ParClick: A Scalable Algorithm for EM-based Click Models

Khandel, P.; Markov, I.; Yates, A.; Varbanescu, A.-L.

DOI
10.1145/3485447.3511967

Publication date
2022

Document Version
Final published version

Published in
WWW'22

ParClick: A Scalable Algorithm for EM-based Click Models

Pooya Khandel
University of Amsterdam
Amsterdam, Netherlands
p.khandel@uva.nl

Ilya Markov
Independent researcher*
Amsterdam, Netherlands
ilya.markov@gmail.com

Andrew Yates
University of Amsterdam
Amsterdam, Netherlands
a.c.yates@uva.nl

Ana-Lucia Varbanescu
University of Amsterdam
Amsterdam, Netherlands
a.l.varbanescu@uva.nl

ABSTRACT

Research on click models usually focuses on developing effective approaches to reduce biases in user clicks. However, one of the major drawbacks of existing click models is the lack of scalability. In this work, we tackle the scalability of Expectation-Maximization (EM)-based click models by introducing ParClick, a new parallel algorithm designed by following the Partitioning-Communication-Aggregation-Mapping (PCAM) method. To this end, we first provide a generic formulation of EM-based click models. Then, we design an efficient parallel version of this generic click model following the PCAM approach: we partition user click logs and model parameters into separate tasks, analyze communication among them, and aggregate these tasks to reduce communication overhead. Finally, we provide a scalable, parallel implementation of the proposed design, which maps well on a multi-core machine. Our experiments on the Yandex relevance prediction dataset show that ParClick scales well when increasing the amount of training data and computational resources. In particular, ParClick is 24.7 times faster to train with 40 million search sessions and 40 threads compared to the standard sequential version of the Click Chain Model (CCM) without any degradation in effectiveness.

CCS CONCEPTS

• **Information systems** → **Users and interactive retrieval**.

KEYWORDS

Click model, Web Search, User Modeling, Parallel computing

ACM Reference Format:
Pooya Khandel, Ilya Markov, Andrew Yates, and Ana-Lucia Varbanescu. 2022. ParClick: A Scalable Algorithm for EM-based Click Models. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3511967

1 INTRODUCTION

Users actively seek various kinds of information through search engines by submitting queries and interacting with search engine results pages (SERPs). User interactions, such as clicks, mouse movements, and timings, contain valuable information that is used to enhance ranking performance [25]. Among these user-generated data, the most widely used information source is *user clicks* [9, 25].

User clicks suffer from various types of biases, such as position bias or attention bias [9]. To predict observed clicks accurately, including correcting for biases, a number of click models were proposed based on probabilistic graphical models (PGMs) [9] and neural networks [3, 4, 8].

The most effective click models used in practice, e.g., Position-Based Model (PBM) and Click Chain Model (CCM), employ Expectation-Maximization (EM) to learn their parameters from a click log [9]. This learning approach suffers from a lack of scalability. For example, training the PBM model with 8 million search sessions using the standard PyClick[1] library takes about 12.5 hours on an Intel(R) Xeon(R) Gold 5118 CPU running at 2.30GHz, while scaling up the number of search sessions to 40 million is infeasible with this approach, due to the large memory footprint of the training (i.e., the training crashes out-of-memory, even though the machine has 256GB of RAM). At the same time, a large web search engine, such as Google, processes up to 3.5 billion search sessions (≈ 437 times more data) in a single day.[2] This means that, with existing algorithms, training even a simple click model, such as PBM, on a single Google-day's worth of click logs is infeasible.

Improving the scalability of click models for large click logs is extremely beneficial for real-world applications, because it enables a *significant processing speed-up* and/or *fast processing of massive datasets*. Concretely, better scalability (i) makes the analysis of larger click logs feasible, which in turn provides better query coverage, as more unique queries can be seen during training,[3] (ii) enables frequent retraining, thus providing efficient support for user-behavior variations over time, and (iii) provides opportunities for employing increasingly complex and more effective click models.

PGM-based click models are widely used for online and counterfactual learning to rank [1, 18, 20, 28, 30]. These models are usually trained with Maximum Likelihood Estimation (MLE) or EM algorithms; however, it has been shown that click models trained with EM perform better [15]. In our work, we specifically address the scalability of EM-based click models, aiming to devise a *generic* solution that supports the diversity of applications, which manifests itself in different parameters, different estimation functions, and diverse statistical distributions of click log data

In this paper, we propose ParClick, a generic algorithm that considerably improves the scalability of EM-based click models through efficient parallelization. Specifically, we employ the Partitioning-Communication-Aggregation-Mapping (PCAM) [13] design method

*This research was done while at the University of Amsterdam.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3511967

[1] https://github.com/markovi/PyClick
[2] https://www.internetlivestats.com/google-search-statistics
[3] Many click models only work with queries they have seen during training.

to create a new parallel algorithm that correctly and efficiently partitions user click logs and model parameters, and assigns the processing of different partitions to parallel tasks. Our algorithm is generic, in that it supports different processing tasks (supporting *multiple click-models*), different partitioning and load-balancing strategies, thus enabling close-to-optimal strong-scaling, and can be implemented on any parallel system or programming model.

Our prototype, evaluated in this paper, includes two click models, two load balancing policies, and is implemented for a multi-core machine. Concretely, we demonstrate our generic algorithm on two case-studies: the relatively simple PBM model, as a representative of position-based click models, and the more complex CCM model, as a representative of cascade click models. We evaluate the scalability of ParClick applied to PBM and CCM using the Yandex relevance prediction dataset [26]. Our results indicate that ParClick scales well when increasing the size of a click log and the allocated resources (i.e., threads for the algorithm and cores for the machine). For example, in our largest experiment, we trained CCM with 40 million search sessions using 40 threads (one thread per task) and observed a 24.7 speed-up compared to the standard sequential version. Overall, our analysis demonstrates that ParClick can be employed for training EM-based click models with a large number of search sessions, e.g., 40–50 million, in a matter of tens of minutes, without any degradation of effectiveness.

