FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems[*]

Hui Li[†], Tsz Nam Chan[§], Man Lung Yiu[§], Nikos Mamoulis[†]

[†]The University of Hong Kong [§]Hong Kong Polytechnic University

[†]{hli2, nikos}@cs.hku.hk [§]{cstnchan, csmlyiu}@comp.polyu.edu.hk

ABSTRACT

Recommender systems have many successful applications in e-commerce and social media, including Amazon, Netflix, and Yelp. Matrix Factorization (MF) is one of the most popular recommendation approaches; the original user-product rating matrix $\mathbf{R}$ with millions of rows and columns is decomposed into a user matrix $\mathbf{Q}$ and an item matrix $\mathbf{P}$, such that the product $\mathbf{Q}^T\mathbf{P}$ approximates $\mathbf{R}$. Each column $\mathbf{q}$ ($\mathbf{p}$) of $\mathbf{Q}$ ($\mathbf{P}$) holds the latent factors of the corresponding user (item), and $\mathbf{q}^T\mathbf{p}$ is a prediction of the rating to item $\mathbf{p}$ by user $\mathbf{q}$. Recommender systems based on MF suggest to a user in $\mathbf{q}$ the items with the top-k scores in $\mathbf{q}^T\mathbf{P}$. For this problem, we propose a Fast and EXact Inner PROduct retrieval (FEXIPRO) framework, based on sequential scan, which includes three elements. First, FEXIPRO applies an SVD transformation to $\mathbf{P}$, after which the first several dimensions capture a large percentage of the inner products. This enables us to prune item vectors by only computing their partial inner products with $\mathbf{q}$. Second, we construct an integer approximation version of $\mathbf{P}$, which can be used to compute fast upper bounds for the inner products that can prune item vectors. Finally, we apply a lossless transformation to $\mathbf{P}$, such that the resulting matrix has only positive values, allowing for the inner products to be monotonically increasing with dimensionality. Experiments on real data demonstrate that our framework outperforms alternative approaches typically by an order of magnitude.

1. INTRODUCTION

Recommender systems have become de facto tools for suggesting items that are of potential interest to users. Real applications include product recommendation (Amazon), TV recommendation (Netflix) and restaurant recommendation (Yelp). A fundamental task in modern recommender systems is *top-k recommendation* [14], which suggests to a user the top-k predicted ratings on items that she has not rated yet. Top-k recommendation is typically performed in two phases: *learning* and *retrieval* [7] (see Figure 1).

Specifically, consider a set of users and a set of items: each user can rate any item. In the *learning phase*, the missing ratings of the not-yet-rated items are estimated using methods from machine learning, approximation theory, and various heuristics [2]. Example techniques include collaborative filtering [33], user-item graph models [4], regression based models [37] and matrix factorization (MF) [26]. In this paper, we focus on MF, due to its prevalence in dealing with large user-item rating matrices. Let $\mathbf{R}$ be a $m \times n$ matrix with the ratings of m users on n items. MF approximately factorizes $\mathbf{R}$ and computes the mapping of each user and item to a d-dimensional factor vector, where $d \ll \min\{m, n\}$. The output of the learning phase based on MF is a user matrix $\mathbf{Q} \in \mathbb{R}^{d\times m}$, where the i-th column is the factor vector of the i-th user, and an item matrix $\mathbf{P} \in \mathbb{R}^{d\times n}$, where the i-th column is the factor vector of the i-th item. Given a user vector $\mathbf{q}$ and an item vector $\mathbf{p}$, their inner product $\mathbf{q}^T\mathbf{p}$ is the predicted rating of the corresponding user to the corresponding item. A higher inner product implies a higher chance of the user to be satisfied by the item. The power of MF has been proved in the Netflix Prize [10]. Besides its superior performance, another strength of MF, making it widely used, is that additional information besides the existing ratings can easily be integrated into the model to further increase its accuracy. Such information includes social network data [27], locations of users and items [28] and visual appearance [20].

Given the output from the learning phase, the task of the *retrieval phase* is to generate a top-k recommendation list for any target user. This task is challenging because the number of items n is typically very large (e.g., millions). Materializing the entire ratings prediction list for all users is practically infeasible [7, 18]. Thus, reducing the online top-k retrieval cost becomes critical, especially when designing a real-world large-scale recommender system that handles tens of thousands of queries per second.

Most of the previous works focus on the learning phase of recommender systems; there are only a few studies about the retrieval phase [7, 36, 35, 32]. LEMP [36, 35] is a sequential scan algorithm designed to compute the top-k items for all users in $\mathbf{Q}$. Besides, the algorithm of [32] uses a tree structure to index $\mathbf{Q}$, in order to speed up retrieval. As pointed out in [7], such batch computational approaches assume that $\mathbf{Q}$ is static; therefore, they may not be suitable for recommender systems (e.g., FindMe [13, 3] and Microsoft Xbox [7, 31, 24]) where the target user's vector $\mathbf{q}$ is updated online by some ad-hoc contextual information (e.g., user behavior) before computing $\mathbf{q}^T\mathbf{P}$.

In this paper, we propose FEXIPRO, a Fast and EXact Inner PROduct retrieval framework, which takes as input a *single* user vector $\mathbf{q}$ and finds the top-k item vectors $\mathbf{p}$ with the largest $\mathbf{q}^T\mathbf{p}$. FEXIPRO is orthogonal to how the learning phase (MF) is implemented and does not assume a static $\mathbf{Q}$, therefore, it does not affect recommendation quality and can be applied even for dynamically adjusted user vectors. Our framework is based on sequential scan. Besides applying two popular heuristics (early termination

[*]Work supported by grant 17205015 from Hong Kong RGC.

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA

© 2017 ACM. ISBN 978-1-4503-4197-4/17/05...$15.00

DOI: http://dx.doi.org/10.1145/3035918.3064009

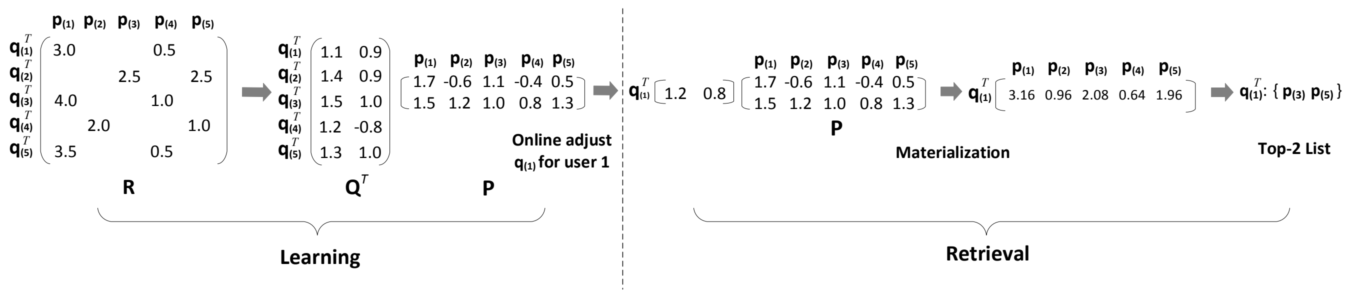

Figure 1: Overview of Top-k IP Retrieval in Recommender Systems Based on Matrix Factorization (MF)

using Cauchy-Schwarz inequality, incremental pruning) [29, 36, 35], FEXIPRO uses three techniques to accelerate search:

SVD Transformation. We apply a lossless SVD transformation to $\mathbf{P}$ in a preprocessing stage, which results in a new matrix $\bar{\mathbf{P}}$. Given a query vector $\mathbf{q}$, we convert $\mathbf{q}$ to a $\bar{\mathbf{q}}$, such that $\mathbf{q}^T\mathbf{P}$=$\bar{\mathbf{q}}^T\bar{\mathbf{P}}$. The transformation introduces skew to the scalars of $\bar{\mathbf{q}}$, such that the absolute value at each dimension is larger than the next one with high probability. For a given $\bar{\mathbf{q}}^T\bar{\mathbf{p}}$, the skew results in a large percentage of the inner product after processing only few dimensions. This leads to a tight upper bound that can facilitate pruning $\bar{\mathbf{p}}$ without having to compute the entire product.

Integer Approximation. Arithmetic operations on integers are much faster than those on floating-point numbers. FEXIPRO is the first framework to exploit this opportunity. We generate an approximation of $\mathbf{P}$ that keeps the integral parts of the original values after scaling them up (in order to minimize accuracy loss). The fast-to-compute inner product between the integer approximations of $\mathbf{q}$ and $\mathbf{p}$ can be converted to a tight upper bound of $\mathbf{q}^T\mathbf{p}$ that can be used to prune $\mathbf{p}$ without computing the exact product.

Reduction for Monotonicity. We propose a reduction technique that transforms all values of $\mathbf{P}$ (and $\mathbf{q}$) to positive ones, hence rendering inner products to monotonically increase as more dimensions are processed. Monotonicity allows the application of tighter pruning bounds derived from partially computed products and further accelerates FEXIPRO.

After integrating all the above techniques into FEXIPRO, our framework computes the top-k recommendations for a given user vector $\mathbf{q}$ orders of magnitude faster compared to alternative techniques, as shown by extensive experimentation on real datasets.

2. PRELIMINARIES

In this section, we first define the top-k inner product retrieval problem in recommender systems and explain why it is a challenging one. Then, we review the sequential scan based approach also adopted by [36]. The notation we are using throughout the paper is summarized in Table 1.

2.1 Problem Definition

The problem of *exact top-k Inner Product (IP) retrieval* in recommender systems is to generate the top-k items list for a given (target) user. The problem can be formally defined as follows:

PROBLEM 1 (IP). *Given a d-dimensional vector* $\mathbf{q}$*, find the k d-dimensional vectors* $\mathbf{p} \in \mathbf{P}$*, for which the* $\mathbf{q}^T\mathbf{p}$ *values are the largest in* $\mathbf{q}^T\mathbf{P}$*. Ties are broken arbitrarily.*

The IP problem is hard to solve in general, because the scoring function $\mathbf{q}^T\mathbf{p}$ differs from common similarity measures used for ranking in that it does not obey the triangular inequality; therefore pruning bounds that use it cannot be leveraged. Besides, due to the dimensionality curse [19], index-based similarity search approaches can easily become worse than naive sequential scan methods. Although matrix factorization reduces the original very high dimensional space (millions of dimensions, i.e., the number of items n) to a space of d dimensions (d is typically tens to hundreds in order for MF to be effective) in the learning phase, d is still too large for index-based similarity search. What is worse, large inner products do not necessarily correspond to nearby vectors by any metric in the vector space. Thus metric space techniques cannot directly be used [5]. In summary, it is difficult to solve the IP retrieval problem efficiently without special assumptions about the data. The factor vectors in recommender systems have some additional properties that introduce challenges:

- Vectors in $\mathbf{Q}$ and $\mathbf{P}$ are dense with few zeros in each dimension. Thus, similarity search approaches for sparse vectors based on inverted indexes [9, 6] are not effective.
- The lengths of the vectors can be different, i.e., the vectors are not normalized.
- Vectors may include negative values; hence partially computed inner products may not be effective in deriving tight pruning bounds [17].

Table 1: Notation

Symbols	Description
$\mathbf{q}$, $\mathbf{p}$, $\mathbf{Q}$, $\mathbf{P}$	original vectors and matrices
$\bar{\mathbf{q}}$, $\bar{\mathbf{p}}$	vectors after applying SVD transformation (Section 3)
$\hat{\mathbf{q}}$, $\hat{\mathbf{p}}$	scaled vectors for integer upper bound (Section 4)
$\hat{\bar{\mathbf{q}}}$, $\hat{\bar{\mathbf{p}}}$	vectors after applying reduction (Section 5)
$\mathbf{p}_{(\mathbf{i})}$	the i-th vector in $\mathbf{P}$
p_s	the s-th scalar of vector $\mathbf{p} = (p_1, p_2, ..., p_d)^T \in \mathbb{R}^d$
$\|\|\mathbf{p}\|\|$	the *length* (magnitude/norm) of $\mathbf{p}$: $\sqrt{\sum_{s=1}^{d}(p_s)^2}$
m, n	number of users/items
d	number of dimensions
t	threshold: the k-th inner product found so far
w	checking dimension in incremental pruning
ρ	parameter used for selecting w (Section 3)
e	scaling parameter for integer upper bound (Section 4)

2.2 Sequential Scan

Sequential Scan (SS) was adopted and optimized in previous work on IP retrieval in recommender systems [36, 35] and search engines (i.e., find relevant documents to a keyword query) [29]. Although there are different details in each method, the core of SS is common, as illustrated by Figure 2 and Algorithm 1.

Algorithm 1 uses the Cauchy-Schwarz inequality (i.e., $\mathbf{q}^T\mathbf{p} \leq ||\mathbf{q}||\,||\mathbf{p}||$) to derive a fast-to-compute upper bound for the inner products. SS sorts the columns of $\mathbf{P}$ before the scan (Line 4), in decreasing order of their lengths (i.e., their $||\cdot||$ norms). Sorting is independent of the query vector $\mathbf{q}$, therefore it is performed only once in a preprocessing phase (then, SS can be applied multiple times for different query vectors). Thus, SS scans the vectors of $\mathbf{P}$ in decreasing order of $||\mathbf{p}||$. When SS reaches a vector $\mathbf{p}$, it first checks whether $||\mathbf{q}||\,||\mathbf{p}||$ is less than or equal to the k-th largest inner product computed so far (i.e., t). If $||\mathbf{q}||\,||\mathbf{p}|| \le t$, scan stops and the current top-k items are reported as the results, since it is impossible for $\mathbf{p}$ or any item vector that follows in the scan order to enter the results. Cauchy-Schwarz inequality is also used in cosine similarity search problems [6], since the lengths of the vectors can be precomputed at a preprocessing phase and $||\mathbf{q}||$ can be computed very fast (and only once) when a query $\mathbf{q}$ is given.

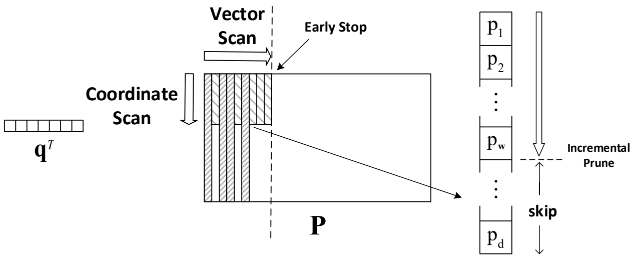

Figure 2: Sequential Scan

Algorithm 1 Basic Sequential Scan for Inner Product Retrieval

```
1: procedure RETRIEVAL($\mathbf{q}, \mathbf{P}, k$)
2:     Priority Queue $r \leftarrow \varnothing$                ▷ maintain top-$k$ results
3:     $t \leftarrow -\infty$                                   ▷ smallest IP in $r$
4:     Sort $\mathbf{P}$ by length in a descending order
5:     for each $\mathbf{p} \in \mathbf{P}$ do                  ▷ vector scan
6:         if $||\mathbf{q}||\,||\mathbf{p}|| \le t$ then       ▷ termination condition
7:             return $r$
8:         else
9:             $v \leftarrow$ CoordinateScan($\mathbf{q}, \mathbf{p}, t$)   ▷ incremental prune
10:            if $v > t$ then
11:                Push $\mathbf{p}$ into $r$ and update $t$
12:    return $r$
```

LEMP [36, 35] is the state-of-the-art exact IP retrieval method, which adapts SS. LEMP groups the vectors of $\mathbf{P}$ into buckets. To form the buckets, first $\mathbf{P}$ is sorted so that the vectors are in decreasing order of their lengths, and then consecutive vectors are packed into disjoint buckets, so that each bucket can fit into the L2 cache. The original task of LEMP is to find all pairs $(\mathbf{q}, \mathbf{p})$, such that $\mathbf{q} \in \mathbf{Q}$, $\mathbf{p} \in \mathbf{P}$, and $\mathbf{q}^T\mathbf{p} \ge t$ (i.e., the above-t problem). For the top-k IP retrieval problem, LEMP uses for each $\mathbf{q}$ the current k-th largest entry in the top-k list of $\mathbf{q}$ as t. LEMP retrieves the result list by finding all $\mathbf{p}'$ with a cosine similarity to $\mathbf{q}'$ at least $\frac{t}{||\mathbf{q}||\,||\mathbf{p}||}$, where $\mathbf{q}'$ and $\mathbf{p}'$ are the normalized versions of vectors $\mathbf{q}$ and $\mathbf{p}$, respectively, since:

$$\mathbf{q}^T\mathbf{p} \ge t \Leftrightarrow cos(\mathbf{q}, \mathbf{p}) \ge \frac{t}{||\mathbf{q}||\,||\mathbf{p}||} \Leftrightarrow \mathbf{q}'^T\mathbf{p}' \ge \frac{t}{||\mathbf{q}||\,||\mathbf{p}||}.$$

From the above equation, we can see that what LEMP uses is actually $||\mathbf{q}||\,||\mathbf{p}|| \ge \frac{t}{\mathbf{q}'^T\mathbf{p}'}$, which is similar to the pruning by SS using Cauchy-Schwarz inequality: $\mathbf{q}^T\mathbf{p} \le ||\mathbf{q}||\,||\mathbf{p}||$. After bucketizing $\mathbf{P}$, LEMP applies SS by accessing the buckets of $\mathbf{P}$ one-by-one.[1] After accessing a vector $\mathbf{p}$, scan can terminate if $||\mathbf{q}||\,||\mathbf{p}|| < t$; i.e., for a single query $\mathbf{q}$, LEMP can use the same termination condition as SS.

