q} \frac{\mathcal{D}_i \cdot c_i}{\mathcal{Q}_i} \,, \tag{29}$$

which is essentially an unbiased estimation of Eq. (27) since

$$\begin{aligned} \mathbb{E}[\text{Metric}_{IPS}] &= \sum_{i=1}^{n_q} \frac{\mathcal{D}_i \cdot P(\mathcal{C} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)}{P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)} \\ &= \sum_{i=1}^{n_q} \frac{\mathcal{D}_i \cdot P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h) \cdot r_i}{P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)} = \sum_{i=1}^{n_q} \mathcal{D}_i \cdot r_i \,. \end{aligned} \tag{30}$$

More generally, if we further consider the benefit b_i of each item, then we have an unbiased metric of utility based on implicit feedbacks as

$$\text{Metric} = \sum_{i=1}^{n_q} \frac{\mathcal{D}_i}{\mathcal{Q}_i} \cdot c_i \cdot b_i \,, \tag{31}$$

where discount factor $\mathcal{D}_i$ and propensity $\mathcal{Q}_i$, as we mentioned before, correspond to $P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)$ and $P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)$, respectively. Following the general form in Eq. (31), we summarize and compare several existing (counterfactual) learning to rank methods in Table 1.

The main difficulty of estimating the utility metric in Eq. (1) from the users' click through logs lies in the estimation of the underlying examination probability $P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)$, since we do not have any direct supervised signal telling whether the item i is observed by the user at position k_i. Due to the lack of supervision, strong assumptions are made on $P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)$ in different models to make the propensity estimation feasible. For instance, in PropSVMrank [27] and PBM [49], the propensity is only related to position k_i; in CPBM [17], the propensity is related to both position k_i and query q. Still, we need to meet other strict requirements like online randomization [27] or multiple rankers [17, 49]. Jointly learning a propensity model with a relevance model merely from click logs [4, 22] does not need to meet these requirements, but is also challenging. In such methods, the propensity estimation and the relevance estimation strongly depend on the unbiasedness of each other. Due to the lack of direct supervision, there is no clear evidence to demonstrate the unbiasedness of either model.

In this work, we do not rely on the separate estimation of relevance and propensity, which simplify the problem. Thus, we can estimate a much more loose assumption that the examination probability is a function of position k_i and feature f_i (containing user/query features, item features and context features). Binding the estimation of discount factor with the estimation of propensity, we notice that

$$\frac{\mathcal{D}_i}{\mathcal{Q}_i} = \frac{P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)P(r_i = 1)}{P(\mathcal{O} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)P(r_i = 1)} = \frac{P(\mathcal{C} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)}{P(\mathcal{C} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)} \,. \tag{32}$$

Since the relevance term r_i is removed in Eq. (32), without relying on an accurate relevance estimation, a ratio between discount factor and propensity can be obtained. We only need to estimate the click probability of the item i at position k_i. Based on users' click through logs, we derive an unbiased metric of utility, inspired by Eq. (31) and Eq. (32), as

$$U_q' = \sum_{i=1}^{n_q} \frac{P(\mathcal{C} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i)}{P(\mathcal{C} = 1|\mathcal{F} = f_i, \mathcal{K} = k_i^h)} \cdot c_i \cdot b_i \,. \tag{33}$$

This is exactly the objective we optimize for in Section 4.2.2.

5 SEMI-SYNTHETIC EXPERIMENTS

The semi-synthetic setup is widely applied in the community of counterfactual learning to rank [4, 17, 22, 27] as it allows us to explore a range of different settings. In this section, we will evaluate the performance of our proposed approaches and the baseline approaches with three semi-synthetic benchmark datasets[5].

5.1 Datasets

We base the semi-synthetic experiments on three real-world benchmark datasets. The detailed description of these three datasets is as below.

[5]The code with running instructions for our experiments is available at https://bit.ly/32J1pSF

- **Yahoo! LETOR set 1**[6] is the dataset used in Yahoo! Learning-to-Rank Challenge. It contains 29,921 queries with 710k documents. The 700 features are extracted from query-URL pairs with all the features normalized to be in the [0,1] range. The relevance judgments can take five different values from 0 (irrelevant) to 4 (perfectly relevant).
- **MSLR-WEB10K**[7] is a large-scale dataset released by Microsoft Research Asia in May 2010. Microsoft datasets contains 10,000 queries and 1,200,193 documents. There are in total 136 features extracted from query-URL pairs. The relevance judgements are obtained from a retired labeling set of Microsoft Bing search engine, which take five values from 0 (irrelevant) to 4 (perfectly relevant).
- **Istella-SLETOR**[8] [32] is released by Istella in 2016, which is one of the largest public available datasets. Istella-S is composed of 33,018 queries and 220 features representing each query-document pair. The average examples in one query is 104. The relevance judgement ranges from 0 (irrelevant) to 4 (perfectly relevant).

5.2 Click Data Generation

We mainly follow Fang et al. [17] to generate synthetic click data with item-wise examination probability for Yahoo! LETOR, MSLR-WEB10K and Istella-S LETOR datasets. In the following part, *oracle model* refers to this click generation model. First of all, we follow the given train/validation/test splits of the three datasets, and queries without relevant documents are filtered. Following [17], to generate the initial positions of the items, we learn two ranking models by running SVMRank [23] on two small randomly sampled subsets of the queries in the training data. Specifically, the two ranking models are trained with 22 shared queries and 92 distinguished queries. Then, the initial positions of the items for the remaining queries is generated by the two learned ranking models. Each query-item pair in the remaining queries corresponds to two initial positions, generated by the two ranking model, respectively. Note that two initial ranking models are required in [17] but not in our method. We follow this setting for fair comparison. The maximal position k_{max} is set to be 10 in our experiments.

