Reverse Maximum Inner Product Search: How to efficiently find users who would like to buy my item?

DAICHI AMAGATA, Osaka University, JST PRESTO, Japan
TAKAHIRO HARA, Osaka University, Japan

The MIPS (maximum inner product search), which finds the item with the highest inner product with a given query user, is an essential problem in the recommendation field. It is usual that e-commerce companies face situations where they want to promote and sell new or discounted items. In these situations, we have to consider a question: who are interested in the items and how to find them? This paper answers this question by addressing a new problem called reverse maximum inner product search (reverse MIPS). Given a query vector and two sets of vectors (user vectors and item vectors), the problem of reverse MIPS finds a set of user vectors whose inner product with the query vector is the maximum among the query and item vectors. Although the importance of this problem is clear, its straightforward implementation incurs a computationally expensive cost.

We therefore propose Simpfer, a simple, fast, and exact algorithm for reverse MIPS. In an offline phase, Simpfer builds a simple index that maintains a lower-bound of the maximum inner product. By exploiting this index, Simpfer judges whether the query vector can have the maximum inner product or not, for a given user vector, in a constant time. Besides, our index enables filtering user vectors, which cannot have the maximum inner product with the query vector, in a batch. We theoretically demonstrate that Simpfer outperforms baselines employing state-of-the-art MIPS techniques. Furthermore, our extensive experiments on real datasets show that Simpfer is about 500–8000 times faster than the baselines.

CCS Concepts: • **Information systems** → **Recommender systems**; **Proximity search**.

Additional Key Words and Phrases: reverse maximum inner product search, high dimensional data, algorithm

ACM Reference Format:
Daichi Amagata and Takahiro Hara. 2021. Reverse Maximum Inner Product Search: How to efficiently find users who would like to buy my item?. In *Fifteenth ACM Conference on Recommender Systems (RecSys '21), September 27-October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3460231.3474229

1 INTRODUCTION

The MIPS (maximum inner product search) problem, or k-MIPS problem, is an essential tool in the recommendation field. Given a query (user) vector, this problem finds the k item vectors with the highest inner product with the query vector among a set of item vectors. The search result, i.e., k item vectors, can be used as recommendation for the user, and the user and item vectors are obtained via Matrix Factorization, which is well employed in recommender systems [4, 7, 12, 29]. Although some learned similarities via MLP (i.e., neural networks) have also been devised, e.g., in [35, 37], [25] has actually demonstrated that inner product-based (i.e., Matrix Factorization-based) recommendations show better performances than learned similarities. We hence focus on inner product between d-dimensional vectors that are obtained via Matrix Factorization.

Manuscript submitted to ACM

Table 1. Example of reverse MIPS where $\mathbf{q} = \mathbf{p}_5$. The rows at right illustrate the result of MIPS on $\mathbf{P}$ for each $\mathbf{u} \in \mathbf{Q}$.

$\mathbf{Q}$		$\mathbf{P}$		$\mathbf{p}^*$ (MIPS result)	
$\mathbf{u}_1$	$\langle 3.1, 0.1 \rangle$	$\mathbf{p}_1$	$\langle 2.8, 0.6 \rangle$	$\mathbf{u}_1$	$\mathbf{p}_3$
$\mathbf{u}_2$	$\langle 2.5, 2.0 \rangle$	$\mathbf{p}_2$	$\langle 2.5, 1.8 \rangle$	$\mathbf{u}_2$	$\mathbf{p}_2$
$\mathbf{u}_3$	$\langle 1.5, 2.2 \rangle$	$\mathbf{p}_3$	$\langle 3.2, 1.0 \rangle$	$\mathbf{u}_3$	$\mathbf{p}_5$
$\mathbf{u}_4$	$\langle 1.8, 3.2 \rangle$	$\mathbf{p}_4$	$\langle 1.4, 2.6 \rangle$	$\mathbf{u}_4$	$\mathbf{p}_5$
		$\mathbf{p}_5$	$\langle 0.5, 3.4 \rangle$		

1.1 Motivation

The k-MIPS problem is effective for the case where a user wants to know items that s/he prefers (i.e., user-driven cases), but e-commerce companies usually face situations where they want to advertise an item, which may be new or discounted one, to users, which corresponds to item-driven cases. Trivially, an effective advertisement is to recommend such an item to users who would be interested in this item.

In the context of the k-MIPS problem, if this item is included in the top-k item set for a user, we should make an advertisement of the item to this user. That is, *we should find a set of such users*. This paper addresses this new problem, called *reverse k-MIPS problem*. To ease of presentation, this section assumes that $k = 1$ (the general case is defined in Section 2). Given a query vector $\mathbf{q}$ (the vector of a target item) and two sets of d-dimensional vectors $\mathbf{Q}$ (set of user vectors) and $\mathbf{P}$ (set of item vectors), the reverse MIPS problem finds all user vectors $\mathbf{u} \in \mathbf{Q}$ such that $\mathbf{q} = \arg\max_{\mathbf{p} \in \mathbf{P} \cup \{\mathbf{q}\}} \mathbf{p} \cdot \mathbf{u}$.

Example 1. *Table 1 illustrates* $\mathbf{Q}$, $\mathbf{P}$, *and the MIPS result, i.e.,* $\mathbf{p}^* = \arg\max_{\mathbf{p} \in \mathbf{P}} \mathbf{u} \cdot \mathbf{p}$, *of each vector in* $\mathbf{Q}$. *Let* $\mathbf{q} = \mathbf{p}_5$, *and the result of reverse MIPS is* $\{\mathbf{u}_3, \mathbf{u}_4\}$ *because* $\mathbf{p}_5$ *is the top-1 item for* $\mathbf{u}_3$ *and* $\mathbf{u}_4$. *When* $\mathbf{q} = \mathbf{p}_1$, *we have no result, because* $\mathbf{p}_1$ *is not the top-1 item* $\forall \mathbf{u} \in \mathbf{Q}$. *Similarly, when* $\mathbf{q} = \mathbf{p}_2$, *the result is* $\{\mathbf{u}_2\}$.

From this example, we see that, if an e-commerce service wants to promote the item corresponding to $\mathbf{p}_5$, this service can obtain the users who would prefer this item through the reverse MIPS, and sends them a notification about this item.

The reverse k-MIPS problem is an effective tool not only for item-driven recommendations but also market analysis. Assume that we are given a vector of a new item, $\mathbf{q}$. It is necessary to design an effective sales strategy to gain a profit. Understanding the features of users that may prefer the item is important for the strategy. Solving the reverse k-MIPS of the query vector $\mathbf{q}$ supports this understanding.

1.2 Challenge

The above practical situations clarify the importance of reverse MIPS. Because e-commerce services have large number of users and items, $|\mathbf{Q}|$ and $|\mathbf{P}|$ are large. In addition, a query vector is not pre-known and is specified on-demand fashion. The reverse k-MIPS is therefore conducted online and is computationally-intensive task. Now the question is how to efficiently obtain the reverse MIPS result for a given query.

