Accelerating Exact Inner Product Retrieval by CPU-GPU Systems

Long Xiang, Bo Tang, Chuan Yang
Department of Computer Science and Engineering, Southern University of Science and Technology
PCL Research Center of Networks and Communications, Peng Cheng Laboratory
{11749127@mail., tangb3@, 11612732@mail.}sustech.edu.cn

ABSTRACT

Recommender systems are widely used in many applications, e.g., social network, e-commerce. Inner product retrieval (IPR) is the core subroutine in Matrix Factorization (MF) based recommender systems. It consists of two phases: i) inner product computation and ii) top-k items retrieval. The performance bottleneck of existing solutions is inner product computation phase. Exploiting Graphics Processing Units (GPUs) to accelerate the computation intensive workloads is the gold standard in data mining and machine learning communities. However, it is not trivial to apply CPU-GPU systems to boost the performance of IPR solutions due to the nature complex of the IPR problem.

In this work, we analyze the time cost of each phase in IPR solutions at first. Second, we exploit the characteristics of CPU-GPU systems to improve performance. Specifically, the computation tasks of IPR solution are heterogeneously processed in CPU-GPU systems. Third, we demonstrate the efficiency of our proposal on four standard real datasets.

ACM Reference Format:
Long Xiang, Bo Tang, Chuan Yang. 2019. Accelerating Exact Inner Product Retrieval by CPU-GPU Systems. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3331184.3331376

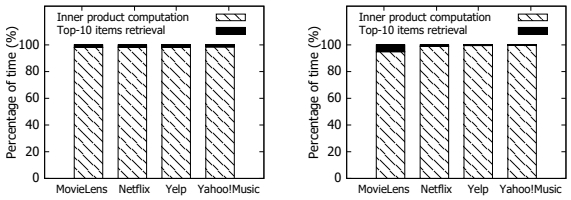
1 INTRODUCTION

Recommender systems [1, 3, 4] have been widely deployed in industry applications, e.g., product recommendation (Alibaba, Amazon), friends recommendation (Facebook, Twitter). The objective of the recommender system is suggesting top-k items to each user. There are two parts, i.e., *learning part* and *retrieval part*, in a typical recommender system. Many works in literature studied *learning part*, i.e., how to obtain size-m user matrix $\mathbf{Q} \in \mathbb{R}^{d\times m}$ and size-n item matrix $\mathbf{P} \in \mathbb{R}^{d\times n}$ accurately with missing values in user-rating-item matrix $\mathbb{R}^{m\times n}$. In this paper, we focus on the *retrieval part*, i.e., computing a top-k item list for each user. *Retrieval part* has two phases:

(1) Inner product computation, for each user $\mathbf{q} \in \mathbf{Q}$, it computes $\mathbf{q}^T\mathbf{p}$ with every item $\mathbf{p} \in \mathbf{P}$.

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331376



(a) SeqScan (b) FEXIPRO

Figure 1: Percentage of each phase on real datasets

(2) Top-k items retrieval, it retrieves top-k items with largest $\mathbf{q}^T\mathbf{p}$ values for all users in $\mathbf{Q}$.

The performance bottleneck of existing solutions for *retrieval part* is the inner product computation. To verify this, we execute brute-force solution (SeqScan) and the state-of-the-art solution (FEXIPRO [1]) for exact inner product retrieval (IPR) problem on four real-world datasets. We will introduce the details of both solutions in Section 2 shortly. Figure 1(a) and (b) depict the running time percentage of each phase on four datasets, respectively.

As illustrated in Figure 1, inner product computation contributes at least 98% and 96% of the running time (on all datasets) in brute force solution (SeqScan) and state-of-the-art solution (FEXIPRO), respectively. It motivates us to improve the performance of IPR solutions by accelerating inner product computations. In this work, we utilize the characteristics of CPU-GPU systems to boost the efficiency of IPR solutions. Specifically, we accomplish that by (i) heterogeneous processing inner product computation and top-k list retrieval in CPU-GPU systems, (ii) reducing the data transfer cost between CPU (host memory) and GPU (video memory), and (iii) avoiding unnecessary computation costs by early termination techniques.