2.2.1 Incremental Pruning

When scan reaches vector $\mathbf{p}$ and $||\mathbf{q}||\,||\mathbf{p}|| > t$, the exact inner product of $\mathbf{q}^T\mathbf{p}$ should be calculated (denoted as *coordinate scan* in Figure 2). LEMP uses an incremental pruning technique, which can potentially reduce the number of coordinates that need to be scanned for each $\mathbf{p}$. Let w be a selected *checking* dimension, where $1 \le w < d$. Let $\mathbf{q}^{\ell^T} = (q_1, ..., q_w)$ be the *partial vector* comprising the first w coordinates of $\mathbf{q}$ and $\mathbf{q}^{h^T} = (q_{w+1}, ..., q_d)$ be the *residue vector* comprising the remaining coordinates. The partial and residue vectors $\mathbf{p}^\ell$ and $\mathbf{p}^h$ for $\mathbf{p}$ are similarly defined. Then:

$$\mathbf{q}^T\mathbf{p} \le \mathbf{q}^{\ell^T}\mathbf{p}^\ell + ||\mathbf{q}^h||\,||\mathbf{p}^h||, \tag{1}$$

due to $\mathbf{q}^T\mathbf{p} = \mathbf{q}^{\ell^T}\mathbf{p}^\ell + \mathbf{q}^{h^T}\mathbf{p}^h$ and $\mathbf{q}^{h^T}\mathbf{p}^h \le ||\mathbf{q}^h||\,||\mathbf{p}^h||$. In addition, since $\mathbf{q}^{\ell^T}\mathbf{p}^\ell + ||\mathbf{q}^h||\,||\mathbf{p}^h|| \le ||\mathbf{q}||\,||\mathbf{p}||$, the right side of Equation 1 is a tighter bound compared to the Cauchy-Schwarz inequality. Therefore, after computing $\mathbf{q}^{\ell^T}\mathbf{p}^\ell$ using the first w dimensions only, we can *check* whether $\mathbf{q}^{\ell^T}\mathbf{p}^\ell + ||\mathbf{q}^h||\,||\mathbf{p}^h|| \le t$ and terminate the IP computation if the condition is true.

Algorithm 2 shows how this *incremental pruning* is integrated into the procedure that scans $\mathbf{q}$ and $\mathbf{p}$ to compute $\mathbf{q}^T\mathbf{p}$. After scanning the first w coordinates and computing the bound, coordinate scan terminates if $\mathbf{p}$ cannot make it to the top-k results. In order for this approach to be effective, the $||\mathbf{p}^h||$ norms for all vectors should be precomputed and stored together with them (i.e., as we do for the $||\mathbf{p}||$ norms). Thus, w should be determined and fixed before query time.[2]

Algorithm 2 Incremental Pruning

```
1: procedure COORDINATESCAN($\mathbf{q}, \mathbf{p}, t$)
2:     $v \leftarrow 0$                                          ▷ IP accumulator
3:     for $s$ in $1 \ldots w$ do
4:         $v \leftarrow v + q_s \cdot p_s$
5:     if $v + ||\mathbf{q}^h||\,||\mathbf{p}^h|| \le t$ then   ▷ pruning test
6:         return $-\infty$
7:     else
8:         for $s$ in $(w+1) \ldots d$ do
9:             $v \leftarrow v + q_s \cdot p_s$
10:        return $v$
```

In order for the incremental pruning bound (Equation 1) to be effective, we should set an appropriate value for w. Larger values of w result in more effective pruning power as $\mathbf{q}^{\ell^T}\mathbf{p}^\ell$ is closer to $\mathbf{q}^T\mathbf{p}$, but they also increase the cost of accumulating the partial IP for each item vector. In addition, the effectiveness of pruning would change if the dimensions are considered in a different order. Intuitively, putting the dimensions with the largest absolute values first would maximize the exact part ($\mathbf{q}^{\ell^T}\mathbf{p}^\ell$) and minimize the uncertain part ($||\mathbf{q}^h||\,||\mathbf{p}^h||$) of the bound, making it more tight. Still, the optimal order differs for different $\mathbf{q}$ and $\mathbf{p}$. To determine a good value for w and an ordering of dimensions for each bucket of $\mathbf{P}$, LEMP [36, 35] applies a small number of sample queries, in a preprocessing phase.

Our proposal adopts the basic framework of SS as shown in Figure 2 and Algorithms 1 and 2. In the following sections, we pro-

[1] Bucketization can help to reduce CPU cache misses, in the case where multiple queries are to be executed together (i.e., the original task of LEMP). For our top-k IP retrieval problem taking a single query $\mathbf{q}$ as input, bucketization is not necessary and LEMP reduces to SS, with the differences being the normalization of vectors and cosine similarity computation.

[2] The overhead of attempting incremental pruning multiple times, i.e., at different values of w, outweighs the pruning effectiveness. In addition, there is additional space overhead for the partial norms.

pose three techniques that progressively improve the efficiency of SS, resulting in our FEXIPRO framework.

3. SVD TRANSFORMATION

A natural question is whether we can come up with a *global reordering* of the dimensions, which would maximize the power of incremental pruning (Equation 1) for every query. In this case, we could reorder $\mathbf{P}$ at a preprocessing stage and perform coordinate scans sequentially, after simply reordering $\mathbf{q}$ only once. An effective global reordering would put the dimensions with the largest absolute values first for every $\mathbf{p} \in \mathbf{P}$. However, this approach seems hard to apply on the original $\mathbf{P}$, since the values in the same dimension may vary a lot for different $\mathbf{q}$ and $\mathbf{p}$ and it is hard to find a global order which can fit every vector well.

To solve this problem, we propose to apply a transformation on matrix $\mathbf{P}$ using *singular value decomposition* (SVD). The goal of the transformation is to change the value distributions in the vectors such that the first dimensions would have higher absolute values compared to the ones that follow. The transformation only needs to be performed once, before the application of any query. Each time a query $\mathbf{q}$ comes, it is transformed accordingly at a very low cost. Although the matrix $\mathbf{P}$ and vector $\mathbf{q}$ have been changed, the top-k inner product result for a given query is the same as the result for the original $\mathbf{P}$ and $\mathbf{q}$. The use of SVD transformation to facilitate top-k IP retrieval was first proposed in [7], however, not for the purpose of improving the effectiveness of incremental pruning. Instead, the few most important components of the transformed vectors are used to create a PCATree that supports approximate top-k IP retrieval accurately and efficiently.

In details, our FEXIPRO framework applies SVD on the items matrix $\mathbf{P} \in \mathbb{R}^{d \times n}$ to obtain three matrices $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$, such that $\mathbf{P} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. $\mathbf{U}$ is a $d \times d$ unitary matrix, $\mathbf{\Sigma}$ is a $d \times n$ matrix with non-negative real numbers on the diagonal and $\mathbf{V}$ is a $n \times n$ unitary matrix. $\mathbf{\Sigma}$ and $\mathbf{V}$ can be further divided into two parts, respectively:

$$\mathbf{\Sigma} = [\mathbf{\Sigma_d}|\mathbf{O}],\ \mathbf{V} = [\mathbf{V_1}|\mathbf{V_2}], \tag{2}$$

where $\mathbf{\Sigma_d}$ is the $d \times d$ diagonal matrix with singular values $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_d$, $\mathbf{O}$ is the $d \times (n-d)$ zero matrix, and $\mathbf{V_1}$ and $\mathbf{V_2}$ are the $n \times d$ and $n \times (n-d)$ matrices respectively. FEXIPRO transfers $\mathbf{q}$ and $\mathbf{P}$ from the original space to a new space which can give us a much tighter bound for incremental pruning:

THEOREM 1. *The inner product of* $\mathbf{q}$ *and* $\mathbf{P}$ *in the original space is completely preserved in a new space obtained by SVD with the same dimensionality. That is,* $\mathbf{q}^T\mathbf{P} = \bar{\mathbf{q}}^T\bar{\mathbf{P}}$*, where* $\bar{\mathbf{q}} = \mathbf{\Sigma_d}\mathbf{U}^T\mathbf{q}$ *and* $\bar{\mathbf{P}} = \mathbf{V_1}^T$.

PROOF. Given the definition of SVD, Equation 2 and a vector $\mathbf{q}$, we have: $\mathbf{q}^T\mathbf{P} = \mathbf{q}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{q}^T\mathbf{U}[\mathbf{\Sigma_d}|\mathbf{O}][\mathbf{V_1}|\mathbf{V_2}]^T$ $= \mathbf{q}^T\mathbf{U}\mathbf{\Sigma_d}\mathbf{V_1}^T = [\mathbf{\Sigma_d}\mathbf{U}^T\mathbf{q}]^T\mathbf{V_1}^T = \bar{\mathbf{q}}^T\bar{\mathbf{P}}$. □

Therefore, for $\bar{\mathbf{P}} = \mathbf{V_1}^T$ and $\bar{\mathbf{q}} = \mathbf{\Sigma_d}\mathbf{U}^T\mathbf{q}$, FEXIPRO converts the original goal, which is to find the top-k maximum values in $\mathbf{q}^T\mathbf{P}$, to the task of finding the top-k maximum values in $\bar{\mathbf{q}}^T\bar{\mathbf{P}}$. The fact that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$ increases the chance that $|\bar{q}_i| > |\bar{q}_j|$ for $i < j$. Therefore, we can expect that in the computation of $\bar{\mathbf{q}}^T\bar{\mathbf{p}}$ the first accumulated products will be larger than the remaining ones; in other words, the partial inner product $\bar{\mathbf{q}}^{\ell^T}\bar{\mathbf{p}}^\ell$ is expected to be a large percentage of the overall inner product $\bar{\mathbf{q}}^T\bar{\mathbf{p}}$.[3] This makes incremental pruning much more effective, compared to the case where the values in the vectors are more uniform. More importantly, the order of dimensions is fixed, i.e., the same for any query. Thus, the transformation of $\mathbf{P}$ to $\bar{\mathbf{P}}$ only needs to be done once at preprocessing and can be used for any query. For a query vector $\mathbf{q}$, the time complexity of transformation is $O(d^2)$. Since d is relatively small, the actual cost is not high; in addition, the conversion is done only once for the given query before the sequential scan. The SVD transformation does not physically 'reorder' the dimensions, but it achieves the same goal (i.e, a global scanning order for any query by decreasing absolute values).

[3]We show the effectiveness of our SVD transformation in Appendix B - *Effectiveness of SVD*.

When applying the SVD transformation to $\mathbf{P}$, we should consider that standard SVD would be slow when $\mathbf{P}$ is large. Although the decomposition of $d \times n$ matrix $\mathbf{P}$ is much faster than factorizing the initial $m \times n$ ratings matrix $\mathbf{R}$ in the learning phase, the overhead is still high as the complexity of performing SVD on $\mathbf{P}$ is $O(dn^2)$. Fortunately, the transformation has a nice property which can reduce the complexity significantly. Observe from the Proof of Theorem 1 that only the $n \times d$ matrix $\mathbf{V_1}$ is used in the final transformation, while the $n \times (n-d)$ matrix $\mathbf{V_2}$ which occupies the most of $\mathbf{V}$ is abandoned. Similarly, only $\mathbf{\Sigma_d}$ is needed and the majority of matrix $\mathbf{\Sigma}$ is not used. Thus, we can perform a *thin* SVD (i.e., $\mathbf{P} = \mathbf{U}\mathbf{\Sigma_d}\mathbf{V_1}^T$) instead of the complete SVD. Thin SVD is commonly used in practice and it is significantly faster than SVD if $n \gg d$; the complexity is reduced from $O(dn^2)$ to $O(d^2n)$ [21, 11]. As demonstrated in our experiments, thin SVD is fast even when handling a very large $\mathbf{P}$.

We now turn to the question of how to set the parameter w, which is used to determine the portion $\mathbf{q}^{\ell^T}\mathbf{p}^\ell$ of the IP that will be computed before attempting incremental pruning (see Section 2.2.1). To determine a good value for w, LEMP [36, 35] applies a small number of sample queries, testing different w values, in the preprocessing phase. Besides the extra cost in sampling and testing queries, this approach is problematic in the case where the queries that are eventually issued for recommendation are dynamically adjusted. As discussed in Section 1, this is common in recommender systems like FindMe [13, 3] and Microsoft Xbox [7, 31, 24]. FEXIPRO uses a different strategy to determine w: we set w to the smallest value such that the fraction $\frac{\sum_{s=1}^{w}\sigma_s}{\sum_{s=1}^{d}\sigma_s}$ accumulates to a certain ratio ρ of the total sum (we observed the best performance is obtained when $\rho = 0.7$). The rationale behind this approach is that FEXIPRO incorporates σ into $\mathbf{q}$, so that the values of $\bar{\mathbf{q}}$ have a similar distribution to σ (i.e., the first σ values are a large percentage of the total sum). Intuitively, the partial inner product becomes large and effective for pruning when ρ is large enough. The computed w by this strategy is the same for all queries, since Σ_d is determined only by $\mathbf{P}$.

4. INTEGER-BASED PRUNING

Floating-point operations invoke more CPU cycles and are usually much slower than integer computations. If the scalars in the vectors were integers instead of floating numbers, the cost of top-k retrieval would be significantly lower. Besides, using integers can help us reduce the CPU cache miss rate; floating numbers occupy much more space compared to integers, so it is more likely that the data to be accessed are already in cache when using integers. However, simply truncating floating numbers to integers results in loss of accuracy and potentially incorrect results. In this section, we show how FEXIPRO uses integer approximations without losing any information during top-k IP retrieval.

4.1 Integer Upper Bound

By using only the integer part of the scalars in two vectors $\mathbf{q}$ and

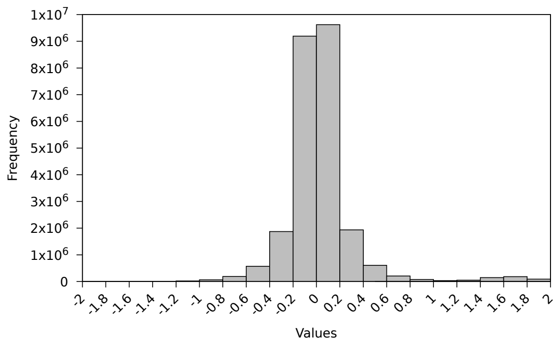

Figure 3: Value Distribution in Netflix Dataset

$\mathbf{p}$, we can define an upper bound for $\mathbf{q}^T\mathbf{p}$ as follows:

THEOREM 2. *The inner product of* $\mathbf{q}$ *and* $\mathbf{p}$ *has an integer upper bound* $IU(\mathbf{q},\mathbf{p}) = \sum_{s=1}^{d}\left(\lfloor q_s\rfloor\cdot\lfloor p_s\rfloor + \big|\lfloor q_s\rfloor\big| + \big|\lfloor p_s\rfloor\big| + 1\right)$, *where* $\lfloor q_s\rfloor$ *is the integer part of* q_s *(i.e., the largest integer less than or equal to* q_s*) and* $\big|\lfloor q_s\rfloor\big|$ *is the absolute value of* $\lfloor q_s\rfloor$.