A straightforward approach is to run a state-of-the-art exact MIPS algorithm *for every vector in* $\mathbf{Q}$ and check whether or not $\mathbf{q} = \arg\max_{\mathbf{p} \in \mathbf{P} \cup \{\mathbf{q}\}} \mathbf{u} \cdot \mathbf{p}$. This approach obtains the exact result, but it incurs unnecessary computation. The poor performance of this approach is derived from the following observations. First, we do not need the MIPS result of $\mathbf{u}$ when $\mathbf{q}$ does not have the maximum inner product with $\mathbf{u}$. Second, this approach certainly accesses all user vectors in $\mathbf{Q}$, although many of them do not contribute to the reverse MIPS result. However, it is not trivial to skip evaluations of

some user vectors without losing correctness. Last, its theoretical cost is the same as the brute-force case, i.e., $O(nmd)$ time, where $n = |\mathbf{Q}|$ and $m = |\mathbf{P}|$, which is not appropriate for online computations. These concerns pose challenges for solving the reverse MIPS problem efficiently.

1.3 Contribution

To address the above issues, we propose *Simpfer*, a simple, fast, and exact algorithm for reverse MIPS. The general idea of Simpfer is to efficiently solve the decision version of the MIPS problem. Because the reverse MIPS of a query $\mathbf{q}$ requires a yes/no decision for each vector $\mathbf{u} \in \mathbf{Q}$, it is sufficient to know whether or not $\mathbf{q}$ can have the maximum inner product for $\mathbf{u}$. Simpfer achieves this in $O(1)$ time in many cases by exploiting its index built in an offline phase. This index furthermore supports a constant time filtering that prunes vectors in a batch if their answers are no. We theoretically demonstrate that the time complexity of Simpfer is lower than $O(nmd)$.

The summary of our contributions is as follows:

- We address the problem of reverse k-MIPS. To our knowledge, this is the first work to study this problem.
- We propose Simpfer as an exact solution to the reverse MIPS problem. Simpfer solves the decision version of the MIPS problem at both the group-level and the vector-level efficiently. Simpfer is surprisingly simple, but our analysis demonstrates that Simpfer theoretically outperforms a solution that employs a state-of-the-art exact MIPS algorithm.
- We conduct extensive experiments on four real datasets, MovieLens, Netflix, Amazon, and Yahoo!. The results show that Simpfer is about 500–8000 times faster than baselines. For example, Simpfer can process a reverse 10-MIPS query on a set of 2 million user vectors within 0.5 seconds, while the baselines need more than a half hour to deal with it.
- Simpfer is easy to deploy: *if recommender systems have user and item vector sets that are designed in the inner product space, they are ready to use Simpfer via our open source implementation*[1]. This is because Simpfer is unsupervised and has only a single parameter (the maximum value of k) that is easy to tune and has no effect on the running time of online processing.

Organization. The rest of this paper is organized as follows. We formally define our problem in Section 2. We review related work in Section 3. Our proposed algorithm is presented in Section 4, and the experimental results are reported in Section 5. Last, we conclude this paper in Section 6.

2 PROBLEM DEFINITION

Let $\mathbf{P}$ be a set of d-dimensional real-valued item vectors, and we assume that d is high [21, 26]. Given a query vector, the maximum inner product search (MIPS) problem finds

$$\mathbf{p}^* = \arg\max_{\mathbf{p} \in \mathbf{P}} \mathbf{p} \cdot \mathbf{q}.$$

The general version of the MIPS problem, i.e., the k-MIPS problem, is defined as follows:

Definition 1 (k-MIPS problem). *Given a set of vectors $\mathbf{P}$, a query vector $\mathbf{q}$, and k, the k-MIPS problem finds k vectors in $\mathbf{P}$ that have the highest inner products with $\mathbf{q}$.*

[1] https://github.com/amgt-d1/Simpfer

For a user (i.e., query), the k-MIPS problem can retrieve k *items* (e.g., vectors in $\mathbf{P}$) that the user would prefer. Different from this, the reverse k-MIPS problem can retrieve *a set of users* who would prefer a given item. That is, in the reverse k-MIPS problem, a query can be an item, and *this problem finds users attracted by the query item.* Therefore, the reverse k-MIPS is effective for advertisement and market analysis, as described in Section 1. We formally define this problem[2].

Definition 2 (Reverse k-MIPS problem). *Given a query (item) vector* $\mathbf{q}$*,* k*, and two sets of vectors* $\mathbf{Q}$ *(set of user vectors) and* $\mathbf{P}$ *(set of item vectors), the reverse* k*-MIPS problem finds all vectors* $\mathbf{u} \in \mathbf{Q}$ *such that* $\mathbf{q}$ *is included in the* k*-MIPS result of* $\mathbf{u}$ *among* $\mathbf{P} \cup \{\mathbf{q}\}$.

Note that $\mathbf{q}$ can be $\mathbf{q} \in \mathbf{P}$, as described in Example 1. We use n and m to denote $|\mathbf{Q}|$ and $|\mathbf{P}|$, respectively.

Our only assumption is that there is a maximum k that can be specified, denoted by k_{max}. This is practical, because k should be small, e.g., $k = 5$ [15] or $k = 10$ [3], to make applications effective. (We explain how to deal with the case of $k > k_{max}$ in Section 4.1.) This paper develops an exact solution to the new problem in Definition 2.

3 RELATED WORK

Exact k-MIPS Algorithm. The reverse k-MIPS problem can be solved exactly by conducting an exact k-MIPS algorithm for each user vector in $\mathbf{Q}$. The first line of solution to the k-MIPS problem is a tree-index approach [8, 16, 24]. For example, [24] proposed a tree-based algorithm that processes k-MIPS not only for a single user vector but also for some user vectors in a batch. Unfortunately, the performances of the tree-index algorithms degrade for large d because of the curse of dimensionality.

LEMP [27, 28] avoids this issue and significantly outperforms the tree-based algorithms. LEMP uses several search algorithms according to the norm of each vector. In addition, LEMP devises an early stop scheme of inner product computation. During the computation of $\mathbf{u} \cdot \mathbf{q}$, LEMP computes an upper-bound of $\mathbf{u} \cdot \mathbf{q}$. If this bound is lower than an intermediate k-th maximum inner product, q cannot be in the final result, thus the inner product computation can be stopped. LEMP is actually designed for the top-k inner product join problem: for each $\mathbf{u} \in \mathbf{Q}$, it finds the k-MIPS result of $\mathbf{u}$. Therefore, LEMP can solve the reverse k-MIPS problem, but it is not efficient as demonstrated in Section 5.

FEXIPRO [18] further improves the early stop of inner product computation of LEMP. Specifically, FEXIPRO exploits singular value decomposition, integer approximation, and a transformation to positive values. These techniques aim at obtaining a tighter upper-bound of $\mathbf{u} \cdot \mathbf{q}$ as early as possible. [18] reports that state-of-the-art tree-index algorithm [24] is completely outperformed by FEXIPRO. Maximus [1] takes hardware optimization into account. It is, however, limited to specific CPUs, so we do not consider Maximus. Note that LEMP and FEXIPRO are heuristic algorithms, and $O(nmd)$ time is required for the reverse k-MIPS problem.