2 PRELIMINARIES

We define the exact inner product retrieval (IPR) problem formally in Problem 1.

Problem 1. *(IPR Problem) Given a user matrix $\mathbf{Q} \in \mathbb{R}^{d\times m}$ and a item matrix $\mathbf{P} \in \mathbb{R}^{d\times n}$, the exact inner product retrieval (IPR) problem returns the items with top-k largest $\mathbf{q}^T\mathbf{p}$ in $\mathbf{q}^T\mathbf{P}$ for each user $\mathbf{q} \in \mathbf{Q}$. Ties are broken arbitrarily.*

IPR problem is computation intensive as the number of users and items are typically millions or billions in a real-world recommender system, e.g., there are almost 6 billions active products in Taobao[1].

2.1 Existing Solutions

Sequential scan (SeqScan) is the brute-force solution for IPR problem. I.e., for every user $\mathbf{q}$ in $\mathbf{Q}$, it computes the inner product $\mathbf{q}^T\mathbf{p}$

[1] http://taobao.com

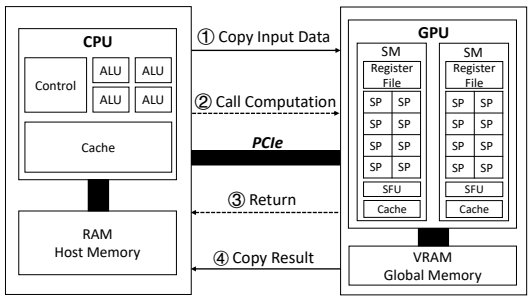


Figure 2: CPU-GPU system work flow

with each item $\mathbf{p}$ in $\mathbf{P}$. The top-k largest inner products $\mathbf{q}^T\mathbf{p}$ (in $\mathbf{q}^T\mathbf{P}$) are maintained by a size-k priority queue $\mathcal{H}$. Finally, it returns the items in $\mathcal{H}$ to the user $\mathbf{q}$. The time complexity of the brute-force solution is $O(mn(d + \log(k)))$. It is impractical to offer online top-k items retrieval in a real-world recommender system as n and m are millions, or even billions. Recently, several sequential scan based solutions [1, 3] for IPR problem are devised to improve the performance of IPR problem in commodity CPUs. To our knowledge, FEXIPRO [1] outperforms alternative approaches (e.g., [3], SeqScan) by an order of magnitude. However, it is still impractical for online top-k items recommendation in large-scale real-world applications.

We test the performance of SeqScan and FEXIPRO for IPR problem by varying k in all four datasets. As the SeqScan and FEXIPRO curves shown in Figure 3, we conclude (i) SeqScan performs the worst in all cases. It is insensitive to k as the factor $\log(k)$ is a small constant for all $k \leq 100$, and (ii) the time cost of FEXIPRO grows dramatically with the rising of k on all datasets. For example, it takes 2247.5 seconds in **Yahoo!Music** for top-10 items recommendation, as FEXIPRO illustrated in Figure 3(d). Hence, it is also impractical for large-scale recommendation systems.

2.2 CPU-GPU System

Nowadays, Graphics Processing Units (GPUs) have been widely used in data mining and machine learning applications as they are good candidates for parallelization [2]. It is important to consider how to exploit CPU-GPU systems to improve the efficiency of IPR problem. Figure 2 illustrates the work flow of a modern commodity chip which integrated both CPU and GPU. Typically, CPU is the host. For GPU computation tasks, the host issues instructions to move data from host memory to video memory via the PCIe interconnect (step ① in Figure 2). After that, the host calls GPU (step ②) (a.k.a., kernel invocation) to execute multiple instances (called *threads*) to accomplish the computation task with the data store in video memory. Kernel invocation is asynchronous, so the driver will return control to the host immediately after it has launched the kernel (step ③). The computation results will be moved back to host memory via PCIe bus for further processing in the host (step ④).

The IPR problem is well suited to CPU-GPU systems as it is a data parallel task and GPUs are massively parallel processors. However, it is not trivial to use CPU-GPU systems to improve the efficiency of IPR solutions. The challenges are: (i) how to fully utilize the limited video memory (a few GBs), and (ii) how to reduce data movement overhead between host memory and video memory.