In summary, the main contributions of our work are as follows:

(C1) We formulate a generic algorithm for training EM-based click models.
(C2) We design a parallel version of our generic algorithm, following the PCAM method, thus providing the first scalable, generic algorithm to train EM-based click models.
(C3) Through extensive empirical analysis, we demonstrate that ParClick scales well with larger click logs and more computational resources.
(C4) We provide an open source, efficient implementation of ParClick, applicable on any shared memory parallel machine.[4] Furthermore, this implementation is faster to train and requires less memory compared to PyClick.

The rest of this paper is structured as follows. We overview related work in Section 2. Section 3 describes the PBM and CCM models, and explains how their parameters are estimated with the EM algorithm. We explain in detail the design and implementation of ParClick in Section 4. Our empirical evaluation and results are presented in Section 5. Finally, we conclude the paper in Section 6.

2 RELATED WORK

To place our research in context, this section presents an overview of recent advances in click models and in how they address scalability.

One class of click models is developed based on probabilistic graphical models (PGMs) [19]. These models differ in how they distinguish two primary events: a user examining a document and a user attracted by a document. For example, the PBM model assumes that examination depends on rank [9]. The cascade model [10], instead, assumes that examination depends on whether higher-ranked documents were examined and/or clicked. Various extensions of the PBM and cascade models were proposed [6, 12, 16, 17]. More recent work focuses on exploiting further information about users, results, and other search characteristics to enhance the accuracy of click models [7, 27, 29, 32].

Another class of click models is formed by neural-based approaches. Borisov et al. [3] proposed a general neural framework and modeled user browsing behavior as a sequence of vectors. Yu et al. [31] introduced the Rank-Biased Neural Network to learn input representation of queries and documents automatically. Through investigating the relationship between the predicted clicks and the estimated relevance scores, Chen et al. [8] introduced the Context-Aware Click Model. Furthermore, Dai et al. [11] exploit imitation learning to address exposure bias and inferior estimation. More recently, Lin et al. [21] proposed GraphCM that is based on graph neural networks to extract sessions information and address the data sparsity problem.

All these approaches focus on improving the effectiveness of click models. However, computation efficiency and scalability have received little attention so far. Most research on improving efficiency focused on the EM algorithm itself, known to be computationally intensive. For example, [2] proposed a parallel variant of EM using GPUs, optimized for the application of Gaussian Mixture Models (GMM), but *not* directly applicable for efficient click models. Instead, online approaches [5, 24], are built to converge faster, and form the basis of the OnlineEM method Markov et al. [23] used to update parameters of an already trained model for new logged query sessions, without re-training from scratch. OnlineEM, however, cannot improve the efficiency of training click models from scratch. The closest research to our work is [22] that focuses on efficient training of bayesian browsing model at large scale; however, their method is not directly generalized for EM-based click models.

In this work, we propose *the first generic parallel algorithm* that directly addresses the scalability of EM-based click models. Therefore, our approach is complementary to current developments in the field of click models: our solution can be combined with these approaches to further improve the efficiency of existing and upcoming click models.

3 BACKGROUND

In this Section, we present the details of the PBM and CCM click models that are needed to introduce our ParClick in Section 4. Table 1 presents the notation used in the paper.

3.1 Position-Based Model (PBM)

In the PBM click model, the probability that a user clicks on a search result $P(C_d = 1)$ depends on the probability that the user examines that search result $P(E_d = 1)$ and finds it relevant/attractive $P(A_d = 1)$ [9]. The examination probability depends on the rank of the search result, while the attractiveness probability depends on the query-result pair:

$$P(C_d = 1) = P(E_d = 1) \cdot P(A_d = 1) = \gamma_r \cdot \alpha_{qd}, \tag{1}$$

where r is the rank of result d and γ_r and α_{qd} are the parameters of PBM. They are estimated iteratively using EM as follows [9]:

[4] https://github.com/uva-sne/ParClick

Table 1: Notation used in this paper.

Symbol	Description
(t)	Iteration
M	Number of EM iterations
q	User's query
d	Document
r	Rank of a document
α	Attractiveness parameter
γ	Examination parameter
S	Set of all query sessions
s	Query session
Φ_x	Update function for a parameter of type x in the EM algorithm
X	Set
X_y	Set that has concept y
V	Set containing parameters of a click model
A	Subset of V with all attractiveness parameters
Γ	Subset of V with all examination parameters
v_i	Parameter of a click model
$H(v_i)$	Type of parameter v_i
C_M	EM-based click model
T_i	Parallel task
$D(T_i)$	Data associated with task T_i
Ψ_{v_i}	Set of all parameters that affect v_i when updating
N	Number of tasks

$$\gamma_r^{(t+1)} = \frac{1}{|S|} \sum_{s \in S} \Phi_\gamma \left(\alpha_{qd}^{(t)}, \gamma_r^{(t)}, s \right), \tag{2}$$

$$\alpha_{qd}^{(t+1)} = \frac{1}{|S_{qd}|} \sum_{s \in S_{qd}} \Phi_{\alpha_{PBM}} \left(\alpha_{qd}^{(t)}, \gamma_r^{(t)}, s \right), \tag{3}$$

where t is the iteration count, d in the result shown at rank r in the query session s when query q is issued, S is the set of all training query sessions, and S_{qd} is the subset of training sessions containing query q and result d, Φ_γ and $\Phi_{\alpha_{PBM}}$ are estimation functions for examination and attractiveness parameters of PBM.

3.2 Click Chain Model (CCM)

CCM is one of cascade click models, that is, it assumes that users scan search results from top to bottom. Similarly to PBM, this model has attractiveness parameters; however, it differs from PBM in the way examination probability of a result is computed. In CCM, the probability of click can be calculated through the following equation:

$$P(C_d = 1) = \alpha_{qd} \cdot P(E_r = 1), \tag{4}$$

Unlike PBM, in CCM it assumed that the first ranked result is always examined and the examination of other ranks depends on previous ranks' examinations and clicks. Accordingly, for CCM, $P(E_r = 1)$ is computed based on three types of examination parameters $\gamma_1, \gamma_2, \gamma_3$ through Eq. (5).

$$\begin{aligned} P(E_{r+1} = 1) = P(E_r = 1) \cdot \Big(\Big(1 - \alpha_{qd_r} \Big) \cdot \gamma_1 + \\ \alpha_{qd_r} \cdot \Big(\gamma_2 \cdot \Big(1 - \alpha_{qd_r} \Big) + \gamma_3 \cdot \alpha_{qd_r} \Big) \Big) \end{aligned} \tag{5}$$

CCM parameters are also estimated using EM over multiple iterations similar to (2) and (3) [9].