Approximation k-MIPS Algorithm. To solve the k-MIPS problem in sub-linear time by sacrificing correctness, many works proposed approximation k-MIPS algorithms. There are several approaches to the approximation k-MIPS problem: sampling-based [6, 21, 34], LSH-based [14, 23, 26, 32], graph-based [20, 22, 38], and quantization approaches [9, 13]. They have both strong and weak points. For example, LSH-based algorithms enjoy a theoretical accuracy guarantee. However, they are empirically slower than graph-based algorithms that have no theoretical performance guarantee. Literature [3] shows that the MIPS problem can be transformed into the Euclidean nearest neighbor search problem,

[2] Actually, the reverse top-k query (and its variant), a similar concept to the reverse k-MIPS problem, has been proposed in [30, 31, 36]. It is important to note that these works do not suit recent recommender systems. First, they assume that d is low (d is around 5), which is not probable in Matrix Factorization. Second, they consider the Euclidean space, whereas inner product is a non-metric space. Because the reverse top-k query processing algorithms are optimized for these assumptions, they cannot be employed in Matrix Factorization-based recommender systems and cannot solve (or be extended for) the reverse k-MIPS problem.

but it still cannot provide the correct answer. Besides, existing works that address the (reverse) nearest neighbor search problem assume low-dimensional data [33] or consider approximation algorithms [19].

Since this paper focuses on the exact result, these approximation k-MIPS algorithms cannot be utilized. In addition, approximate answers may lose effectiveness of the reverse k-MIPS problem. If applications cannot contain users, who are the answer of the k-MIPS problem, these users may lose chances of knowing the target item, which would reduce profits. On the other hand, if applications contain users, who are *not* the answer of the k-MIPS problem, as an approximate answer, they advertise the target item to users who are not interested in the item. This also may lose future profits, because such users may stop receiving advertisements if they get those of non-interesting items.

4 PROPOSED ALGORITHM

To efficiently solve the reverse MIPS problem, we propose Simpfer. Its general idea is to efficiently solve the decision version of the k-MIPS problem.

Definition 3 (k-MIPS decision problem). *Given a query* $\mathbf{q}$*,* k*, a user vector* $\mathbf{u}$*, and* $\mathbf{P}$*, this problem returns yes (no) if* $\mathbf{q}$ *is (not) included in the* k*-MIPS result of* $\mathbf{u}$*.*

Notice that *this problem does not require the complete* k*-MIPS result*. We can terminate the k-MIPS of $\mathbf{u}$ whenever it is guaranteed that $\mathbf{q}$ is (not) included in the k-MIPS result.

To achieve this early termination efficiently, it is necessary to obtain a lower-bound and an upper-bound of the k-th highest inner product of $\mathbf{u}$. Let ϕ and μ respectively be a lower-bound and an upper-bound of the k-th highest inner product of $\mathbf{u}$ on $\mathbf{P}$. If $\phi \geq \mathbf{u} \cdot \mathbf{q}$, it is guaranteed that $\mathbf{q}$ does not have the k highest inner product with $\mathbf{u}$. Similarly, if $\mu \leq \mathbf{u} \cdot \mathbf{q}$, it is guaranteed that $\mathbf{q}$ has the k highest inner product with $\mathbf{u}$. This observation implies that we need to efficiently obtain ϕ and μ. Simpfer does pre-processing to enable it in an offline phase. Besides, since $n = |\mathbf{Q}|$ is often large, accessing all user vectors is also time-consuming. This requires a filtering technique that enables the pruning of user vectors that are not included in the reverse k-MIPS result *in a batch*. During the pre-processing, Simpfer arranges $\mathbf{Q}$ so that batch filtering is enabled. Simpfer exploits the data structures built in the pre-processing phase to quickly solve the k-MIPS decision problem.

4.1 Pre-processing

The objective of this pre-processing phase is to build data structures that support efficient computation of a lower-bound and an upper-bound of the k-th highest inner product for each $\mathbf{u_i} \in \mathbf{Q}$, for arbitrary queries. We utilize Cauchy–Schwarz inequality for upper-bounding. Hence we need the Euclidean norm $\|\mathbf{u_i}\|$ for each $\mathbf{u_i} \in \mathbf{Q}$. To obtain a lower-bound of the k-th highest inner product, we need to access at least k item vectors in $\mathbf{P}$. The norm computation and lower-bound computation are independent of queries (as long as $k \leq k_{max}$), so they can be pre-computed. In this phase, Simpfer builds the following array for each $\mathbf{u_i} \in \mathbf{Q}$.

Definition 4 (Lower-bound array). *The lower-bound array* L_i *of a user vector* $\mathbf{u_i} \in \mathbf{Q}$ *is an array whose* j*-th element,* L_i^j*, maintains a lower-bound of the* j*-th inner product of* $\mathbf{u_i}$ *on* $\mathbf{P}$*, and* $|L_i| = k_{max}$*.*

Furthermore, to enable batch filtering, Simpfer builds a *block*, which is defined below.

Algorithm 1: Pre-Processing of Simpfer

Input: $\mathbf{Q}$, $\mathbf{P}$, and k_{max}

```
for each u_i ∈ Q do
    Compute ||u_i||
for each p_j ∈ P do
    Compute ||p_j||
Sort Q and P in descending order of norm size
P' ← the first O(k_max) vectors in P
for each u_i ∈ Q do
    R ← k_max vectors p ∈ P' that maximize u_i · p
    for j = 1 to k_max do
        L_i^j ← u_i · p, where p provides the j-th highest inner product with u_i in R
B ← ∅, B ← a new block
for each u_i ∈ Q do
    Q(B) ← Q(B) ∪ {u_i}
    for j = 1 to k_max do
        L^j(B) ← min{L^j(B), L_i^j}
    if |Q(B)| = O(log n) then
        B ← B ∪ {B}
        B ← a new block
```

Definition 5 (Block). *A block* $\mathbf{B}$ *is a subset of* $\mathbf{Q}$. *The set of vectors belonging to* $\mathbf{B}$ *is represented by* $\mathbf{Q}(\mathbf{B})$. *Besides, we use* $L(\mathbf{B})$ *to represent the lower-bound array of this block, and*

$$L^j(\mathbf{B}) = \min_{\mathbf{u_i} \in \mathbf{Q}(\mathbf{B})} L_i^j \tag{1}$$

The block size $|\mathbf{Q}(\mathbf{B})|$ can be arbitrarily determined, and we set $|\mathbf{Q}(\mathbf{B})| = O(\log n)$ to avoid system parameter setting.

Pre-processing algorithm. Algorithm 1 describes the pre-processing algorithm of Simpfer.

(1) Norm computation: First, for each $\mathbf{u} \in \mathbf{Q}$ and $\mathbf{p} \in \mathbf{P}$, its norm is computed. Then, $\mathbf{Q}$ and $\mathbf{P}$ are sorted in descending order of norm.

(2) Lower-bound array building: Let $\mathbf{P'}$ be the set of the first $O(k_{max})$ vectors in $\mathbf{P}$. For each $\mathbf{u_i} \in \mathbf{Q}$, L_i is built by using $\mathbf{P'}$. That is, $L_i^j = \mathbf{u_i} \cdot \mathbf{p}$, where $\mathbf{p} \in \mathbf{P'}$ yields the j-th highest inner product for $\mathbf{u} \in \mathbf{P'}$. The behind idea of using the first $O(k_{max})$ item vectors in $\mathbf{P}$ is that vectors with large norms tend to provide large inner products [20]. This means that we can obtain a tight lower-bound at a lightweight cost.