PROOF. Let $\lfloor q_s\rfloor$ and $\Delta q_s = q_s - \lfloor q_s\rfloor$ be the integer and fractional parts of a scalar q_s, respectively. Due to $\Delta q_s \leq 1$, we have:

$$\begin{aligned}\mathbf{q}^T\mathbf{p} &= \sum_{s=1}^{d} q_s p_s = \sum_{s=1}^{d}(\lfloor q_s\rfloor + \Delta q_s)(\lfloor p_s\rfloor + \Delta p_s)\\ &= \sum_{s=1}^{d}\Big(\lfloor q_s\rfloor\cdot\lfloor p_s\rfloor + \Delta q_s\cdot\lfloor p_s\rfloor + \Delta p_s\cdot\lfloor q_s\rfloor + \Delta p_s\cdot\Delta q_s\Big) \quad (3)\\ &\leq \sum_{s=1}^{d}\Big(\lfloor q_s\rfloor\cdot\lfloor p_s\rfloor + \big|\lfloor q_s\rfloor\big| + \big|\lfloor p_s\rfloor\big| + 1\Big). \quad \Box\end{aligned}$$

Recall that, during SS, if $||\mathbf{q}||\,||\mathbf{p}|| > t$ for the next vector $\mathbf{p}$, then FEXIPRO computes the inner product $\mathbf{q}^T\mathbf{p}$, attempting incremental pruning after w dimensions. Before computing $\mathbf{q}^T\mathbf{p}$, we can calculate the integer upper bound $IU(\mathbf{q},\mathbf{p})$, which may possibly help us to prune $\mathbf{p}$. Because all calculations in $IU(\mathbf{q},\mathbf{p})$ are applied on integers, computing it is much faster compared to the exact IP $\mathbf{q}^T\mathbf{p}$. If $IU(\mathbf{q},\mathbf{p}) \leq t$, there is no chance that $\mathbf{q}^T\mathbf{p}$ can be in the top-k IPs, so we can avoid the more expensive exact $\mathbf{q}^T\mathbf{p}$ computation. Thus, together with each vector $\mathbf{p} \in \mathbf{P}$, we can store an integer vector approximation and apply the integer bound check before computing the exact $\mathbf{q}^T\mathbf{p}$.

However, the integer upper bound will be very loose, if the scalars in each dimension are within a small range, which is common in recommender systems. We illustrate this by extracting statistics from the Netflix dataset[4], which comes from a well-known movie recommendation system. After applying the matrix factorization method of [41] with $d = 50$, there are in total 24,897,950 values in $\mathbf{Q}$ and $\mathbf{P}$. Figure 3 shows the distribution of all these values. Observe that most of the values in $\mathbf{q}$ and $\mathbf{p}$ are in the range $[-1, 1]$. This can be explained by the principle of low-rank matrix factorization, which tries to minimize the root-mean-square error ($RMSE = \sqrt{\frac{1}{mn}\sum_{r_{ij}\in\mathbf{R}}(r_{ij} - \mathbf{q}_{(\mathbf{i})}{}^T\mathbf{p}_{(\mathbf{j})})^2}$) between real ($r_{ij}$) and predicted ratings ($\mathbf{q}_{(\mathbf{i})}{}^T\mathbf{p}_{(\mathbf{j})}$) during the learning phase. Since the maximum rating (5 in this example) is typically much smaller than the dimensionality d, the values of q_s and p_s tend to be in a narrow range around 0 after the iterative optimization that minimizes $RMSE$. Another reason is the regularization term, which is

[4]The statistics are similar for the other three datasets that we use. In order to save space, we show these in Appendix B - *Value Distributions*.

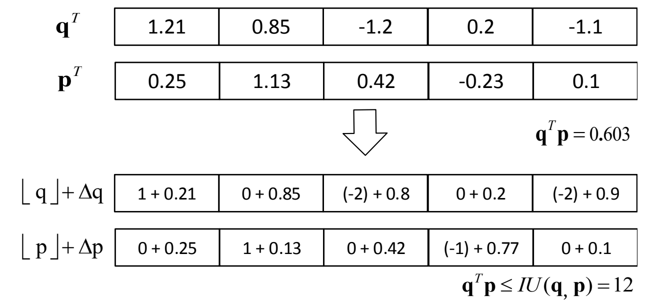
$\mathbf{q}^T$ 1.21 0.85 -1.2 0.2 -1.1
$\mathbf{p}^T$ 0.25 1.13 0.42 -0.23 0.1
$\mathbf{q}^T\mathbf{p} = 0.603$
$\lfloor q\rfloor + \Delta q$ 1 + 0.21 0 + 0.85 (-2) + 0.8 0 + 0.2 (-2) + 0.9
$\lfloor p\rfloor + \Delta p$ 0 + 0.25 1 + 0.13 0 + 0.42 (-1) + 0.77 0 + 0.1
$\mathbf{q}^T\mathbf{p} \leq IU(\mathbf{q},\mathbf{p}) = 12$

Figure 4: An Example of Integer Upper Bound

commonly used for low-rank matrix factorization, to penalize large parameters (i.e., q_s and p_s) and avoid over-fitting during training.

Figure 4 shows via an example how loose the integer upper bound can be on vectors containing scalars with values from such a narrow range. $\mathbf{q}$ and $\mathbf{p}$ are two randomly chosen vector instances in Netflix. For simplicity, we only show the values of first 5 dimensions. The inner product is 0.603, while the integer upper bound is $IU(\mathbf{q},\mathbf{p}) = 12$ ($\gg$ 0.603). The integer upper bound remains very loose when all dimensions are considered.

4.2 Scaled Integer Upper Bound

In order to make the upper bound tighter, we scale the original scalars (floating numbers) to numbers within a range $[-e, e]$. The scaling is done by first normalizing each scalar p_s (resp. q_s) by dividing it by the largest absolute value $max_{\mathbf{P}}$ (resp. $max_{\mathbf{q}}$) of the scalars in $\mathbf{P}$ (resp. $\mathbf{q}$) and then multiplying it by e, as shown in Equation 4:

$$\hat{\mathbf{p}}^T = \Big(\frac{e\cdot p_1}{max_{\mathbf{P}}}, ..., \frac{e\cdot p_d}{max_{\mathbf{P}}}\Big), \hat{\mathbf{q}}^T = \Big(\frac{e\cdot q_1}{max_{\mathbf{q}}}, ..., \frac{e\cdot q_d}{max_{\mathbf{q}}}\Big) \quad (4)$$

Recall that $\mathbf{P}$ is known in advance, thus we can get the maximum absolute value in $\mathbf{P}$ (denoted by $max_{\mathbf{P}}$) during the preprocessing phase. For $\mathbf{q}$, the maximum absolute value $max_{\mathbf{q}}$ is obtained online very fast. The transformation of vectors by Equation 4 preserves the order of top-k IP results for any query $\mathbf{q}$, since:

$$\hat{\mathbf{q}}^T\hat{\mathbf{p}} = \sum_{s=1}^{d}\frac{e^2\cdot q_s\cdot p_s}{max_{\mathbf{q}}\cdot max_{\mathbf{P}}} = \frac{e^2}{max_{\mathbf{q}}\cdot max_{\mathbf{P}}}\mathbf{q}^T\mathbf{p} \quad (5)$$

As the example of Figure 5 illustrates, for the same vectors as in Figure 4 and $e = 100$, scaling can result in a much tighter integer upper bound. The ratio of new upper bound ($IU(\hat{\mathbf{q}},\hat{\mathbf{p}}) = 5,726$) to the exact IP value (5,206.28) on the scaled vectors is only 1.1, while the ratio before scaling is 19.9 (=12/0.603).

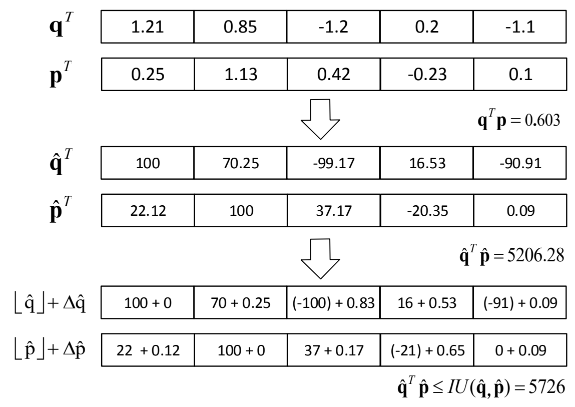
$\mathbf{q}^T$ 1.21 0.85 -1.2 0.2 -1.1
$\mathbf{p}^T$ 0.25 1.13 0.42 -0.23 0.1
$\mathbf{q}^T\mathbf{p} = 0.603$
$\hat{\mathbf{q}}^T$ 100 70.25 -99.17 16.53 -90.91
$\hat{\mathbf{p}}^T$ 22.12 100 37.17 -20.35 0.09
$\hat{\mathbf{q}}^T\hat{\mathbf{p}} = 5206.28$
$\lfloor \hat{q}\rfloor + \Delta\hat{q}$ 100 + 0 70 + 0.25 (-100) + 0.83 16 + 0.53 (-91) + 0.09
$\lfloor \hat{p}\rfloor + \Delta\hat{p}$ 22 + 0.12 100 + 0 37 + 0.17 (-21) + 0.65 0 + 0.09
$\hat{\mathbf{q}}^T\hat{\mathbf{p}} \leq IU(\hat{\mathbf{q}},\hat{\mathbf{p}}) = 5726$

Figure 5: Integer Upper Bound After Scaling

The proposed integer upper bound becomes more accurate when e increases, as shown in Theorem 5 (in Appendix A). In particular, the error is inversely proportional to e.

5. MONOTONICITY

SVD based global reordering benefits incremental pruning, since the value distribution becomes more skewed, i.e., the first dimensions have larger absolute values than the latter ones. Another problem that previous studies have not addressed is the monotonicity. MF results in matrices with both negative and positive values. Thus, large absolute values in the first dimensions do not necessarily mean that the partial inner product ${\mathbf{q}^{\ell}}^T\mathbf{p}^{\ell}$ before incremental pruning is attempted occupies a large percentage of the exact inner product (i.e., $\mathbf{q}^T\mathbf{p}$). In order to alleviate this problem, we present a *reduction* that converts all vectors in $\mathbf{P}$ to ones that have positive values only. The reduction is also applied on $\mathbf{q}$. We show that after the reduction, the order of $\mathbf{p}$ vectors with respect to their inner products with the reduced query vector are preserved. This means that we can apply IP retrieval in the reduced space and obtain the exact top-k results as we would in the original space. By applying IP retrieval on vectors having only positive values, we can take advantage of the monotonically increasing IP during coordinate scan to obtain tighter bounds for incremental pruning. In Section 5.1, we first present an approach from previous work that reduces IP retrieval to k-NN search and motivates our approach. Our novel reduction is described in Section 5.2.

5.1 Existing Reduction of IP to Previous Problems

Bachrach et al. [7] propose an order preserving transformation to reduce IP retrieval to k-NN search using Euclidean distance and cosine similarity search (CSS):

THEOREM 3. *Top-k IP retrieval in the d-dimensional space can be reduced to k-NN retrieval or CSS problem in the $(d+1)$-dimensional space, by adding one dimension to each $\mathbf{q}^T$ and $\mathbf{p}^T$: $\tilde{\mathbf{q}}^T = (0, q_1, q_2, ..., q_d)$ and $\tilde{\mathbf{p}}^T = (\sqrt{b^2 - ||\mathbf{p}||^2}, p_1, p_2, ..., p_d)$ in preprocessing, where $b = max_{\mathbf{p}\in\mathbf{P}}||\mathbf{p}||$.*

PROOF. Proved in [7]. □

Based on this transformation, Bachrach et al. [7] reduce top-k IP retrieval to a k-NN problem and use a PCATree data structure to retrieve the approximate top-k results for any query $\mathbf{q}$.

5.2 Reduction for Monotonicity

The problem of the transformation by Theorem 3 is that when we change the top-k IP retrieval problem to a k-NN search in the Euclidean space, we lose the opportunity to use the low-overhead Cauchy-Schwarz termination check for SS (i.e., Line 6 of Algorithm 1). Thus, we consider how to transform the data so that top-k IP retrieval is still based on SS and monotonicity holds during IP computations by coordinate scan. By doing so, the retrieval process can benefit from both the Cauchy-Schwarz based early termination and monotonicity in incremental pruning. In the following, we will use $\equiv$ to denote that the order of results after some operations is preserved. For instance, we use $min\ ||\tilde{\mathbf{q}} - \tilde{\mathbf{p}}||^2 \equiv max\ \mathbf{q}^T\mathbf{p}$ to indicate that the order of IPs of $\mathbf{p}$ and $\mathbf{q}$ by maximum values is preserved by the order of Euclidean distances by minimum values between $\tilde{\mathbf{q}}$ and $\tilde{\mathbf{p}}$ after using Theorem 3.

While our goal is not to apply k-NN techniques for solving top-k IP retrieval, Theorem 3 inspired us to convert the values in $\mathbf{P}$ to positive ones (which is not something that Theorem 3 does). Euclidean distance has a property: after adding the same value to the same dimensions of two vectors the distance between them remains unchanged; i.e., the distance between vectors $\mathbf{q}$ and $\mathbf{p}$ equals the distance between vectors $\mathbf{q} + \mathbf{c}$ and $\mathbf{p} + \mathbf{c}$, where $\mathbf{c}$ is a vector of constants with the same number of dimensions as $\mathbf{q}$ and $\mathbf{p}$ and notation $+$ indicates vector addition. Accordingly, we change Theorem 3 so that all the values in vectors are positive, as follows:

LEMMA 1. *Top-k IP retrieval in the d-dimensional space can be reduced to k-NN retrieval in the $(d+1)$-dimensional space, by adding one dimension to $\mathbf{q}$ and $\mathbf{p}$ as follows: $\acute{\mathbf{p}}^T = (\sqrt{b^2 - ||\mathbf{p}||^2}, p_1 + c_1, p_2 + c_2, ..., p_d + c_d)$, where $b = max_{\mathbf{p}\in\mathbf{P}}||\mathbf{p}||$ and $c_s \geq \max\{1, |p_{min}|\}, \forall s \in \{1, 2, ..., d\}$. $|p_{min}|$ is the absolute value of the minimum value in $\mathbf{P}$. At query time, $\acute{\mathbf{q}}^T = (0, \frac{q_1}{||\mathbf{q}||} + c_1, \frac{q_2}{||\mathbf{q}||} + c_2, ..., \frac{q_d}{||\mathbf{q}||} + c_d)$. Values in each dimension of $\acute{\mathbf{q}}$ and $\acute{\mathbf{p}}$ are positive.*

PROOF. $||\acute{\mathbf{q}} - \acute{\mathbf{p}}||^2 = ||\frac{\tilde{\mathbf{q}}}{||\mathbf{q}||} - \tilde{\mathbf{p}}||^2$. By Theorem 3, $\min ||\frac{\tilde{\mathbf{q}}}{||\mathbf{q}||} - \tilde{\mathbf{p}}||^2 \equiv \max\ (\frac{\mathbf{q}}{||\mathbf{q}||})^T\mathbf{p}$ and hence we have $\min ||\acute{\mathbf{q}} - \acute{\mathbf{p}}||^2 \equiv \max\ (\frac{\mathbf{q}}{||\mathbf{q}||})^T\mathbf{p}$. For a query $\mathbf{q}$, $||\mathbf{q}||$ does not affect the order of inner product $\mathbf{q}^T\mathbf{p}$, thus $\min ||\acute{\mathbf{q}} - \acute{\mathbf{p}}||^2 \equiv \max\ (\mathbf{q}^T\mathbf{p})$. Since $p_s \geq -|p_{min}|$, we have $p_s + c_s \geq 0$ and the values of $\acute{\mathbf{p}}$ are nonnegative. Given $-1 \leq \frac{q_s}{||\mathbf{q}||} \leq 1$ and $c_s \geq 1$ for $1 \leq s \leq d$, we have $\frac{q_s}{||\mathbf{q}||} + c_s \geq 0$ and values of $\acute{\mathbf{q}}$ are nonnegative. □

Now the problem is how to convert k-NN retrieval to top-k IP retrieval, while keeping the nonnegativity of all scalars. Based on Lemma 1, FEXIPRO maps the original d-dimensional $\mathbf{p}$ and $\mathbf{q}$ vectors to $(d+2)$-dimensional vectors $\hat{\acute{\mathbf{p}}}$ and $\hat{\acute{\mathbf{q}}}$ using the following new Theorem 4. Vectors $\hat{\acute{\mathbf{p}}}$ and $\hat{\acute{\mathbf{q}}}$ have three properties: (1) all values in $\hat{\acute{\mathbf{p}}}$ are nonnegative, while the first dimension in $\hat{\acute{\mathbf{q}}}$ is negative and the remaining $(d+1)$ dimensions are nonnegative; (2) $max\ \hat{\acute{\mathbf{q}}}^T\hat{\acute{\mathbf{p}}} \equiv max\ \mathbf{q}^T\mathbf{p}$; (3) $\hat{\acute{\mathbf{p}}}$ and $\hat{\acute{\mathbf{q}}}$ support Cauchy-Schwarz based pruning. By using the new transformation, FEXIPRO can benefit from both early termination during SS and incremental pruning in coordinate scans.