4 PARCLICK: A SCALABLE ALGORITHM FOR EM-BASED CLICK MODELS

In this section we explain the design of our generic parallel click model training, and present the core aspects of its implementation in ParClick. We first present our generic formulation for EM-based click models in Section 4.1. Next, our design follows the PCAM methodology outlined in [13], in which the parallel algorithm to solve a problem is built in four steps: partitioning (Section 4.2.1), communication (Section 4.2.2), aggregation (Section 4.2.3), and mapping (Section 4.2.4).

4.1 Generic EM-based Click Models

A generic EM-based click model, C_M, can be formulated using a finite set of parameters $V = \{v_1, v_2, \ldots v_k\}$ of different types, $H(v_i)$. Click models usually have $m \geq 2$ different types of parameters. For example, PBM has two types of parameters (thus, $m = 2$): attractiveness parameters (possibly billions) and examination parameters (e.g., ten for web search). CCM, on the other hand, has four types of parameters (thus, $m = 4$): similarly to PBM, attractiveness parameters (possibly billions), and three types of examination parameters (one of each type).

For all EM-based click models, the training procedure is the same: we iterate over all query sessions $s_j \in S$ and calculate any required updates for parameters $v_i \in V$ that are affected by the current query session s_j. A generic way to calculate new estimates of model parameters in each iteration of EM is the following (note, for example, that Eq. (6) generalizes Eqs. (2) and (3)):

$$v_i^{(t+1)} = \frac{1}{|S_{v_i}|} \sum_{s \in S_{v_i}} \Phi_{H(v_i)} \left(\Psi_{v_i}^{(t)}, s \right), \ \forall v_i \in V, \tag{6}$$

where $v_i^{(t+1)}$ is the new estimate of model parameter v_i at iteration $(t+1)$, $\Phi_{H(v_i)}$ is the estimation function for a parameter of type $H(v_i)$ as derived in [9], $S_{v_i} \subseteq S$ is the set of all query sessions that affect the value of v_i, and $\Psi_{v_i} = \{v_j \mid v_j \in V, v_j \text{ is required to update the value of } v_i\}$ is a set of other parameters (from the previous iteration (t)) that affect v_i.

Our ultimate goal is to provide a scalable parallel algorithm to train a generic EM-based click model using Eq. (6). The design of this parallel algorithm is described next.

4.2 Parallelization

When training a click model sequentially, a single task needs to calculate all the parameters in V, iterating, for every parameter v_i, over a subset of sessions, $S_{v_i} \subseteq S$. With $|S|$ and $|V|$ being very large (in the order of billions), traditional sequential training becomes prohibitively slow, and, therefore, it does not scale to real-world scenarios such as web search. To tackle this lack of scalability, we follow the Partitioning-Communication-Aggregation-Mapping

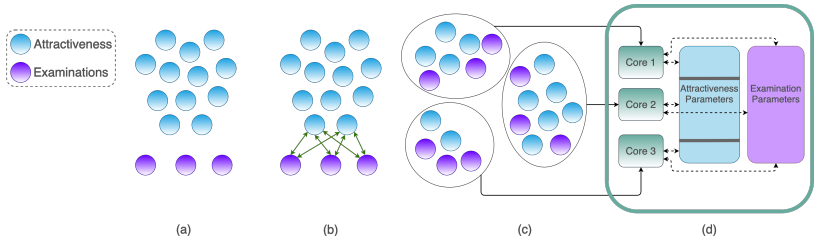

Figure 1: Defining tasks based on PCAM in ParClick: (a) Partitioning: One task per parameter is defined. (b) Communication: Data from different tasks is required by other tasks. (c) Aggregation: Ultimately, each task contains multiple attractiveness and all examination parameters. (d) Mapping: Map each task to a distinct thread, and each thread executed on a single core.

(PCAM) approach [13] to design a parallel algorithm to train EM-based click models. In the following paragraphs, we describe the specific actions we take at each stage of PCAM to define parallel tasks. Figure 1 shows overall steps performed through PCAM[5].

4.2.1 PCAM: Partitioning. In PCAM, partitioning refers to splitting the problem into tasks that *could* execute in parallel, i.e., the tasks with minimal dependencies. Our generic click model (Eq. (6)) indicates that a parameter v_i can be updated in iteration $t+1$ with a subset of the input data, $S_{v_i} \subseteq S$, and a subset of model parameters from the previous iteration, $\Psi_{v_i}^{(t)}$. Based on this fundamental observation, our generic parallel algorithm for click-models training aims to calculate multiple v_i parameters in parallel.

Broadly speaking, there are two partitioning approaches: domain decomposition (i.e., data-driven partitioning, where computation follows the data) and functional partitioning (i.e., task-driven partitioning, where data follows the computation). For our problem, domain decomposition requires identifying an efficient partitioning of S and $\bigcup_i \Psi_{v_i}$ and, for every partition, identifying which v_i parameters can be computed with data from this partition. With this approach, calculating certain v_i parameters would likely require access to multiple partitions, which could be expensive. Instead, functional partitioning requires identifying a partitioning of parameter computations into tasks and associating the required data to each task. In this case, if the same data is required by multiple tasks, data sharing and/or replication might be required. For both cases, the next two steps – communication and aggregation – aim to reduce the impact of non-ideal partitions.

Given the formulation in Eq. (6), we opt to start from a functional partitioning and we define a task T_i for calculating each v_i. Consequently, each task requires access to data $D(T_i) = S_{v_i} \cup \Psi_{v_i}$ Based on this definition, there would be $|V|$ tasks $T = \{T_1, T_2, \ldots, T_{|V|}\}$.

4.2.2 PCAM: Communication. The communication phase focuses on analyzing the dependencies between the proposed tasks, that is, identifying the data required by multiple tasks, which, in turn, should be communicated across tasks. For this analysis, we split tasks into *independent* and *dependent*, as follows: two tasks T_a and T_b are *independent* (and, therefore, can execute in parallel) iff $D(T_a) \cap D(T_b) = \emptyset$, and *dependent* otherwise. Consequently, two independent tasks share no data, and, therefore, require no communication or synchronization with each other.