Following [17], the examination probability which is related to both position and the item is calculated by $P(o_{i,k_i} = 1|k_i, x_i) = \frac{1}{k_i^{\max(w \cdot x_i + 1, 0)}}$. In our setting, x_i is set of item features, while in the setting of [17], x_i is the set of query features which is shared among all the items for a same query. The parameter vector w is drawn from a uniform distribution over $[-\eta, \eta)$ and is normalized such that $\sum_{j=1} w_j = 0$. The parameter η controls how the examination probability varies with context.

The relevance probability is defined as $P(r_i = 1) = \epsilon + (1 - \epsilon)\frac{2^{y_i} - 1}{2^{y_{\max}} - 1}$ (following [22]), where y_i denotes the relevance label of x_i and y_{max} is the highest level of relevance. ϵ is set to 0.1, which denotes the click probability of irrelevant documents.

5.3 Baselines

We implement eight baselines that explore the performance of two standard learning to rank methods (i.e., **SVMRank** [23] and **LambdaRank** [10]), with four propensity estimation methods, which are detailed as follows.

- **None** uses the original click data without debiasing.
- **Randomization** [27] estimates propensity with online randomized experiments.
- **CPBM** [17] estimates examination probability w.r.t different queries based on intervention harvesting.

[6] https://webscope.sandbox.yahoo.com
[7] https://www.microsoft.com/en-us/research/project/mslr/
[8] http://quickrank.isti.cnr.it/istella-dataset/

Table 2. Comparison of different (counterfactual) learning to rank models on Yahoo! LETOR set 1.

Ranking model		Relevance-based					Utility-based	
		MAP	nDCG@1	nDCG@3	nDCG@5	nDCG@10	# Click	CTR
SVMRank	None	0.702	0.653	0.680	0.728	0.845	0.599	0.0641
	Randomization	0.639	0.498	0.571	0.640	0.787	0.533	0.0573
	CPBM	0.707	0.661	0.689	0.735	0.849	0.599	0.0645
	Groundtruth	*0.718*	*0.680*	*0.709*	*0.752*	*0.859*	*0.612*	*0.0655*
LambdaRank	None	0.707	0.665	0.691	0.735	0.850	0.608	0.0648
	Randomization	0.680	0.605	0.650	0.703	0.828	0.582	0.0621
	CPBM	0.718	0.683	0.703	0.747	0.857	0.613	0.0651
	Groundtruth	*0.719*	*0.684*	*0.709*	*0.751*	*0.859*	*0.618*	*0.0657*
DNN	DLA	0.665	0.500	0.582	0.654	0.792	0.593	0.0587
	CTR-1	0.647	0.499	0.581	0.650	0.792	0.552	0.0577
U-rank		0.721*	0.692*	0.714*	0.755*	0.862*	0.614	0.0659*
U-rank+$_{sinkhorn}$		0.714	0.685	0.703	0.743	0.856	0.619*	0.0657*
U-rank+$_{lambda}$		**0.722***	**0.694***	**0.716***	**0.758***	**0.863***	**0.632***	**0.0662***
KM (oracle model)		*0.935*	*0.981*	*0.986*	*0.974*	*0.988*	*0.697*	0.0737

∗ denotes statistically significant improvement (t-test with p-value<0.05) over all baselines (except *Groundtruth*).

- **Groundtruth** uses the groundtruth examination probability for *oracle model* as propensity. The result of this method is the upper bound of the results of other IPS approaches based on the same ranking model.

Other methods that we include for comparison are as follows.

- **CTR-1** [20] is the position-aware click model used in our framework which assigns position 1 to each item during online inference.
- **DLA** [4] is a dual learning algorithm that jointly learns an unbiased ranker and an unbiased propensity model. The ranker is a deep neural network.

We also explore the performance of **KM (oracle model)** which solves the maximum-weight graph matching problem via Kuhn-Munkres (KM) algorithm with $O(N^3)$ time complexity [28, 36], given the groundtruth click probability . It is supposed to produce the best utility that can be achieved on the testing data. As for our method, ***U-rank*** is the preliminary version of this work (published in [14]), which is a lambdaloss based implementation without the debiasing of selection bias. In this work, we further debias the selection bias and propose ***U-rank+***$_{\textbf{sinkhorn}}$ and ***U-rank+***$_{\textbf{lambda}}$ based on the sinkhorn solver and lambdaloss solver, respectively. For fair comparison, the DNN based ranking models adopt the same network architecture. We use a fully-connected neural network with 4 hidden layers, where the hidden sizes are 1024,1024,512,50, respectively.

5.4 Overall Performance

In this section, we assume the benefit of each item brought to the system as 1 in order to consistently and fairly compare *U-rank+* with existing (counterfactual) learning to rank methods that rank the items according to the relevance they estimate. We evaluate the performance of the baseline approaches and our proposed model in terms of the relevance-based metrics, i.e., *MAP* and *nDCG* (including nDCG@1, nDCG@3, nDCG@5, nDCG@10), and utility-based metrics, i.e., *# Click* and *CTR*. Here, *# Click* and *CTR* are utility metrics based on oracle click model denoting *number of clicks per query* and *click probability per document*, respectively. Specifically, *# Click* is average number of clicks over the test queries where the click are sampled once for each query-item pair. *CTR* is the average of the exact click probability for each query-item pair on the test set.

The overall performance on the three benchmark datasets is shown in Table 2, Table 3, and Table 4. From the tables we have the following observations:

Table 3. Comparison of different (counterfactual) learning to rank models on MSLR- WEB10K.