(3) Block building: After that, blocks are built, so that user vectors in a block keep the order and each block is disjoint. Given a new block $\mathbf{B}$, we insert user vectors $\mathbf{u_i} \in \mathbf{Q}$ into $\mathbf{Q}(\mathbf{B})$ in sequence while updating $L^j(\mathbf{B})$, until we have $|\mathbf{Q}(\mathbf{B})| = O(\log n)$. When $|\mathbf{Q}(\mathbf{B})| = O(\log n)$, we insert $\mathbf{B}$ into a set of blocks $\mathcal{B}$, and make a new block.

Example 2. *Fig. 1 illustrates an example of block building. For ease of presentation, we use* b *as a block size and* $n = 3b$. *For example,* $\mathbf{Q}(\mathbf{B_1}) = \{\mathbf{u_1}, ..., \mathbf{u_b}\}$, *and* $\|\mathbf{u_1}\| \geq ... \geq \|\mathbf{u_b}\|$.

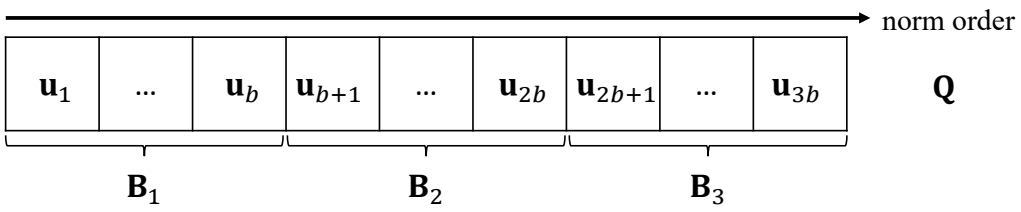


Fig. 1. Example of block building.

Generally, this pre-processing is done only once. An exception is the case where a query with $k > k_{max}$ is specified. In this case, Simpfer re-builds the data structures then processes the query. This is actually much faster than the baselines, as shown in Section 5.7.

Analysis. We here prove that the time complexity of this pre-processing is reasonable. Without loss of generality, we assume $n \geq m$, because this is a usual case for many real datasets, as the ones we use in Section 5.

Theorem 1. *Algorithm 1 requires $O(n(d + \log n))$ time.*

Proof. The norm computation requires $O((n + m)d) = O(nd)$ time, and sorting requires $O(n \log n)$ time. The building of lower-bound arrays needs $O(n \times k_{max})$ time, since $O(|\mathbf{P}'|) = O(k_{max})$. Because $k_{max} = O(1)$, $O(n \times k_{max}) = O(n)$. The block building also requires $O(n \times k_{max}) = O(n)$ time. In total, this pre-processing requires $O(n(d + \log n))$ time. □

The space complexity of Simpfer is also reasonable.

Theorem 2. *The space complexity of the index is $O(n)$.*

Proof. The space of the lower-bound arrays of user vectors is $O(\sum_n |L_i|) = O(n)$, since $O(|L_i|) = O(1)$. Blocks are disjoint, and the space of the lower-bound array of a block is also $O(1)$. We hence have $O(\frac{n}{\log n})$ lower-bound arrays of blocks. Now this theorem is clear. □

4.2 Upper- and Lower-bounding for the k-MIPS Decision Problem

Before we present the details of Simpfer, we introduce our techniques that can quickly answer the k-MIPS decision problem for a given query $\mathbf{q}$. Recall that $\mathbf{Q}$ and $\mathbf{P}$ are sorted in descending order of norm. Without loss of generality, we assume that $\|\mathbf{u_i}\| \geq \|\mathbf{u_{i+1}}\|$ for each $i \in [1, n-1]$ and $\|\mathbf{p_j}\| \geq \|\mathbf{p_{j+1}}\|$ for each $j \in [1, m-1]$, for ease of presentation.

Given a query $\mathbf{q}$ and a user vector $\mathbf{u_i} \in \mathbf{Q}$, we have $\mathbf{u_i} \cdot \mathbf{q}$. Although our data structures are simple, they provide effective and "light-weight" filters. Specifically, we can quickly answer the k-MIPS decision problem on $\mathbf{q}$ through the following observations[3].

Lemma 1. *If $\mathbf{u_i} \cdot \mathbf{q} \leq L_i^k$, it is guaranteed that $\mathbf{q}$ is not included in the k-MIPS result of $\mathbf{u_i}$.*

Proof. Let $\mathbf{p}$ be the vector in $\mathbf{P}$ such that $\mathbf{u_i} \cdot \mathbf{p}$ is the k-th highest inner product in $\mathbf{P}$. The fact that $L_i^k \leq \mathbf{u_i} \cdot \mathbf{p}$ immediately derives this lemma. □

It is important to see that the above lemma provides "no" as the answer to the k-MIPS decision problem on $\mathbf{q}$ in $O(1)$ time (after computing $\mathbf{u_i} \cdot \mathbf{q}$). The next lemma deals with the "yes" case in $O(1)$ time.

[3]Existing algorithms for top-k retrieval, e.g., [10, 11], use similar (but different) bounding techniques. They use a bound (e.g., obtained by a block) to *early stop* linear scans. On the other hand, our bounding is designed to *avoid* linear scans and to filer multiple user vectors in a batch.

Algorithm 2: LINEAR-SCAN(u)

```
Input: u ∈ Q, P, q, and k
I ← {u · q}, τ ← 0
for each p_i ∈ P do
    if u · q ≥ ‖u‖‖p_i‖ or τ ≥ ‖u‖‖p_i‖ then
        return 1 (yes)
    γ ← u · p_i
    if γ > τ then
        I ← I ∪ {γ}
        if |I| > k then
            Delete the (k + 1)-th inner product from I
            τ ← the k-th inner product in I
        if τ > u · q then
            return 0 (no)
```

LEMMA 2. *If* $\mathbf{u_i} \cdot \mathbf{q} \geq \|\mathbf{u_i}\|\|\mathbf{p_k}\|$, *it is guaranteed that* $\mathbf{q}$ *is included in the k-MIPS result of* $\mathbf{u_i}$.

PROOF. From Cauchy–Schwarz inequality, we have $\mathbf{u_i} \cdot \mathbf{p_j} \leq \|\mathbf{u_i}\|\|\mathbf{p_j}\|$. Since $\|\mathbf{p_k}\|$ is the k-th highest norm in $\mathbf{P}$, $\mathbf{u_i} \cdot \mathbf{p} \leq \|\mathbf{u_i}\|\|\mathbf{p_k}\|$, where $\mathbf{p}$ is defined in the proof of Lemma 1. That is, $\|\mathbf{u_i}\|\|\mathbf{p_k}\|$ is an upper-bound of $\mathbf{u_i} \cdot \mathbf{p}$. Now it is clear that $\mathbf{q}$ has $\mathbf{u_i} \cdot \mathbf{q} \geq \mathbf{u_i} \cdot \mathbf{p}$ if $\mathbf{u_i} \cdot \mathbf{q} \geq \|\mathbf{u_i}\|\|\mathbf{p_k}\|$. □

We next introduce a technique that yields "no" as the answer for *all user vectors* in a block $\mathbf{B}$ in $O(1)$ time.