Table 1: Statistics of the Four Real Datasets

Datasets	m	n	d	$\mathbf{Q}^T\mathbf{P}$ size (GB)
MovieLens	247753	33670	50	31.08
Netflix	480189	17770	50	31.79
Yelp	552339	77079	50	158.60
Yahoo!Music	1000990	624961	50	2330.47

2.3 Experimental Setting

All algorithms are implemented by C++. The experiments of CPU-based solutions are run with single thread. The operating system is CentOS 7.4. The GPU is NVIDIA GeForce(R) Titan Xp, 12 GB video memory with 547.7GB/s memory bandwidth. Each reported measurement is the average of 10 times experiments.

We used four standard real datasets in previous researches on recommender systems [1]. Each dataset contains a matrix $\mathbf{Q}$ and a matrix $\mathbf{P}$. Table 1 summarizes the size of each matrix in four datasets. The last column is the size of the matrix $\mathbf{Q}^T\mathbf{P} \in \mathbb{R}^{m\times n}$ for each dataset.

3 CPU-GPU BASED IPR SOLUTION

In this section, we exploit CPU-GPU systems to boost the performance of IPR solution. We first devise a matrix partition scheme (utilizing video memory fully) to accelerate inner product computation in Section 3.1. Then we improve the efficiency of IPR solution by customizing Bitonic sorting in GPU in Section 3.2.

3.1 GPU-based Inner Product Computation

The inner product computation between each user $\mathbf{q} \in \mathbf{Q}$ with item matrix $\mathbf{P}$ can be highly parallel computed in GPU. While inner product computation $\mathbf{q}^T\mathbf{P}$ is readily parallelized, how to fully unlock the power of GPU is still under exploration.

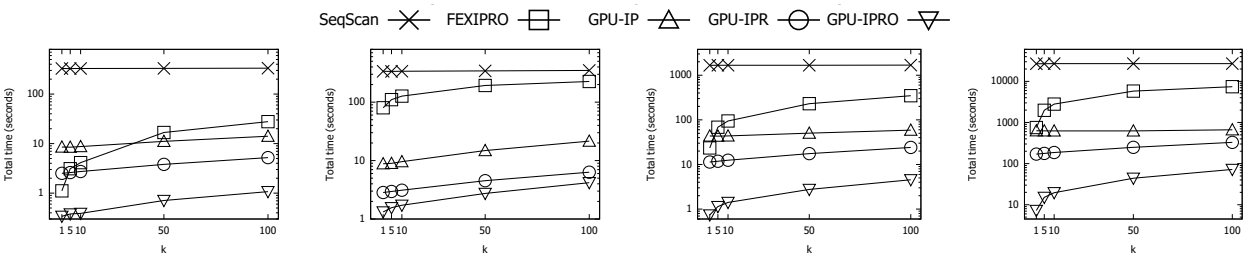
The simplest scheme is *all-in-one*, i.e., compute $\mathbf{Q}^T\mathbf{P}$ parallel in GPU. However, it is impractical in many applications as the matrix size $\mathbf{Q}^T\mathbf{P}$ ($\mathbb{R}^{m\times n}$) is always larger than the video memory size in GPU, e.g., the video memory is 2-12 GB in commodity GPUs, but the minimum matrix size $\mathbf{Q}^T\mathbf{P}$ is 31.08 GB in our experimental datasets (see last column in Table 1).

Due to the limited size of video memory, we propose to compute the inner products *batch-by-batch*. Suppose the video memory size is $\mathcal{B}$, it is constant when the hardware is specified. The remaining question is to decide the batch processing subset of $\mathbf{Q}$ and $\mathbf{P}$, denotes as $\mathbf{Q}_s$ and $\mathbf{P}_s$, respectively.

We formulate the problem as follows, where $Cost(\mathbf{Q}_s, \mathbf{P}_s)$ is the computation cost of $\mathbf{Q}_s^T\mathbf{P}_s$, and $Size(\mathbf{Q}_s^T\mathbf{P}_s)$ is the result size of $\mathbf{Q}_s^T\mathbf{P}_s$.