[5]Based on Figure 2.1 from [13]

Two dependent tasks can still execute in parallel if the shared data is made available to both tasks (e.g., through shared memory). However, this shared data still needs to be processed by both tasks, which reduces parallelism efficiency (as more work is done than in the sequential case) and hampers scalability. The larger the shared data is, volume-wise, the higher its negative scalability impact is. Thus, for ParClick, we must identify which tasks are dependent and which are independent; for dependent tasks, we must also estimate the shared data volume.

Our analysis is based on parameter types. Calculating new values of attractiveness parameters for PBM depends on a subset of query sessions, S_{qd}, and all examination parameters, but does not depend on other attractiveness parameters. Calculating new values of examination parameters, however, might require all query sessions in S and a number of attractiveness parameters. The same holds for CCM.

Thus, all our tasks are dependent, and require some data sharing. However, we can still separate parameters (and the tasks that calculate them) between *attractiveness* and *examination* by communication volume, as the data shared between *attractiveness tasks* is minimal in terms of volume, while data shared between *examination tasks* is significantly larger.

4.2.3 PCAM: Aggregation. In general, PCAM uses aggregation to reduce the amount of communication and increase task granularity. To this end, its goal is to aggregate dependent tasks into larger tasks, which can become independent from each other.

To aggregate our initial tasks $T = \{T_1, T_2, \ldots, T_{|V|}\}$, we aim to (i) aggregate dependent tasks with significant data sharing, and (ii) preserve a sufficient number of tasks (to utilize the target machine well). As different attractiveness tasks only share a limited number of examination parameters (at most 10), they are not ideal candidates for (i). Different examination tasks share a lot of query data and attractiveness parameters, but they are too few to meet (ii). Therefore, we opt for a *hybrid* solution: we first aggregate all examination tasks in a subtask by replicating all examination parameters into that, and then merge each attractiveness task with the subtask. As a consequence, an additional step is required to calculate the global examination parameters *after* all tasks have completed, which is performing synchronization for the examination parameters among all the tasks. However, through this aggregation, we avoid any extra replication of query sessions.

After this first aggregation of initial tasks T into T', we reduce the number of tasks to as many as the attractiveness parameters $|A|$. For EM-based click models in real applications, $|A|$ might be in the

order of billions, making the management and synchronization of all these tasks very inefficient on any shared memory machine (and on most supercomputers, too). Thus, we must aggregate tasks further, by merging several existing tasks T'. After this aggregation, the new task T''_k, will likely use many more query sessions to calculate *more* attractiveness parameters, and the same examination parameters. With this aggregation, no extra dependencies are created, but the number of tasks to be executed in parallel can be reduced at will.

Algorithm 1 PCAM: Aggregation

```
1: Create tasks T' = {T'_1, T'_2, ..., T'_|A|}
2: for T'_i ∈ T' do
3:     k = π(T'_i)                          ▷ using Eq. (7) or (8)
4:     Merge S_T''_k with S_T'_i
5:     Merge A_T''_k with A_T'_i
6: end for
```

One final aggregation consideration needs addressing: given the synchronization required to compute the examination parameters, the load balancing of the tasks becomes essential for scalability; a task that takes much longer to compute than the others will slow down the entire computation, and directly reduce scalability. However, load balancing for these tasks is not easy to achieve: $|S_{T'_i}|$ varies over the tasks as there are a different number of query sessions in the user click log for each query. To attain a balanced workload among the new tasks, we merge the tasks based on policy π as explained in Algorithm 1. Based on the statistical distribution of user click logs, different policies can be used. In this work, we adopt two policies - Round-Robin (RR) and Minimum Utilization (MU) - and compare them with each other.

In the RR policy, we iterate over tasks $T' = \{T'_1, T'_2, \ldots, T'_{|A|}\}$ and merge them into new tasks successively:

$$\pi(T'_i) = i \bmod N. \tag{7}$$

This policy can only produce a balanced workload if all T'_i have very similar sizes; however, in reality, this is rarely the case. To tackle this problem, we also define the MU policy, where we iterate over the tasks in T' and assign them to the least occupied task from $T'' = \{T''_1, T''_2, \ldots, T''_N\}$:

$$\pi(T'_i) = \arg\min\{|S_{T''_k}| \mid k \in \{1, 2, \ldots, N\}\}. \tag{8}$$

MU policy will produce more balanced tasks, while requiring very limited additional pre-processing; thus, the processing time should be similar to the RR policy. The evaluation of these policies is further discussed in Section 5.

4.2.4 PCAM: Mapping. Finally, the last stage of our design is the mapping from the theoretical algorithm to a practical application. Ideally, with a good aggregation, this step is a purely technical one, for the actual implementation. However, in this step, additional knowledge about the machine architecture should be taken into account to potentially improve the previous step's aggregation.

In our case, we execute the training on a multi-core machine. Thus, we map each task to a distinct thread, and each thread is executed on a single core. Since tasks calculate new estimates of attractiveness parameters independently, attractiveness parameters can be assumed *task-local*, i.e., they need to be accessed by one single task, and no synchronization is needed across tasks. Therefore, we (virtually) distribute these parameters across threads. Examination parameters are first estimated by each task independently, and then combined in a shared set. Therefore, we keep a task-local copy of the examination parameters for each thread, allow threads to compute independently, synchronize to ensure all tasks have calculated their examinations, and combine the results in the shared examinations. We implement the synchronization using *barriers* as defined in [14].