Ranking model		Relevance-based					Utility-based	
		MAP	nDCG@1	nDCG@3	nDCG@5	nDCG@10	# Click	CTR
SVMRank	None	0.496	0.443	0.509	0.570	0.735	0.818	0.0827
	Randomization	0.433	0.351	0.416	0.484	0.686	0.777	0.0799
	CPBM	0.494	0.434	0.502	0.564	0.732	0.827	0.0823
	Groundtruth	*0.513*	*0.481*	*0.525*	*0.588*	*0.747*	*0.863*	*0.0871*
LambdaRank	None	0.500	0.456	0.508	0.569	0.736	0.829	0.0830
	Randomization	0.451	0.380	0.440	0.507	0.700	0.813	0.0808
	CPBM	**0.508**	**0.461**	**0.515**	**0.580**	**0.740**	0.836	0.0836
	Groundtruth	*0.516*	*0.482*	*0.525*	*0.588*	*0.747*	*0.891*	*0.0886*
DNN	DLA	0.457	0.373	0.454	0.522	0.705	0.823	0.0828
	CTR-1	0.476	0.425	0.478	0.545	0.722	0.829	0.0814
U-rank		0.492	0.428	0.484	0.554	0.724	0.906*	0.0915*
U-rank+$_{sinkhorn}$		0.473	0.398	0.463	0.532	0.712	0.928*	0.0918*
U-rank+$_{lambda}$		0.479	0.416	0.468	0.540	0.716	**0.951***	**0.0921***
KM (oracle model)		*0.711*	*0.763*	*0.795*	*0.809*	*0.881*	*0.972*	*0.0969*

∗ denotes statistically significant improvement (t-test with p-value<0.05) over all baselines (except *Groundtruth*).

Table 4. Comparison of different (counterfactual) learning to rank models on Istella-SLETOR.

Ranking model		Relevance-based					Utility-based	
		MAP	nDCG@1	nDCG@3	nDCG@5	nDCG@10	# Click	CTR
SVMRank	None	0.769	0.613	0.626	0.681	0.806	0.936	0.0937
	Randomization	0.742	0.572	0.591	0.649	0.787	0.912	0.0911
	CPBM	0.770	0.619	0.630	0.684	0.808	0.939	0.0938
	Groundtruth	*0.775*	*0.639*	*0.646*	*0.695*	*0.816*	*0.953*	*0.0952*
LambdaRank	None	0.773	0.623	0.631	0.685	0.810	0.940	0.0941
	Randomization	0.747	0.584	0.597	0.652	0.790	0.920	0.0917
	CPBM	0.776	0.631	0.638	0.691	0.813	0.945	0.0946
	Groundtruth	*0.781*	*0.633*	*0.642*	*0.695*	*0.815*	*0.948*	*0.0948*
DNN	DLA	0.690	0.363	0.435	0.515	0.703	0.840	0.0836
	CTR-1	0.733	0.537	0.562	0.620	0.771	0.895	0.0895
U-rank		**0.782***	0.637*	0.643*	0.695	0.815	0.952*	0.0953*
U-rank+$_{sinkhorn}$		0.776	**0.640***	0.642*	0.693	0.815	0.954*	0.0952*
U-rank+$_{lambda}$		0.780*	0.637*	**0.646***	**0.697***	**0.816**	**0.957***	**0.0954***
KM (oracle model)		*0.993*	*0.993*	*0.995*	*0.995*	*0.995*	*1.128*	*0.1126*

∗ denotes statistically significant improvement (t-test with p-value<0.05) over all baselines (except *Groundtruth*).

- Our methods, including *U-rank*, *U-rank+$_{lambda}$*, and *U-rank+$_{sinkhorn}$*, achieve consistently the best performance than the state-of-the-art baseline approaches on the utility-oriented metrics, i.e., *#Click* and *CTR*. For example, *U-rank+$_{lambda}$* achieves 1.7% improvement in Yahoo! LETOR set 1 and 10.2% improvement in MSLR-WEB10K on *CTR*, compared to the best baseline methods (the baseline in italic use the information from oracle click model, so they are not included for comparison). Besides, *U-rank+$_{lambda}$* improves substantially over *U-rank* on the utility-oriented metrics on those three benchmark datasets, which demonstrates the effectiveness of debiasing selection bias. Compared to lambdaloss-based *U-rank+$_{lambda}$* method, *U-rank+$_{sinkhorn}$* performs slightly worse.
- *U-rank*, *U-rank+$_{lambda}$*, and *U-rank+$_{sinkhorn}$* also outperform most of the baselines in terms of the relevance-based metrics, i.e., MAP and nDCG, and *U-rank+$_{lambda}$* performs best among those three methods. Though it does not always perform the best, especially on MSLR-WEB10K dataset where CPBM generates the best ranking w.r.t. MAP and nDCG. We also observe that even the KM algorithm, which computes the best item-position matching with the oracle examination

probabilities, does not achieve a high MAP/nDCG on MSLR-WEB10K. This observation shows that a model optimizing the utility-based metrics does not always optimize a relevance-based ranking metric, which indicates that traditional ranking metrics such as nDCG are not proper utility metrics. However, on Yahoo! LETOR set 1 and Istella-S LETOR, where the disagreement is smaller, KM algorithm achieves almost the highest MAP and nDCG and our methods also perform well on relevance-based metrices.

- The method *Groundtruth* achieves the best utility among the counterfactual learning approaches, which demonstrates the effectiveness of the IPS-based framework when the propensity estimation is *accurate*. Randomization does not perform well because it assumes that the examination probability only relates to the position, which is not valid in our setting where the examination probability relies on both the position and the item features. CPBM achieves the second-best utility among IPS-based methods since it models the propensity by considering both position and query features.
- *U-rank* and CTR-1 share the same click model. However, *U-rank* outperforms CTR-1 mainly because CTR-1 ranks items by their estimated click probability at position 1, which is suboptimal in case of item-wise examination probability. Our methods also outperform DLA since DLA relies heavily on the accuracy of estimated propensity, which is hard to achieve.