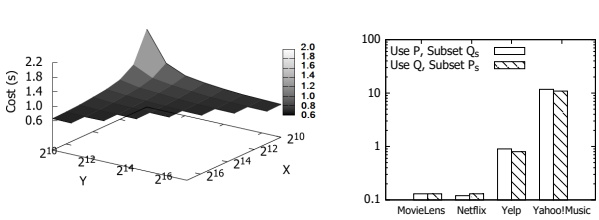
$$\begin{aligned} \text{Minimize:} \quad & Cost(\mathbf{Q}, \mathbf{P}) = \textstyle\sum_{\mathbf{Q}_s \in \mathbf{Q}} \sum_{\mathbf{P}_s \in \mathbf{P}} Cost(\mathbf{Q}_s, \mathbf{P}_s) \\ \text{subject to:} \quad & Size(\mathbf{Q}_s^T\mathbf{P}_s) \leq \mathcal{B} \\ & \mathbf{Q}_s \in \mathbb{R}^{x\times d}, x \in [1, m] \\ & \mathbf{P}_s \in \mathbb{R}^{y\times d}, y \in [1, n] \end{aligned} \tag{1}$$

Inner product computation is a highly parallelizable task well suited to GPUs. Theoretically, the minimum computation cost achieves by fully utilizing the video memory budget $\mathcal{B}$. This is also confirmed



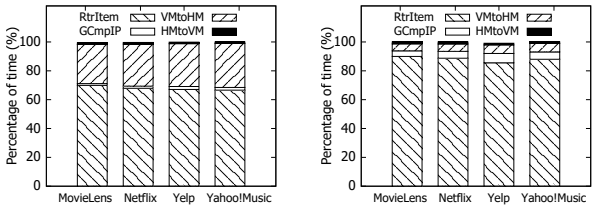
(a) **MovieLens** (b) **Netflix** (c) **Yelp** (d) **Yahoo!Music**

Figure 3: Performance of IPR solutions



(a) Test on **Yelp** (b) Performance testing

Figure 4: Matrix partition scheme



(a) GPU-IP (b) GPU-IPR

Figure 5: Time cost break down

by the testing results in Figure 4(a), the larger of $Size(\mathbf{Q}_S^T\mathbf{P}_s)$, the cheaper of $Cost(\mathbf{Q},\mathbf{P})$, as the diagonal of x-y plane in Figure 4(a). For simplicity, the matrix partition scheme in our solution is: (i) use the whole set of $\mathbf{Q}$, and determine $\mathbf{P}_S$ by $\mathcal{B}$ and $Size(\mathbf{Q})$, or (ii) vice verse. We test the performance of the above two ideas in Figure 4(b), Option (i) performs slightly better than Option (ii) in large datasets (i.e., **Yelp** and **Yahoo!Music**). We use Option (i) in the rest of this paper.

Experimental Evaluation: Here, we denote the IPR solution with GPU-paralleled $\mathbf{Q}^T\mathbf{P}$ (GPU-IP) computation. It computes the inner products parallel in the GPU side, and retrieves top-k items in the CPU side. Its performance on all datasets is illustrated as GPU-IP curve in Figure 3. Obviously, GPU-IP always performs better than CPU-based solutions. Besides, the running time of GPU-IP solution is also insensitive to k.

In order to identify its performance bottleneck, we break down the time cost of GPU-IP with $k = 10$ in each dataset in Figure 5. The time cost components of GPU-IP are (1) copy data from host memory to video memory (HMtoVM), (2) compute inner products in GPU side (GCmpIP), (3) copy inner product values from video memory to host memory (VMtoHM) and (4) retrieve top-k item list (RtrItem).

Through the cost break down results in Figure 5, we have the following three observations.

(1) The time cost of GPU based inner product computation (GCmpIP) at most 2.05% in all datasets, it contributes 96% and 98% of total cost in CPU-based SeqScan and FEXIPRO (see Figure 1).
(2) The majority of GPU-IP (67%-71%) time cost is retrieving top-k item list (RtrItem).
(3) Copying inner product values from video memory to host memory (VMtoHM) also incurs a large cost (27%-31%).

In summary, computing inner products is not the performance bottleneck in GPU-IP. The performance bottleneck turns to retrieve top-k item list, and copying inner product values from video memory to host memory also plays an important role.