4.3 Parallel EM for Generic EM-based Click Models

In this section, we present our parallel EM algorithm for click models, and describe the concrete implementation of the workload of each task (i.e., the so-called *thread function*). As explained in Section 4.2.3, we have N tasks $T'' = \{T''_1, T''_2, \ldots, T''_N\}$. These tasks will be launched in parallel (lines 1–3, Algorithm 2), and they process $S_{T''_i}$ to estimate new values of C_M parameters in each task $V_{T''_i}$. In particular, each task T''_i calculates new estimates for the set of associated attractiveness parameters $\mathrm{A}_{T''_i}$ *and* task-local copy of values estimated for examination parameters $\Gamma_{T''_i}$. By design (see Section 4.2), $\mathrm{A}_{T''_i}$ parameters do not depend on any other $\mathrm{A}_{T''_j}$, $i \neq j$; thus, they need not be synchronized between tasks. Conversely, task-local estimates of examination parameters $\Gamma_{T''_i}$ need to be synchronized after each iteration of EM to attain exact estimates of examination parameters Γ. We discuss the estimation process for corresponding model parameters of each task ($V_{T''_i} = \mathrm{A}_{T''_i} \cup \Gamma_{T''_i}$) below.

Each parallel task starts with initializing the values of model parameters $V_{T''_i}$ (line 5, Algorithm 2), such that examination parameters are initialized similarly in each task. During iteration t of EM (line 6, Algorithm 2), new estimates of all model parameters are calculated with Eq. (6) (line 7, Algorithm 2). Note that, as described in Section 4.2.3, calculating exact estimates of examination parameters requires synchronization among tasks. To synchronize these parameters and obtain correct values for them, each task waits until all other tasks finish estimations at iteration t and reach the synchronization barrier as described in 4.2.4 (line 8, Algorithm 2). Then, each task reads $\{\Gamma^{(t+1)}_{T''_h}\}$ from all others tasks (line 9, Algorithm 2). Finally, all tasks combine $\Gamma^{(t+1)}_{T''_i}$and $\{\Gamma^{(t+1)}_{T''_h}\}$ into $\Gamma^{(t+1)}$ as follows (line 10, Algorithm 2):

$$\gamma^{(t+1)} = \frac{1}{|S_\gamma|} \sum_{i=1}^{N} \left(\gamma^{(t+1)}_{T''_i} \cdot |S_{T''_i,\gamma}| \right), \ \forall \gamma \in \Gamma, \tag{9}$$

where $\gamma^{(t+1)}_{T''_i}$ is the new value of the examination parameter γ computed by the i-th task on iteration $(t+1)$ (line 7, Algorithm 2) with the set of query sessions in corresponding task that affect its value $S_{T''_i,\gamma}$.

The training procedure (lines 6 to 11, Algorithm 2) continues for M iterations. When the training is finished, each task i contains a part of attractiveness parameters $\mathrm{A}^{(T)}_{T''_i}$ and a local-task copy of the examination parameters $\Gamma^{(T)}$. These final estimations values, i.e., $\bigcup_{i \in \{1,\ldots,N\}} \mathrm{A}^{(T)}_{T''_i}$ and $\Gamma^{(T)}$, represent the trained click model C_M,

which is identical to the sequential C_M as Algorithm 2 preserves the order of computations.

Algorithm 2 Parallel Expectation-Maximization (EM) For Click-Models Training

1: **parfor** $i = 1..N$ **do**
2: *Estimate* C_M *Params*$(S_{T_i''})$ ▷ Launch parallel tasks
3: **end parfor**

4: **function** *Estimate* C_M *Params*$(S_{T_i''})$
5: Initialize $V_{T_i''}^{(0)} : v_i^{(0)} = 0.5 \mid \forall v_i \in V_{T_i''}$
6: **for** $t = 0$ to $t = M$ **do** ▷ EM iterations
7: Estimate $V_{T_i''}$ ▷ using Eq. (6)
8: Synchronization *barrier* ▷ as described in 4.2.4
9: $\forall h \in \{1, \ldots, N\}, h \neq i$: Read $\Gamma_{T_h''}^{(t+1)}$
10: Update $\Gamma^{(t+1)}$ ▷ using Eq. (9)
11: **end for**
12: **end function**

4.4 Implementation Details

We implemented ParClick as a multi-threaded application in C++[6]. Specifically, the N tasks that estimate C_M parameters are implemented as threads. Given that no synchronization for attractiveness parameters is required, they are stored locally in each thread, but the examination parameters are stored as shared arrays, i.e., they are accessible for all threads.

To optimize memory usage, we do not preserve new estimations through all iterations. Instead, we first calculate parameter initialization variables and copy them into a container as current values. Then, we calculate new estimations at each iteration, and by the end of each iteration (line 10, Algorithm 2), we copy new estimations into the current container. We call this step as post-processing.

Note that we extended our parallel implementation to also cover the evaluation of ParClick, but the details of this implementation are omitted due to the lack of space.

5 EVALUATION

This section presents the evaluation of our multi-core version of ParClick, focusing on *strong scalability* and support for increasingly large datasets.

5.1 Experimental Setup

Datasets. We run all our experiments on three subsets of the Yandex relevance prediction dataset [26]: D1, D10, and D50, consisting of 1, 10, and 50 million first query sessions, respectively. The training sets consist of the first 80% records in the datasets, and the test sets consist of the remaining 20%. Similar to the standard experimental setup for click models evaluation [9], we remove from each test set those queries that do not appear in the corresponding training set. Table 2 reports the properties of each dataset: the number of query sessions in the training and test datasets, and the number of unique queries in the training and test data.

Table 2: Datasets used in this paper.

Dataset		D1	D10	D50
#Query Sessions	Total	1,000,000	10,000,000	50,000,000
	Training	800,000	8,000,000	40,000,000
	Test	111,953	1,361,903	7,779,226
#Unique Queries	Training	342,810	2,630,411	10,486,900
	Test	20,987	215,084	931,829

Baselines. To assess scalability, we compare all our parallel versions against efficient sequential versions of PBM and CCM. Because no prior work on large-scale training of EM-based click models exists, these baseline C++ versions are implemented in-house. We further note that ParClick is, by virtue of its parallel design and C++ implementation, orders of magnitude faster than state-of-the-art Python models, like PyClick; we consider such an apples-to-oranges comparison unfair, and we do not pursue it in this work.

Computational infrastructure. We perform our experiments on a multi-core machine featuring a 4-socket, 48-core Intel(R) Xeon(R) Gold 5118 CPU running at 2.30GHz, and 256 GB of memory. Baseline experiments run on a single core; ParClick runs using 2, 4, 8, 16, 32, and 40 threads, with one thread per core[7].