5.5 Empirical Analysis

In this section, we conduct empirical analysis to answer the following research questions:

RQ1 How effective is the position propensity model in reducing selection bias?
RQ2 Can our proposed methods deal with position bias?
RQ3 How do our methods conduct direct revenue maximization?
RQ4 Does our framework generalize to different ranking model architectures?

5.5.1 RQ1: How effective is the position propensity model in reducing selection bias? In this section, we study the effect of position propensity model in reducing selection bias. We use two kinds of test datasets, biased and random datasets. The initial ranked lists of the biased test set are generated from two same ranking models as in the training set while the initial ranked lists of the random test set are randomly shuffled. Then the same click generation procedure as in Section 5.2 is used to generate clicks based on the given ranked lists in the two kinds of datasets. We train *U-rank* and *U-rank+*$_{lambda}$ on the same (biased) training set and then evaluate on the two kinds of test datasets. The evaluation is based on two metrics: AUC (Area Under the ROC curve) and Logloss (cross entropy). Usually, higher AUC and lower Logloss indicate better performance. Both the click model and position propensity model are fully-connected neural networks. The network architecture are the same as the ranking model.

The performance of *U-rank* and *U-rank+*$_{lambda}$ on those two kinds of test datasets is shown in Table 5. *U-rank+*$_{lambda}$ and *U-rank* achieve similar performance on biased test set; however, *U-rank+*$_{lambda}$ works consistently better than *U-rank* on random test set, which verifies that *U-rank+*$_{lambda}$ is less affected by selection bias while still maintains a reasonable accuracy rate.

Table 5. Comparison of different click models on three datasets.

Model	Metric	MSLR-WEB10K		Yahoo! LETOR set 1		Istella-SLETOR	
		biased	random	biased	random	biased	random
U-rank	AUC	0.6974	0.7456	0.7830	0.7463	0.7563	0.6711
	Logloss	0.2699	0.1999	0.2019	0.2001	0.2695	0.1865
U-rank+$_{lambda}$	AUC	0.6973	0.7464	0.7831	0.7475	0.7566	0.6725
	Logloss	0.2699	0.1994	0.2020	0.1998	0.2697	0.1839

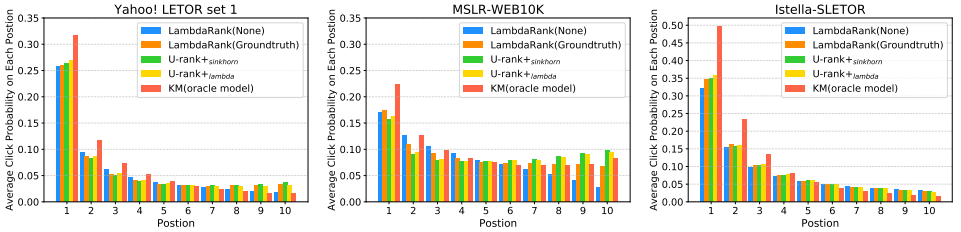

Fig. 5. Average click probability on each position.

5.5.2 RQ2: Can our proposed methods deal with position bias? In Figure 5, we show the average click probability on each position of *U-rank+*$_{lambda}$, *U-rank+*$_{sinkhorn}$, and LambdaRank based approaches since they are the best baselines on average in terms of utility as shown in Table 2 and Table 3. We also plot the results of *KM (oracle model)* for reference.

Comparing the results of the three datasets in Figure 5, we observe a steeper decline of average click probability to positions of the KM (oracle model) method on the Yahoo! and Istella datasets than that on the MSLR dataset. It means that the optimal matching tends to display the most relevant items at the Yahoo! and Istella datasets' top positions, which suggests that positions have a powerful impact on users' clicks in these two datasets. Thus, to optimize the utility, a well-performed approach should put more relevant items at higher positions. In MSLR-WEB10K, on the other hand, we find that the average click probability of the optimal matching tends to be equally distributed on the positions, compared to the Yahoo! and Istella datasets. We find that *our methods are adaptive to different severity of position bias*. In the Yahoo! and Istalla datasets, our models, *U-rank+*$_{lambda}$ and *U-rank+*$_{sinkhorn}$, focus more on top positions than LambdaRank, while in the MSLR dataset, our models learn a flatter distribution. Notably, our models achieve a more considerable sum of click probabilities in these three datasets over all the positions than LambdaRank.

5.5.3 RQ3: How do our methods conduct direct revenue maximization? We analyze the result of a single query in detail. The experiment is conducted on the first query of the MSLR-WEB10K dataset. Figure 6 shows the click probabilities of the ten items for this query and their click probabilities if placed at each position according to our oracle click data generation model. The position of each item assigned by different methods is denoted in orange color. We can see that although LambdaRank performs better in nDCG with a groundtruth propensity. It, however, achieves a lower click probability than our methods, *U-rank+*$_{lambda}$ and *U-rank+*$_{sinkhorn}$. This is because similar to the KM (oracle model), *our methods will take the position sensitivity of different items into consideration*. For example, document 6 is of high relevance and relatively not sensitive to the position change. LambdaRank displays it in the second position while our methods and KM both display it at a lower position so that the second position is kept for an item that is more sensitive to the position change.