THEOREM 4. *Monotonicity holds in IP computation, after adding two dimensions to $\mathbf{p}$ and $\mathbf{q}$ and transforming them to: $\hat{\acute{\mathbf{p}}}^T = (||\acute{\mathbf{p}}||^2, \acute{p}_1, ..., \acute{p}_{d+1})$, $\hat{\acute{\mathbf{q}}}^T = (-1, 2\acute{q}_1, ..., 2\acute{q}_{d+1})$. The order of the IPs in the original d-dimensional space (i.e., in $\mathbf{q}^T\mathbf{P}$) is preserved by the inner products ${\hat{\acute{\mathbf{q}}}_{\mathbf{i}}}^T\hat{\acute{\mathbf{p}}}_{\mathbf{j}}$ in the $(d+2)$-dimensional space: i.e., $max\ \hat{\acute{\mathbf{q}}}^T\hat{\acute{\mathbf{p}}} \equiv max\ \mathbf{q}^T\mathbf{p}$.*

PROOF. $\hat{\acute{\mathbf{q}}}^T\hat{\acute{\mathbf{p}}} = -||\acute{\mathbf{p}}||^2 + 2\acute{\mathbf{q}}^T\acute{\mathbf{p}}$. For a given query $\mathbf{q}$, $||\acute{\mathbf{q}}||$ is fixed. Thus $-||\acute{\mathbf{p}}||^2 + 2\acute{\mathbf{q}}^T\acute{\mathbf{p}} \equiv -||\acute{\mathbf{p}}||^2 + 2\acute{\mathbf{q}}^T\acute{\mathbf{p}} - ||\acute{\mathbf{q}}||^2$. We have $||\acute{\mathbf{q}} - \acute{\mathbf{p}}||^2 = ||\acute{\mathbf{p}}||^2 - 2\acute{\mathbf{q}}^T\acute{\mathbf{p}} + ||\acute{\mathbf{q}}||^2$. Thus finding the top-$k$ maximum $\hat{\acute{\mathbf{q}}}^T\hat{\acute{\mathbf{p}}}$ is equivalent to finding the top-k minimum $||\acute{\mathbf{q}} - \acute{\mathbf{p}}||^2$. From Lemma 1 we have $max\ \hat{\acute{\mathbf{q}}}^T\hat{\acute{\mathbf{p}}} \equiv min\ ||\acute{\mathbf{q}} - \acute{\mathbf{p}}||^2 \equiv max\ \mathbf{q}^T\mathbf{p}$. In addition, all $\acute{p}_s$ and $\acute{q}_s$ are nonnegative, where $1 \leq s \leq d+1$. Thus all the values in $\hat{\acute{\mathbf{q}}}$ and $\hat{\acute{\mathbf{p}}}$ are nonnegative, except $\hat{\acute{q}}_1$ which is -1. □

Theorem 4 can also be used to reduce the k-NN search problem to a top-k IP retrieval problem. Although Bachrach et al. [7] provide a transformation that reduces k-NN search to top-k IP retrieval, it does not guarantee monotonicity.

Data distribution (i.e., the existence of skew) is determined by $\mathbf{q}$, $\mathbf{p}$, and $\mathbf{c}$. We can set $\mathbf{c}$ in any way such that $c_s \geq \max(1, |p_{min}|), \forall c_s \in \{c_1, c_2, ..., c_d\}$. Recall that the distributions of $\mathbf{q}$ vectors after using SVD based reordering (Section 3) follow the distribution of Σ_d to some extent; therefore, here we use a similar way to set $\mathbf{c}$, i.e., $c_s = \max(1, |p_{min}|) + \frac{\sigma_s}{\sigma_d}, 1 \leq s \leq d$. Our experimental results suggest that this simple approach gives promising results, even though more elaborate methods for setting $\mathbf{c}$ are possible. By using the reduction of Theorem 4 after using SVD based global reordering, incremental pruning should be more effective since the values are nonnegative and the data distributions are more skewed compared to the original vectors.

6. FEXIPRO: PUTTING ALL TOGETHER

In this section, we will illustrate how FEXIPRO uses the techniques (i.e., SVD transformation, integer upper bound, and monotonicity reduction) introduced in the previous sections together for fast and exact inner product retrieval in recommender systems. Three key problems when combining these techniques are (1) how to integrate the integer upper bound with incremental pruning; (2) how to switch between the different inner product spaces; (3) the order of applying the three techniques.

Integer Based Incremental Pruning. Recall that we use $||\mathbf{q}||\,||\mathbf{p}||$ (i.e., Cauchy-Schwarz inequality) and $\sum_{s=1}^{d}\left(\lfloor q_s\rfloor\cdot\lfloor p_s\rfloor+\left|\lfloor q_s\rfloor\right|+\left|\lfloor p_s\rfloor\right|+1\right)$ (i.e., Equation 3) as upper bounds for the inner product between $\mathbf{q}$ and $\mathbf{p}$. There is no guarantee which one is tighter, however, $||\mathbf{q}||\,||\mathbf{p}||$ is faster to compute and should be used first in general. While computing the integer upper bound for $\mathbf{q}$ and $\mathbf{p}$, we can apply incremental pruning as follows:

$$\mathbf{q}^T\mathbf{p} \leq \sum_{s=1}^{w}\left(\lfloor q_s^\ell\rfloor\cdot\lfloor p_s^\ell\rfloor+\left|\lfloor q_s^\ell\rfloor\right|+\left|\lfloor p_s^\ell\rfloor\right|+1\right)+||\mathbf{q}^h||\,||\mathbf{p}^h|| \quad (6)$$

In other words, we can compute the integer IP only partially using the integral parts of the first w dimensions and then use Equation 6 to prune $\mathbf{p}$ if the right part of the equation is not greater than the k-th product so far (i.e., t). If $\mathbf{p}$ is not pruned, we compute the integer IP for the remaining $d-w$ dimensions and use the complete integer upper bound (i.e., Equation 3) to attempt pruning $\mathbf{p}$.

To apply the integer upper bound, we should first scale the values of the vectors, as discussed in Section 4.2. In order to make the bounds tighter and pruning more effective, instead of only using $max_{\mathbf{P}}$ and $max_{\mathbf{q}}$ as in Equation 4, we use different maximum absolute values for the scaling of the first w and the last $d-w$ dimensions of each vector as follows:

$$\hat{\mathbf{q}}^{\ell^T}=\left(\frac{e\cdot q_1}{max_{\mathbf{q}^\ell}},...,\frac{e\cdot q_w}{max_{\mathbf{q}^\ell}}\right),\ \hat{\mathbf{p}}^{\ell^T}=\left(\frac{e\cdot p_1}{max_{\mathbf{P}^\ell}},...,\frac{e\cdot p_w}{max_{\mathbf{P}^\ell}}\right)$$
$$\hat{\mathbf{q}}^{h^T}=\left(\frac{e\cdot q_{w+1}}{max_{\mathbf{q}^h}},...,\frac{e\cdot q_d}{max_{\mathbf{q}^h}}\right),\hat{\mathbf{p}}^{h^T}=\left(\frac{e\cdot p_{w+1}}{max_{\mathbf{P}^h}},...,\frac{e\cdot p_d}{max_{\mathbf{P}^h}}\right) \quad (7)$$

The reason for using different maximum absolute values is that the value distributions after the SVD transformation become more skewed. Scaling helps to avoid small values, as illustrated in Figures 4 and 5, and makes the integer upper bound tight. However, after the SVD transformation, the first few dimensions tend to have much higher values (e.g., 20) than the dimensions at the tails of the vectors (e.g., 0.01).[5] Therefore, scaling using the maximum absolute values of $\mathbf{P}$ and $\mathbf{q}$ is not effective for the smaller values of the vectors and may result in them having small or 0 integral parts, which makes the integer upper bound loose. By using different scaling for the two parts of $\mathbf{P}$ and $\mathbf{q}$, the upper bound for the residue vectors becomes much tighter.

Switching between Different Inner Product Spaces. For a given pair $(\mathbf{q},\mathbf{p})$ of vectors FEXIPRO considers four inner products: the original inner product $\mathbf{q}^T\mathbf{p}$, the inner product $\bar{\mathbf{q}}^T\bar{\mathbf{p}}$ after applying SVD transformation, the scaled inner product $\hat{\mathbf{q}}^T\hat{\mathbf{p}}$ for integer-based pruning, and the monotonic inner product $\hat{\hat{\mathbf{q}}}^T\hat{\hat{\mathbf{p}}}$. Incremental pruning can be attempted in all these representations. Since $\mathbf{q}^T\mathbf{p}$ equals $\bar{\mathbf{q}}^T\bar{\mathbf{p}}$, we only need to consider how to switch between the last three inner products.

The scaled inner product $\hat{\mathbf{q}}^{\ell^T}\hat{\mathbf{p}}^\ell$ ($\hat{\mathbf{q}}^{\mathbf{h}^T}\hat{\mathbf{p}}^{\mathbf{h}}$) can be computed based on $\mathbf{q}^{\ell^T}\mathbf{p}^\ell$ ($\mathbf{q}^{\mathbf{h}^T}\mathbf{p}^{\mathbf{h}}$) using Equation 7. From Lemma 1 and Theorem 4, we can find that we can derive $\hat{\hat{\mathbf{q}}}^T\hat{\hat{\mathbf{p}}}$ fast if we have already computed $\mathbf{q}^T\mathbf{p}$:

$$\hat{\hat{\mathbf{q}}}^T\hat{\hat{\mathbf{p}}}=\frac{2\mathbf{q}^T\mathbf{p}}{||\mathbf{q}||}+2\sum_{s=1}^{d}\left(\frac{c_s\cdot q_s}{||\mathbf{q}||}+c_s\cdot p_s+c_s^2\right)-||\acute{\mathbf{p}}||^2, \quad (8)$$

where $2\sum_{s=1}^{d}\left(c_s\cdot p_s+c_s^2\right)-||\acute{\mathbf{p}}||^2$ can be computed during preprocessing and $2\sum_{s=1}^{d}\frac{c_s\cdot q_s}{||\mathbf{q}||}$ can be obtained online while computing $||\mathbf{q}||$. Similarly, we can get $\hat{\hat{\mathbf{q}}}^{\ell^T}\hat{\hat{\mathbf{p}}}^\ell$ from $\mathbf{q}^{\ell^T}\mathbf{p}^\ell$.

Workflow of FEXIPRO. The SVD transformation (abbreviated as S) helps making the values of the first several dimensions larger, making incremental pruning more effective. Therefore, it should be applied before integer based incremental pruning (abbreviated as I) in order to improve the effectiveness of I. In addition, the rationale behind our reduction approach (abbreviated as R), introduced in Section 5, is to transform the original data so that the values become positive. Since performing SVD over positive values could make part of the data negative again and cancel the effect of R, S should be performed before R. In summary, the most promising orders of the three techniques are SIR and SRI. Experimentally, we found that in most cases the SIR order is superior to SRI, so we adopted it in the standard workflow of FEXIPRO.

Our framework FEXIPRO is described by Algorithms 3, 4 and 5. Algorithm 3 illustrates the preprocessing phase of FEXIPRO (i.e., SVD transformation, scaling for deriving integer upper bounds, and reduction). For each vector $\mathbf{p}$ of $\mathbf{P}$, FEXIPRO computes and stores together (1) the corresponding scaled vector $\hat{\mathbf{p}}=\hat{\mathbf{p}}^\ell|\hat{\mathbf{p}}^h$ (i.e., different scaling) and its integral approximation $\lfloor\hat{p}\rfloor$ and (2) the corresponding reduced vector $\hat{\hat{\mathbf{p}}}$ having positive values only. Together, FEXIPRO precomputes and stores the length of the original vector and the lengths of the residue vectors used in incremental pruning ($||\bar{\mathbf{p}}^h||$ and $||\hat{\hat{\mathbf{p}}}^h||$).

Algorithms 4 and 5 show how FEXIPRO conducts top-k IP retrieval. No matter whether the current task is to compute a whole inner product (i.e., $\mathbf{q}^T\mathbf{p}$) or a partial inner product (i.e., $\mathbf{q}^{\ell^T}\mathbf{p}^\ell$ and $\mathbf{q}^{h^T}\mathbf{p}^h$), FEXIPRO first tries the length upper bound $||\mathbf{q}||\,||\mathbf{p}||$; this is a cheap test because the length of each $\mathbf{p}$ is pre-computed and the length of each query $\mathbf{q}$ can be calculated online in a short time. In coordinate scan (Algorithm 5), FEXIPRO first computes the integer upper bound and tries to use it to prune the candidate vector $\mathbf{p}$ (Lines 2-8). If pruning based on the integer upper bound fails, the exact inner product $\mathbf{q}^{\ell^T}\mathbf{p}^\ell$ for the first w dimensions is computed (Lines 9-11). Incremental pruning is then attempted using Equation 1 (Lines 12-13). Then, the more costly incremental pruning based on monotonicity reduction is attempted (Lines 14-17 in Algorithm 5), before computing the exact IP for the residue of the vectors (Lines 18-20 in Algorithm 3). By using SVD transformation, the integer upper bound, and the reduced vectors alternately, FEXIPRO manages to prune a large number of candidates, and this results in a significant drop of the processing cost, as shown in the next section.

[5]Recall that making the first several dimensions larger than the others is the rationale behind our SVD transformation.