Goals and evaluation metrics. We analyze the following performance aspects of ParClick: scalability, parallelization cost, and memory usage. The metrics we use for the analysis are: (M1) execution time in seconds; (M2) speed-up, i.e., the ratio between the sequential processing time and the parallel processing time; (1) **A**verage Percentage of **C**omputation Time within Parameter Estimation Stage per It**e**ration (ACE) (time spent in line 7), **A**verage Percentage of **S**yncrhonization Time per It**e**ration (ASE) (time spent in lines 8 to 9), and **A**verage Percentage of **P**ost-Processing Time per It**e**ration (APE) as described in 4.4; and, (M3) memory footprint.

Effectiveness. The effectiveness of click models is usually evaluated by calculating log-likelihood or perplexity [9]. As the proposed algorithm 2 preserves the order of computations, no effectiveness degradation will occur for ParClick. To validate this claim, we measured the log-likelihood for all our experiments; we observed exactly the same log-likelihood as the standard sequential version for each dataset, regardless of the number of threads.

5.2 Experiments and Results

We performed $2 \times 2 \times 7 \times 3 = 84$ experiments, as we measured execution time and its components for 2 models, 2 load-balancing policies, 3 increasingly large datasets, and 7 different numbers of threads. All results are presented in Tables 3 and 4, in terms of training execution time (in seconds), speed-up, and ACE.

We observe that using more threads improves performance in all cases for both small and large datasets. Moreover, in our largest experiment, i.e., using D50, ParClick allows for 23.8 and 24.7 faster

[6]The code is available online (link not included to preserve the double-blind evaluation process).

[7]Due to the way the machine is configured, we cannot use all 48 cores for a single experiment.

Table 3: Training time, speed-up, and ACE for PBM for different datasets and thread counts.

Threads	D1						D10						D50					
	Training[s]		Speed-up		ACE [%]		Training[s]		Speed-up		ACE [%]		Training[s]		Speed-up		ACE [%]	
	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU
1	224	228	1.0	1.0	77.0	77.4	2,079	2,041	1.0	1.0	79.5	79.5	11,036	11,566	1.0	1.0	83.6	83.8
2	113	114	1.9	1.9	77.1	77.0	1,075	1,072	1.9	1.9	80.5	80.7	5,477	5,550	2.0	2.0	82.2	81.1
4	73.3	66.2	3.0	3.4	66.4	70.0	597	603	3.4	3.3	76.9	76.5	3,163	3,048	3.4	3.7	80.4	79.5
8	30	33	7.3	6.8	72.8	69.2	317	282	6.5	7.2	72.7	76.6	1,356	1,351	8.1	8.5	78.5	80.6
16	18	17	12.1	12.8	68.8	69.7	150	156	13.8	13.0	73.9	72.3	804	738	13.7	15.6	70.6	75.3
32	13	12	16.6	18.4	64.1	65.3	118	112	17.6	18.0	66.8	69.3	547	536	20.1	21.5	70.9	72.1
40	12	11	18.0	**19.0**	62.6	64.1	106	104	19.5	**19.5**	67.7	68.5	505	484	21.8	**23.8**	71.6	74.2

Table 4: Training time, speed-up, and ACE for CCM for different datasets and thread counts.

Threads	D1						D10						D50					
	Training[s]		Speed-up		ACE [%]		Training[s]		Speed-up		ACE [%]		Training[s]		Speed-up		ACE [%]	
	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU	RR	MU
1	2,216	2,219	1.0	1.0	98.0	98.0	22,745	22,704	1.0	1.0	98.4	98.4	119,408	118,559	1.0	1.0	98.7	98.7
2	1,142	1,118	1.9	2.0	95.6	97.1	11,867	11,188	1.9	2.0	97.1	97.9	58,484	58,014	2.0	2.0	97.7	98.2
4	635	616	3.5	3.6	90.7	94.5	6,215	6,062	3.7	3.7	96.2	96.3	30,843	31,455	3.9	3.8	96.4	96.8
8	331	313	6.7	7.1	88.8	95.2	3,409	3,176	6.7	7.1	89.9	95.0	16,339	15,775	7.3	7.5	91.9	95.6
16	194	168	11.4	13.1	79.8	92.4	1,925	1,705	11.8	13.3	84.5	93.5	9,933	8,408	12.0	14.1	80.1	93.2
32	141	111	15.6	20.0	68.2	89.8	1,297	1,111	17.5	20.4	75.3	89.5	6,220	5,552	19.2	21.4	78.3	89.5
40	113	100	19.6	**22.1**	76.0	88.4	1,201	960	18.9	**23.6**	72.9	94.1	6,360	4,804	18.8	**24.7**	68.3	94.5

training than the sequential versions for PBM and CCM, respectively. Thus, with ParClick, the training time is drastically reduced from several hours to a few minutes for PBM, and from more than a day to less than an hour for CCM.

In the following paragraphs, we discuss these results in more detail, focusing on our evaluation goals: scalability, parallelization cost, and memory footprint.

Scalability. There are two dimensions of scalability to discuss: input-data scaling (i.e., how does our algorithm handle increasing input data) and strong scaling (i.e., how does our algorithm respond when more resources are made available for the same dataset).

To assess input-data scalability, we observe that ParClick can process all three datasets correctly. For D50 (the largest dataset in our analysis), the execution time of training CCM is about 33 hours sequentially. Moreover, we observe that the algorithm's execution time is increasing linearly with the size of the training dataset, i.e., the training for D10 and D50 is roughly 10 and 50 times larger than that for D1. This indicates our algorithm scales to large datasets *without* any loss of efficiency due to parallelism.

To assess strong scaling, we evaluate the speed-up achieved by ParClick when increasing the number of threads made available for training. In the ideal case, as we increase the number of threads, we should see a proportional increase in speed-up. Figures 2 and 3 illustrate the observed speed-ups for PBM and CCM, respectively.