5.5.4 RQ4: Does our framework generalize to different ranking model architectures? In our initial implementation, the ranking model is a neural network. To study the influence of different ranking model architectures, we change our ranking model from a NN-based model to a Tree-based model. We train the *U-rank*(Tree) and *U-rank+ (Tree)* with a similar procedure in LambdaMART [9] according to our loss function. Note that here *U-rank+ (Tree)* is implemented with the lambdaloss solver. As presented in Table 6, Table 7, and Table 8, *U-rank+ (Tree)* works consistently better than LambdaMART w.r.t. utility-based metrics on three datasets. Our methods also outperform all baselines in terms of the relevance-based metrics on Yahoo! and Istella datasets, which demonstrates

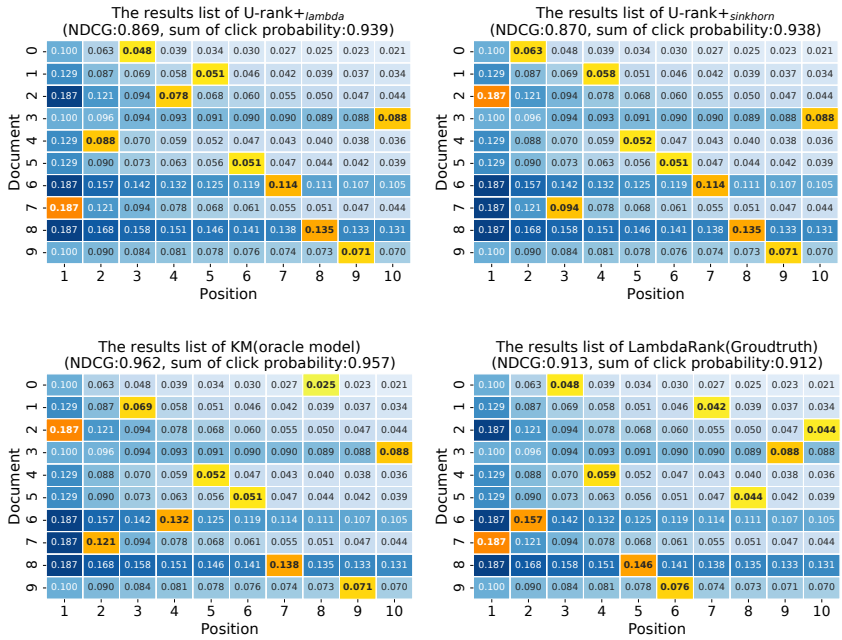

Fig. 6. Comparison of the result lists of different methods on the first query of MSLR-WEB10K.

Table 6. Comparison of different Tree-based models on Yahoo! LETOR set 1.

Ranking model		Relevance-based					Utility-based	
		MAP	nDCG@1	nDCG@3	nDCG@5	nDCG@10	# Click	CTR
LambdaMART	None	0.714	0.694	0.707	0.747	0.657	0.611	0.0655
	Randomization	0.701	0.679	0.689	0.732	0.850	0.606	0.0650
	CPBM	0.717	**0.698**	0.709	0.749	0.859	0.615	0.0657
	Groundtruth	*0.718*	*0.696*	*0.710*	*0.749*	*0.859*	*0.619*	*0.0658*
U-rank (Tree)		0.720	0.692	**0.714***	0.756*	0.862	0.621*	0.0659
U-rank+ (Tree)		**0.722***	0.694	**0.714***	**0.757***	**0.863**	**0.622***	**0.0660***
KM (oracle model)		*0.935*	*0.981*	*0.986*	*0.974*	*0.988*	*0.697*	0.0737

$*$ denotes statistically significant improvement (measured by t-test with p-value<0.05) over all baselines.

Table 7. Comparison of different Tree-based models on MSLR-WEB10K.

Ranking model		Relevance-based					Utility-based	
		MAP	nDCG@1	nDCG@3	nDCG@5	nDCG@10	# Click	CTR
LambdaMART	None	0.509	**0.493**	0.528	0.581	0.747	0.845	0.0835
	Randomization	0.501	0.476	0.516	0.573	0.741	0.835	0.0826
	CPBM	**0.517**	0.492	**0.530**	**0.589**	**0.749**	0.845	0.0835
	Groundtruth	*0.520*	*0.495*	*0.538*	*0.592*	*0.752*	*0.851*	*0.0844*
U-rank (Tree)		0.474	0.405	0.467	0.537	0.716	0.912*	0.0909*
U-rank+ (Tree)		0.479	0.412	0.471	0.538	0.717	**0.916***	**0.0912***
KM (oracle model)		*0.710*	*0.763*	*0.795*	*0.809*	*0.881*	*0.972*	*0.0969*

$*$ denotes statistically significant improvement (measured by t-test with p-value<0.05) over all baselines.

the generalizability of our methods, taking both NN and Tree models as the ranking model. Though our models do not always perform the best in regard to relevance-based metrics, especially on MSLR-WEB10K dataset, it accords with the results in Table 3 and it is explainable. Besides, *U-rank+*

Table 8. Comparison of different Tree-based models on Istella-SLETOR.

Ranking model		Relevance-based					Utility-based	
		MAP	nDCG@1	nDCG@3	nDCG@5	nDCG@10	# Click	CTR
LambdaMART	None	0.772	0.609	0.619	0.675	0.803	0.932	0.0931
	Randomization	0.773	0.617	0.628	0.682	0.807	0.943	0.0936
	CPBM	0.775	0.623	0.630	0.686	0.809	0.930	0.0938
	Groundtruth	*0.780*	*0.627*	*0.637*	*0.691*	*0.812*	*0.938*	*0.0941*
U-rank (Tree)		0.773	0.623	0.624	0.678	0.806	0.942	0.0947*
U-rank+ (Tree)		**0.778**	**0.629***	**0.640***	**0.690**	**0.814***	**0.970***	**0.0952***
KM (oracle model)		*0.993*	*0.993*	*0.995*	*0.995*	*0.995*	*1.128*	*0.1126*

$*$ denotes statistically significant improvement (measured by t-test with p-value<0.05) over all baselines.

(Tree) also outperforms *U-rank (Tree)* on both relevance-based metrics and utility-based metrics, which indicates the effectiveness of debiasing selection bias.

6 ONLINE A/B TESTING

In order to verify the effectiveness of our proposed model in real-world applications, we conduct experiments on the data from a recommendation scenario and an online advertising scenario in a mainstream App store. This App store has hundreds of millions of daily active users who create hundreds of billions of user logs everyday in the form of implicit feedback such as browsing, clicking and downloading behaviors.