Algorithm 3 Preprocessing in FEXIPRO

```
1: procedure PREPROCESS($\rho$, $e$, $\mathbf{P}$)
2:     Sort $\mathbf{P}$ by length in a descending order
3:     $\mathbf{U}, \mathbf{\Sigma_d}, \mathbf{V_1} \leftarrow SVD(\mathbf{P})$        ▷ Thin SVD
4:     Calculate $w$ according to $\rho$
5:     $i \leftarrow 0$
6:     for each $\mathbf{p} \in \mathbf{P}$ do
7:         $\bar{\mathbf{p}} \leftarrow \mathbf{V_{1i}}$        ▷ $i$-th column of $\mathbf{V_1}$, Theorem 1
8:         Compute $\hat{\mathbf{p}}^\ell$, $\hat{\mathbf{p}}^h$ and $\lfloor \hat{p} \rfloor$ for $\bar{\mathbf{p}}$        ▷ Equation 7
9:         Calculate $\hat{\bar{\mathbf{p}}}$        ▷ Theorem 4
10:        $\mathbf{p}.length \leftarrow ||\mathbf{p}||$        ▷ length of original vector
11:        $\bar{\mathbf{p}}.rightLength \leftarrow ||\bar{\mathbf{p}}^h||$
12:        $\hat{\bar{\mathbf{p}}}.rightLength \leftarrow ||\hat{\bar{\mathbf{p}}}^h||$
13:        $i \leftarrow i + 1$
14:    return $w, \mathbf{U}, \mathbf{\Sigma_d}, \mathbf{P}$
```

Algorithm 4 Retrieval in FEXIPRO

```
1: procedure RETRIEVAL($w$, $e$, $\mathbf{U}$, $\mathbf{\Sigma_d}$, $\mathbf{q}$, $\mathbf{P}$, $k$)
2:     Priority Queue $r \leftarrow \varnothing$        ▷ maintain top-k results
3:     $t \leftarrow -\infty$        ▷ threshold for current top-$k$ $\mathbf{q}^T\mathbf{p}$
4:     $t' \leftarrow -\infty$        ▷ threshold for current top-$k$ $\hat{\bar{\mathbf{q}}}^T\hat{\bar{\mathbf{p}}}$
5:     $\bar{\mathbf{q}} \leftarrow \mathbf{\Sigma_d}\mathbf{U}^T\mathbf{q}$        ▷ Theorem 1
6:     Compute $\hat{\bar{\mathbf{q}}}$, $\hat{\mathbf{q}}^\ell$, $\hat{\mathbf{q}}^h$ and $\lfloor \hat{q} \rfloor$ for $\bar{\mathbf{q}}$        ▷ Theorem 4 and Equation 7
7:     $\mathbf{q}.length \leftarrow ||\mathbf{q}||$
8:     $\bar{\mathbf{q}}.rightLength \leftarrow ||\bar{\mathbf{q}}^h||$
9:     $\hat{\bar{\mathbf{q}}}.rightLength \leftarrow ||\hat{\bar{\mathbf{q}}}^h||$
10:    for each $\mathbf{p} \in \mathbf{P}$ do        ▷ descending order of $\mathbf{p}$'s length
11:        if $\mathbf{q}.length \cdot \mathbf{p}.length \leq t$ then
12:            return $r$
13:        else
14:            $v \leftarrow$ CoordinateScan($\mathbf{p}$, $\mathbf{q}$, $w$, $e$, $t$, $t'$)
15:            if $v > t$ then
16:                Push $\mathbf{p}$ into $r$ and update $t$
17:                Calculate $t'$ based on $t$        ▷ Equation 8 for $\hat{\bar{\mathbf{q}}}^T\hat{\bar{\mathbf{p}}}$
18:    return $r$
```

Algorithm 5 Coordinate Scan in FEXIPRO

```
1: procedure COORDINATESCAN($\mathbf{p}$, $\mathbf{q}$, $w$, $e$, $t$, $t'$)
2:     $b^\ell \leftarrow \frac{IU(\hat{\mathbf{q}}^\ell, \hat{\mathbf{p}}^\ell) \cdot \max \bar{\mathbf{q}}^\ell \cdot \max \bar{\mathbf{P}}^\ell}{e^2}$        ▷ Equation 7
3:     $ub_1 \leftarrow \bar{\mathbf{p}}.rightLength \cdot \bar{\mathbf{q}}.rightLength$
4:     if $b^\ell + ub_1 < t$ then        ▷ Equation 6
5:         return $-\infty$
6:     $b^h \leftarrow \frac{IU(\hat{\mathbf{q}}^h, \hat{\mathbf{p}}^h) \cdot \max \bar{\mathbf{q}}^h \cdot \max \bar{\mathbf{P}}^h}{e^2}$        ▷ Equation 7
7:     if $b^\ell + b^h < t$ then        ▷ Equation 3
8:         return $-\infty$
9:     $v \leftarrow 0$
10:    for $s$ in $1 \ldots w$ do
11:        $v \leftarrow v + \bar{p}_s \cdot \bar{q}_s$
12:    if $v + ub_1 < t$ then        ▷ Equation 1
13:        return $-\infty$
14:    Obtain $\hat{\bar{\mathbf{q}}}^{\ell T}\hat{\bar{\mathbf{p}}}^\ell$ based on $v$        ▷ Equation 8 for $\hat{\bar{\mathbf{q}}}^{\ell T}\hat{\bar{\mathbf{p}}}^\ell$
15:    $ub_2 \leftarrow \hat{\bar{\mathbf{p}}}.rightLength \cdot \hat{\bar{\mathbf{q}}}.rightLength$
16:    if $\hat{\bar{\mathbf{q}}}^{\ell T}\hat{\bar{\mathbf{p}}}^\ell + ub_2 < t'$ then        ▷ Lemma 1 and Theorem 4
17:        return $-\infty$
18:    for $s$ in $(w + 1) \ldots d$ do
19:        $v \leftarrow v + \bar{p}_s \cdot \bar{q}_s$
20:    return $v$
```

7. EXPERIMENTS

We compared FEXIPRO with several previous approaches for top-k IP retrieval in recommender systems. All methods were implemented in C++ using standard libraries and -O3 optimization flag. To implement thin SVD, we used Armadillo[6], a C++ linear algebra library. The experiments were conducted on a Intel(R) Core(TM) CPU i5-4590 @ 3.30 GHz machine running Ubuntu 16.04, with 16 GB of main memory.

7.1 Experimental Setup

We compare FEXIPRO with four competitors: Naive, BallTree, FastMKS and SS-L. The Naive method sequentially scans the columns of $\mathbf{P}$ and computes all $\mathbf{q}^T\mathbf{p}$; while doing so it keeps track of the top-k products with the help of a priority queue. The BallTree method [32] uses a space-partitioning tree, which is optimized for top-k IP search. We set the maximum capacity of leaf nodes in the BallTree to 20, as suggested in [32]. We did not implement its advanced version, DualTree, as it was reported to be not better than BallTree in previous studies [32, 36]. FastMKS [16, 15] uses another tree, called CoverTree, for efficient top-k IP retrieval. The base parameter of CoverTree is set to 1.3, as suggested in [16]. SS-L is a version of SS (Section 2.2) that applies the optimizations of LEMP that were found to be the most effective ones in [36, 35] for the problem studied there (i.e., the computation of top-k IP results for all $\mathbf{q} \in \mathbf{Q}$). In particular, IP computation in SS-L applies on the *normalized* vectors $\mathbf{q}$ and $\mathbf{p}$ and SS-L applies coordinate-based pruning before incremental pruning. We used publicly available source codes for FastMKS[7] and LEMP[8] (for the implementation of IP computation and pruning in SS-L).

We ran five versions of FEXIPRO: F-S, F-I, F-SI, F-SR, and F-SIR where S, I and R represent SVD transformation, integer upper bound, and monotonicity reduction respectively. F-S only adopts SVD based transformation and performs incremental pruning, as discussed in Sections 2.2.1 and 3. F-I uses the scaled integer upper bound described in Section 4.2 and integer-based incremental pruning (i.e., Equation 6), before calculating the exact inner products. F-SI is a combined version of F-S and F-I, which applies Algorithm 5 for coordinate scan, excluding Lines 14-17. F-SR uses monotonicity reduction after SVD reordering, as shown in Section 5.2; i.e., it only applies Lines 9-11 and 14-19 of Algorithm 5. F-SIR uses all three methods, as described in Section 6. The default values of the parameters used for integer scaling and for determining the checking dimension w are $e = 100$ and $\rho = 0.7$, respectively.

Table 2 summarizes the four datasets that we have used, taken from real applications and widely used by previous research on recommender systems. MovieLens[9], Yelp[10] and Netflix[11] include 5-point ratings. Yahoo! Music[12] uses a 100-point scale; we map all ratings in it to a 5-point scale, in order to be able to compare performance using the same scaling values e and $RMSE$ standard as other datasets. To facilitate the repeatability of our experiments, we used the open-source MF library LIBPMF[13] to factorize each dataset into 50-dimensional vectors with the same factorization parameters as those reported in [41]. We put results over other values of d in Appendix B - *Performance with Different Values of d.* The original training/test split in the datasets is used in MF.

7.2 Results And Analysis

Overall Performance. Table 4 shows the total wall-clock time (in

[6]http://arma.sourceforge.net

[7]http://mlpack.org

[8]http://dws.informatik.uni-mannheim.de/en/resources/software/lemp

[9]MovieLens latest dataset (April 2016): http://grouplens.org/datasets/movielens

[10]Round 7 of Yelp Data Challenge: http://www.yelp.com/dataset_challenge

[11]Netflix Prize: http://netflixprize.com

[12]Yahoo! Music: http://webscope.sandbox.yahoo.com/catalog.php?datatype=r

[13]http://www.cs.utexas.edu/~rofuyu/libpmf

Table 2: Statistics of the Four Real Datasets

Dataset	m	n	# of ratings
MovieLens	247,753	33,670	22,884,377
Yelp	552,339	77,079	2,155,421
Netflix	480,189	17,770	100,480,507
Yahoo! Music	1,000,990	624,961	256,804,235

Table 3: Average Number of Entire $\mathbf{q}^T\mathbf{p}$ Computations ($k = 1$)

	BallTree	SS-L	F-S	F-SI	F-SIR
MovieLens	6570.50	1212.45	152.85	56.39	6.84
Yelp	18419.74	8191.13	1180.53	50.08	1.94
Netflix	12795.68	7813.12	5427.29	1776.34	12.70
Yahoo! Music	78659.87	32011.48	10180.79	3697.55	8.22

seconds) spent by each method to compute the top-1 inner product of all queries in $\mathbf{Q}$. The numbers in brackets indicate the costs for preprocessing (e.g., applying thin SVD, scaling numbers, computing vector lengths). Results for other values of k can be found in Table 8 in Appendix B. Figure 6 demonstrates the speedup of F-SIR (in total time including both retrieval and preprocessing) over other methods. Overall, the sequential scan based methods SS-L and FEXIPRO are much faster than the tree based methods (i.e., BallTree and FastMKS) using different values of k. The tree based methods are sometimes even slower than Naive, due to the curse of dimensionality. F-SIR is the fastest among all the approaches regardless of k, outperforming BallTree, FastMKS, and SS-L, typically by an order of magnitude. In Netflix, the gap between the approaches is not significant as in the other three datasets. Specifically, SS-L only achieves a two-fold runtime reduction compared to Naive and the speedup of F-SIR over SS-L is about 4. We will explain the different behavior on Netflix later, when analyzing sensitivity to k and retrieval time for individual queries.

The preprocessing cost of all methods is affordable. Compared to other methods (except FastMKS), F-S, F-SI, F-SR and F-SIR have higher preprocessing cost because of the SVD transformation they apply. Still, this extra cost is not excessive, considering that preprocessing only needs to be conducted once. The low cost for preprocessing also demonstrates the benefit of using thin SVD, due to the property brought by Theorem 1. When comparing different versions of FEXIPRO, we can find that each version has a better performance than its simple version (e.g., F-SIR is better than F-SI and F-SI is better than F-S), which demonstrates that the different parts of FEXIPRO (i.e., SVD reordering, integer upper bound and monotonicity reduction) complement each other.

Pruning Power. Table 3 averages for each query vector $\mathbf{q}$ the number of $\mathbf{p}$ vectors for which the entire exact inner product $\mathbf{q}^T\mathbf{p}$ is computed (for $k = 1$). The distribution is shown in Appendix B - *Number of Entire* $\mathbf{q}^T\mathbf{p}$ *Computations*. We compare BallTree, SS-L and three versions of FEXIPRO. For other values of k, the results are shown in Table 7 in Appendix B. Observe the significant drop in the number of entire product computations by the different versions of FEXIPRO, compared to previous approaches; this explains why FEXIPRO is much faster than previous work. F-SIR ends up with a much smaller number of products to compute entirely, compared to F-S and F-SI, at the cost of partial IP computations and more operations for pruning checks. As a result, the gap between the retrieval time of F-SIR and those of F-S and F-SI is not as large as their differences in Table 3.

Sensitivity to k. Figure 7 illustrates the performance of SS-L and F-SIR using different k. The performance of the two sequential scan based methods FEXIPRO and SS-L degrades with k. In order to explain the behaviors of FEXIPRO and SS-L, we conduct an analysis of the data. We calculate the average inner product for the k-th returned item, as a function of k (see Figure 8). We found that when k is small, the difference between the smallest inner products in the top-k sets is quite large. As k gets larger, the differences between the smallest inner products in the result become smaller (e.g., notice the small gap between the 40-th and the 50-th largest product). Therefore, as k increases it becomes harder for FEXIPRO and SS-L to prune $\mathbf{p}$ vectors using length products and incremental pruning, as their differences to the ones in the top-k set becomes smaller. From Figure 8 we can also see that for dataset Netflix the top-k inner products show a slower rate of decay: the average inner product decreases by only 0.45 when k increases from 1 to 50. This indicates that the differences between the IPs in Netflix are smaller; therefore, top-k IP retrieval is harder for this dataset. In practice, recommender systems use a small value of k (e.g., around 5) in order not to overwhelm the users.

Costs of Individual Queries. Figure 9 shows the distribution of the costs of individual queries $\mathbf{q}$ (the shadowed area of each plot is the total retrieval cost for all queries). For datasets MovieLens, Yelp and Yahoo! Music, the great majority of queries have a very low cost, therefore the overall cost shown in Table 4 is not dominated by queries which need long retrieval times. On the other hand, on Netflix, queries have a more uniform cost and most of them need moderate retrieval time. This explains the less significant average improvement of FEXIPRO over Naive on Netflix (about 6x), compared to other datasets (75x-300x).

Performance Using Different Parameters. FEXIPRO requires two parameters ρ (for choosing the checking dimension w in incremental pruning) and e (for the scaling of scalars in order to apply the integer upper bound). Figure 10 illustrates the performance of SS-L, F-S and F-SIR as well as the value of selected w for different ρ. Observe that the best performance is achieved when $\rho = 0.7$ or $\rho = 0.8$. When $\rho = 0.7$, the corresponding value of w is between 6 and 15 which is relatively small compared to the number of dimensions (d = 50 in our experiments). Therefore, FEXIPRO can prune the great majority of vectors by only scanning 10%-30% of their coordinates. The retrieval cost is not very sensitive to ρ, as it remains within a narrow margin for different ρ values. Figure 11 shows how the value of e affects the performance. Observe that when e gets larger than 100 the cost converges. Therefore, it suffices to set $e = 100$ in order to achieve good performance while also scaling the values to relatively small integers (which require less space). This indicates the potential of FEXIPRO to benefit more from the computational power of modern hardware architectures, by using SIMD (Single Instruction Multiple Data) instruction set. Integer approximations in a small range (e.g, $[-128, 127]$) can fit in small integer types (e.g., int8), and the registers can accommodate more data, while the pruning power remains the same as when using large integer types. Hence, performance can further be improved by processing the products of multiple integers in parallel. We plan to explore this direction of further accelerating FEXIPRO in our future work.

