Journal of Artificial Intelligence Research 70 (2021) 1441-1479 Submitted 09/2020; published 04/2021

Efficient Retrieval of Matrix Factorization-Based Top-k Recommendations: A Survey of Recent Approaches

Dung D. Le DDLE.2015@SMU.EDU.SG
Hady W. Lauw HADYWLAUW@SMU.EDU.SG
School of Computing and Information Systems
Singapore Management University
80 Stamford Road, Singapore 178902

Abstract

Top-k recommendation seeks to deliver a personalized list of k items to each individual user. An established methodology in the literature based on matrix factorization (MF), which usually represents users and items as vectors in low-dimensional space, is an effective approach to recommender systems, thanks to its superior performance in terms of recommendation quality and scalability. A typical matrix factorization recommender system has two main phases: *preference elicitation* and *recommendation retrieval*. The former analyzes user-generated data to learn user preferences and item characteristics in the form of latent feature vectors, whereas the latter ranks the candidate items based on the learnt vectors and returns the top-k items from the ranked list. For *preference elicitation*, there have been numerous works to build accurate MF-based recommendation algorithms that can learn from large datasets. However, for the *recommendation retrieval* phase, naively scanning a large number of items to identify the few most relevant ones may inhibit truly real-time applications. In this work, we survey recent advances and state-of-the-art approaches in the literature that enable *fast* and *accurate* **retrieval** for MF-based personalized recommendations. Also, we include analytical discussions of approaches along different dimensions to provide the readers with a more comprehensive understanding of the surveyed works.

1. Introduction

Recommender system is a staple feature in e-applications, be it commerce (e.g., Amazon), entertainment (e.g., Netflix), or social media (e.g., Facebook). This is driven by sheer necessity, as the exploding number of choices implores product and service providers to narrow the myriad of possibilities down to a manageable number k to be presented to each customer. To cater to customers' idiosyncratic preferences, such top-k recommendation lists should be customized. Personalization is made possible via recommendation algorithms that learn from users' historical feedback, which may be explicit (e.g., ratings) (Salakhutdinov & Mnih, 2008) or implicit (e.g., click behaviors) (Rendle et al., 2009). Example techniques include collaborative filtering (Sarwar et al., 2001), user-item graph models (Aggarwal et al., 1999), regression based models (Vucetic & Obradovic, 2005), deep learning-based models (Zhang et al., 2019).

A preponderance of recommendation algorithms in the literature are based on matrix factorization (MF) techniques (Koren et al., 2009), which become prevalent after the Netflix competition (2006) when Netflix announced a prize money of $1 million to those who would improve its root-mean-square performance by at least 10%. The core idea of MF-based recommendation algorithms is to learn low-dimensional representations of users and items from either explicit user-item associations such as user-item ratings or implicit feedback such as playcounts and dwell time. The association (or preference) of a user to an item is modelled by the inner product between the corresponding vectors. MF-based methods have improved upon the efficiency of other collaborative filtering approaches such as neighborhood-based models. By representing users and items as low-dimensional vectors, MF-based methods can avoid the expensive similarity weight computation (user-to-user or item-to-item) in high-dimensional space of neighborhood-based collaborative filtering models.

1.1 Two Phases of MF Recommender Systems

While focusing on the prevalent class of MF-based recommendation algorithms, we should be cognizant of the two distinct phases that contribute towards their efficacious deployment (Bachrach et al., 2014; Li et al., 2017). Figure 1 illustrates these two phases for a system of m users and n items.

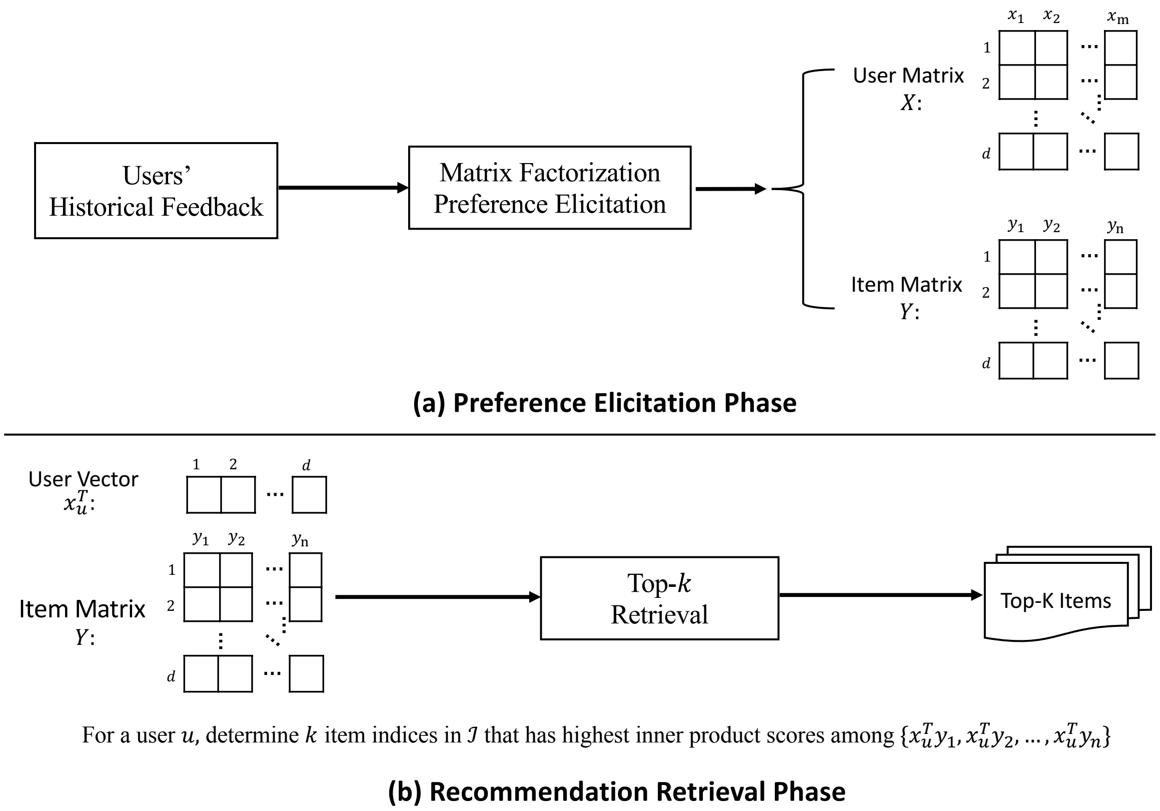
Users' Historical Feedback
Matrix Factorization Preference Elicitation
User Matrix X:
x_1 x_2 x_m
1 2 d
Item Matrix Y:
y_1 y_2 y_n
(a) Preference Elicitation Phase
User Vector x_u^T:
Item Matrix Y:
Top-k Retrieval
Top-K Items
For a user u, determine k item indices in $\mathcal{I}$ that has highest inner product scores among $\{x_u^T y_1, x_u^T y_2, \ldots, x_u^T y_n\}$
(b) Recommendation Retrieval Phase

Figure 1: Two Phases of Matrix Factorization Recommendation
(The focus of this survey is on the efficiency of the recommendation retrieval phase).

- One phase, which could be offline, is *preference elicitation* (Figure 1(a)). This phase analyzes users' historical feedback (e.g., ratings, click behaviors, etc.), which could express preference signals of users. The learnt insight about user preferences will be modelled for prediction. Specifically, given a collective interaction rating matrix $\mathcal{R} \in \mathbb{R}^{m \times n}$, where m and n denote the number of users and items respectively, the element r_{ui} denotes the rating given by user u to item i. The observed elements constitute a small subset Ω of the rating matrix $\mathcal{R}$. MF-based algorithms work by decomposing the partially observed user-item interaction matrix $\mathcal{R}$

into the product of two rectangular matrices X and Y. The former matrix X can be referred to as the user preference matrix, where each column of X is a latent vector $x_u \in \mathbb{R}^d$ for a user u, with d being the vector dimensionality. The latter matrix Y is the item matrix where each column of Y is a latent vector $y_i \in \mathbb{R}^d$ for an item i.

Generally, MF-based models can be categorized into either rating-based or ranking-based approaches. *Rating-based MF* models try to fit the values of observed ratings and the learnt user and item vectors are used to predict the missing entries in the matrix $\mathcal{R}$. *Ranking-based MF* methods learn from users' ordinal preferences to derive a ranking of the non-observed items for recommendation. In both cases, the degree of preference of u for i is modelled as the inner product ${x_u}^T y_i$. The emphasis of this phase is primarily the *accuracy* in identifying a user's preferred items. For reference, we list the common notations used throughout the paper in Table 1.

- The next phase, commonly online, is *recommendation retrieval* (Figure 1(b)). Upon the appearance of a target user u, we immediately construct a recommendation list of k items for u. In the context of MF models, given the user query vector x_u, the top-k items with the highest inner product scores ${x_u}^T y_i, \forall 1 \leq i \leq n$ should be quickly identified for recommendation. In the scenarios where there are bias terms in modelling user-item interaction, i.e., $r_{ui} \propto {x_u}^T y_i + b_u + b_i$, we can convert each user vector x_u to $\tilde{x}_u = [x_u, b_u, 1]$ and each item vector y_i to $\tilde{y}_i = [y_i, 1, b_i]$. The problem now becomes identifying the top-k items with the highest inner product scores $\tilde{x}_u^T \tilde{y}_i, \forall 1 \leq i \leq n$. For ease of readability, heretofore we refer to $\tilde{x}_u$ and $\tilde{y}_i$ as x_u and y_i respectively. The real-time nature of the task is necessitated by the response time expected by end users. Thus, the emphases in this phase encompass both *accuracy* and *retrieval efficiency*.

  Ideally, all item vectors in $\{y_1, y_2, \ldots, y_n\}$ are examined and sorted according to their inner product scores to vector x_u, and the top-k items in the rank list will be returned. However, the modern catalog of items is often too large to allow an exhaustive computation of all the inner products within a budgeted retrieval time. Therefore, having a faster alternative for this process of recommendation retrieval is critical and desirable.

The concern in traditional recommendation literature frequently stops at the preference elicitation phase, preoccupied with accuracy as the sole arbiter in any comparison of methods. Towards a more holistic discourse on the concerns of recommender systems, in this survey we cover approaches to managing the dual objectives of accuracy and retrieval efficiency of the top-k recommendations (Figure 1(b)). Note that, some of the surveyed methods are aimed at recommendation specifically, whereas some are proposed for more general retrieval problems. For the latter, we discuss these methods in the context of MF-based recommendation retrieval specifically.