LEMMA 3. *Given a block* $\mathbf{B}$, *let* $\mathbf{u_i}$ *be the first vector in* $\mathbf{Q(B)}$. *If* $\|\mathbf{u_i}\|\|\mathbf{q}\| \leq L^k(\mathbf{B})$, *for all* $\mathbf{u_j} \in \mathbf{Q(B)}$, *it is guaranteed that* $\mathbf{q}$ *is not included in the k-MIPS result of* $\mathbf{u_j}$.

PROOF. From Cauchy–Schwarz inequality, $\|\mathbf{u_i}\|\|\mathbf{q}\|$ is an upper-bound of $\mathbf{u_j} \cdot \mathbf{q}$ for all $\mathbf{u_j} \in \mathbf{Q(B)}$, since $\mathbf{Q(B)} = \{\mathbf{u_i}, \mathbf{u_{i+1}}, ...\}$. We have $L^k(\mathbf{B}) \leq L^k_j$ for all $\mathbf{u_j} \in \mathbf{Q(B)}$, from Equation (1). Therefore, if $\|\mathbf{u_i}\|\|\mathbf{q}\| \leq L^k(\mathbf{B})$, $\mathbf{u_j} \cdot \mathbf{q}$ cannot be the k highest inner product. □

If a user vector $\mathbf{u_i}$ cannot obtain a yes/no answer from Lemmas 1–3, Simpfer uses a linear scan of $\mathbf{P}$ to obtain the answer. Let τ be a threshold, i.e., an intermediate k-th highest inner product for $\mathbf{u}$ during the linear scan. By using the following corollaries, Simpfer can obtain the correct answer and early terminate the linear scan.

COROLLARY 1. *Assume that* $\mathbf{q}$ *is included in an intermediate result of the k-MIPS of* $\mathbf{u_i}$ *and we now evaluate* $\mathbf{p_j} \in \mathbf{P}$. *If* $\mathbf{u_i} \cdot \mathbf{q} \geq \|\mathbf{u_i}\|\|\mathbf{p_j}\|$, *it is guaranteed that* $\mathbf{q}$ *is included in the final result of the k-MIPS of* $\mathbf{u_i}$.

PROOF. Trivially, we have $j \geq k$. Besides, $\|\mathbf{u_i}\|\|\mathbf{p_j}\| \geq \mathbf{u_i} \cdot \mathbf{p_l}$ for all $k \leq l \leq m$, because $\mathbf{P}$ is sorted. This corollary is hence true. □

From this corollary and the fact that $\mathbf{u_i} \cdot \mathbf{q} \geq \tau$ if $\mathbf{q}$ is included in an intermediate result, we also have:

COROLLARY 2. *Assume that* $\mathbf{q}$ *is included in an intermediate result of the k-MIPS of* $\mathbf{u_i}$ *and we now evaluate* $\mathbf{p_j} \in \mathbf{P}$. *If* $\tau \geq \|\mathbf{u_i}\|\|\mathbf{p_j}\|$, *it is guaranteed that* $\mathbf{q}$ *is included in the final result of the k-MIPS of* $\mathbf{u_i}$.

COROLLARY 3. *When we have* $\tau > \mathbf{u_i} \cdot \mathbf{q}$, *it is guaranteed that* $\mathbf{q}$ *is not included in the final result of the k-MIPS of* $\mathbf{u_i}$.

Algorithm 2 summarizes the linear scan that incorporates Corollaries 1–3.

Algorithm 3: SIMPFER

```
   Input: Q, P, q, k, and B
1  Q_r ← ∅, Compute ||q||
2  for each B ∈ B do
3      u ← the first user vector in Q(B)
4      if ||u|||q|| > L^k(B) then
5          for each u_i ∈ Q(B) do
6              γ ← u_i · q
7              if γ > L_i^k then
8                  if ||u_i|||p_k|| > γ then
9                      f ← LINEAR-SCAN(u_i)
10                     if f = 1 then
11                         Q_r ← Q_r ∪ {u_i}
12                 else
13                     Q_r ← Q_r ∪ {u_i}
14 return Q_r
```

4.3 The Algorithm

Now we are ready to present Simpfer. Algorithm 3 details it. To start with, Simpfer computes $\|\mathbf{q}\|$. Given a block $\mathbf{B} \in \mathcal{B}$, Simpfer tests Lemma 3 (line 4). If the user vectors in $\mathbf{Q}(\mathbf{B})$ may have yes as an answer, for each $\mathbf{u_i} \in \mathbf{Q}(\mathbf{B})$, Simpfer does the following. (Otherwise, all user vectors in $\mathbf{Q}(\mathbf{B})$ are ignored.) First, it computes $\mathbf{u_i} \cdot \mathbf{q}$, then tests Lemma 1 (line 7). If $\mathbf{u_i}$ cannot have the answer from this lemma, Simpfer tests Lemma 2. Simpfer inserts $\mathbf{u_i}$ into the result set $\mathbf{Q_r}$ if $\mathbf{u_i} \cdot \mathbf{q} \geq \|\mathbf{u_i}\|\|\mathbf{p_k}\|$. Otherwise, Simpfer conducts LINEAR-SCAN($\mathbf{u_i}$) (Algorithm 2). If LINEAR-SCAN($\mathbf{u_i}$) returns 1 (yes), $\mathbf{u_i}$ is inserted into $\mathbf{Q_r}$. The above operations are repeated for each $\mathbf{B} \in \mathcal{B}$. Finally, Simpfer returns the result set $\mathbf{Q_r}$.

The correctness of Simpfer is obvious, because it conducts LINEAR-SCAN($\cdot$) for all vectors that cannot have yes/no answers from Lemmas 1–3. Besides, Simpfer accesses blocks sequentially, so it is easy to parallelize by using multicore. Simpfer hence further accelerates the processing of reverse k-MIPS, see Section 5.6.

4.4 Complexity Analysis

We theoretically demonstrate the efficiency of Simpfer. Specifically, we have:

THEOREM 3. *Let α be the pruning ratio ($0 \leq \alpha \leq 1$) of blocks in $\mathcal{B}$. Furthermore, let m' be the average number of item vectors accessed in* LINEAR-SCAN($\cdot$). *The time complexity of Simpfer is $O((1-\alpha)nm'd)$.*

PROOF. Simpfer accesses all blocks in $\mathcal{B}$, and $|\mathcal{B}| = O(\frac{n}{\log n})$. Assume that a block $\mathbf{B} \in \mathcal{B}$ is not pruned by Lemma 3. Simpfer accesses all user vectors in $\mathbf{Q}(\mathbf{B})$, so the total number of such user vectors is $(1-\alpha) \times O(\frac{n}{\log n}) \times O(\log n) = O((1-\alpha)n)$. For these vectors, Simpfer computes inner products with $\mathbf{q}$. The evaluation cost of Lemmas 1 and 2 for these user vectors is thus $O((1-\alpha)nd)$. The worst cost of LINEAR-SCAN($\cdot$) for vectors that cannot obtain the answer from these lemmas is $O((1-\alpha)nm'd)$. Now the time complexity of Simpfer is

$$O(\frac{n}{\log n} + (1-\alpha)nd + (1-\alpha)nm'd) = O(\frac{n}{\log n} + (1-\alpha)nm'd) \tag{2}$$
$$= O((1-\alpha)nm'd)$$

Consequently, this theorem holds. □

Remark. There are two main observations in Theorem 3. First, because we practically have $m' < m$ and $\alpha > 0$, Simpfer outperforms a k-MIPS-based solution that incurs $O(nmd)$ time. (Our experimental results show that $m' = O(k)$ in practice.) The second observation is obtained from Equation (2), which implies the effectiveness of blocks. If Simpfer does not build blocks, we have to evaluate Lemma 1 *for all* $\mathbf{u} \in \mathbf{Q}$. Equation (2) suggests that the blocks theoretically avoids this.