Batch Query Processing. In our problem setting, we consider the efficient top-k IP retrieval for a *single* user vector $\mathbf{q}$. Therefore, in our experiments so far, we assumed that multiple user vectors are considered independently. However, a real recommender system should be able to handle multiple query requests simultaneously in order to improve its performance. Toward this direction, we first consider the possibility of using high-performance matrix kernel libraries (e.g., Armadillo, BLAS, and Intel MKL) to process multiple top-k IP queries in batch, with the potential use of multi-threading. For this purpose, we use the *dgemm* routine with *cBLAS*

Table 4: Total Retrieval and Preprocessing Times (in seconds) for All Top-1 IP Queries

	MovieLens		Yelp		Netflix		Yahoo! Music	
	Retrieve	Preprocess	Retrieve	Preprocess	Retrieve	Preprocess	Retrieve	Preprocess
Naive	441.53	-	2203.27	-	443.99	-	31706.75	-
BallTree	222.62	(0.23)	1961.13	(0.55)	868.52	(0.09)	21096.32	(5.10)
FastMKS	338.15	(0.23)	2695.58	(4.92)	973.00	(0.90)	16340.25	(51.14)
SS-L	68.22	(0.12)	273.97	(0.29)	251.77	(0.12)	4990.64	(1.92)
F-S	1.91	(0.31)	23.75	(0.64)	109.18	(0.17)	714.57	(4.87)
F-I	**0.95**	(0.10)	24.79	(0.23)	149.28	(0.05)	731.61	(1.88)
F-SI	1.53	(0.29)	16.79	(0.75)	138.17	(0.12)	600.53	(6.74)
F-SR	1.66	(0.34)	21.06	(0.66)	85.57	(0.17)	599.32	(5.03)
F-SIR	1.47	(0.44)	**13.14**	(0.89)	**69.56**	(0.25)	**420.11**	(8.30)

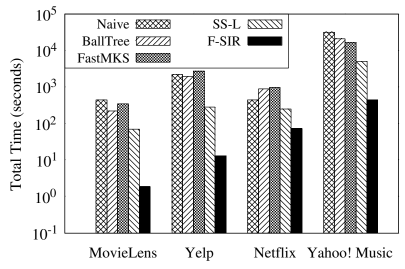

Figure 6: Total Cost (k=1)

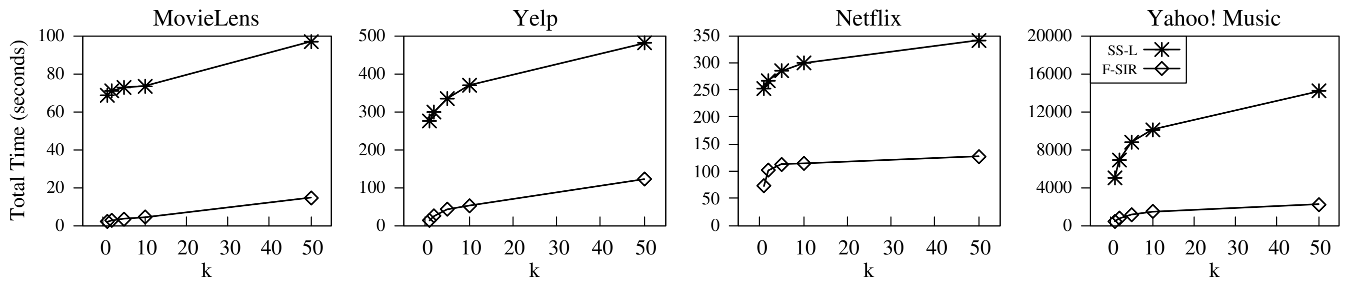

Figure 7: Total Retrieval Times for All Queries (Varying k)

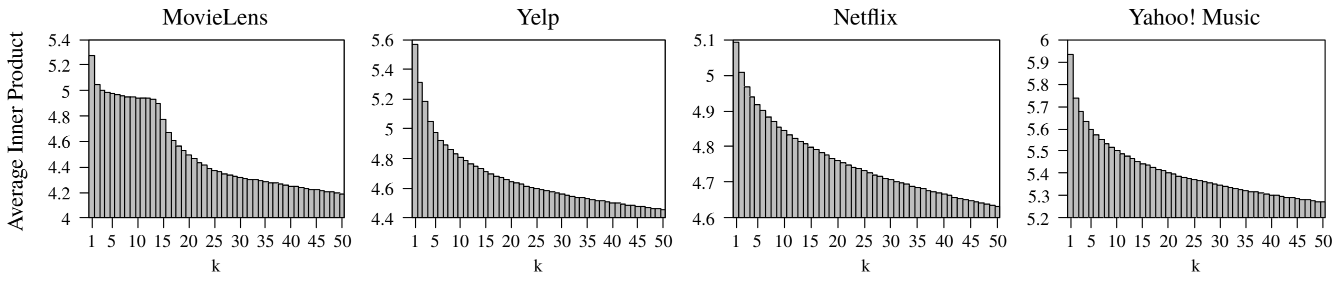

Figure 8: Average k-th Inner Product per Query

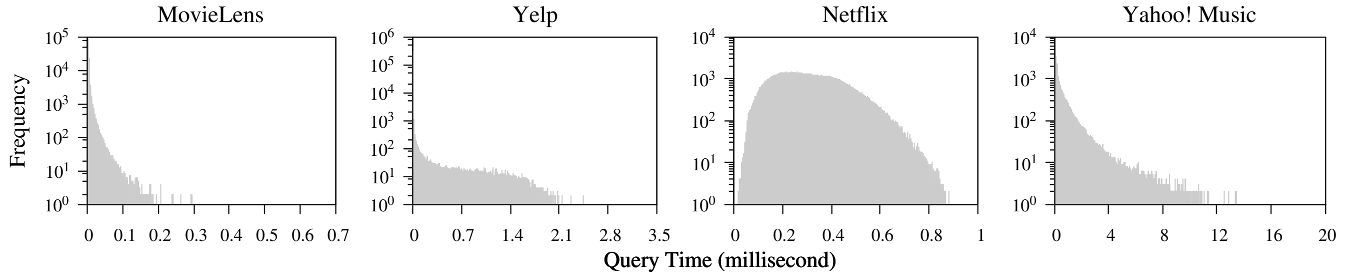

Figure 9: Distribution of Individual Query Costs (F-SIR, $k = 1$)

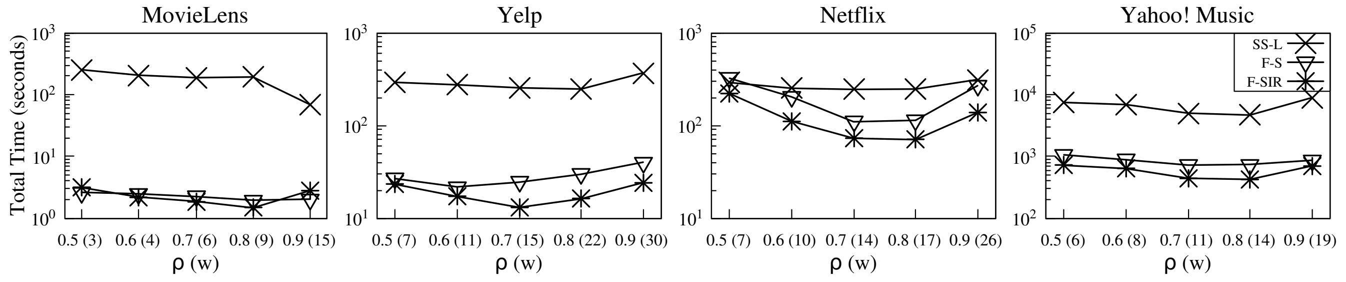

Figure 10: Total Retrieval Times for All Queries for Different $\rho\,(w)$ Values ($k = 1$)

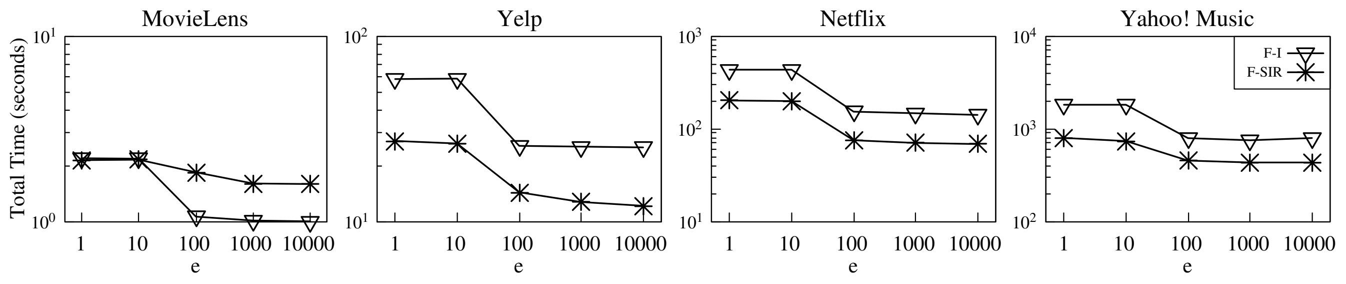

Figure 11: Total Retrieval Times for All Queries for Different e Values ($k = 1$)

in Intel MKL for matrix multiplication[14] to implement a *MiniBatch* version of the Naive method. In particular, we process the entire $\mathbf{Q}$ workload of users, by taking a batch of user vectors at a time, computing the top k results for all user vectors in the batch by multiplying them with the item matrix $\mathbf{P}$, and then proceeding to process the next batch in the same way. By default, MKL uses one thread for one physical core and the machine we use has four cores (four threads in total). Table 5 shows the results for $k = 1$ (due to memory constraints, we did not test Yahoo! Music with batch size of 10,000). By comparison to Table 4, we can observe that when using highly optimized matrix kernel libraries, MiniBatch can achieve quite a good performance. However, FEXIPRO as well as all its competitors in our previous experiments were implemented using single threading and the standard C++ library. Given that Intel MKL is highly optimized on Intel's microprocessor platforms and the block matrix multiplication method further improves cache locality (FEXIPRO only processes one query each time due to our problem setting), these significant performance gains are expected. Even though we have not employed any hardware-aware optimizations (e.g., SIMD, CPU cache locality) and multithreading, FEXIPRO still outperforms MiniBatch by 10, 6, 2.5 times on MovieLens, Yelp and Yahoo! Music, respectively. Netflix is a data set with different distribution and all pruning based methods perform poorly as shown in Table 4. Therefore, FEXIPRO does not outperform Intel MKL on Netflix, but it has a comparable performance to multi-threaded MiniBatch (BatchSize=1). The performance of FEXIPRO is expected to improve if we incorporate optimized matrix kernel libraries into it.

Table 5: MiniBatch Using Intel MKL (in seconds)

Dataset	batch size=1		batch size=100		batch size=10000	
	single-threaded	multi-threaded	single-threaded	multi-threaded	single-threaded	multi-threaded
MovieLens	209.06	127.41	30.45	17.43	25.36	15.42
Yelp	1106.5	791.69	157.82	85.15	129.37	75.38
Netflix	157.57	68.83	31.60	16.55	26.06	15.37
Yahoo! Music	21478.3	18677.2	2299.22	1250.37	-	-

We also show the performance of LEMP [36] (i.e., LEMP-LI in original implementation), which is the state-of-the-art algorithm for batch top-k IP evaluation (i.e., the top-k inner product join problem). We apply LEMP to obtain the top-k IP results of all queries in $\mathbf{Q}$, using the original LEMP implementation, which applies several optimizations (i.e., sampling for tuning w dynamically, processing queries in batches, using a cache-friendly bucketization). Table 6 shows the retrieval time of LEMP for different values of k. By comparison to Table 4 (and Table 8 in Appendix B), note that FEXIPRO is much faster than LEMP in all datasets (except Netflix), although FEXIPRO processes the queries one-by-one in a for-loop without employing any batch-processing optimizations. Since sampling and bucketization used by LEMP are orthogonal to our framework, we plan to embed these approaches into FEXIPRO in the future and develop a unified framework for both single and batch inner product retrieval tasks in recommender systems.

[14] https://software.intel.com/en-us/node/529735

Table 6: Batch Query Processing by LEMP (in seconds)

Dataset	k=1	k=2	k=5	k=10	k=50
MovieLens	45.32	47.66	52.18	54.98	62.52
Yelp	124.56	145.22	196.47	212.11	278.15
Netflix	78.45	96.12	114.63	137.23	221.16
Yahoo! Music	2589.11	3417.65	5178.01	6289.18	8561.09

8. RELATED WORK

Top-k IP retrieval is related to, but not equivalent to, two well-known problems: k nearest neighbor search (KNN) and cosine similarity search (CSS) in vector spaces. The three problems are equivalent if all vectors have the same length, which is not true in MF-based recommender systems. Thus, many previous techniques for KNN [40] and CSS [9, 6, 44, 39] cannot directly be used to solve the top-k IP retrieval problem. Top-k IP retrieval over normalized vectors (i.e., CSS) is well studied in document retrieval using keyword queries [29, 12]. Besides the normalization of the lengths, another important difference between document vectors and MF-based item vectors is that the former are very sparse, having a very small percentage of non-zero values. This allows for their effective indexing using inverted files.

As a result, a number of specialized techniques have been proposed for speeding up top-k IP retrieval. In Euclidean embedding approaches [22, 23], users and items are embedded in the Euclidean space; the top-k items to be recommended to a target user are then the k nearest item vectors of the user vector. Along this line, Fraccaro et al. [18] recently proposed an indexable probabilistic matrix factorization approach that uses Geodesic Monte Carlo as inference procedure. KNN can directly be applied to the output of this framework. Although these modified learning techniques reduce the retrieval time, they deviate from the widely used inner product based matrix factorization framework and they may not be considered equally effective. Besides, the resulting vector space also has high dimensionality and it is resistant to indexing. The approach we propose in this paper belongs to the class of methods that apply top-k IP retrieval on the output of traditional MF. These methods are orthogonal to the learning phase; therefore, existing systems can use them without having to change their learning phase. Three categories of top-k IP retrieval approaches have been proposed in this direction.

Hash-based Retrieval. Techniques in the first category approximate user and item vectors using hash codes in order to speed up retrieval, albeit at the risk of reducing the quality of recommendations. Some approaches apply Locality Sensitive Hashing (LSH) [19], which has been shown effective in solving nearest neighbor search problems. LSH based methods [34, 30, 5] are independent of the dimensionality d of the user and item vectors, but they usually require long hash bits and multiple hash tables with high storage overhead that may hinder their applicability. Other approaches [43, 42] apply *learning to hash* to convert user and item vectors into binary hash codes. Top-k IP retrieval is then transformed to searching for item hash codes with a small Hamming distance to the user code. Zhou et al. [43] construct binary codes such that the inner product of user and item vectors can be preserved by the Hamming distance between their respective binary codes. Zhang et al. [42] argue that the IP between two vectors is fundamentally different from the similarity between them. They improve the approach of [43] by constructing binary codes that preserve the ranking order of IPs instead of the similarity between vectors. Learning to hash methods, in addition being approximate, are not suitable for recommender systems which modify query vectors online using contextual information, since the corresponding binary codes cannot be updated dynamically.

Tree-based Methods. Tree based approaches usually suffer from the cost of random accesses and from the curse of dimensionality. The performance of tree based methods degrades fast and they eventually become worse than sequential scan as the dimensionality increases to a few tens of dimensions [38]. The metric-tree based approach [25] indexes the item vectors and uses a simple branch-and-bound algorithm to retrieve the top-k items. However, this method shows no improvement over the naive approach as k getting larger. Thus, Koenigstein et al. [25] also propose a method which clusters users and uses the top-k results of cluster centers as approximation. DualTree based methods [32, 16, 15] use space-partitioning trees to index $\mathbf{Q}$ and $\mathbf{P}$ and branch-and-bound algorithms in order to boost the efficiency. [7] applies an SVD based transformation to the vectors and indexes the first few components of the new vectors by a PCATree, in order to solve the top-k IP retrieval problem approximately. While the motivation is similar to our SVD transformation technique, SVD in [7] is applied for creating the PCATree which is used for approximate search. In addition, [7] applies KNN search in the Euclidean space, while FEXIPRO searches in the original inner product space. As we show in Appendix B - *Comparsion with PCATree*, our framework is faster than the PCATree approach, although we find the exact top-k IPs.

Sequential Scan. LEMP [36, 35] is a sequential scan based approach which solves the top-k IP retrieval problem exactly. Our framework shares some common modules with LEMP (i.e., early termination using Cauchy-Schwarz inequality, incremental pruning), but it also has significant differences. First of all, LEMP focuses on a different problem; that of retrieving all IP values in $\mathbf{Q}^T\mathbf{P}$ above a threshold t; therefore LEMP uses a bucketization approach for $\mathbf{P}$ and $\mathbf{Q}$, which does not offer any benefit to top-k IP retrieval for an individual $\mathbf{q}$. In contrast, our FEXIPRO framework focuses on individual top-k IP retrieval and our techniques are also useful to recommenders which dynamically change $\mathbf{q}$ before search, like Microsoft Xbox [7]. Second, LEMP reduces IP computation to CSS computation and uses the directions of the vectors for indexing and pruning; however, these techniques are not proved to be as effective as incremental pruning. FEXIPRO focuses on optimizing incremental pruning with the help of three approaches not employed by LEMP, i.e., SVD-based transformation, integer-based pruning, and reduction to monotonic IP computation. These approaches were shown in our experiments to be very effective in practice.

In summary, none of the previous approaches are designed to efficiently solve the exact top-k inner product retrieval problem in recommender systems, for *individual* user vectors $\mathbf{q}$, which can potentially be changed online.