1.2 The Prohibitive Complexity of Linear Scanning

Top-k retrieval of MF-based recommendation is equivalent to the task of ranking every item i ($\forall 1 \leq i \leq n$) for each user u with respect to the preference score ${x_u}^T y_i$. This is reducible to the fundamental problem of finding the item with the highest inner product (Equation 1).

Symbol	Description
$\mathcal{U}$	collection of all users
$\mathcal{I}$	collection of all items
$\mathcal{R}$	user-item ratings matrix
Ω	set of observed ratings
u	a specific user
i	a specific item
r_{ui}	rating of user u for item i
x_u	latent vector representation for user u
y_i	latent vector representation for item i
d	latent space dimension
k	the number of recommendations
m	the number of users
n	the number of items

Table 1: List of Notations.

Problem 1 (Maximum Inner Product Search - MIPS) *For each vector $x_u, u \in \mathcal{U}$, determine the item $i \in \mathcal{I}$ such that:*

$$i = \arg\max_{1 \leq i \leq n} x_u^T y_i \tag{1}$$

Problem 1, known as *Maximum Inner Product Search* or MIPS, arises naturally in many large scale tasks (Shrivastava & Li, 2014; Auvolat & Vincent, 2015), when inner product-based comparisons are done between the embedding vector of a query and many candidate objects' vectors.

The straightforward solution for Problem 1 is to *compute the user preference scores ${x_u}^T y_i$ of all n items and rank these scores descendingly*. Computationally, per-user, the cost of such naive exhaustive approach is $\mathcal{O}(n \times d)$, which scales linearly with the number of items n and the number of latent factors d. Given the current scale of the number of items and the desired personalization purpose, achieving real-time performance for each user by examining all possible items, repeatedly for the many users of a large-scale system, is not practical. For instance, recent estimates[1] put the number of unique products at Amazon.com at hundreds of millions. For digital artefacts, the scale could be even larger. The number of photos on Facebook[2] or Flickr[3] are estimated to number in bi1lions. Therefore, an alternative solution to exhaustive search over all items is also desirable.

On the other hand, pre-computing and then storing the personalized recommendations for all m users would require a storage cost of $\mathcal{O}(m \times k)$. This is impractical when users and items number in the millions. It also has a calcifying effect, making the system less malleable to the dynamically changing preference of users (e.g., user vector is updated online according to their behaviors) and adoptions of new items. A more promising direction would be retrieval strategies with a smaller memory footprint than full pre-computation, yet with greater computational efficiency than linear scanning.

1. https://bit.ly/2n6kadm, http://bit.ly/2g7wVUI
2. https://bit.ly/3uazjey
3. http://bit.ly/1vlm6zR

1.3 Contributions and Organization

In recent years there emerge such a class of recommendation algorithms as well as search strategies to optimize the recommendation retrieval efficiency of MF-based models, without adversely affecting recommendation accuracy. In this paper, we survey recent state-of-the-art approaches in the literature that enable fast and accurate retrieval for MF-based personalized recommendations.

Our focus on recommendation retrieval renders this work complementary to the existing recommender systems surveys on current generation of preference elicitation methods based on matrix completion (Adomavicius & Tuzhilin, 2005; Das et al., 2017; Ramlatchan et al., 2018), on social recommendation (Yang et al., 2014), on transfer learning for recommendation (Pan, 2016), on context-aware recommendation (Abdi et al., 2018), on explainable recommendation (Zhang & Chen, 2020), and on deep learning based methods (Zhang et al., 2019), etc. This survey is expected to be a useful resource for those interested in recommender systems, preference analytics, and information retrieval to understand the current state-of-the-art in efficient retrieval for recommender systems.

Contributions. To our best knowledge, this is the first written work to systematically document recent advances for efficient retrieval of MF-based personalized recommendations, which is our *first* contribution. As our *second* contribution, we develop a novel taxonomization of the surveyed approaches, overviewed in Section 2. This taxonomy categorizes related works in terms of their pertinent concepts or strategies that yield the improvements in retrieval efficiency. As our *third* contribution, we discuss issues surrounding recommendation retrieval, such as the trade-off between accuracy, retrieval efficiency, and storage cost, as well as the theoretical complexities of the different processing stages.

Organization. Section 2 describes an overview of the recommendation retrieval pipeline with two main steps: *candidate generation* and *candidate ranking*, which are then subsequently elaborated in detail in Section 3 and Section 4 respectively. In Section 5, we conclude the survey and discuss several open research questions that would be interesting to explore further. For completeness, we provide the necessary background knowledge of several popular efficient retrieval data structures in Appendix A.

2. Survey Overview

Top-k matrix factorization recommendation retrieval can be seen as a similarity search problem. Indeed, when a user u goes to the website, the system will request for recommendations given the user vector x_u as the query. The website is expected to return those item vectors with highest inner product scores to the query vector x_u.

As illustrated in Figure 2, retrieving a top-k recommendation list for each user involves the following two requisite steps[4]:

1. **Candidate Generation** Given a user vector x_u as query and item vectors $\{y_1, y_2, \ldots, y_n\}$, an efficient filtering procedure determines a candidate set C_u for recommendation. The objective of this step is typically to remove items which are potentially not in the top-k recommendation

4. Note that these two steps are also applicable to other techniques for recommender systems (Covington et al., 2016). We focus on matrix factorization collaborative filtering models in this work and will expand the coverage to other techniques in future work.

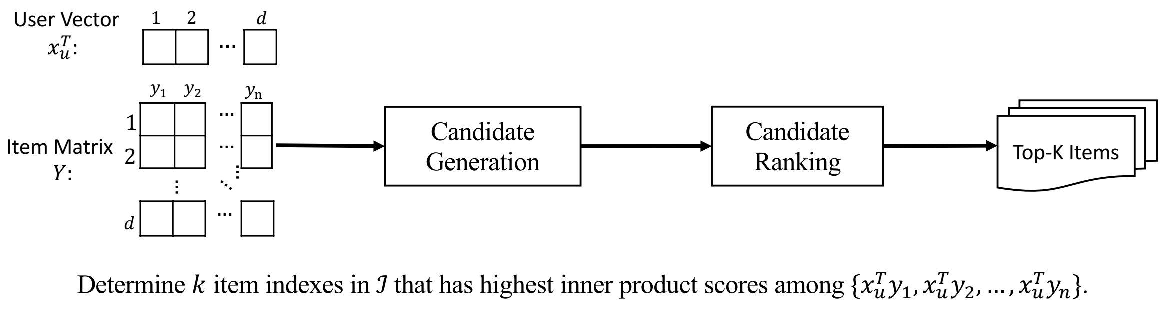
User Vector
x_u^T:
1 2 ... d
y_1 y_2 ... y_n
Item Matrix
Y:
1
2
d
Candidate Generation
Candidate Ranking
Top-K Items
Determine k item indexes in $\mathcal{I}$ that has highest inner product scores among $\{x_u^T y_1, x_u^T y_2, \ldots, x_u^T y_n\}$.

Figure 2: Top-k MF Recommendation Retrieval Pipeline

for user u. This effectively reduces the number of items investigated in the second step, i.e., $|C_u| < n$.

2. **Candidate Ranking** For the candidate items in C_u, we compute their inner products against the query vector x_u, and sort them accordingly to identify the top-k recommendations. This step can be referred to as the *re-ranking* step, which aims to get better ranking among the candidates, before arriving at the final top-k items for recommendation. The computational complexity of this process is $O\left(|C_u| \times d + |C_u| \times \log k\right)$.

For the exhaustive search approach (linear scanning), Step 1 essentially means doing nothing, passing all n items as candidates, i.e., $|C_u| = n$ to the ranking step. Step 2 involves $O\left(n \times d\right)$ operations for computing n inner product scores $\{x_u^T y_1, x_u^T y_2, \ldots, x_u^T y_n\}$ and $O\left(n \times \log k\right)$ operations for ranking these scores and maintain the top-k as we process the candidates.

As majority of items are irrelevant to the users, the two aforementioned steps are usually employed to achieve high efficiency in personalized recommendation (Covington et al., 2016; Kang & McAuley, 2019). On one hand, the *candidate generation* component relies on heuristics or retrieval-efficient structures to recall a small subset of relevant items, to reduce the number of similarity evaluations in the ranking step (Guo et al., 2020). On the other hand, the *candidate ranking* component applies fine-grained re-ranking methods to items produced by the *candidate generation* step to obtain the final top-k. In this survey, we assume that both components are performed using the low-dimensional vectors produced by the *preference elicitation* phase only.

Any effort to optimize the retrieval efficiency would have to improve the running times of either or both of the steps outlined above. Therefore, we use these steps as the primary axis for taxonomizing the works, defining two categories of approaches: *efficient candidate generation* and *efficient candidate ranking* (as shown in Figure 2), in which the former aims at reducing the item space and the latter aims at reducing the complexity of similarity computation in the d-dimensional vector space.

2.1 Efficient Candidate Generation

The idea for any efficient candidate generation method is to speed up the inference time of top-k recommendations with high precision as compared to exhaustive linear scanning. Figure 3 organizes works that seek to quickly discard potentially irrelevant items, resulting in a candidate set C_u in which $|C_u|$ is substantially smaller than n (however, usually larger than k). We define two lines of such strategies under this category as follows:

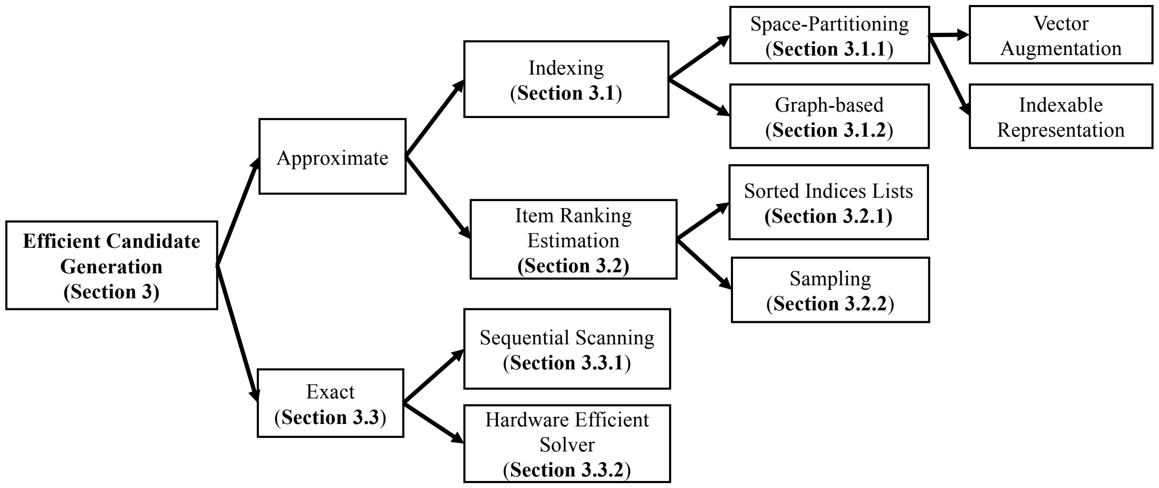
Efficient Candidate Generation (Section 3)
Approximate
Exact (Section 3.3)
Indexing (Section 3.1)
Item Ranking Estimation (Section 3.2)
Sequential Scanning (Section 3.3.1)
Hardware Efficient Solver (Section 3.3.2)
Space-Partitioning (Section 3.1.1)
Graph-based (Section 3.1.2)
Sorted Indices Lists (Section 3.2.1)
Sampling (Section 3.2.2)
Vector Augmentation
Indexable Representation