6.1 Recommendation Scenario

6.1.1 Setups. We conduct A/B testing in a recommendation scenario in a mainstream App store with multiple sub-scenarios such as "Must-have Apps" and "Novel and Fun". The proposed model *U-rank* is compared with the current production baseline DeepFM [19]. The whole online experiment lasts 24 days, from May 6, 2020 to May 29, 2020. We monitor the results of A/A testing for the first seven days, conduct A/B testing for the following ten days, and conduct A/A testing again in the last seven days. 15% of the users are randomly selected as the experimental group and another 15% of the users are in the control group. During A/A testing, all the users are served by DeepFM model [19]. During A/B testing, users in the control group are presented with recommendation by DeepFM, while users in the experimental group are presented with the recommendation by our proposed model *U-rank*. Note that the click model of *U-rank* shares the same network architecture and parameter complexity with DeepFM in order to verify whether the improvement is brought by the objective function design of the ranker in *U-rank*.

To deploy *U-rank*, we utilize a single node with 48 core Intel Xeon CPU E5-2670 (2.30 GHZ), 400 GB RAM and as well as 2 NVIDIA TESLA V100 GPU cards, which is the same as the training environment of the baseline DeepFM. For model training, *U-rank* requires minor changes to the current training procedure due to the pair-wise loss function. For model inference, *U-rank* shares the same pipeline as DeepFM, which means there is no extra engineering work needed in model inference, to upgrade DeepFM model (or other similar deep models) to *U-rank*.

6.1.2 Metrics. We examine two metrics in the online evaluation. They are *Click-through rate*: $CTR = \frac{\#downloads}{\#impressions}$ and *Conversion rate*: $CVR = \frac{\#downloads}{\#users}$, where # downloads, # impressions and #users are the number of downloads, impressions and visited users, respectively.

6.1.3 Results. Figure 7 and Figure 8 show the improvement of the experimental group over the control group w.r.t. CTR and CVR, respectively. We can see that the system is rather stable where both CTR and CVR fluctuated within 8% during the A/A testing. Our *U-rank* model is launched to the live system on Day 8. From Day 8, we observe a significant improvement over the baseline model

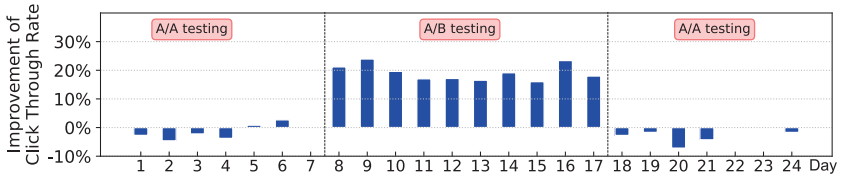

Fig. 7. Online experimental results of click through rate (recommendation scenario).

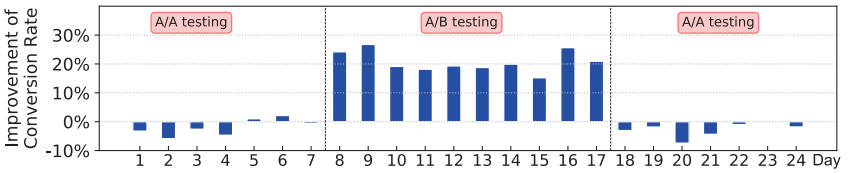

Fig. 8. Online experimental results of conversion rate (recommendation scenario).

w.r.t. both CTR and CVR. The average improvement of CTR is 19.2% and the average improvement of CVR is 20.8% over the ten days of A/B testing. These results clearly demonstrate the high effectiveness of our proposed model in improving the total utility which refers to the number of downloads in this scenario. From Day 18, we conduct A/A testing again to replace our *U-rank* model with the baseline model in the experimental group. We observe a sharp drop in the performance of the experimental group, which once more verify that the improvement of online performance in the experimental group is indeed introduced by our proposed model.

6.2 Online Advertising Scenario

6.2.1 Setups. We also conduct A/B testing in an online advertising scenario "Boutique Apps" in the same App store. In this scenario, each App is related to a benefit which is corresponding to the price the campaign would pay to the platform if this App is downloaded once. The performance of a model would be evaluated by how much profit it brings to the platform. In order to compare the proposed model *U-rank+* with the current production baseline model $\mathscr{A}$[9], we conduct online experiment for 25 days, from October 21, 2020 to November 14, 2020. We monitor the results of A/A testing for the first nine days, conduct A/B testing for the following sixteen days. The A/B testing consists of several phases. In the beginning, 5% of the users are randomly selected as the experimental group on October 30 (Day 10 in Figure 9). As we observe the positive performance of *U-rank+*, we gradually increase the traffic of the experimental group to 17% (Day 15 in Figure 9) and to 20% (Day 19 in Figure 9). Another 20% of the users are randomly selected as the control group. During A/A testing, all the users are served by $\mathscr{A}$. During A/B testing, users in the control group are presented with recommendation by $\mathscr{A}$, while users in the experimental group are presented with the recommendation by our proposed model *U-rank+*. Note that the click model of *U-rank+* shares the same network architecture and parameter complexity with $\mathscr{A}$, in order to verify whether the improvement is brought by the objective function design of the ranker in *U-rank+*. The deployment of *U-rank+* is the same as *U-rank*.

6.2.2 Metrics. We examine *effective Cost Per Mille*: eCPM $= \frac{\sum_i bid_i \times \mathbb{1}_i}{\#impressions} \times 1000$ in the online advertising scenario where for the i-th instance in the log of the day of testing (referred to as platform revenue in previous parts), bid_i is the bid of the item in the instance, $\mathbb{1}_i$ is the indicator

[9]The baseline model is a point-wise CTR prediction model, which is not clearly described due to some commercial concerns.