Related Problems. There is a variant of the top-k IP retrieval problem called top-k all-pairs inner product search (AIP) [8]. AIP aims at finding the top-k largest values in $\mathbf{Q}^T\mathbf{P}$. To avoid direct computation of inner products for all pairs, a sampling approach which selects diamonds from the weighted tripartite representations of $\mathbf{Q}$ and $\mathbf{P}$ is proposed in [8]. Since AIP targets a different objective, the solutions proposed for AIP are not tailored to our problem.

9. CONCLUSION AND DISCUSSION

In this paper, we proposed a framework FEXIPRO for fast top-k inner product (IP) retrieval in recommender systems based on matrix factorization, which builds upon a basic sequential scan approach. Besides applying two popular heuristics (early termination using Cauchy-Schwarz inequality, incremental pruning), we propose three approaches for improving the pruning effectiveness. Our framework, is significantly faster than alternative approaches, typically by an order of magnitude.

Although FEXIPRO is designed for MF based recommender systems, its three techniques can be adopted by other applications of retrieval based on inner products. Note that FEXIPRO is suited for IP retrieval over dense vectors; for sparse vectors, inverted index based methods can be a better choice. We now briefly discuss the conditions under which we expect these techniques to be effective for general inner product retrieval tasks.

- SVD transformation introduces skew to the vectors, making it easier to prune item vectors after computing their partial inner products and applying incremental pruning. However, if the differences between the eigenvalues are small, SVD will not give much speedup. For example, if $\mathbf{P}$ has high entropy (i.e., the distribution of values in $\mathbf{P}$ is close to uniform), then the singular values (square roots of eigenvalues) are roughly the same and our SVD transformation will not be effective.
- Integer approximation represents the scalars in the vectors as integers after scaling them up. This method is effective when the values are within a small range, which is common for MF based recommender systems. If the values vary a lot, we do not expect the technique to be very effective. Operations on integers are typically faster compared to operations on floating-point numbers. Besides, using small integers (e.g., int8) can take advantage of SIMD instructions in modern CPUs (e.g., by performing multiple int8 operations per cycle).
- Monotonicity reduction transforms the search space to a new space where vectors only have positive values; this allows the derivation of tighter bounds from the partial IP by exploiting the monotonicity in IP computation. This technique can be used to accelerate general inner product retrieval tasks, where the percentage of negative values is high. We have such distributions in MF-based recommender systems (e.g., see Figure 3). In applications (e.g., social link prediction and document similarity search), where values are already positive after a specific factorization (e.g., non-negative matrix factorization), the reduction is not expected to speedup the retrieval phase.

In the future, we plan to study the effectiveness of our framework on other top-k IP computation problems, such as computing the above-t [36] or the top-k largest values in $\mathbf{Q}^T\mathbf{P}$ [8]. Our proposed transformations can also be used by other retrieval methods and thus we also plan to explore the direction of using the output of our transformations as input to alternative methods (including approximate methods [35]) in order to improve their efficiency.

10. REFERENCES

[1] R. A. Adams and C. Essex. Calculus: a complete course. page 69, 2010. Toronto: Pearson Canada.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[3] C. C. Aggarwal. Recommender systems - the textbook. pages 183–184, 2016. Springer.

[4] C. C. Aggarwal, J. L. Wolf, K. Wu, and P. S. Yu. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *KDD*, pages 201–212, 1999.

[5] T. D. Ahle, R. Pagh, I. P. Razenshteyn, and F. Silvestri. On the complexity of inner product similarity join. In *PODS*, pages 151–164, 2016.

[6] D. C. Anastasiu and G. Karypis. L2AP: fast cosine similarity search with prefix L-2 norm bounds. In *ICDE*, pages 784–795, 2014.

[7] Y. Bachrach, Y. Finkelstein, R. Gilad-Bachrach, L. Katzir, N. Koenigstein, N. Nice, and U. Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *RecSys*, pages 257–264, 2014.

[8] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri. Diamond sampling for approximate maximum all-pairs dot-product (MAD) search. In *ICDM*, pages 11–20, 2015.

[9] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[10] R. M. Bell and Y. Koren. Lessons from the netflix prize challenge. *SIGKDD Explorations*, 9(2):75–79, 2007.

[11] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[12] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Y. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, pages 426–434, 2003.

[13] R. D. Burke, K. J. Hammond, and B. C. Young. The FindMe approach to assisted browsing. *IEEE Expert*, 12(4):32–40, 1997.

[14] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.

[15] R. R. Curtin, A. G. Gray, and P. Ram. Fast exact max-kernel search. In *SDM*, pages 1–9, 2013.

[16] R. R. Curtin and P. Ram. Dual-tree fast exact max-kernel search. *Statistical Analysis and Data Mining*, 7(4):229–253, 2014.

[17] A. P. de Vries, N. Mamoulis, N. Nes, and M. L. Kersten. Efficient k-nn search on vertically decomposed data. In *SIGMOD Conference*, pages 322–333, 2002.

[18] M. Fraccaro, U. Paquet, and O. Winther. Indexable probabilistic matrix factorization for maximum inner product search. In *AAAI*, pages 1554–1560, 2016.

[19] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[20] R. He and J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016.

[21] M. P. Holmes, A. G. Gray, and C. L. Isbell. Fast SVD for large-scale matrices. In *Workshop on Efficient Machine Learning at NIPS*, volume 58, pages 249–252, 2007.

[22] M. Khoshneshin and W. N. Street. Collaborative filtering via euclidean embedding. In *RecSys*, pages 87–94, 2010.

[23] N. Koenigstein and Y. Koren. Towards scalable and accurate item-oriented recommendations. In *RecSys*, pages 419–422, 2013.

[24] N. Koenigstein and U. Paquet. Xbox movies recommendations: variational bayes matrix factorization with embedded feature selection. In *RecSys*, pages 129–136, 2013.

[25] N. Koenigstein, P. Ram, and Y. Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*, pages 535–544, 2012.

[26] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[27] H. Li, D. Wu, W. Tang, and N. Mamoulis. Overlapping community regularization for rating prediction in social recommender systems. In *RecSys*, pages 27–34, 2015.

[28] D. Lian, C. Zhao, X. Xie, G. Sun, E. Chen, and Y. Rui. GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *KDD*, pages 831–840, 2014.

[29] Y. Low and A. X. Zheng. Fast top-k similarity queries via matrix compression. In *CIKM*, pages 2070–2074, 2012.

[30] B. Neyshabur and N. Srebro. On symmetric and asymmetric LSHs for inner product search. In *ICML*, pages 1926–1934, 2015.

[31] U. Paquet and N. Koenigstein. One-class collaborative filtering with random graphs. In *WWW*, pages 999–1008, 2013.

[32] P. Ram and A. G. Gray. Maximum inner-product search using cone trees. In *KDD*, pages 931–939, 2012.

[33] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[34] A. Shrivastava and P. Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS*, pages 2321–2329, 2014.

[35] C. Teflioudi and R. Gemulla. Exact and approximate maximum inner product search with lemp. *ACM Trans. Database Syst.*, 42(1):5:1–5:49, 2016.

[36] C. Teflioudi, R. Gemulla, and O. Mykytiuk. LEMP: fast retrieval of large entries in a matrix product. In *SIGMOD Conference*, pages 107–122, 2015.

[37] S. Vucetic and Z. Obradovic. Collaborative filtering using a regression-based approach. *Knowl. Inf. Syst.*, 7(1):1–22, 2005.

[38] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[39] C. Xiao, W. Wang, X. Lin, and H. Shang. Top-k set similarity joins. In *ICDE*, pages 916–927, 2009.

[40] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, pages 311–321, 1993.

[41] H. Yu, C. Hsieh, S. Si, and I. S. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, pages 765–774, 2012.

[42] Z. Zhang, Q. Wang, L. Ruan, and L. Si. Preference preserving hashing for efficient recommendation. In *SIGIR*, pages 183–192, 2014.

[43] K. Zhou and H. Zha. Learning binary codes for collaborative filtering. In *KDD*, pages 498–506, 2012.

[44] S. Zhu, J. Wu, H. Xiong, and G. Xia. Scaling up top-k cosine similarity search. *Data Knowl. Eng.*, 70(1):60–83, 2011.

APPENDIX

A. TIGHTNESS OF INTEGER BOUND

THEOREM 5. $\lim_{e \to \infty} \frac{max_{\mathbf{q}} \cdot max_{\mathbf{P}}}{e^2} IU(\hat{\mathbf{q}}, \hat{\mathbf{p}}) = \mathbf{q}^T \mathbf{p}$.

PROOF. Since $\hat{q}_s - 1 \leq \lfloor \hat{q}_s \rfloor \leq \hat{q}_s + 1$, we have $\frac{e \cdot q_s}{max_{\mathbf{q}}} - 1 \leq \lfloor \hat{q}_s \rfloor \leq \frac{e \cdot q_s}{max_{\mathbf{q}}} + 1$. By dividing both sides by e and applying limit, we get $\lim_{e \to \infty} (\frac{q_s}{max_{\mathbf{q}}} - \frac{1}{e}) \leq \lim_{e \to \infty} \frac{\lfloor \hat{q}_s \rfloor}{e} \leq \lim_{e \to \infty} (\frac{q_s}{max_{\mathbf{q}}} + \frac{1}{e})$. Using Squeeze Theorem [1], we have $\lim_{e \to \infty} \frac{\lfloor \hat{q}_s \rfloor}{e} = \frac{q_s}{max_{\mathbf{q}}}$. Similarly, we can get $\lim_{e \to \infty} \frac{||\lfloor \hat{q}_s \rfloor||}{e^2} = 0$, $\lim_{e \to \infty} \frac{\lfloor \hat{p}_s \rfloor}{e} = \frac{p_s}{max_{\mathbf{P}}}$ and $\lim_{e \to \infty} \frac{||\lfloor \hat{p}_s \rfloor||}{e^2} = 0$. Then:

$$\lim_{e \to \infty} \frac{max_{\mathbf{q}} \cdot max_{\mathbf{P}}}{e^2} \sum_{s=1}^{d} \Big(\lfloor \hat{q}_s \rfloor \cdot \lfloor \hat{p}_s \rfloor + \big|\lfloor \hat{q}_s \rfloor\big| + \big|\lfloor \hat{p}_s \rfloor\big| + 1 \Big)$$

$$= \lim_{e \to \infty} max_{\mathbf{q}} \cdot max_{\mathbf{P}} \sum_{s=1}^{d} \Big((\frac{\lfloor \hat{q}_s \rfloor}{e})(\frac{\lfloor \hat{p}_s \rfloor}{e}) + \frac{||\lfloor \hat{q}_s \rfloor||}{e^2} + \frac{||\lfloor \hat{p}_s \rfloor||}{e^2} + \frac{1}{e^2} \Big)$$

$$= \lim_{e \to \infty} max_{\mathbf{q}} \cdot max_{\mathbf{P}} \Big(\sum_{s=1}^{d} (\frac{q_s}{max_{\mathbf{q}}} \frac{p_s}{max_{\mathbf{P}}} + 0 + 0 + 0) \Big) = \sum_{s=1}^{d} q_s \cdot p_s$$

$$= \mathbf{q}^T \mathbf{p}. \quad \square$$

Theorem 5 implies that the integer based upper bound becomes tighter as e increases. This effect is also demonstrated in Figure 11.

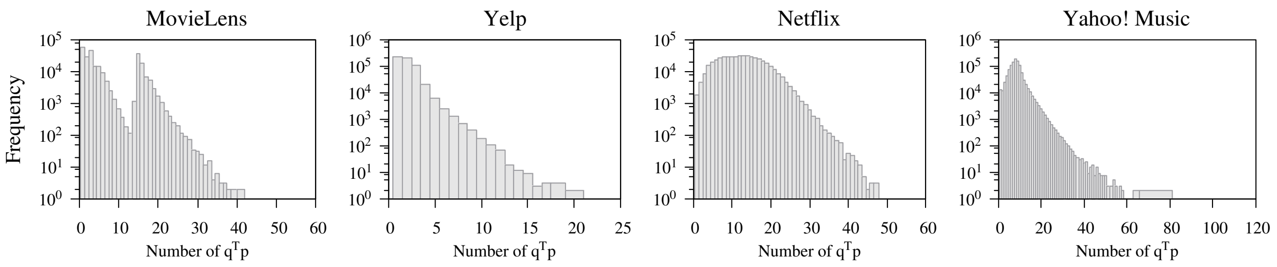

Figure 12: Number of Entire $\mathbf{q}^T\mathbf{p}$ Computations in F-SIR ($k = 1$)

B. ADDITIONAL EXPERIMENTS

Value Distributions. Figure 14 illustrates the histogram of original values in $\mathbf{q}$ and $\mathbf{p}$. The original values are within a small range and close to each other, which supports our argument in Section 4.1 that the value distributions in the original MF matrices are not suitable for applying the integer upper bound.

Number of Entire $\mathbf{q}^T\mathbf{p}$ Computations. Figure 12 shows the distribution of the number of entire $\mathbf{q}^T\mathbf{p}$ computations per query when $k = 1$ and F-SIR is adopted, of which the average numbers have already been shown in Table 3.

Effectiveness of SVD. As explained in Section 3, the desirable effect of SVD transformation is that the query vectors become skewed and that the first few dimensions have the largest maximum absolute values. The ultimate goal is that the partial product $\bar{\mathbf{q}}^{\ell T}\bar{\mathbf{p}}^{\ell}$ becomes large and incremental pruning becomes more effective. Accordingly, we conduct an analysis on the four datasets that shows the effect of the transformation. Specifically, we compute all inner products in $\mathbf{Q}^T\mathbf{P}$ and we average the cumulative IP after having processed each dimension. We repeat this experiment before (i.e., Naive method) and after the SVD transformation (i.e., F-S). Figure 15 shows the results. Observe that before the transformation, the IP is distributed evenly to all dimensions, whereas in F-S the first few dimensions accumulate a large percentage of the IP. We also analyze the value distributions before and after SVD transformation to show the effectiveness of the transformation. Figures 16 and 17 show for each dimension the average absolute scalars in the $\mathbf{q}$ and $\mathbf{p}$ vectors, respectively, before and after SVD transformation. From Figure 16 observe that the distribution of $\mathbf{q}$ values becomes very skewed after the SVD transformation. Besides, the resulting dimensions are naturally ordered by decreasing average absolute value; hence the partial inner products $\mathbf{q}^{\ell T}\mathbf{p}^{\ell}$ using the first dimensions can be used to derive tight bounds for the entire products. In addition, note that the $\mathbf{p}$ values are converted to smaller ones in a narrow range as shown in Figure 17; therefore for an IP which is gradually computed by cumulating the products at each dimension, the fluctuation during the last computations is expected to be small. Figures 18 and 19 show the average skew in the absolute values of the original vectors, in $\mathbf{P}$ and $\mathbf{Q}$, respectively, after reordering their dimensions by decreasing absolute value. For example, two vectors $(-1, 2, -4)$ and $(3, -1, -2)$ will become $(4, 2, 1)$ and $(3, 2, 1)$ respectively after changing their values to absolute ones and sorting. The average vectors shown in the figures have the mean in each dimension (e.g., $(3.5, 2, 1)$ if the original vectors are $(-1, 2, -4)$ and $(3, -1, -2)$). Intuitively, these values show the best ordering we could get for incremental pruning on the original vectors (e.g., by a dynamic reordering of dimensions for each individual query [17, 36]). Note that the value distributions in these average vectors are not as skewed as the distributions we get after our SVD transformation (note that Figure 16 is in log-scale), a fact that justifies the superior performance of FEXIPRO over previous work that also applies incremental pruning (i.e., [36]).

Performance with Different Values of k. Table 7 demonstrates the pruning power of the different approaches in terms of exact IP computations that need to be performed. Table 8 shows the performance of all tested methods on the four datasets, when different values of k are used. From Tables 7 and 8 we can see that FEXIPRO outperforms all other methods for different values of k.