Figure 3: Approaches for Efficient Candidate Generation

- We emphasize on the class of *approximate* methods, which are applicable in scenarios where scalability is desirable at an acceptable cost of slightly different top-k to linear scanning. The underlying idea of approximate methods is to **trade off** the cost of slightly lower recall from potentially missing out on the false negatives, for the benefit of faster retrieval through smaller candidate sets. We further categorize these approximate candidate generation methods into two classes: using *indexing* structures (Section 3.1) or *item ranking estimation* methods (Section 3.2).

  - **Indexing** refers to techniques that pre-process item vectors for efficient similarity search later. This pre-processing is query-independent and only involves the item vectors. We further categorize this approach to *graph-based* and *space-partitioning* index structures. The former constructs a similarity graph where each node represents an item and the edges connect highly similar items. At the querying step, a greedy search algorithm traverses the constructed graph to arrive at the most similar nodes (i.e., items) to the query vector x_u. The latter splits the item vectors into neighborhood regions, each of which contains highly similar items. At the querying step, only items in the region that contains the query vector or its neighboring regions are potential recommendation candidates. For the *space-partitioning* approach, there is a further consideration on whether one should perform a post-learning *vector augmentation* step or learn an *indexable representation* preference elicitation model to produce "indexing-friendly" vectors that can achieve high retrieval accuracy. We provide a more detailed review of these structures in Appendix A.

  - **Item Ranking Estimation** refers to the class of methods that *estimate the ranking of inner product scores* $\{x_u^T y_i | \forall i, 1 \leq i \leq n\}$ without computing these inner products. The top items in the estimated ranking will be selected as candidates for recommendations. *Sorted Indices Lists* assumes that $x_u^T y_i > x_u^T y_j \iff \max\limits_{1 \leq l \leq d} x_u^{(l)} y_i^{(l)} > \max\limits_{1 \leq l \leq d} x_u^{(l)} y_j^{(l)}$ and uses this assumption to generate the candidate set by searching for top elements in the matrix $Z = \left[x_u^{(l)} y_i^{(l)} \right], \forall i, 1 \leq i \leq n; \forall l, 1 \leq l \leq d$. *Sampling* methods propose sampling strategies in which each item $i \in \mathcal{I}$ is associated with a random variable score whose probability is proportional to $x_u^T y_i$. The sampling process results in an array of item scores $[c_1, c_2, \dots, c_n]$ (c_i is the score for item i) that is correlated to the ranking of the inner product scores $\left[x_u^T y_1, x_u^T y_2, \dots, x_u^T y_n\right]$ and the items with highest scores will be the candidates. We further consider and review different sampling strategies in Section 3.2.2.

- In addition to approximate methods, there are also a class of efficient methods for *exact* top-k retrieval, which attempts to return an identical top-k to linear scanning. These methods are most suitable for the low or moderate data dimension, and if perfect recall is indeed critical to the applications. However, their performance will degrade rapidly as the number of features d increases (Li et al., 2017; Shrivastava & Li, 2014). For completeness, we also include these methods in the survey. Particularly, we describe two classes of exact top-k recommendation retrieval methods, namely *sequential scanning* and *hardware efficient solver* in Section 3.3.

Section 3 describes these efficient candidate generation strategies in detail along with comprehensive qualitative analyses and comparisons.

2.2 Efficient Candidate Ranking

Figure 4 organizes works that attempt to improve the efficiency of candidate ranking step by reducing the cost of computing $|C_u|$ inner product scores of the candidate vectors in C_u to the query vector x_u, which operate in the $d-$dimensional real-valued latent feature space. Efficient candidate ranking can be achieved either via *discrete representation*, which formulates user-item preference score as a function that is more computationally efficient such as *Hamming distance* in the binary space (instead of real-valued space) or via *quantization* techniques, which approximate the inner product computations in the d dimensional space.

- For recommendation retrieval, *discrete representation* (Section 4.1) reduces the computational cost of operating on real-valued latent vectors by representing users and items as binary codes in the Hamming space. The inner product score computation can be converted to computing the Hamming distance between user and item binary vectors, requiring efficient XOR operations (Zhang et al., 2016). That speeds up the generation of recommendations, even if we have to exhaustively scan over all items. Depending on the discretization strategies, we can categorize these methods into either *quantization-based* discretization, which first learns a real-valued vectors and then rounds up the closest solution in the Hamming space, or *optimization-based discretization*, which learns the user and item binary codes directly from the user-item interactions observations.

- *Quantization* is another approach for efficient maximum inner product search (Section 4.2), in which each $d-$dimensional vector is expressed through a set of representative vectors called the *codebooks*. The inner product in the original d-dimensional space is approximated as the sum of inner products of some of these codebooks vectors. As the number of vectors in the codebooks is typically significantly smaller than the number of items n, vector quantization methods reduce the cost of computing inner products between original vectors. We further categorize this approach into *vector quantization*, which performs clustering in the $d-$ dimensional space, and *product quantization*, which decomposes the $d-$dimensional space into the product of M subspaces with dimension d' and performs vector quantization for each subspace separately.

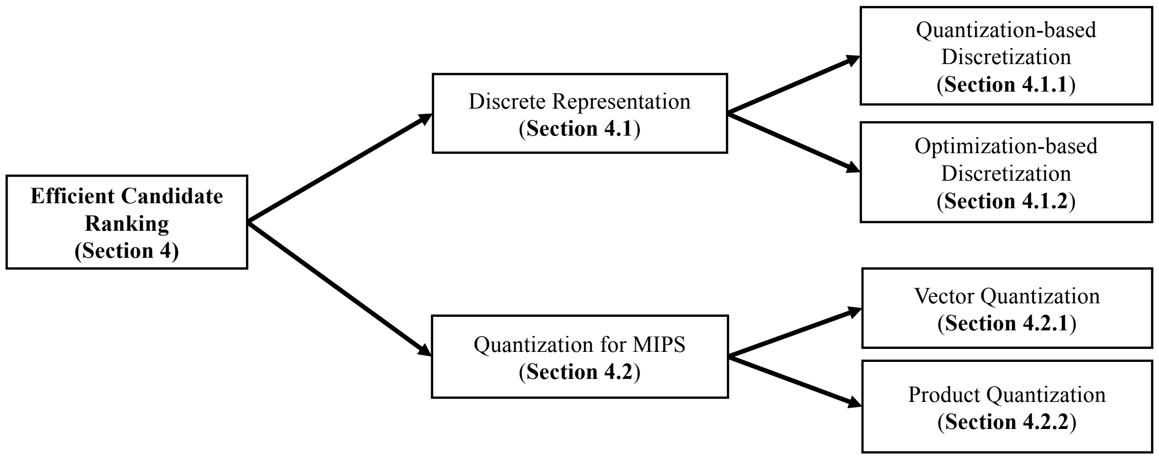
Efficient Candidate Ranking (Section 4)
Discrete Representation (Section 4.1)
Quantization for MIPS (Section 4.2)
Quantization-based Discretization (Section 4.1.1)
Optimization-based Discretization (Section 4.1.2)
Vector Quantization (Section 4.2.1)
Product Quantization (Section 4.2.2)

Figure 4: Approaches for Efficient Candidate Ranking

We will elaborate further on the complexities and qualitative comparisons of the methods under these two ideas in Section 4.

3. Efficient Candidate Generation

We first discourse on the class of approximate methods for efficient candidate generation, comprising retrieval-efficient structures and sampling methods. Thereafter, we discuss a couple of pertinent issues such as the accuracy versus retrieval efficiency trade-off (Section 3.4.1) and budgeted queries (Section 3.4.2).

3.1 Efficient Candidate Generation via Indexing

This approach performs query-independent preprocessing of item vectors and stores them in an *indexing* data structure that supports efficient candidate filtering upon query, probably in sub-linear time with respect to the number of items n. There are several possible structures, including space-partitioning methods such as Locality Sensitive Hashing (Indyk & Motwani, 1998), spatial trees such as KD-tree (Bentley, 1975) and graph-based methods such as Hierarchical Navigable Small World - HNSW (Malkov & Yashunin, 2019) (see Appendix A for a background on indexing structures).

The pre-processing is query-independent and only involves item vectors. At the retrieval step, the built structure can quickly retrieve the candidates for each user u, probably in sub-linear time with respect to the number of items n. By indexing item latent vectors, we can quickly retrieve a small candidate set for the "most relevant" items to the user query vector, as illustrated in Figure 5. Specifically, all indexing strategies adopt the same method to achieve sub-linear searching time: given a query, it reduces the search space by automatically discarding a large amount of potentially irrelevant data points. Figure 5 depicts two phases of a top-k recommender system with the aid of indexing structures, consisting two main steps:

1. *Index Construction:* indexing item vectors $\{y_i\}_{i \in \mathcal{I}}$ obtained from the *preference elicitation phase*.

2. *Top-k Retrieval:* given the user vector x_u as query, using the built index to get the candidates for top-k items, ranking the candidates, and returning the final top-k.

As opposed to exhaustive search, indexing offers a speed advantage, at the cost of some storage. As opposed to full pre-computation that scales with the number of items and users, indexing is significantly more storage-efficient (only items need to be indexed), while offering greater flexibility for k, the size of recommendation list to be retrieved. Popular indexing structures include locality-sensitive hashing (LSH) (Shrivastava & Li, 2014) hash tables, spatial indexing (Bentley, 1975), and inverted index (Bhowmik et al., 2016) (see Appendix A). We categorize these structures under *space-partitioning* approach. Another category is *graph-based*, based on the recent advances on using similarity graph exploration for nearest neighbors search. In the following, we describe each category in detail.