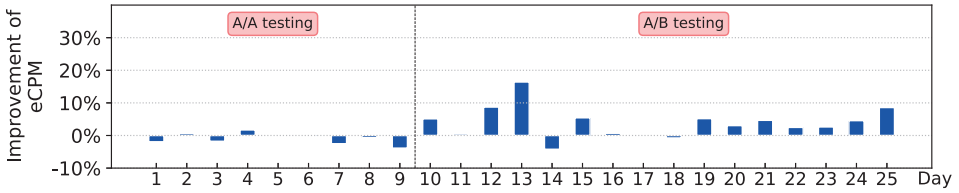

Fig. 9. Online experimental results of eCPM (online advertising scenario).

function which equals 1 if the i-th instance is a positive sample, and # impressions is the number of impressions.

6.2.3 Results. Figure 9 shows the improvement of the experimental group over the control group w.r.t. eCPM. We can see that the system is stable where the improvement of eCPM fluctuated within 5% during the A/A testing. The average improvement of the experimental group during the nine days of A/A testing is −0.91%. Our *U-rank+* model is launched to the live system on Day 10. From Day 10, we observe a significant improvement, 5.12%, over the baseline model. However, the improvement drops to 0.5% on Day 11, then, it increases to 8.72%, 16.41% for the following two days. The fluctuation of the improvement is caused by the small traffic of the experimental group. Thus, we increase the traffic to 17% and 20% on Day 15 and Day 19, respectively. We can see that the improvement of eCPM becomes stable as we increase of traffic of the experimental group. The average improvement of eCPM is 4.19% over the sixteen days of A/B testing. Considering the fluctuation of A/A testing, the actual improvement of eCPM is $4.19\% - (-0.91\%) = 5.1\%$. These results clearly demonstrate the high effectiveness of *U-rank+* in improving the total utility which refers to the income of the platform in this scenario.

7 CONCLUSION

In this work, we propose a general graph matching framework for utility-oriented learning to rank with logged user feedback. By formulating the ranking objective as the maximum-weight matching on the item-position bipartite graph, we optimize the expected utility directly based on clicks without any extra assumptions on relevance nor examination. Considering different utility forms in real world applications, a general weighted form of ranking metric is proposed. We first estimate the weight of the bipartite graph with a position-aware deep CTR model, which models the examination bias explicitly by taking user context and item attribute into consideration. Besides, a propensity model is learned in advance and used to correct for selection bias in the logging data, alleviating the overestimation of examination bias. Then we propose two different solver, Sinkhorn and Lambdaloss, to solve the maximum-weight matching problem, i.e., to optimize the expected utility. Sinkhorn algorithm learns a Doubly-Stochastic Matrices (DSM)-based ranking function and solves the graph matching problem in an end-to-end manner. Lambdaloss solves the matching problem by learning a scoring function with pairwise permutations, and reduces the complexity in inference stage from $O(N^3)$ to $O(N)$. Theoretical analysis shows that the lambdaloss objective proposed in this work is an upper bound of the graph matching objective, and also a natural extension of previous counterfactual learning to rank objectives. Extensive studies on three benchmark datasets have shown the effectiveness of our work. We also deploy this ranking framework on two different application scenarios, including recommendation and online advertising, where we observe a large utility improvement over the production baselines.

ACKNOWLEDGEMENT

Weinan Zhang is supported by "New Generation of AI 2030" Major Project (2018AAA0100900) and National Natural Science Foundation of China (62076161, 61772333, 61632017). The work is also sponsored by Huawei Innovation Research Program. We thank MindSpore [1] for the partial support of this work, which is a new deep learning computing framework.

REFERENCES

[1] 2020. MindSpore. https://www.mindspore.cn/
[2] Ryan Prescott Adams and Richard S Zemel. 2011. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925* (2011).
[3] Aman Agarwal, Ivan Zaitsev, Xuanhui Wang, Cheng Li, Marc Najork, and Thorsten Joachims. 2019. Estimating position bias without intrusive interventions. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 474–482.
[4] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W Bruce Croft. 2018. Unbiased learning to rank with unbiased propensity estimation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 385–394.
[5] Xiao Bai, Reza Abasi, Bora Edizel, and Amin Mantrach. 2019. Position-aware deep character-level CTR prediction for sponsored search. *IEEE Transactions on Knowledge and Data Engineering* (2019).
[6] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine learning* 79, 1-2 (2010), 151–175.
[7] Alexey Borisov, Ilya Markov, Maarten De Rijke, and Pavel Serdyukov. 2016. A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*. 531–541.
[8] Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning (ICML-05)*. 89–96.
[9] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.
[10] Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*. 193–200.
[11] Olivier Chapelle and Ya Zhang. 2009. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*. 1–10.
[12] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2020. Bias and Debias in Recommender System: A Survey and Future Directions. *IEEE Transactions on Knowledge and Data Engineering* (2020).
[13] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*. 87–94.
[14] Xinyi Dai, Jiawei Hou, Qing Liu, Yunjia Xi, Ruiming Tang, Weinan Zhang, Xiuqiang He, Jun Wang, and Yong Yu. 2020. U-rank: Utility-oriented Learning to Rank with Implicit Feedback. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2373–2380.
[15] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 293–296.
[16] Georges E Dupret and Benjamin Piwowarski. 2008. A user browsing model to predict search engine click data from past observations.. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 331–338.
[17] Zhichong Fang, Aman Agarwal, and Thorsten Joachims. 2019. Intervention Harvesting for Context-Dependent Examination-Bias Estimation. In *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Paris, France) *(SIGIR 2019)*. ACM, New York, NY, USA, 825–834. https://doi.org/10.1145/3331184.3331238
[18] Fan Guo, Chao Liu, Anitha Kannan, Tom Minka, Michael Taylor, Yi-Min Wang, and Christos Faloutsos. 2009. Click chain model in web search. In *Proceedings of the 18th international conference on World wide web*. 11–20.
[19] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. 1725–1731. https://doi.org/10.24963/ijcai.2017/239

[20] Huifeng Guo, Jinkai Yu, Qing Liu, Ruiming Tang, and Yuzhou Zhang. 2019. PAL: A Position-Bias Aware Learning Framework for CTR Prediction in Live Recommender Systems. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Copenhagen, Denmark) *(RecSys 2019)*. Association for Computing Machinery, New York, NY, USA, 452–456. https://doi.org/10.1145/3298689.3347033

[21] James J Heckman. 1979. Sample selection bias as a specification error. *Econometrica: Journal of the econometric society* (1979), 153–161.