3.2 GPU-based Top-k Items Retrieval

In this section, we boost the performance GPU-IP by reducing the cost of retrieving top-k items for each user. Bitonic sort is a parallel sorting algorithm. However, its performance scales poorly with the input array size [2]. The research question in this section is *how to utilize GPU computing performance and bandwidth fully.*

Our solution framework GPU-IPR is illustrated in Figure 6. The operations in IPR solution are heterogeneously processed by a combination of processors (i.e., CPU and GPU). It computes the inner products on GPU, GPU-IPR computes the inner product values between user matrix $\mathbf{Q}$ and a subset of items in $\mathbf{P}$, i.e., $\mathbf{Q}^T\mathbf{P}_s$. This matrix partition scheme also provides benefits to Bitonic sorting in the next step. The inner product values of each user ($\mathbf{q}^T\mathbf{P}_s$) are sorted parallel in different GPU threads groups with Bitonic sorting function. The key point is determining the number of inner product values for each user (see g_s in Figure 6) as it affects the performance of Bitonic sorting seriously. The size of g_s should fit in the shared memory of each threads group as it incurs minimum cache access latency in GPU cache hierarchy. Hence, $g_s = 1024$ in our experiments. Finally, the top-k inner product values for each user return to CPU (i.e., host memory). They update the result priority queue $\mathcal{H}_i$ for every user $\mathbf{q}_i \in Q$.

The GPU-IPR curves in Figure 3 show the performance of GPU-IPR on all datasets. It offers 2.6-6.11 times for RtrItem and 2.02-3.83 times speedup over GPU-IP in all cases. We then break down the cost of GPU-IPR in Figure 5(b). Obviously, the cost of VMtoHM (copy inner product values from video memory to host memory) is reduced significantly as it only returns the top-k result of each user in each batch. In addition, the majority of GPU-IPR (85% -90%)

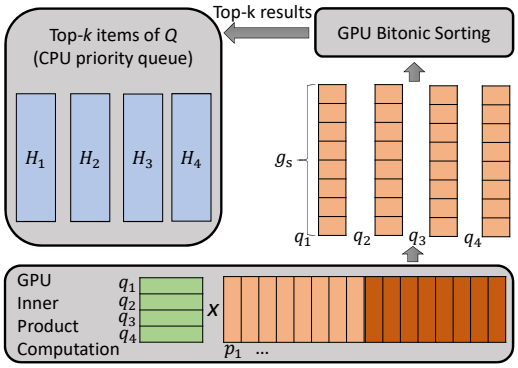


Figure 6: GPU-IPR framework

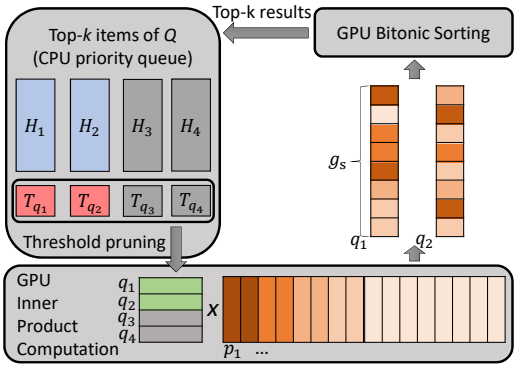


Figure 7: GPU-IPRO framework

is still retrieving top-k items for all users. It consists of the cost of Bitonic sorting in GPU (80%-96%) and the cost of priority queue maintenance in CPU (4%-20%).

4 PERFORMANCE OPTIMIZATION

In this section, we improve the efficiency of GPU-IPR for IPR problem by avoiding unnecessary computation cost with early termination technique. The core idea is exploiting Cauchy-Schwarz inequality for early termination, as introduced in Lemma 1. Interestingly, it not only reduces the number of inner products computations as usual but also accelerates the performance of Bitonic sorting as it has fewer unsorted values.