3.1.1 SPACE-PARTITIONING INDEXING

Here, we focus on space-partitioning indexing structures for efficient candidate generation. The principal idea is to build a partition of $\mathcal{R}^d$ into many neighborhood regions and split the data accordingly.

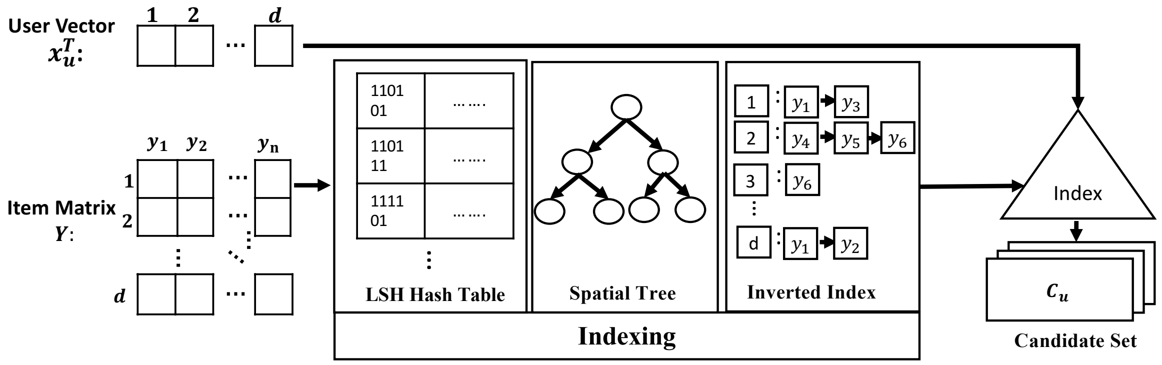

Figure 5: An Illustration of Candidate Screening with Indexing

Early attempts towards having efficient MF-based retrieval include those by Koenigstein et al. (2012), which constructs a ball tree with item vectors and employs a branch and bound algorithm and Ram and Gray (2012), which proposes a dual-tree based search using cone trees to handle many queries simultaneously. However, these solutions partition the data space based on Euclidean distance, which could produce different top-k recommendations from the inner product. The reason is because inner product does not form a proper metric space: for $\forall x \in \mathcal{R}^d$, you can always find $y_1, y_2 \in \mathcal{R}^d$ so that $x^T y_1 < x^T x < x^T y_2$. A point might not be its own nearest neighbor. This violates the triangle inequality and invalidates some common approaches for approximate nearest neighbors search. This means that the inner product kernel is not compatible with the operations of a spatial tree index relying on Euclidean distance, or an inverted index relying on cosine similarity. To resolve this issue, there are two following possible solutions.

1. **Vector Augmentation** The *first* solution is to reduce MIPS problem to either *Nearest Neighbor Search* (NNS) problem defined as:

$$i = \arg \min_{1 \le i \le n} ||x_u - y_i|| \tag{2}$$

or *Maximum Cosine Similarity Search* (MCSS) problem:

$$i = \arg \max_{1 \le i \le n} \frac{x_u^T y_i}{||x_u||.||y_i||} \tag{3}$$

The three problems are equivalent if all item vectors are of the same length. This can be solved via post-learning transformations applied to user and item vectors. This typically adds extra dimensions to user vectors $\{x_u\}_{u \in \mathcal{U}}$ and item vectors $\{y_i\}_{i \in \mathcal{I}}$ so that solving MIPS in the original space is equivalent to solving NNS/MCSS in the transformed space. After the reduction, there are several indexing solutions for the transformed NNS/MCSS such as locality sensitive hashing (LSH) (Shrivastava & Li, 2014, 2015; Neyshabur & Srebro, 2015), PCA-tree (Bachrach et al., 2014).

There are many choices for the transformation itself. For example, (Shrivastava & Li, 2014, 2015) augment the vectors to $(d+m)$-dimensional space. Later, (Neyshabur & Srebro, 2015; Bachrach et al., 2014) extend the output latent vectors by one dimension to equalize the magnitude of item vectors:

$$x_u \to \left[\frac{x_u}{||x_u||}, 0\right] \forall u \in \mathcal{U}, \tag{4}$$

$$y_i \to \left[\frac{y_i}{\max_{i \in \mathcal{I}} ||y_i||}, \sqrt{1 - ||\frac{y_i}{\max_{i \in \mathcal{I}} ||y_i||}||^2}\right] \forall i \in \mathcal{I}, \tag{5}$$

Extensions include (Huang et al., 2018) that minimizes the distortion error in reducing MIPS to NNS via Asymmetric LSH scheme and Query Normalized First transformation and (Keivani et al., 2018) that uses randomized partition trees instead of LSH for a better theoretical guarantee on choosing the best MIPS-to-NNS/MCSS reduction strategies. (Yan et al., 2018) points out that (Neyshabur & Srebro, 2015) suffers from long tails in the 2-norm distribution of real-datasets and proposes to partition a dataset to sub-datasets and build a hash index for each sub-dataset independently.

Also under this category, (Bhowmik et al., 2016) recently presents a post-MF-learning processing with sparsity mapping scheme that derives sparse representation for each user and item from their respective dense real-valued latent vectors. Two close points are mapped to sparse vectors with similar sparsity pattern (i.e., significant overlap of non-zero indices). Two distant points are mapped to vectors with different sparsity patterns. It then uses inverted indexing (See Appendix A) with the resulting sparse vectors to generate candidates for top-K recommendation retrieval.

2. **Indexable Representations** The *second* solution is to avoid the need for post learning vector augmentation by designing recommendation algorithms whose latent output vectors can be immediately sublinearly searchable using indexing data structures.

- **Metric learning** takes as input distances (or their ordinal relationships) and outputs low-dimensional latent coordinates for each point that would preserve the input as much as possible. Because they operate in the Euclidean space, the coordinates support NNS using geometric index structures such as spatial trees. For instance, CFEE (Khoshneshin & Street, 2010) fits a rating $\hat{r}_{ui}$ by user u on item i in terms of the squared Euclidean distance between x_u and y_i, i.e., $\hat{r}_{ui} = \mu + b_u + b_i - ||x_u - y_i||^2$. On the other hand, COE (Le & Lauw, 2016) and CML (Hsieh et al., 2017) seek to ensure that an item i liked by a user u would be placed closer to the user on the Euclidean representation than a less preferred item j, implying $||x_u - y_i|| < ||x_u - y_j||$. As these methods use Euclidean distance to model the user preference over items, the retrieval of top-K recommendations becomes top-K nearest neighbor search (NNS) with Euclidean distance.

- One insight towards achieving geometric compatibility is to **desensitize the effect of vector magnitudes**. The key reason behind the incompatibility between inner product search that matrix factorization relies on, and the aforesaid index structures is how a user u's degree of preference for an item i, expressed as the inner product $x_u{}^T y_i$, is sensitive to the respective magnitude of the latent vectors $||x_u||, ||y_i||$. There are several approaches in this direction. One is IPMF (Fraccaro et al., 2016), which extends the Bayesian Probabilistic Matrix Factorization (Salakhutdinov & Mnih, 2008), by making the item latent vectors natively of fixed length. *Indexable Bayesian Personalized Ranking* or IBPR (Le & Lauw, 2017), on the other hand, proposes the use of angular distance kernel, evaluated as the `arccos` of the inner product between the normalized vectors, to model pairwise ordinal preferences. Both IPMF and IBPR produce output vectors in which MIPS is equivalent to NNS and MCSS. Recently, (Le & Lauw, 2020) proposes SRPR, an indexable method that learns the user and item vectors that are robust to the stochasticity of randomly generated LSH hash functions. As as result, SRPR shows better performances post-LSH-indexing for top-k recommendation compared to IPMF and IBPR, when LSH is the designated indexing structure for recommendation retrieval.

3.1.2 GRAPH-BASED INDEXING

The idea of using similarity graph, i.e., where each vertex is connected to the most similar vertices for nearest neighbor search problem has been well studied in literature (see Appendix A for a background). Some of the current approaches attempt to use this technique to solve the maximum inner product search problem efficiently (Equation 1).

(Morozov & Babenko, 2018) introduces the concept of IP-*Delaunay* graph, which is the smallest graph that can guarantee the return of exact solutions for maximum inner product search by greedy seach algorithm. Specifically, the proposed framework *ip-NSW* is based on Navigable Small World (NSW) approach to approximate the IP-*Delaunay* graph. Based on the assumption that similar item vectors may constitute relevant results to the same query, *ip-NSW* (Morozov & Babenko, 2018) proposes to solve MIPS with the use of similarity graph based on the inner product between item vectors.

1. Before observing any query, ip-NSW constructs a s-Delaunay similarity graph based on the inner product between item vectors. At each step, ip-NSW adds the next item i node to the

graph. Vertex i is connected by directed edges to H vertices, corresponding to most similar item vectors that are already added to the graph.

2. At the query phase, greedy walks on this graph are employed to efficiently determine the candidates for recommendation. At this stage, ip-NSW maintains a priority queue of size Q of neighbors that should be visited by the search process. Both H and Q determine the balance/trade-off between the run-time and search accuracy.

Recently, (Liu et al., 2020) proposes *ip-NSW+*, which further improves *ip-NSW* by introducing an angular similarity graph, in addition to the inner product similarity graph of *ip-NSW*. The search process first starts on the angular graph to find the angular neighbors for the query. These angular neighbors are used to initialize the candidate pool when searching on the inner product sinilarity graph of *ip-NSW*.

Another recent work (Zhou et al., 2019), based on the Mobius transformation on the item vectors, connects graph-based indices for maximum inner product search and approximate nearest neighbor search. (Zhou et al., 2019) also points out that there is an isomorphism between the IP-*Delaunay* graph before the transformation and a subgraph of the Delaunay triangulation with respect to the Euclidean distance between the transformed data points. Based on this observation, the authors propose to approximate the IP-*Delaunay* graph as follows:

1. Applying Mobius transformation to each item vector y_i:

$$y_i \mapsto y_i' = \frac{y_i}{||y_i||^2}$$

   Let $\tilde{Y} = \{y_i', \forall i = 1, n\} \cup \{0\}$ be the transformed set (0 is the origin). Constructing approximate L_2-*Delaunay* graph for $\tilde{Y}$. Let N_0 be the set of neighbors of 0 on the constructed graph. Then remove 0 and its edges from the graph, and replace y_i' by the original y_i.