5 EXPERIMENT

This section reports our experimental results. All experiments were conducted on a Ubuntu 18.04 LTS machine with a 12-core 3.0GHz Intel Xeon E5-2687w v4 processor and 512GB RAM.

5.1 Setting

Datasets. We used four popular real datasets: MovieLens[4], Netflix, Amazon[5], and Yahoo![6]. The user and item vectors of these datasets were obtained by the Matrix Factorization in [5]. These are 50-dimensional vectors (the dimensionality setting is the same as [18, 28][7]). The other statistics is shown in Table 2. We randomly chose 1,000 vectors as query vectors from $\mathbf{P}$.

Table 2. Dataset statistics

	MovieLens	Netflix	Amazon	Yahoo!
$\lvert\mathbf{Q}\rvert$	138,493	480,189	1,948,882	2,088,620
$\lvert\mathbf{P}\rvert$	26,744	17,770	98,211	200,941

Evaluated algorithms. We evaluated the following three algorithms.

- LEMP [28]: the state-of-the-art *all-k*-MIPS algorithm. LEMP originally does k-MIPS for all user vectors in $\mathbf{Q}$.
- FEXIPRO [18]: the state-of-the-art k-MIPS algorithm. We simply ran FEXIPRO for each $\mathbf{u} \in \mathbf{Q}$.
- Simpfer: the algorithm proposed in this paper. We set $k_{max} = 25$.

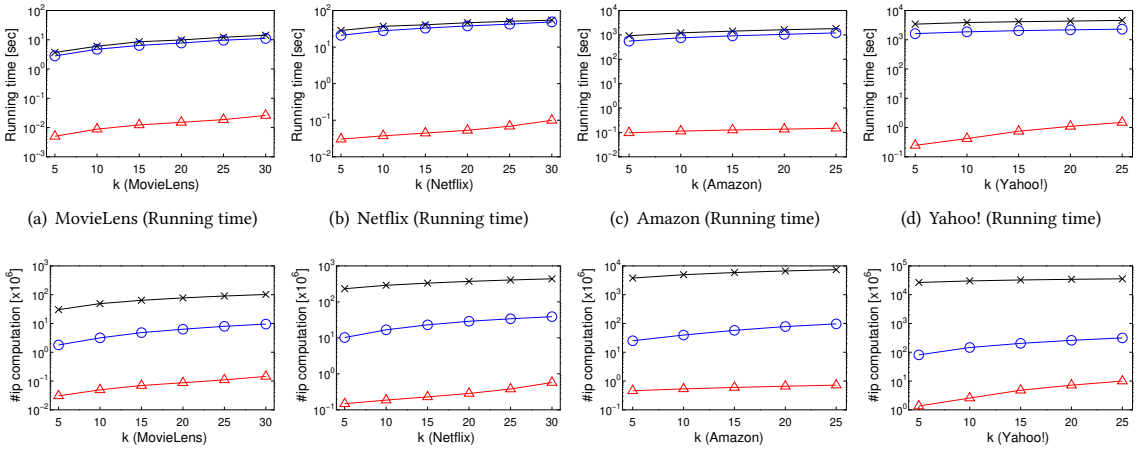
These algorithms were implemented in C++ and compiled by g++ 7.5.0 with -O3 flag. We used OpenMP for multicore processing. These algorithms return the exact result, so we measured their running time.

Note that [18, 28] have demonstrated that the other exact MIPS algorithms are outperformed by LEMP and FEXIPRO, so we did not use them as competitors. (Recall that this paper focuses on the exact answer, thus approximation algorithms are not appropriate for competitors, see Section 3.) In addition, LEMP and FEXIPRO also have a pre-processing (offline) phase. We did not include the offline time as the running time.

5.2 Result 1: Effectiveness of blocks

We first clarify the effectiveness of blocks employed in Simpfer. To show this, we compare Simpfer with Simpfer without blocks (which does not evaluate line 4 of Algorithm 3). We set $k = 10$.

[4]https://grouplens.org/datasets/movielens/
[5]https://jmcauley.ucsd.edu/data/amazon/
[6]https://webscope.sandbox.yahoo.com/
[7]As our theoretical analysis shows, the time of Simpfer is trivially proportional to d, thus its empirical impact is omitted.



(a) MovieLens (Running time) (b) Netflix (Running time) (c) Amazon (Running time) (d) Yahoo! (Running time)

(e) MovieLens (#ip computations) (f) Netflix (#ip computations) (g) Amazon (#ip computations) (h) Yahoo! (#ip computations)

Fig. 2. Impact of k: Running time (top) and #ip computations (bottom). "×" shows LEMP, "○" shows FEXIPRO, and "△" shows Simpfer.

On MovieLens, Netflix, Amazon, and Yahoo!, Simpfer (Simpfer without blocks) takes 8.8 (25.2), 37.5 (135.8), 113.3 (446.2), and 419.8 (617.8) [msec], respectively. This result demonstrates that, although the speed-up ratio is affected by data distributions, blocks surely yield speed-up.

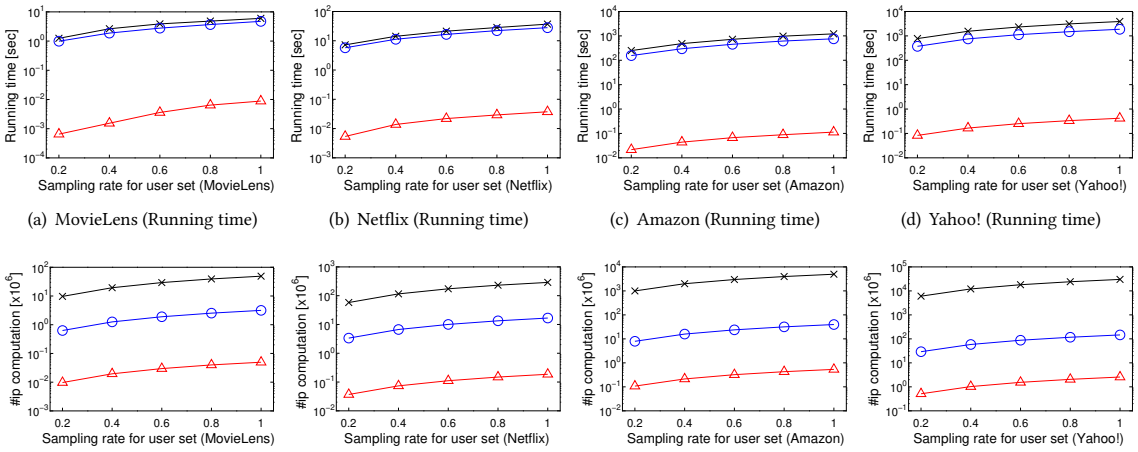
5.3 Result 2: Impact of k

We investigate how k affects the computational performance of each algorithm by using a single core. Fig. 2 depicts the experimental results.