Table 7: Average Number of Entire $\mathbf{q}^T\mathbf{p}$ Computations

	Dataset	BallTree	SS-L	F-S	F-SI	F-SIR
k=2	MovieLens	6994.98	3014.11	263.66	102.24	16.20
	Yelp	27883.71	10114.54	1925.00	105.57	3.63
	Netflix	16790.02	11469.15	11068.29	3160.26	16.98
	Yahoo! Music	109892.50	45125.53	30482.14	6383.33	15.73
k=5	MovieLens	9080.48	3870.12	437.47	163.88	24.63
	Yelp	38061.82	13215.19	3260.06	276.95	9.30
	Netflix	17030.21	12794.66	11753.46	3836.12	37.59
	Yahoo! Music	129947.41	69806.65	43436.31	9793.95	33.58
k=10	MovieLens	8244.51	4997.16	596.26	216.20	31.40
	Yelp	42214.07	19119.88	4482.94	519.19	21.55
	Netflix	17069.63	14918.81	12314.32	4440.52	65.06
	Yahoo! Music	164314.89	110157.22	55390.69	13158.15	62.00
k=50	MovieLens	14257.87	16921.11	2275.53	999.71	154.81
	Yelp	57971.54	25762.11	8462.07	1910.11	110.51
	Netflix	17407.25	15921.78	13838.04	6396.28	246.08
	Yahoo! Music	220110.42	196771.99	94536.23	25467.89	266.71

Performance with Different Values of d. We investigate the impact of d (i.e., the rank of the factorization) on SS-L and FEXIPRO. Figure 20 shows the results on factorized matrices using different values of d (i.e., $d = 10, 50, 80, 100$) and $k = 1$. Observe that the performance gap between the two methods is not sensitive to d, i.e., F-SIR consistently ourperforms SS-L.

Comparsion with PCATree. PCATree [7] is an approximate method for IP retrieval. When $k = 1$, it needs 52.44 (0.16), 260.57 (0.36), 50.32 (0.28) and 3772.80 (0.98) seconds for MovieLens, Yelp, Netflix, Yahoo! Music, respectively. The numbers in brackets are the preprocessing time. Though FEXIPRO is an exact retrieval method, it is much faster than the approximate PCATree approach in three out of the four datasets (it loses marginally by PCATree in Netflix). To measure the accuracy of the top-k results by PCATree, we use root-mean-square error: $RMSE@k = \sqrt{\frac{1}{mk}\sum_{i=1}^{m}\sum_{s=1}^{k}\left(L_{rec}(i,s) - L_{opt}(i,s)\right)^2}$, where $L_{rec}(i,s)$ and $L_{opt}(i,s)$ are the scores (predicted ratings) of the s-th recommended item in the approximated list and the optimal list for user i, respectively. The optimal list can be obtained by any exact method (e.g., Naive or FEXIPRO). Figure 13 shows the $RMSE@k$ of PCATree. The parameters of PCATree can be tuned to achieve a better trade-off between accuracy and speedup. Still, PCATree is an *approximate* method of lower recommendation quality compared to our *exact* FEXIPRO framework.

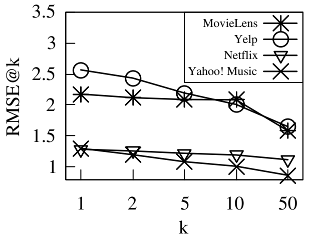

Figure 13: RMSE@k

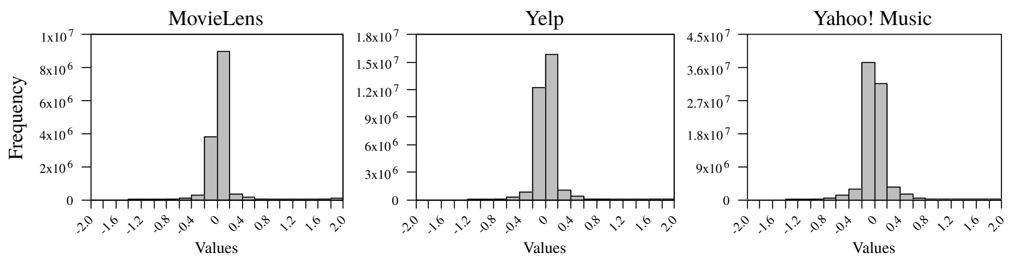

Figure 14: Value Distributions

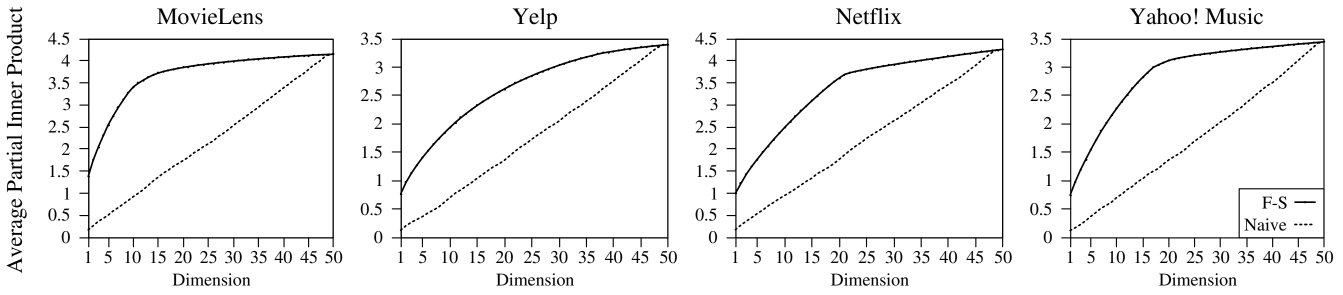

Figure 15: Partial Inner Product Before and After SVD Transformation

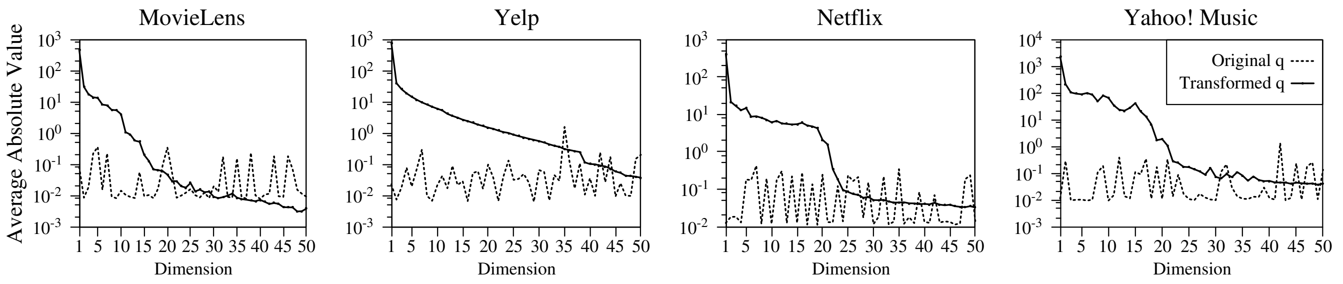

Figure 16: Value Distributions of q Before and After SVD Transformation

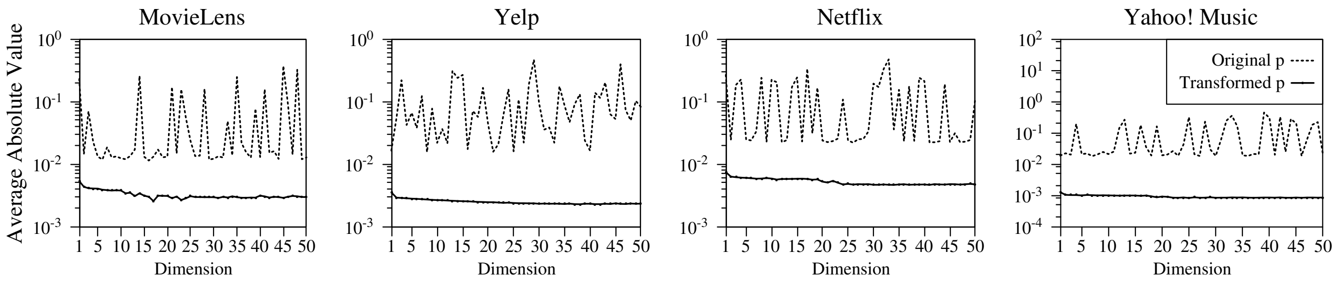

Figure 17: Value Distributions of p Before and After SVD Transformation

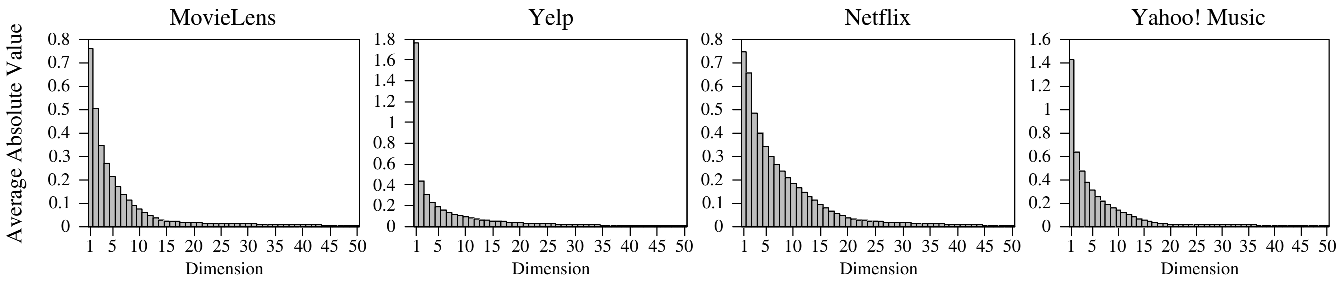

Figure 18: Value Distributions of Reordered q

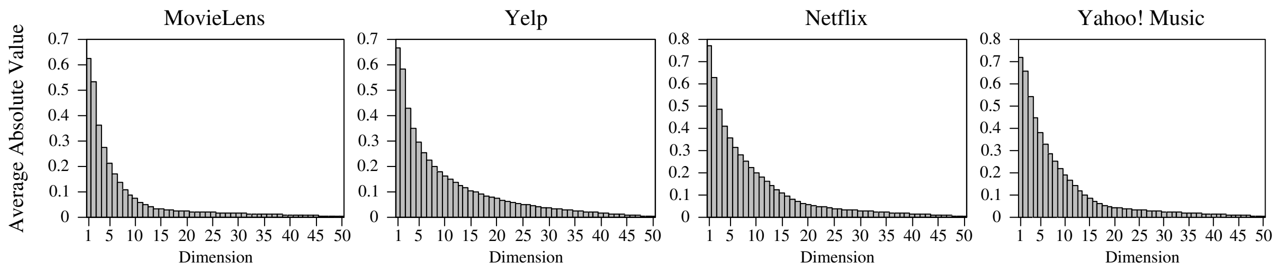

Figure 19: Value Distributions of Reordered p

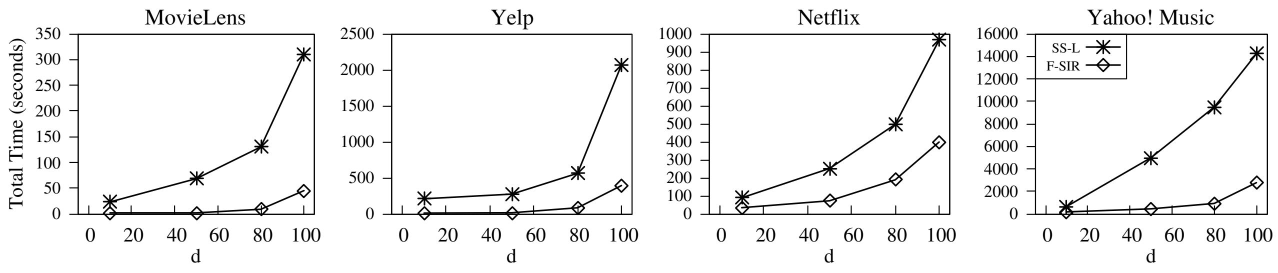

Figure 20: Total Retrieval Times for All Queries (Varying d)

Table 8: Total Retrieval and Preprocessing Times (in seconds) for All Top-k IP Queries

		MovieLens		Yelp		Netflix		Yahoo! Music	
		Retrieve	Preprocess	Retrieve	Preprocess	Retrieve	Preprocess	Retrieve	Preprocess
k=2	Naive	441.91	-	2213.27	-	445.65	-	31504.95	-
	BallTree	267.92	(0.23)	2198.93	(0.55)	878.12	(0.09)	21396.32	(5.10)
	FastMKS	343.25	(0.23)	2745.58	(4.92)	985.00	(0.90)	16571.12	(50.94)
	SS-L	70.69	(0.12)	298.06	(0.29)	267.64	(0.12)	6861.29	(1.92)
	F-S	2.31	(0.31)	39.94	(0.64)	114.31	(0.17)	924.61	(4.87)
	F-I	**1.65**	(0.11)	36.91	(0.23)	154.98	(0.05)	949.91	(1.98)
	F-SI	2.06	(0.29)	33.13	(0.75)	149.11	(0.12)	969.51	(6.74)
	F-SR	2.16	(0.34)	31.06	(0.66)	145.57	(0.17)	899.32	(5.03)
	F-SIR	2.10	(0.44)	**23.20**	(0.89)	**101.12**	(0.25)	**712.21**	(8.23)
k=5	Naive	441.85	-	2204.32	-	443.61	-	30955.50	-
	BallTree	275.44	(0.24)	2984.92	(0.55)	883.32	(0.08)	21940.22	(5.05)
	FastMKS	429.21	(0.32)	3897.32	(4.95)	1049.24	(0.90)	25329.27	(50.95)
	SS-L	72.59	(0.13)	334.05	(0.29)	284.46	(0.12)	8783.39	(1.93)
	F-S	4.29	(0.31)	63.87	(0.65)	266.19	(0.17)	1556.08	(4.89)
	F-I	**2.85**	(0.10)	68.21	(0.23)	168.54	(0.05)	1686.53	(1.89)
	F-SI	3.21	(0.30)	48.70	(0.75)	184.36	(0.13)	1412.43	(6.62)
	F-SR	3.58	(0.31)	56.88	(0.66)	168.46	(0.17)	1317.25	(5.03)
	F-SIR	3.11	(0.44)	**41.21**	(0.88)	**112.78**	(0.24)	**1134.56**	(8.27)
k=10	Naive	440.00	-	2209.33	-	444.81	-	31093.85	-
	BallTree	311.07	(0.23)	3474.89	(0.54)	889.17	(0.09)	23999.29	(5.12)
	FastMKS	471.41	(0.29)	4489.99	(4.92)	1090.32	(0.91)	28867.93	(51.02)
	SS-L	73.35	(0.13)	368.99	(0.30)	299.12	(0.12)	10112.80	(1.95)
	F-S	5.55	(0.34)	88.99	(0.64)	283.24	(0.17)	2004.00	(4.87)
	F-I	4.38	(0.10)	94.88	(0.23)	179.96	(0.05)	2163.86	(1.89)
	F-SI	4.17	(0.30)	70.28	(0.75)	203.40	(0.12)	1858.92	(6.81)
	F-SR	4.67	(0.35)	76.95	(0.65)	175.17	(0.17)	1736.03	(5.04)
	F-SIR	**3.82**	(0.44)	**51.13**	(0.88)	**114.12**	(0.25)	**1429.16**	(8.43)
k=50	Naive	443.24	-	2223.56	-	454.55	-	31189.81	-
	BallTree	413.08	(0.23)	4639.86	(0.52)	908.78	(0.09)	30827.95	(5.22)
	FastMKS	636.15	(0.34)	5625.83	(4.94)	1168.01	(0.89)	37518.79	(51.02)
	SS-L	96.77	(0.14)	479.79	(0.33)	341.49	(0.16)	14132.80	(1.95)
	F-S	23.13	(0.34)	180.89	(0.66)	357.33	(0.17)	3519.14	(4.88)
	F-I	18.09	(0.10)	178.80	(0.23)	225.41	(0.05)	3746.08	(1.88)
	F-SI	16.38	(0.29)	155.33	(0.77)	271.73	(0.12)	3314.83	(6.58)
	F-SR	17.44	(0.31)	148.91	(0.67)	219.11	(0.17)	3039.67	(5.04)
	F-SIR	**14.23**	(0.44)	**121.11**	(0.87)	**127.34**	(0.25)	**2232.19**	(8.31)