2. At the querying step, let N_0 be the initial vertices of the greedy search algorithms described in Figure 10.

Since the graph is built for L_2- distance metric, it can preserve the advantageous features of metric similarity graph. Also, the transformation provides good starting vertices for the greedy search process. Thus, the performance of the proposed method significantly improves that of *ip-NSW* (Morozov & Babenko, 2018).

Recent work on learning to route on similarity graphs (Baranchuk et al., 2019) and learning to partition space for nearest neighbor search (Dong et al., 2020) could open a new class of methods to improve even the greedy search algorithm on the similarity graph, which is used extensively for most of graph-based indexing methods.

3.1.3 GRAPH-BASED VS. SPACE-PARTITIONING INDEXING

Compared to space-partitioning methods, searching for top-k recommendations with graph-based indexing may start from a distant point to the query, but they usually approach closer to the query quickly. The reason is that the similarity graphs can express the neighbor relationships better than space-partitioning indexing methods, which tend to check more nearby areas to achieve the same level of accuracy (Fu et al., 2019). Another interesting perspective about MF-based recommendation retrieval is that the queries (or user vectors) are usually having clustering structures (Jiang et al., 2020) (i.e., users can be clustered to groups that share the same preferences), which is more suitable for data-dependent methods such as graph-based indexes than data-independent methods such as LSH tables or spatial trees.

However, the cost of constructing the similarity graph is usually significantly expensive. On the other hand, constructing LSH tables or KD-trees can be done efficiently and their querying step is parallelizable. Data-independent methods as space-partition indexes are also easier to perform maintenance operations (e.g., inserting or deleting a data point), compared to data-dependent methods as graph-based indexes. In the context of rapid growing recommender systems, indexes might need to be updated frequently, which is more suitable with structures such as LSH tables or spatial trees.

3.2 Efficient Candidate Generation via Ranking Estimation Strategies

To avoid the need for computing inner product scores and ranking these scores to arrive at the top-k recommendations, one approach is to estimate or approximate this ranking of scores using solely elements of user and item vectors. The estimated ranking of items can be used to generate candidates for recommendations. These candidates can be re-ranked later for more accurate ranking among them and the final top-k will be selected.

Particularly, all methods covered in this section consider the following matrix $Z = [x_u^{(l)} y_i^{(l)}], \forall i \in [1, 2, \ldots, n]$ and $\forall l \in [1, 2, \ldots, d]$:

$$Z = \begin{bmatrix} x_u^{(1)} y_1^{(1)} & x_u^{(1)} y_2^{(1)} & x_u^{(1)} y_3^{(1)} & \ldots & x_u^{(1)} y_n^{(1)} \\ x_u^{(2)} y_1^{(2)} & x_u^{(2)} y_2^{(2)} & x_u^{(2)} y_3^{(2)} & \ldots & x_u^{(2)} y_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_u^{(d)} y_1^{(d)} & x_u^{(d)} y_2^{(d)} & x_u^{(d)} y_3^{(d)} & \ldots & x_u^{(d)} y_n^{(d)} \end{bmatrix} \tag{6}$$

in which, each column i of Z is the element-wise product between vectors x_u and y_i. The sum of each column i of Z is the corresponding inner product score between x_u and y_i.

3.2.1 ITEM RANKING ESTIMATION WITH SORTED INDICES LISTS

To quickly construct the candidate set, (Yu et al., 2017) proposes an algorithm *Greedy-MIPS* based on the following assumption:

$$x_u{}^T y_i > x_u{}^T y_j \Leftrightarrow \max_{1 \le l \le d} x_u^{(l)} y_i^{(l)} > \max_{1 \le l \le d} x_u^{(l)} y_j^{(l)}, \tag{7}$$

In other words, the ranking of inner products between user vector x_u and item vectors $\{y_i\}_{i=1}^n$ can be approximated by the ranking of the maximum element-wise products. For each query x_u, if we can quickly determine the index pair (i, l) such that $x_u^{(l^*)} y_{i^*}^{(l^*)} = \max\{x_u^{(l)} y_i^{(l)}, 1 \le i \le n, 1 \le l \le d\}$, then we can yield an estimate of the inner products and avoid the expensive inner product computations. This is equivalent to determining the largest element in the matrix Z. Towards this end, (Yu et al., 2017) proposes two following steps:

- Before observing any query, *Greedy-MIPS* constructs d different lists, in which the $l-$th list consists of sorted indices of elements in that dimension of all item vectors, i.e., $\{y_i^{(l)}\}$.

- At the querying phase, a max-heap is employed to iteratively traverse (j, l) entries of matrix $Z = [x_u^{(l)} y_j^{(l)}], \forall j \in [1, 2, \ldots, n]$ and $l \in [1, 2, \ldots, d]$ in a greedy sequence, and the first newly visited item indices j will be added to the candidate set C_u.

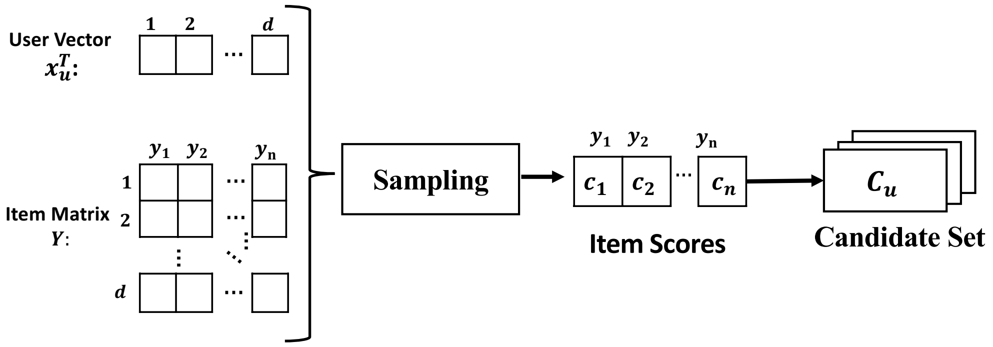

Figure 6: An Illustration of Candidate Screening via Sampling

With assumption (7), *Greedy-MIPS* can avoid inner product computations in the candidate generation process. Although assumption (7) is not guaranteed mathematically, *Greedy-MIPS* shows promising performances for many real-world datasets.

3.2.2 ITEM RANKING ESTIMATION VIA SAMPLING

Sampling is another way to approximate the ranking of inner products. The underlying principle of sampling methods for estimating the item ranking for a user u is to associate each item index $i \in [1, 2, \ldots n]$ with a ranking score, whose probability is proportional to $x_u^T y_i$, without the need to compute all n inner products $\{x_u^T y_i\}_{i=1,n}$. At the end of the sampling process, the spectrum of the item ranking scores, i.e., $\{c_1, c_2, \ldots, c_n\}$ would be highly correlated with the ranking of the original inner product values. The larger the inner product values of $x_u^T y_i$, the higher the score c_i of item index i.

The resulting item scores, i.e., $\{c_1, c_2, \ldots, c_n\}$ enable us to generate C_u using candidate items with the highest scores (illustrated in Figure 6). A re-ranking step with exact inner product computations can be performed afterwards to rank the candidates and to select the top-k items for recommendation.

In the following, we review several existing sampling strategies in the literature and the analysis of their complexities and limitations.

1. **Simple Sampling**: (Drineas et al., 2006) describes a simple sampling scheme, which can be described as follows. For each column i of Y, we sample a row index l with a probability of $x_u^{(l)}/||x_u||_1$, and return $y_i^{(l)}$ as the score. For each column index i, let Z_i be the random variable that denotes the output of this sampling process. The expectation value of the score

Z_i for an item i can be measured as in Eq. 8:

$$\begin{aligned} E\left[Z_i\right] &= \sum_{l=1}^{d} y_i^{(l)} \Pr\left[\text{row index } l \text{ is sampled}\right] \\ &= \sum_{l=1}^{d} \frac{y_i^{(l)} x_u^{(l)}}{||x_u||_1} = \frac{x_u^T y_i}{||x_u||_1} \end{aligned} \tag{8}$$

This means that for each $i \in I$, $||x_u||_1 Z_i$ is an unbiased estimate for $x_u^T y_i$. Therefore, these scores $c_i = ||x_u||_1 Z_i$ can be used to estimate the item ranking and to generate candidate set C_u. This sampling strategy has the cost of $O(n)$, as sampling is performed for each item $i \in I$. To generate B candidates for top-k recommendation, we need to rank the scores $\{c_1, c_2, \ldots, c_n\}$ with the cost of $O(n \log(B))$.

2. **Wedge Sampling**: (Cohen & Lewis, 1997) proposed another efficient sampling technique, called wedge sampling. Wedge sampling needs to pre-compute some statistics in advance, including the sum of all inner products $z = \sum_{j=1}^{n} x_u^T y_j$ and norm-1 of the rows of the item matrix Y, i.e., r_l^{norm} is the norm-1 of the $l-$th row of Y. Since we can pre-compute $r_l^{\text{norm}}, \forall 1 \leq l \leq d$, computing the sum of all inner products $z = \sum_{l=1}^{d} r_l^{\text{norm}} x_u^l$ takes $\mathcal{O}(d)$ time.

   Given the pre-computed sum of all inner products z and the norm-1 of matrix Y's rows $r_l^{\text{norm}}, \forall 1 \leq l \leq d$, wedge sampling randomly samples a column (an item) i of Y with probability $\frac{x_u^T y_j}{z}$ as follows. It first samples a row index l with probability $\frac{x_u^{(l)}}{z} r_l^{\text{norm}}$, and then samples a column (item) index i with probability $y_i^l / r_l^{\text{norm}}$. Each time the item index i is sampled, we increase the score c$_i$ for the i-th item by 1. In this case, c_i acts as a counting variable that summarizes the occurrence of item i in the sample set.

   We have:

$$\begin{aligned} \Pr(\text{item } i \text{ is sampled}) &= \sum_{l=1}^{d} \Pr(i|l).\Pr(l) \\ &= \sum_{l=1}^{d} \frac{y_i^l}{r_l^{\text{norm}}} . \frac{r_l^{\text{norm}} . x_u^{(l)}}{z} = \sum_{l=1}^{d} \frac{x_u^{(l)} . y_i^l}{z} = \frac{x_u^T y_j}{z} \end{aligned} \tag{9}$$

   Wedge sampling achieves lower variance than the basic sampling in practice. Given that the sampling is constant, wedge sampling has the complexity of $O(s)$ for generating scores $\{c_1, c_2, \ldots, c_n\}$ and $O(n \log(B))$ for ranking n scores $\{c_1, c_2, \ldots, c_n\}$ to arrive at a set of B candidates.