We first observe that, as k increases, the running time of each algorithm increases, as shown in Figs. 2(a)–2(d). This is reasonable, because the cost of (decision version of) k-MIPS increases. As a proof, Figs. 2(e)–2(h) show that the number of inner product (ip) computations increases as k increases. The running time of Simpfer is (sub-)linear to k (the plots are log-scale). This suggests that $m' = O(k)$.

Second, Simpfer significantly outperforms LEMP and FEXIPRO. For example, when $k = 10$, Simpfer is 680 (536), 988 (747), 9246 (4449), and 10691 (6703) times faster than LEMP (FEXIPRO) on MovieLens, Netflix, Amazon, and Yahoo!, respectively. This result is derived from our idea of quickly solving the k-MIPS decision problem. The techniques introduced in Section 4.2 can deal with both yes and no answer cases efficiently. Therefore, our approach functions quite well in practice.

Last, an interesting observation is the performance differences between FEXIPRO and Simpfer. Let us compare them with regard to running time. Simpfer *is at least two* orders of magnitude faster than FEXIPRO. On the other hand, with regard to the number of inner product computations, that of Simpfer is *one* order of magnitude lower than that of FEXIPRO. This result suggests that the filtering cost of Simpfer is light, whereas that of FEXIPRO is heavy. Recall that Lemmas 1–3 need only $O(1)$ time, and Corollaries 1–3 need $O(k)$ time in practice. On the other hand, for each user vector in Q, FEXIPRO incurs $\Omega(k)$ time, and its filtering cost is $O(d')$, where $d' < d$. For high-dimensional vectors, the difference between $O(1)$ and $O(d')$ is large. From this point of view, we can see the efficiency of Simpfer.



(a) MovieLens (Running time) (b) Netflix (Running time) (c) Amazon (Running time) (d) Yahoo! (Running time)

(e) MovieLens (#ip computations) (f) Netflix (#ip computations) (g) Amazon (#ip computations) (h) Yahoo! (#ip computations)

Fig. 3. Impact of $|\mathbf{Q}|$: Running time (top) and #ip computations (bottom). "×" shows LEMP, "○" shows FEXIPRO, and "△" shows Simpfer.

5.4 Result 3: Impact of Cardinality of Q

We next study the scalability to $n = |\mathbf{Q}|$ by using a single core. To this end, we randomly sampled $s \times n$ user vectors in $\mathbf{Q}$, and this sampling rate s has $s \in [0.2, 1.0]$. We set $k = 10$. Fig. 3 shows the experimental result.

In a nutshell, we have a similar result to that in Fig. 2. As n increases, the running time of Simpfer linearly increases. This result is consistent with Theorem 3. Notice that the tendency of the running time of Simpfer follows that of the number of inner product computations. This phenomenon is also supported by Theorem 3, because the main bottleneck of Simpfer is Linear-scan$(\cdot)$.

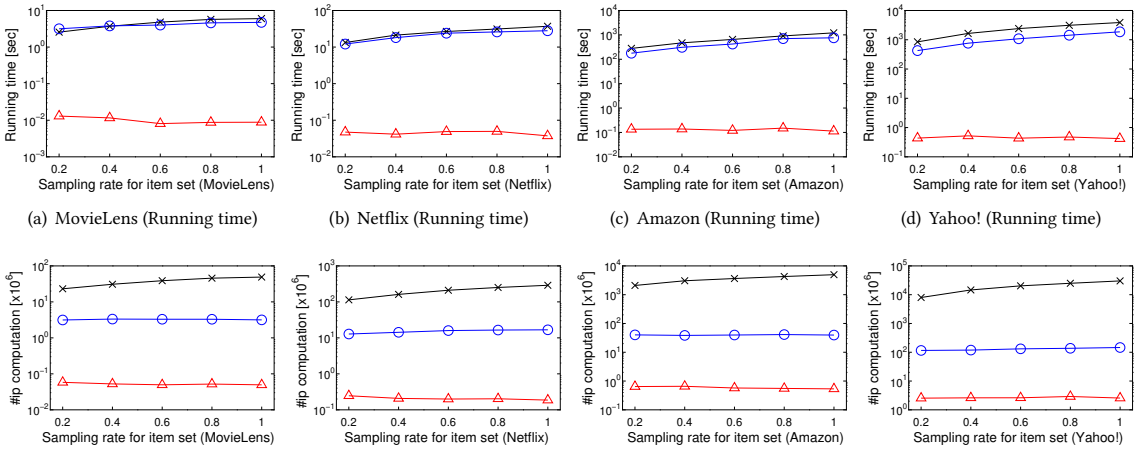
5.5 Result 4: Impact of Cardinality of P

The scalability to $m = |\mathbf{P}|$ by using a single core is also investigated. We randomly sampled $s \times m$ user vectors in $\mathbf{P}$, as with the previous section. Fig. 4 shows the experimental result where $k = 10$. Interestingly, we see that the result is different from that in Fig. 3. The running time of Simpfer is almost stable for different m. In this experiment, n and k were fixed, and recall that $m' = O(k)$. From this observation, the stable performance is theoretically obtained. This scalability of Simpfer is an advantage over the other algorithms, since their running time increases as m increases.

5.6 Result 5: Impact of Number of CPU Cores

We study the gain of multicore processing of Simpfer by setting $k = 10$. We depict the speedup ratios compared with the single-core case in Table 3.

We see that Simpfer receives the benefit of multicore processing, and its running time decreases as the number of available cores increases. We here explain why Simpfer cannot obtain speedup ratio c, where c is the number of available cores. Each core deals with different blocks, and the processing cost of a given block $\mathbf{B}$ is different from those of the others. This is because it is unknown whether $\mathbf{B}$ can be pruned by Lemma 3. Even if we magically know the cost, it is NP-hard to assign blocks so that each core has the same processing cost [2, 17]. Therefore, perfect load-balancing is



(a) MovieLens (Running time) (b) Netflix (Running time) (c) Amazon (Running time) (d) Yahoo! (Running time)

(e) MovieLens (#ip computations) (f) Netflix (#ip computations) (g) Amazon (#ip computations) (h) Yahoo! (#ip computations)

Fig. 4. Impact of $|\mathbf{P}|$: Running time (top) and #ip computations (bottom). "×" shows LEMP, "○" shows FEXIPRO, and "△" shows Simpfer.

Table 3. Speedup ratios of Simpfer

#cores	MovieLens	Netflix	Amazon	Yahoo!
4	3.70	3.44	3.20	3.03
8	6.85	6.12	5.12	4.04
12	8.75	7.79	7.83	4.30

impossible in practice. The Yahoo! case in particular represents this phenomenon. Because many user vectors in Yahoo! have large norms, blocks often cannot be filtered by Lemma 3. This can be seen from the observation in Fig. 3(h): the number of inner product computations on Yahoo! is larger than those on the other datasets. The costs of Corollaries 1–3 are data-dependent (i.e., they are not pre-known), rendering a fact that Yahoo! is a hard case for obtaining a high speedup ratio.