Lemma 1. *Suppose the k-th largest item value of user $\mathbf{q}$ is S_k, if $||\mathbf{q}|| \cdot ||\mathbf{p}|| \leq S_k$, thus, we have $\mathbf{q}^T\mathbf{p} \leq S_k$.*

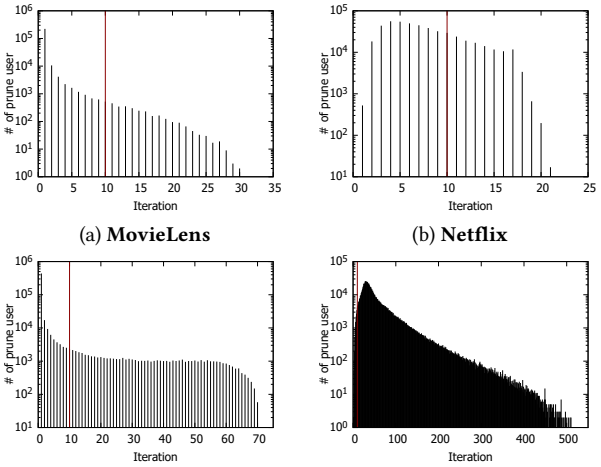
Proof. The proof is trivial as $\mathbf{q}^T\mathbf{p} \leq ||\mathbf{q}|| \cdot ||\mathbf{p}||$. □

We present the optimized GPU-IPR (i.e., GPU-IPRO) for IPR problem in Figure 7. The items in $\mathbf{P}$ are sorted by the descending order of $||\mathbf{p}||$ initially. GPU-IPRO works the same as GPU-IPR at the first iteration. Before the second iteration, GPU-IPRO applies Lemma 1 to prune the users which already obtained top-k item list. In particular, for user $\mathbf{q}$, it the k-th largest item value is compared with $T_{q_i} = ||\mathbf{q}_i|| \cdot ||\mathbf{p}_x||$ (as illustrated in Figure 7), where $\mathbf{p}_x$ has the largest $||\mathbf{p}||$ in the rest item matrix $\mathbf{P} - \mathbf{P}_s$. Only the surviving users, i.e., its $S_k \leq T_{q_i}$, will pass to the next iteration. For example, only $\mathbf{q}_1$ and $\mathbf{q}_2$ will be processed at the second iteration, $\mathbf{q}_3$ and $\mathbf{q}_4$ are pruned (as gray cells shown in Figure 7). Many unnecessary inner products computations are avoided through this. Meanwhile, it reduces the input size of Bitonic sorting in the next step.

The pruning ability of our proposed idea is evaluated on four datasets, the experimental results with $k = 10$ are shown in Figure 8. It prunes 98.88%, 76.61%, 88.69% and 1.57% of users at first 10 iterations in **MovieLens**, **Netflix**, **Yelp** and **Yahoo!Music**, respectively. We evaluate the performance of GPU-IPRO in Figure 3. As shown by GPU-IPRO curves, GPU-IPRO outperforms GPU-IP and GPU-IPR up to 94.57 times and 24.72 times in four datasets, respectively.

5 CONCLUSION

In this paper, we propose several techniques (e.g., matrix partition scheme, CPU-GPU heterogeneous processing) to improve the performance of the exact inner product retrieval (IPR) with recommender system at CPU-GPU systems. The efficiency of our proposal



(a) **MovieLens** (b) **Netflix**

(c) **Yelp** (d) **Yahoo!Music**

Figure 8: Pruning ability Lemma 1 in CPU-GPU system

is demonstrated by extensive experimental studies on four standard real datasets. A promising direction for future work is to exploit the advanced techniques in modern hardware (e.g., branch prediction, hardware prefetching) to further improve the performance of our proposal GPU-IPRO.

6 ACKNOWLEDGMENTS

This work was supported by the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No.JCYJ20180302174301157), the Guangdong Natural Science Foundation (Grant No.2018A030310129), the National Science Foundation of China (NSFC No.61802163), and PCL Future Regional Network Facilities for Large-scale Experiments and Applications (PCL2018KP001).

REFERENCES

[1] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems. In *SIGMOD*. ACM, 835–850.
[2] Takazumi Matsumoto and Man Lung Yiu. 2015. Accelerating exact similarity search on CPU-GPU systems. In *ICDM*. IEEE, 320–329.
[3] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. 2015. LEMP: Fast retrieval of large entries in a matrix product. In *SIGMOD*. ACM, 107–122.
[4] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *SIGIR*. ACM, 183–192.