[22] Ziniu Hu, Yang Wang, Qu Peng, and Hang Li. 2019. Unbiased LambdaMART: An unbiased pairwise learning-to-rank algorithm. In *The World Wide Web Conference*. 2830–2836.

[23] Thorsten Joachims. 2006. Training Linear SVMs in Linear Time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Philadelphia, PA, USA) *(KDD 2006)*. Association for Computing Machinery, New York, NY, USA, 217–226. https://doi.org/10.1145/1150402.1150429

[24] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2017. Accurately interpreting clickthrough data as implicit feedback. In *ACM SIGIR Forum*, Vol. 51. Acm New York, NY, USA, 4–11.

[25] 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 (TOIS)* 25, 2 (2007), 7.

[26] Thorsten Joachims, Laura A Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Sigir*, Vol. 5. 154–161.

[27] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2016. Unbiased Learning-to-Rank with Biased Feedback. *ArXiv* abs/1608.04468 (2016).

[28] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[29] Roderick JA Little and Donald B Rubin. 2019. *Statistical analysis with missing data*. Vol. 793. John Wiley & Sons.

[30] Lori Lorigo, Maya Haridasan, Hrönn Brynjarsdóttir, Ling Xia, Thorsten Joachims, Geri Gay, Laura Granka, Fabio Pellacini, and Bing Pan. 2008. Eye tracking and online search: Lessons learned and challenges ahead. *Journal of the American Society for Information Science and Technology* (2008).

[31] Lori Lorigo, Bing Pan, Helene Hembrooke, Thorsten Joachims, Laura Granka, and Geri Gay. 2006. The influence of task and gender on search and evaluation behavior using Google. *Information Processing & Management* (2006).

[32] 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 *SIGIR*.

[33] Jiaxin Mao, Zhumin Chu, Yiqun Liu, Min Zhang, and Shaoping Ma. 2019. Investigating the Reliability of Click Models. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. 125–128.

[34] Gonzalo Mena, David Belanger, Gonzalo Munoz, and Jasper Snoek. 2017. Sinkhorn networks: Using optimal transport techniques to learn permutations. In *NIPS Workshop in Optimal Transport and Machine Learning*.

[35] Pavel Metrikov, Fernando Diaz, Sebastien Lahaie, and Justin Rao. 2014. Whole page optimization: how page elements interact with the position auction. In *Proceedings of the fifteenth ACM conference on Economics and computation*. 583–600.

[36] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

[37] Harrie Oosterhuis and Maarten de Rijke. 2018. Differentiable unbiased online learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1293–1302.

[38] Harrie Oosterhuis and Maarten de Rijke. 2020. Policy-aware unbiased learning to rank for top-k rankings. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 489–498.

[39] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-based neural networks for user response prediction over multi-field categorical data. *ACM Transactions on Information Systems (TOIS)* 37, 1 (2018), 5.

[40] Filip Radlinski, Madhu Kurup, and Thorsten Joachims. 2008. How does clickthrough data reflect retrieval quality?. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 43–52.

[41] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*. 521–530.

[42] Stephen E Robertson. 1977. The probability ranking principle in IR. *Journal of documentation* 33, 4 (1977), 294–304.

[43] Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. 2017. Deeppermnet: Visual permutation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3949–3957.

[44] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. 2016. Multileave gradient descent for fast online learning to rank. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 457–466.

[45] Huazheng Wang, Sonwoo Kim, Eric McCord-Snook, Qingyun Wu, and Hongning Wang. 2019. Variance reduction in gradient exploration for online learning to rank. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 835–844.

[46] Huazheng Wang, Ramsey Langley, Sonwoo Kim, Eric McCord-Snook, and Hongning Wang. 2018. Efficient exploration of gradient space for online learning to rank. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 145–154.

[47] Jun Wang and Jianhan Zhu. 2010. On statistical analysis and optimization of information retrieval effectiveness metrics. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 226–233.

[48] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 115–124.

[49] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position bias estimation for unbiased learning to rank in personal search. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 610–618.

[50] Xuanhui Wang, Cheng Li, Nadav Golbandi, Mike Bendersky, and Marc Najork. 2018. The LambdaLoss Framework for Ranking Metric Optimization. In *Proceedings of The 27th ACM International Conference on Information and Knowledge Management (CIKM 2018)*. 1313–1322.

[51] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. 2018. Turning clicks into purchases: Revenue optimization for product search in e-commerce. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 365–374.

[52] 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*. 1201–1208.

[53] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.

[54] Yuye Zhang, Laurence AF Park, and Alistair Moffat. 2010. Click-based evidence for decaying weight distributions in search effectiveness metrics. *Information Retrieval* 13, 1 (2010), 46–69.

[55] Yunzhang Zhu, Gang Wang, Junli Yang, Dakan Wang, Jun Yan, and Zheng Chen. 2009. Revenue optimization with relevance constraint in sponsored search. In *Proceedings of the Third International Workshop on Data Mining and Audience Intelligence for Advertising*. 55–60.

[56] Yunzhang Zhu, Gang Wang, Junli Yang, Dakan Wang, Jun Yan, Jian Hu, and Zheng Chen. 2009. Optimizing search engine revenue in sponsored search. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. 588–595.