Table 4. Pre-processing time of Simpfer [sec]

MovieLens	Netflix	Amazon	Yahoo!
1.02	4.08	15.10	15.55

5.7 Result 6: Pre-processing Time

Last, we report the pre-processing time of Simpfer. Table 4 shows the results. As Theorem 1 demonstrates, the pre-processing time increases as n increases. We see that the pre-processing time is reasonable and much faster than the online (running) time of the baselines. For example, the running time of FEXIPRO on Amazon with $k = 25$ is 1206 [sec]. When $k = 25$ (i.e., $k = k_{max}$), the total time of pre-processing and online processing of Simpfer is 15.10 + 0.15 = 15.25 [sec]. Therefore, even if $k > k_{max}$ is specified, re-building blocks then processing the query by Simpfer is much faster.

6 CONCLUSION

This paper introduced a new problem, reverse maximum inner product search (reverse MIPS). The reverse MIPS problem supports many applications, such as recommendation, advertisement, and market analysis. Because even state-of-the-art algorithms for MIPS cannot solve the reverse MIPS problem efficiently, we proposed Simpfer as an exact and efficient solution. Simpfer exploits several techniques to efficiently answer the decision version of the MIPS problem. Our theoretical analysis has demonstrated that Simpfer is always better than a solution that employs a state-of-the-art algorithm of MIPS. Besides, our experimental results on four real datasets show that Simpfer is at least two orders of magnitude faster than the MIPS-based solutions.

ACKNOWLEDGMENTS

This research is partially supported by JST PRESTO Grant Number JPMJPR1931, JSPS Grant-in-Aid for Scientific Research (A) Grant Number 18H04095, and JST CREST Grant Number JPMJCR21F2.

REFERENCES

[1] Firas Abuzaid, Geet Sethi, Peter Bailis, and Matei Zaharia. 2019. To Index or Not to Index: Optimizing Exact Maximum Inner Product Search. In *ICDE*. 1250–1261.
[2] Daichi Amagata and Takahiro Hara. 2019. Identifying the Most Interactive Object in Spatial Databases. In *ICDE*. 1286–1297.
[3] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *RecSys*. 257–264.
[4] Chong Chen, Min Zhang, Yongfeng Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. 2020. Efficient Heterogeneous Collaborative Filtering without Negative Sampling for Recommendation. In *AAAI*. 19–26.
[5] Wei-Sheng Chin, Bo-Wen Yuan, Meng-Yuan Yang, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. 2016. LIBMF: a library for parallel matrix factorization in shared-memory systems. *The Journal of Machine Learning Research* 17, 1 (2016), 2971–2975.
[6] Edith Cohen and David D Lewis. 1999. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms* 30, 2 (1999), 211–252.
[7] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*. 39–46.
[8] Ryan R Curtin, Parikshit Ram, and Alexander G Gray. 2013. Fast exact max-kernel search. In *SDM*. 1–9.
[9] Xinyan Dai, Xiao Yan, Kelvin KW Ng, Jiu Liu, and James Cheng. 2020. Norm-Explicit Quantization: Improving Vector Quantization for Maximum Inner Product Search. In *AAAI*. 51–58.
[10] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *SIGIR*. 993–1002.
[11] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Zien. 2011. Evaluation strategies for top-k queries over memory-resident inverted indexes. *PVLDB* 4, 12 (2011), 1213–1224.
[12] Marco Fraccaro, Ulrich Paquet, and Ole Winther. 2016. Indexable probabilistic matrix factorization for maximum inner product search. In *AAAI*. 1554–1560.
[13] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*. 3887–3896.
[14] Qiang Huang, Guihong Ma, Jianlin Feng, Qiong Fang, and Anthony KH Tung. 2018. Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In *KDD*. 1561–1570.
[15] Jyun-Yu Jiang, Patrick H Chen, Cho-Jui Hsieh, and Wei Wang. 2020. Clustering and Constructing User Coresets to Accelerate Large-scale Top-K Recommender Systems. In *The Web Conference*. 2177–2187.
[16] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*. 535–544.
[17] Richard E Korf. 2009. Multi-Way Number Partitioning.. In *IJCAI*. 538–543.
[18] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *SIGMOD*. 835–850.
[19] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488.
[20] Jie Liu, Xiao Yan, Xinyan Dai, Zhirong Li, James Cheng, and Ming-Chang Yang. 2020. Understanding and Improving Proximity Graph Based Maximum Inner Product Search. In *AAAI*. 139–146.

[21] Rui Liu, Tianyi Wu, and Barzan Mozafari. 2019. A Bandit Approach to Maximum Inner Product Search. In *AAAI*. 4376–4383.
[22] Stanislav Morozov and Artem Babenko. 2018. Non-metric similarity graphs for maximum inner product search. In *NeurIPS*. 4721–4730.
[23] Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *ICML*. 1926–1934.
[24] Parikshit Ram and Alexander G Gray. 2012. Maximum inner-product search using cone trees. In *KDD*. 931–939.
[25] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited. In *RecSys*. 240–248.
[26] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS*. 2321–2329.
[27] Christina Teflioudi and Rainer Gemulla. 2016. Exact and approximate maximum inner product search with lemp. *ACM Transactions on Database Systems* 42, 1 (2016), 1–49.
[28] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. 2015. Lemp: Fast retrieval of large entries in a matrix product. In *SIGMOD*. 107–122.
[29] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *NIPS*. 2643–2651.
[30] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. 2010. Reverse top-k queries. In *ICDE*. 365–376.
[31] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Norvag. 2011. Monochromatic and bichromatic reverse top-k queries. *IEEE Transactions on Knowledge and Data Engineering* 23, 8 (2011), 1215–1229.
[32] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-ranging lsh for maximum inner product search. In *NeurIPS*. 2952–2961.
[33] Shiyu Yang, Muhammad Aamir Cheema, Xuemin Lin, and Wei Wang. 2015. Reverse k nearest neighbors query processing: experiments and analysis. *PVLDB* 8, 5 (2015), 605–616.
[34] Hsiang-Fu Yu, Cho-Jui Hsieh, Qi Lei, and Inderjit S Dhillon. 2017. A greedy approach for budgeted maximum inner product search. In *NIPS*. 5453–5462.
[35] Hamed Zamani and W Bruce Croft. 2020. Learning a Joint Search and Recommendation Model from User-Item Interactions. In *WSDM*. 717–725.
[36] Zhao Zhang, Cheqing Jin, and Qiangqiang Kang. 2014. Reverse k-ranks query. *PVLDB* 7, 10 (2014), 785–796.
[37] Xing Zhao, Ziwei Zhu, Yin Zhang, and James Caverlee. 2020. Improving the Estimation of Tail Ratings in Recommender System with Multi-Latent Representations. In *WSDM*. 762–770.
[38] Zhixin Zhou, Shulong Tan, Zhaozhuo Xu, and Ping Li. 2019. Möbius Transformation for Fast Inner Product Search on Graph. In *NeurIPS*. 8216–8227.