The results indicate that ParClick's performance significantly increases up to 16 threads, proportional to the number of threads. However, we see diminishing returns for more than 16 threads. This decrease is due to the increasing cost of synchronization between threads and the fixed overhead of post-processing which, as the work per thread reduces, becomes more significant. In addition,

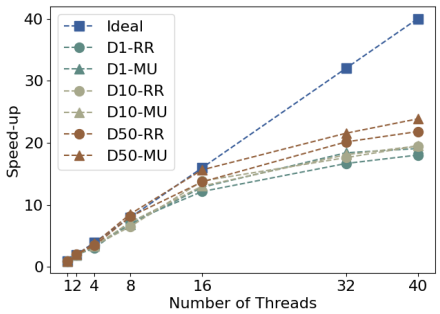

Figure 2: Training speed-up PBM for different number of threads.

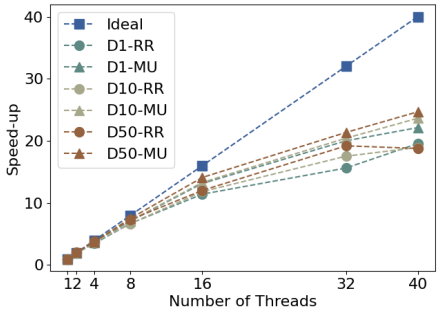

Figure 3: Training speedup of CCM for different number of threads.

NUMA effect, caching effect and hyper-threading are more pronounced in limiting the speed-up for larger threads. Combined,

these factors translate into a less than ideal speed-up gain from increasing threads.

The results further indicate that, indeed, a partitioning scheme that provides better load-balancing (such as MU) provides better speed-up, especially for the more complex models. Thus, this scheme should be preferred for better scalability, despite its slightly increased pre-processing cost.

Parallelization cost. Although parallelization provides significant performance gain for training our models, it does not come for free. The cost of parallelization may become a bottleneck for overall application performance, and, ultimately, may hinder scalability and drastically reduce the efficiency of the parallel application.

We evaluate parallelization cost as the overhead due to other tasks than computation. Thus, ACE is an indication of how well the computational resources are utilized for the actual computation, and ASE and APE represent the parallelization cost. Consequently, higher ACE values indicate a more efficient utilization of computational resources; however, they do not have a linear relationship with speed-up.

Increasing the number of threads usually results in degradation of ACE, because the synchronization and post-processing for more threads takes longer. However, as shown in Tables 3 and 4, the decrease of ACE from 1 to 40 threads is not dramatic. More importantly, ACE *improves* as the input dataset grows larger (e.g., from 64.1 to 74.2 for D1 and D50, respectively, for 40 threads, with the MU load balancing policy). This is a strong indication that ParClick is efficient for large input datasets. Additionally, our results also show that MU consistently outperforms RR in terms of ACE.

To further understand where the efficiency is lost, we also measure the time spent performing synchronization and post-processing, and calculate ASE and APE. As we aim for a high ACE, it follows that ASE and APE should be as close to zero as possible. Figure 4 presents ACE, ASE, and APE for PBM running on D50 with different numbers of threads and load balancing policies. The data indicates that APE is relatively constant for the PBM experiments, taking about 20% of the time per iteration. As more threads are used, the synchronization time also increases, as indicated by a larger ASE. These two overheads hinder the speed-up gain from adding more threads.

Similarly, Figure 5 compares ACE, ASE, and APE for CCM running on D50 with different numbers of threads and load balancing policies. Because CCM is more complex than PBM, ACE is a larger share of execution time per iteration, and APE is negligible compared to it. As expected, ASE is still increasing as the number of threads increases. However, a larger ACE still enables better speed-up gains for CCM than for PBM.

Finally, we observe that the choice of distribution policy influences parallelization cost, as it directly impacts ASE. This happens because synchronization is the time spent by threads waiting for all tasks to complete; if the tasks are severely imbalanced, this wait can become expensive for most threads, and will be reflected in the execution time. As the role of the distribution policy is to balance the workload per task, and MU is superior in this respect to RR, ASE's contribution is lower for MU. Indeed, our results from Figures 4 and 5 show that, for both PBM and CCM, MU leads to better

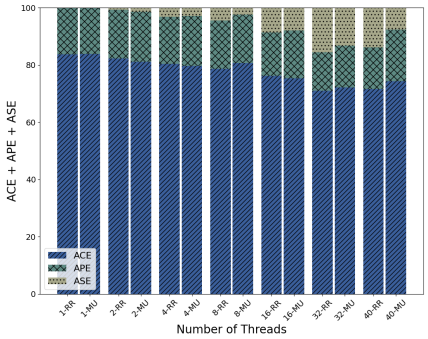

Figure 4: Comparison between ACE, ASE, and APE for PBM experiments with D50

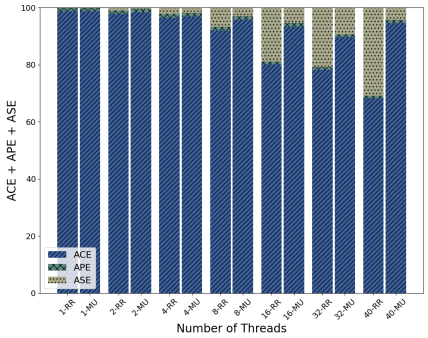

Figure 5: Comparison between ACE, ASE, and APE for CCM experiments with D50

ACE results. However, for PBM, the impact of MU on reducing ASE is small. For CCM, on the other hand, MU drastically reduces ASE.

Memory usage. We also investigate the memory footprint of ParClick with efficiency reports from the machine produced at the end of the experiments. The state-of-the-art implementation of EM-based click models in the PyClick library is memory-intensive: it requires 82.4GB of memory to train and evaluate a simple model like PBM on the D10 dataset; processing the D50 dataset is not feasible within our 256GB. ParClick needs considerably less memory to operate: when training on the D1–D50 datasets, the memory footprint varies from 0.9GB to 32.1GB for PBM, and 0.9GB to 32.3GB for CCM. Consequently, it is now possible to train these models with significantly more than 40 million query sessions. We note that, for our solution, memory usage only increases marginally when increasing the number of threads, because we focus, in our design and implementation, on avoiding any unnecessary data replication.

6 CONCLUSION AND FUTURE WORK

Existing research on click models focuses extensively on improving effectiveness. However, training these models at large scale is, currently, inefficient: most methods and tools are sequential, take excessive training time, and consume too much memory. To address these limitations, we introduce ParClick, the first scalable, generic, efficient parallel algorithm for training EM-based click models

To design ParClick, we first proposed a novel, generic algorithm for training EM-based click models, and further parallelized it using the PCAM design methodology.