3. **Diamond Sampling**: (Ballard et al., 2015) proposes diamond sampling, which first makes use of wedge sampling to return a random column i corresponding to y_i with probability $\frac{x_u^T y_i}{z}$. Given such column index i, then (Ballard et al., 2015) samples a random row index l with probability $\frac{x_u^{(l)}}{||x_u||_1}$ and returns $y_i^{(l)}$ as the result. Let Z_i as the random variable that

denotes the output of the diamond sampling process. The expectation of Z_i can be measured as follows:

$$E[Z_i] = \sum_{l=1}^{d} y_i^{(l)} \frac{x_u^{(l)}}{||x_u||_1} \frac{x_u^T y_i}{z} = \frac{(x_u^T y_i)^2}{z||x_u||_1} \tag{10}$$

This output score Z_i of diamond sampling returns a scaled estimate of $(x_u^T y_i)^2$, identifying candidates with maximum squared inner product $({x_u}^T y_j)^2$. As it follows the wedge sampling strategy, diamond sampling also has the same complexity of $O(s + n\log(B))$ for generating B candidates for top-k MIPS.

4. **Bandit Sampling:** (Liu et al., 2019) introduces a new framework BoundedME, which casts MIPS as Multi-Arm-Bandit (MAB) with bounded pulls problem. The authors treat each item vector y_i as a bandit arm, whose reward has the following true mean of $p_i = \frac{1}{d}\sum_{l=1}^{n} x_u^{(l)} y_i^{(l)} = \frac{x_u^T y_i}{d}$. When pulling an arm y_i, its rewards are generated by sampling without replacement from the reward list $R_i = \{x_u^{(1)} y_i^{(1)}, x_u^{(2)} y_i^{(2)}, \ldots, x_u^{(d)} y_i^{(d)}\}$.

   Unlike previous sampling with replacement methods ((Drineas et al., 2006; Cohen & Lewis, 1997; Ballard et al., 2015)), the MAB framework in (Liu et al., 2019) is a sampling without replacement method. Once an arm is pulled n times, the mean of the returned rewards is exactly the true mean p_i. The MAB framework wants to achieve $epsilon-$optimal arm with probability of at least $1 - \delta$ with as few pulls as possible. This approach provides a flexible mechanism to control and bound suboptimality of the recommendation results with theoretical guarantee.

Advantages of Sampling Methods. One advantage of the sampling approach is that it natively supports budgeted queries (Lorenzen & Pham, 2020). As the number of samples increases, the variance of the estimate decreases, these methods offer a mechanism to govern the trade-off between search efficiency and accuracy. In (Qin Ding, 2019), the authors show that in the scenarios where assumption (7) does not hold, sampling-based methods could be more effective than Greedy-MIPS in (Yu et al., 2017). Another advantage of sampling is that it does not require a pre-built data structure of item vectors as the sampling process is performed at the querying time.

Limitations. The limitation of the above-mentioned sampling methods is that it can only handle non-negative item vectors and user vectors, i.e., when all the elements in the vector are non-negative. For example, diamond sampling's goal is to identify candidates with maximum squared inner product $({x_u}^T y_j)^2$, which is not equivalent to MIPS if inner product scores are negative. To solve this issue, (Qin Ding, 2019) recently proposes a new sampling scheme with the use of alias tables that can handle even negative input vectors. (Lorenzen & Pham, 2020) recently proposes a deterministic wedge-based algorithm, namely *dWedge* that runs significantly faster than methods from (Yu et al., 2017) for budgeted MIPS and (Li et al., 2017) for exact top-k MIPS.

3.3 Exact Methods for Efficient Candidate Generation

In this section, we discuss the class of exact methods for maximum inner product search (MIPS). Exact MIPS retrieval methods attempt to return an identical top-k to linear scanning without examining all the items. For some domains, exact MIPS may be desired than approximate techniques, especially those that are revenue-critical. We will review two different strategies for exact to-k MIPS in the following.

3.3.1 SEQUENTIAL SCANNING

Sequential scanning involves testing potential candidates sequentially, while weeding out unpromising ones. A prototypical approach is LEMP (Teflioudi & Gemulla, 2016; Teflioudi et al., 2015), which exploits the Cauchy-Schwartz inequality ${x_u}^T y_i \leq ||x_u||.||y_i||$.

It first sorts all items in decreasing order of their magnitudes $||y_i||$. For a user query vector x_u, it scans each candidate in the sorted order and maintains a list of the top-k largest inner products seen thus far. As soon as it encounters an item whose inner product upper bound $||x_u||.||y_i||$ is less than or equal to the $k-$th largest inner product computed so far, the scan stops and the current top-k items are returned as the results, since it is impossible for the next items after y_i to enter the result list. This avoids redundant inner product computations.

(Teflioudi & Gemulla, 2016) and (Teflioudi et al., 2015) further improve the efficiency of LEMP by proposing an incremental pruning technique that refines the upper-bounds by computing the partial inner product over the first several dimensions. FEXIPRO in (Li et al., 2017) adopts the same framework and applies singular value decomposition to the user and item matrices to make the first dimensions more meaningful, effectively improving the upper bounds and facilitating the pruning process.

3.3.2 HARDWARE-EFFICIENT MIPS SOLVER

A recent development in (Abuzaid et al., 2019) claims that methods like LEMP or FEXIPRO do not always outperform hardware-efficient approaches for the problem of maximum inner product search. The paper also proposes a solver for MIPS, namely MAXIMUS that takes advantage of hardware efficiency and pruning of the search space. As MAXIMUS is not always better than LEMP or FEXIPRO, (Abuzaid et al., 2019) introduces a data-dependent optimizer, OPTIMUS, that selects online for the best solver for MIPS methods for a given input.

(Xiang et al., 2019) claims that the bottleneck of existing exact solutions is the costly inner product computation. Based on this analysis, the paper proposes an approach that utilizes the characteristics of CPU-GPU systems to accelerate the inner product computation. The three features of the proposed approach include: heterogeneous processing of inner product computation and top-k list retrieval in CPU-GPU systems, reducing the data transfer between CPU and GPU, and avoiding unnecessary computation costs by early termination techniques. (Johnson et al., 2019) introduces FAISS, a library for billion-scale vector similarity search, harnessing the power of GPU for top-k nearest neighbor search and maximum inner product search.

3.4 Discussions

In this section, we discuss several matters concerning the efficient candidate generation methods for top-k MF-based recommendation retrieval and provide some insights for deeper understanding of these methods.

3.4.1 RETRIEVAL-EFFICIENT STRUCTURES

Retrieval-Efficient structures refer to the data structures that support efficient top-k recommendation retrieval such as sorted indices lists (Yu et al., 2017), LSH tables (Le & Lauw, 2017; Bachrach et al., 2014), spatial trees (Keivani et al., 2018), or similarity graph (Morozov & Babenko, 2018). These methods constitute a significant portion of the surveyed approaches for efficient top-k candidate generation. To better understand the cost redistribution undertaken by this strategy, we provide a summary of its various costs, including construction time, candidates generation time, and candidates ranking time in terms of theoretical complexities in Table 2. The methods are grouped according to their approximation strategies as described in Section 3.

- **Complexities.** Table 2 describes the complexities of each method in terms of the number of items n, number of features d, and the method's parameters. This provides an easy way to compare the methods across different recommendation retrieval criteria. As a reference point, we note that exhaustive linear scanning (first line) does not require a pre-processing step. However, it has significantly more expensive candidate ranking time, i.e., $\mathcal{O}(nd + n\log(k))$ than the other approaches that benefit from retrieval-efficient data structures. This is primarily due to the reduction in the number of investigated candidates, which results in faster ranking of candidates later. This benefit comes at the cost of constructing the data structures to store the item vectors in new formats that support efficient similarity search, which is a one-time cost to be amortized over the many query instances, as well as the cost to generate candidates in real-time, which generally is less dominant than the candidate ranking time. Another factor for consideration when using retrieval-efficient structures for recommender systems is the growth rate of the systems. As user preferences may change over time, new items appear, or old items are removed, maintaining a retrieval-efficient structure would require constant updates (e.g., insertion, deletion, or even re-build).

- **Accuracy-Efficiency Trade-Off.** Table 2 also hints at the controls allowed by these strategies to vary the trade-off between accuracy and efficiency. For instance, LSH-based retrieval can achieve higher accuracy, by increasing the number of hash tables L and reducing the number of hash functions b. The cost of this is the increase in terms of construction time, i.e., $\mathcal{O}(ndLb)$ and slower candidate ranking, i.e., $\mathcal{O}(\frac{nL}{2^b}d + \frac{nL}{2^b}\log(k))$. For spatial trees, one can decrease the depth level δ of the tree where the search process operates to get more candidates for the ranking step (i.e., slower ranking time) to gain a higher accuracy. With $s-$ Delaunay approximate similarity graph, one increases the vertex degree H when constructing the graph G_s to improve the quality of greedy walks. Sampling methods can increase the accuracy of top-k results by simply having more samples to better approximate the true ranking of inner product scores. Hence, optimizing for retrieval efficiency is less of a binary switch, and more of a configurable control knob.

For computationally intensive *preference elicitation* models, the process of deriving user and item vectors is usually executed in a periodic manner, followed by constructing the retrieval-efficient structures. To improve the efficiency, one can employ additional interactive *preference elicitation* models (He, Zhang, Kan, & Chua, 2016; Yin, Liu, Li, Dai, Chen, Wu, & Sun, 2019; He, Parra, & Verbert, 2016) to handle user feedback that arrives sequentially. In that scenario, one only need to update the user or item that corresponds to the new data in the structures, without the need to rebuild them entirely (Aytekin & Aytekin, 2019).

3.4.2 BUDGETED QUERIES

Some real-world scenarios may allow recommendation retrieval for different computational budgets depending on the query (Yu et al., 2017). For example, paid users (i.e., queries) could be allocated more computational budgets and expect more quality recommendations as compared to standard users who use the services without fee. For MF-based recommendation retrieval, the challenge is how to generate top-ranked recommendation candidates for a user under a given budget of inner product computations. To handle such budgeted queries scenarios, candidate generation methods should have a flexible control over the number of candidates produced, to be investigated in the ranking step.

Not all methods could handle such scenarios. For example, LSH tables and spatial trees require construction of query-independent structure, which means the query cost and the accuracy-efficiency trade-off are standardized across queries. Among the surveyed approaches, sampling-based methods (Cohen & Lewis, 1997; Ballard et al., 2015; Qin Ding, 2019; Liu et al., 2019) and Greedy-MIPS in (Yu et al., 2017) are capable of keeping the size of C_u within budget in the *candidate generation* step. Graph-based indexing is also a natural fit to budgeted queries scenario, as the greedy-search algorithm on the similarity graph can control the number of generated candidates. The advantage of sampling based methods over other efficient candidate generation methods is that they do not require an overhead cost of pre-processing the item vectors before querying. However, a major drawback of most sampling methods is that they can not handle the non-negative elements in the vectors.

4. Efficient Candidate Ranking

Given the candidate set C_u from *candidate generation* step, there is an unavoidable number of inner product computations, i.e., $|C_u|$ to derive the ranking of the candidates. This process can be made faster via efficient computation of inner product, whose idea is to *reduce* the cost of a specific user-item interaction score. There are two lines of research under this category: one is based on *discrete representation* approach, which learns to encode users and items as binary vectors equipped with efficient XOR operations for preference score computations and the other is based on *quantization* approach to *approximate* inner product computations.

- *Discrete representation* attacks the inefficient real-valued similarity operations when computing user/item vectors similarities by representing the users and items as binary vectors instead. This approach has become popular recently, thanks to the approximate optimization framework for learning binary user/item vectors in (Zhang et al., 2016), which can be adopted for most of the matrix factorization-based recommendation models. Not only do binary representations offer the advantage of cheaper storage (as there are many 0s in the vectors), they also support efficient similarity search with search strategies such as Hamming ranking or hash tables lookup (Zhang et al., 2016) (See Section 5). Section 4.1 provides more detail about the methods under this approach.

- *Quantization* approaches can achieve efficient candidates ranking as they require a smaller number inner product computations in $d-$ dimensional space or cheaper inner product computations in lower dimensional sub-spaces. Particularly, *vector quantization* exploits the clustering structures of the item vectors to approximate each item vector by the cluster's centroid to reduce the number of similarity computations when searching for top-k items. *Product quantization*, on the other hand, attacks the dimensionality d of the latent space by decomposing the feature vectors into several lower sub-components and applying vector quantization for each sub-component. In Section 4.2, we describe these techniques in detail.

	Method	**Construction Time (fixed cost)**	**Candidate Generation Time**	**Candidate Ranking Time**	**Notes**
	Linear Scanning	NA	NA	$\mathcal{O}(nd + n\log(k))$	
Item Ranking Estimation	Sorted Indices Lists (Yu et al., 2017)	$\mathcal{O}(dn\log(n))$	$\mathcal{O}(d\log(d))$	$\mathcal{O}(Bd + B\log(k))$	B-budget for the number of inner product computations
	Simple Sampling (Drineas et al., 2006)	NA	$\mathcal{O}(n + n\log(B))$	$\mathcal{O}(Bd + B\log(k))$	
	Wedge Sampling (Cohen & Lewis, 1997)	$\mathcal{O}(d)$	$\mathcal{O}(s + n\log(B))$	$\mathcal{O}(Bd + B\log(k))$	s-number of samples
	Diamond Sampling (Ballard et al., 2015)				
	BoundedME (Liu et al., 2019)	NA	$\mathcal{O}\left(\frac{n\sqrt{d}}{\epsilon}\sqrt{\log\left(\frac{1}{\delta}\right)}\right)$	$\mathcal{O}(B + B\log(k))$	BoundedME returns a set of B items that is ϵ-optimal with probability $1 - \delta$
Indexing Structures	LSH Hash Table (Neyshabur & Srebro, 2015)	$\mathcal{O}(ndLb)$	$\mathcal{O}(Lbd)$	$\mathcal{O}(\frac{nL}{2^b}d + \frac{nL}{2^b}\log(k))$	L-number of hash tables; b-number of hash functions
	KD Tree (Le & Lauw, 2017)	$\mathcal{O}(nd)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(\frac{n}{2^\delta}d + \frac{n}{2^\delta}\log(k))$	n_t-the number of trees; δ is the depth of the search tree
	PCA Tree (Bachrach et al., 2014)	$\mathcal{O}(nd^2)$			
	Random Partition Tree (Keivani et al., 2018)	$\mathcal{O}(n_t n\log(n)d)$	$\mathcal{O}(n_t\log(n))$	$\mathcal{O}(\frac{n_t}{2^\delta}nd + \frac{n_t}{2^\delta}\log(k))$	
	Sparsity Mapping + Inverted Index (Bhowmik et al., 2016)	$\mathcal{O}(nd\log(d))$	$\mathcal{O}(d\log(d))$	$\mathcal{O}(\frac{n}{D}d + \frac{n}{D}\log(k))$	(Bhowmik et al., 2016) describes a sparsity mapping scheme to convert MF-based user and item vectors to $D-$dimensional sparse vectors.
	s-Delaunay Graph-G_s (Morozov & Babenko, 2018)	$\mathcal{O}(n^2d)$	$\mathcal{O}(Qd_{G_s})$	$\mathcal{O}(\|S\|d + \|S\|\log(k))$	d_{G_s}-the maximum depth of G_s; Q-size of the priority queue; S-sample set generated by the random walk on G_s.

Table 2: Summary of Retrieval-Efficient Methods

x

Note that, some of the methods described in this category can also be used to efficiently generate the recommendation candidates. For example, we can create Locality Sensitive Hashing tables with the output of *discrete representation* methods to quickly retrieve relevant items given the user binary vector as query. With *vector quantization*, we can consider the items whose centroids are most similar to the query as candidates. Here, we focus on the use of these methods in approximating the inner product computations in $d-$dimensional space.

4.1 Discrete Representation for Efficient Recommendation Retrieval

In this approach, each user/item is represented as a sequence of binary values. Let $b_u, d_i \in \{-1, 1\}^r$ be the binary codes that represent user u and item i respectively. The inner product preference score in the Hamming space can be converted to computing the Hamming distance between user and item binary vectors, which requires simple XOR operations (Zhang et al., 2016):

$$\begin{aligned}\text{sim}(u,i) &= \frac{1}{r}\sum_{l=1}^{r}\mathbb{I}(b_u^l = d_i^l)\\ &= \frac{1}{2}\left(\sum_{l=1}^{r}\mathbb{I}(b_u^l = d_i^l) + r - \sum_{l=1}^{r}\mathbb{I}(b_u^l \neq d_i^l)\right)\\ &= \frac{1}{2r}\left(r + \sum_{l=1}^{r} b_u^l d_i^l\right) = \frac{1}{2} + \frac{1}{2r}b_u^T d_i \end{aligned} \tag{11}$$

where $\mathbb{I}(.)$ denotes the indicator function that returns 1 if the statement is true and 0 otherwise.

Given the binary representation of the users and items, one can use different search strategies for nearest neighbor search such as hash tables or Hamming ranking (as described in Section 5). To arrive at the binary vector representation for users and items, there are two main approaches: *quantization-based discretization* or *optimization-based discretization*.

4.1.1 Quantization-Based Discretization

Quantization-based discretization consists of two phases. The first phase is *relaxed optimization* for preference learning with some specific constraints to obtain continuous latent representations for users and items. The second phase is *binary quantization* to convert the continuous latent representations from the first phase into binary codes. We review related work in the following:

(Zhou & Zha, 2012) constructs binary codes such that the Hamming distances of a user and her preferred items are small through minimizing:

$$\begin{aligned}\mathcal{L} &= \sum_{(u,i)\in\Omega}(r_{ui} - \text{sim}(u,i))^2 + \lambda\left(||\sum_u b_u||^2 + ||\sum_i d_i||^2\right)\\ &= \sum_{(u,i)\in\Omega}\left(r_{ui} - \frac{1}{2} - \frac{1}{2r}b_u^T d_i\right)^2 + \lambda\left(||\sum_u b_u||^2 + ||\sum_i d_i||^2\right)\end{aligned} \tag{12}$$

(Zhou & Zha, 2012) first relaxes the solution space of Equation 12 to be real-vectors $\tilde{b}_u, \tilde{d}_i$ in $[-1, 1]^r$ and then rounds the real-valued solutions to the closest binary vectors in $\{\pm 1\}^r$, i.e.,

$$b_u = \underset{b \in \{\pm 1\}^r}{\operatorname{argmin}} ||b - \tilde{b}_u||; \tag{13}$$

$$d_i = \underset{d \in \{\pm 1\}^r}{\operatorname{argmin}} ||d - \tilde{d}_i||. \tag{14}$$

(Zhang et al., 2014) proposes Preference Preserving Hashing or PPH, which is also a two-stage process:

- At the relaxation step for optimization, different from classic matrix factorization, which regularizes the norm of the latent vector to be small. PPH encourages the latent feature norm to reach maximum ratings and hence smaller discrepancy between inner product and cosine similarity is expected. The objective function is as follows:

$$\begin{aligned} \min_{X,Y} \mathcal{L} = \sum_{(u,i) \in \Omega} \left(r_{ui} - x_u^T y_i\right)^2 + \lambda \left(\sum_u \left(||x_u||^2 - \frac{r_{\max}}{2} \right)^2 \right. \\ \left. + \sum_i \left(||y_i||^2 - \frac{r_{\max}}{2} \right)^2 \right) \end{aligned} \tag{15}$$

  where $r_{\max}$ represents the maximal rating value.

- At the quantization step, PPH quantizes each feature vector into $(r-2)$ bit phase codes and 2-bit magnitude codes. The Hamming distance between the generated hashcodes b_u, d_i will preserve the similarity between the original x_u, y_i.

Quantization-based discretization is considered a natural development of existing matrix factorization recommendation models to produce binary vector representation for users and items. However, according to (Zhang et al., 2016), the major drawback of quantization-based discretization methods is that they suffer from large information loss in the binary quantization stage. This could result in a significant degeneration of recommendation accuracy of the original matrix factorization models. This issue motivates the wide adoption of optimization-based discretization approach as an alternative to the two-step approach.

4.1.2 OPTIMIZATION-BASED DISCRETIZATION

Optimization-based discretization adopts classic matrix factorization formulations, while imposing further constraints on balance and decorrelation for the binary codes. The balance constraint is to require each bit to split the dataset as balanced as possible, maximizing the information entropy of the bit. The decorrelation constraint is to guarantee that each bit should be as independent as possible, removing redundancy among the bits (Zhang et al., 2016).