We further implemented a prototype of ParClick as a multi-threaded application and used the PBM and CCM click models as case-studies. Our empirical results indicate that ParClick enables click models' training to scale up to very large click logs. Moreover, for the tested datasets and models, ParClick shows close-to-optimal efficiency when running on up to 16 threads; increasing the number of threads further still improves performance significantly, but the impact of synchronization overhead on parallelization efficiency is more pronounced. Overall, our method makes large-scale training of click models feasible and is considerably more efficient than existing PyClick implementation.

In the near future, we aim to address both the efficiency and effectiveness of more complex click models. We further work on extending our approach to multi-node, large-scale training, and further expand ParClick to use heterogeneous, accelerated architectures.

REFERENCES

[1] 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 (WSDM '19)*. Association for Computing Machinery, 474–482.

[2] Muzaffer Can Altinigneli, Claudia Plant, and Christian Böhm. 2013. Massively Parallel Expectation Maximization Using Graphics Processing Units. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery.

[3] 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 (WWW '16)*. International World Wide Web Conferences Steering Committee, 531–541.

[4] Alexey Borisov, Martijn Wardenaar, Ilya Markov, and Maarten de Rijke. 2018. A Click Sequence Model for Web Search. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. Association for Computing Machinery, 45–54.

[5] Olivier Cappé and Eric Moulines. 2009. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71 (2009), 593–613.

[6] 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 (WWW '09)*. Association for Computing Machinery, 1–10.

[7] Danqi Chen, Weizhu Chen, Haixun Wang, Zheng Chen, and Qiang Yang. 2012. Beyond Ten Blue Links: Enabling User Click Modeling in Federated Web Search. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*. Association for Computing Machinery, 463–472.

[8] Jia Chen, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. A Context-Aware Click Model for Web Search. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM '20)*. Association for Computing Machinery, New York, NY, USA, 88–96.

[9] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search*. Morgan & Claypool.

[10] 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 (WSDM '08)*. 87–94.

[11] Xinyi Dai, Jianghao Lin, Weinan Zhang, Shuai Li, Weiwen Liu, Ruiming Tang, Xiuqiang He, Jianye Hao, Jun Wang, and Yong Yu. 2021. An Adversarial Imitation Click Model for Information Retrieval. In *Proceedings of the Web Conference 2021 (WWW '21)*. Association for Computing Machinery, 1809–1820.

[12] 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 (SIGIR '08)*. Association for Computing Machinery, 331–338.

[13] Ian Foster. 1995. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc.

[14] A. Grama, V. Kumar, A. Gupta, and G. Karypis. 2003. *Introduction to Parallel Computing*. Addison-Wesley.

[15] Artem Grotov, Aleksandr Chuklin, Ilya Markov, Luka Stout, Finde Xumara, and Maarten Rijke. 2015. A Comparative Study of Click Models for Web Search. In *Proceedings of the 6th International Conference on Experimental IR Meets Multilinguality, Multimodality, and Interaction - Volume 9283 (CLEF'15)*. Springer-Verlag, 78–90.

[16] 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 (WWW '09)*. Association for Computing Machinery, 11–20.

[17] Fan Guo, Chao Liu, and Yi Min Wang. 2009. Efficient Multiple-Click Models in Web Search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM '09)*. Association for Computing Machinery, 124–131.

[18] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback *(WSDM '17)*. Association for Computing Machinery, 781–789.

[19] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

[20] Paul Lagrée, Claire Vernade, and Olivier Cappé. 2016. Multiple-Play Bandits in the Position-Based Model. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., 1605–1613.

[21] Jianghao Lin, Weiwen Liu, Xinyi Dai, Weinan Zhang, Shuai Li, Ruiming Tang, Xiuqiang He, Jianye Hao, and Yong Yu. 2021. A Graph-Enhanced Click Model for Web Search. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21)*. Association for Computing Machinery, 1259–1268.

[22] Chao Liu, Fan Guo, and Christos Faloutsos. 2009. BBM: Bayesian Browsing Model from Petabyte-scale Data. In *Proceedings of KDD* (Paris, France). 537–546.

[23] Ilya Markov, Alexey Borisov, and Maarten de Rijke. 2017. *Online Expectation-Maximization for Click Models*. Association for Computing Machinery, 2195–2198.

[24] Radford M. Neal and Geoffrey E. Hinton. 1999. *A View of the EM Algorithm That Justifies Incremental, Sparse, and Other Variants*. MIT Press, 355–368.

[25] 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 (CIKM '08)*. Association for Computing Machinery, 43–52.

[26] Pavel Serdyukov, Nick Craswell, and Georges Dupret. 2012. WSCD 2012: Workshop on Web Search Click Data 2012. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*. Association for Computing Machinery, New York, NY, USA, 771–772.

[27] Si Shen, Botao Hu, Weizhu Chen, and Qiang Yang. 2012. Personalized Click Model through Collaborative Filtering. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*. Association for Computing Machinery, 323–332.

[28] Ali Vardasbi, Maarten de Rijke, and Ilya Markov. 2020. *Cascade Model-Based Propensity Estimation for Counterfactual Learning to Rank*. Association for Computing Machinery, 2089–2092.

[29] Hongning Wang, ChengXiang Zhai, Anlei Dong, and Yi Chang. 2013. Content-Aware Click Modeling. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*. Association for Computing Machinery, 1365–1376.

[30] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. In *WSDM*. ACM, 610–618.

[31] Hai-Tao Yu, Adam Jatowt, Roi Blanco, Joemon M. Jose, and Ke Zhou. 2019. A Rank-Biased Neural Network Model for Click Modeling. In *Proceedings of the 2019 Conference on Human Information Interaction and Retrieval (CHIIR '19)*. Association for Computing Machinery, 183–191.

[32] Yuchen Zhang, Weizhu Chen, Dong Wang, and Qiang Yang. 2011. User-Click Modeling for Understanding and Predicting Search-Behavior. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*. Association for Computing Machinery, 1388–1396.