Generally, the framework for optimization-based discretization can be described as in Equation (16), in which $\mathcal{L}(R, B, D)$ is the loss function over the rating matrix (or user-item interactions), user and item binary codes B and D and the constraints are for the balance and decorrelation properties of B and D.

$$\begin{aligned} \min_{B,D} \mathcal{L}(R, B, D) \text{ such as: } B \in \{\pm 1\}^{r \times m}, D \in \{\pm 1\}^{r \times n}, \\ B1 = 0, D1 = 0; BB^T = I_m, DD^T = I_n \end{aligned} \tag{16}$$

However, solving Equation 16 is challenging since the solution space is $\mathcal{O}\left(2^{(m+n)\times r}\right)$ and involves combinatorial search for the binary codes. (Zhang et al., 2016) proposes to soften the balance and decorrelation constraints by solving the following problem:

$$\begin{aligned} \min_{B,D,X,Y} \mathcal{L}\left(R,B,D\right) - 2\alpha \mathrm{tr}(B^T X) - 2\beta \mathrm{tr}(D^T Y) \quad (17) \\ \text{such as: } B \in \{\pm 1\}^{r\times m}, D \in \{\pm 1\}^{r\times n}, \\ X1 = 0, Y1 = 0; XX^T = I_m, YY^T = I_n \end{aligned}$$

Through joint optimization for the binary codes B, D and the delegate real variables X, Y, the obtained solution for user and item binary codes are nearly balanced and uncorrelated.

We review several representative works in the following. DCF (Zhang et al., 2016) proposes to learn the user and item hashcodes whose inner products reconstruct the observed ratings and laid down an optimization framework for later works to learn user and item binary codes from user-item interactions. For instance, DPR in (Zhang et al., 2017) learns from implicit feedback with ranking-based AUC objective function. (Liu et al., 2019b) proposes DSR to learn user and item codes that considers social information. DCMF (Lian et al., 2017) also takes into account context information (such as user's age and gender, item's category and textual content), while DDL (Zhang et al., 2018b) combines Deep Belief Network (DBN) and Collaborative Filtering (CF). DFM (Liu et al., 2018), (Qu et al., 2020) also learn binary codes for any side feature based on the factorization machine framework. Meanwhile DRMF (Zhang et al., 2018a) is based on each user's pairwise preferences, with self-paced learning. (Li et al., 2019) proposes Discrete Collaborative Hashing, a binary codes learning framework with neural collaborative filtering for efficient recommendation systems. (Guo et al., 2019) introduces discrete trust-aware matrix factorization (DTMF) model to take advantage of both social relations and discrete technique for fast recommendation. (Wang et al., 2019) exploits graph convolutional network (GCN) to model high-order feature from implicit feedback and distill the ranking information derived from GCN to binarized collaborative filtering to improve the efficiency of online recommendation. (Liu et al., 2019a) introduces a new approach Compositional Coding for Collaborative Filtering (CCCF) that represents each user/item with a set of binary vectors, instead of one as in (Zhang et al., 2016), which are associated with a set of sparse real-value weight vectors. CCCF claims to achieve better recommendation efficiency than many other discrete learning methods.

In summary, optimization-based discretization methods have been studied extensively recently, thanks to the rich literature of matrix factorization recommendation models and the new optimization framework proposed in (Zhang et al., 2016). However, the accuracy of optimization-based discretization for collaborative filtering may be affected by the limitted representation capacity of each bit. Therefore, they usually require longer binary codes to meet the desired performance, which might hurt the generalization of the model in the sparse scenarios.

4.2 Efficient Maximum Inner Product Search via Quantization

In this section, we discuss another approach for efficient MIPS, namely *quantization*. The main idea of quantization approach is to approximate the inner product computations in the d dimensional space.

4.2.1 VECTOR QUANTIZATION

Vector quantization is a data compression technique. The idea is to approximate the data points through a smaller collection of representative vectors. In the literature, the collection of these representative vectors is referred to as the *codebook*. We can construct these codebook vectors by using clustering algorithms such as K-means clustering to get the clusters' centroids. Each point is represented by the centroid of the cluster it belongs to. This approach greatly reduces the number of required computations when we perform nearest neighbor search. For a given query, we first determine the closest codebook vector (i.e., centroid). We can then perform similarity computations with data points of that centroid's cluster to determine the top nearest neighbors.

(Du & Wang, 2014) proposes a compositional code approach to approximate inner product search. The idea is based on a vector quantization. Using the composition of several vectors, each of which is selected from one of M source dictionaries $\{C_1, C_2, \ldots, C_M\}$ (where $C_t, \forall 1 \leq t \leq M$ is a matrix of size $d \times n_C$, and n_C is the number of vectors in C_t). Each item vector y_i is represented by a compositional code $\mathbf{b}_i = \left[\mathbf{b}_{i1}^T, \mathbf{b}_{i2}^T, \ldots, \mathbf{b}_{iM}^T\right]^T$ such that:

$$y_i \approx [C_1, C_2, \ldots, C_M] \left[\mathbf{b}_{i1}^T, \mathbf{b}_{i2}^T, \ldots, \mathbf{b}_{iM}^T\right]^T,$$

in which $\mathbf{b}_{it} \in \{0,1\}^{n_C}; ||b_{it}||_1 = 1, \forall 1 \leq t \leq M$. The vectors in source dictionaries $C_t, \forall 1 \leq t \leq M$ and the compositional codes $\mathbf{b}_i, \forall 1 \leq i \leq n$ are learned to minimize the item vector approximation error. The complexity of the learning process is $\mathcal{O}(nM^2 n_C d + d(Mn_C)^2 + (Mn_C)^3)$ for one iteration.

For any query vector x_u, we first compute the inner products between x_u and $M \times n_C$ dictionary vectors with the cost $\mathcal{O}(Mn_C d)$. Since $\sum_{1 \leq t \leq M} n_t < n$, the number of inner product computations is reduced by the order of $\frac{n}{\sum_{1 \leq t \leq M} n_t}$. These inner product values can be used to approximately compute the inner product $x_u^T y_i$ in $\mathcal{O}(M)$ time.

4.2.2 PRODUCT QUANTIZATION

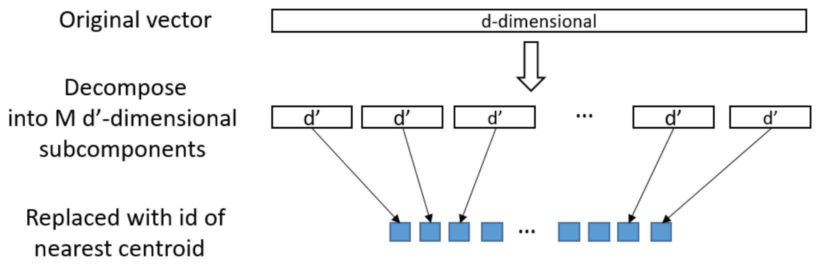
Original vector
d-dimensional
Decompose into M d'-dimensional subcomponents
d'
d'
d'
...
d'
d'
Replaced with id of nearest centroid
...

Figure 7: Illustration for Product Quantization Method

In order to get good search accuracy, vector quantization methods need to use a large value of clusters, i.e., a large number of codebook vectors to better approximate the original data points. *Product quantization* addresses this issue by first decomposing the $d-$dimensional vector space into the product of M sub-spaces with dimension $d' = \left[\frac{d}{M}\right]$ (Figure 7). In each subspace, we quantize each sub-component vectors of the original vectors separately. Each data point is represented by a collection of centroids, one for each sub-component. This approach is interesting, as it turns the curse of dimensionality challenge into an advantage.

(Guo et al., 2016) describes a product quantization technique for maximum inner product search. In particular, each vector is first permuted using a random (but fixed) permutation. Each permuted vector is mapped to M subspaces using simple chunking.

$$x_u = \left[x_u^{(1)}; x_u^{(2)}; \ldots; x_u^{(M)}\right]; \tag{18}$$

$$y_i = \left[y_i^{(1)}; y_i^{(2)}; \ldots; y_i^{(M)}\right] \tag{19}$$

where $x_u^{(t)}, y_i^{(t)} \in \mathbb{R}^l, l = \lceil \frac{d}{M} \rceil$. The t^{th} subspace containing the t^{th} blocks of all the item vectors $\{y_i^{(t)}\}$, is quantized by a code book $U^{(t)} \in \mathbb{R}^{l \times n_C}$, where n_C is the number of quantizers in subspace t. Each item vector y_i is quantized in the t^{th} subspace as $y_i^{(t)} \sim U^{(t)} \alpha_{y_i}^{(t)}$, where $\alpha_{y_i}^{(t)}$ is one-hot indication vector. The inner product of a user query vector and an item vector is approximated as the sum of inner products with the subspace quantizers, with the cost of $\mathcal{O}(M n_C \lceil \frac{d}{M} \rceil)$:

$$x_u{}^T y_i = \sum_{k=1}^{K} x_u^{(k)T} y_i^{(k)} \approx \sum_{k=1}^{K} x_u^{(k)T} U^{(k)} \alpha_{y_i}^{(k)}$$

The codebook $U^{(k)}$ and indication vector $\alpha_{y_i}^{(k)}$ are learned by minimizing the inner product quantization error for held-out queries Z with known top-k, at the cost of $\mathcal{O}(n n_C |Z|)$.

Norm-Explicit Quantization - A Hybrid Approach. Recent work (Dai et al., 2019) points out that existing vector quantization techniques do not allow explicit control of norm error and angular error. For each item vector y_i and its codebook approximation $\bar{y}_i$. Given the user query vector x_u, the paper defines the inner product error, norm error, and angular error as follows:

$$\text{ip} - \text{err} = \left| \frac{x_u^T y_i - x_u^T \bar{y}_i}{x_u^T y_i} \right|; \tag{20}$$

$$\text{norm} - \text{err} = \left| \frac{||y_i|| - ||\bar{y}_i||}{||y_i||} \right|; \tag{21}$$

$$\text{ang} - \text{err} = 1 - \frac{y_i^T \bar{y}_i}{||y_i||.||\bar{y}_i||} \tag{22}$$

The paper also hints that the norm error has more significant influence on inner product than the angular error in most cases. Thus, it is more appropriate to reduce quantization errors in norm to improve the performance of vector quantization methods for MIPS. Specifically, the idea of the proposed method in (Dai et al., 2019) is to quantize the norm $||y_i||$ and the direction vector $\frac{y_i}{||y_i||}$ of the item vectors separately. The objective is to achieve small norm error, while the direction vector can be quantized using previous vector quantization methods. The code-books are divided into two parts. The first M' are norm codebooks and the other $M - M'$ code-books are for the direction vector.

4.3 Discussions

In this section, we discuss the accuracy and efficiency trade-off of the efficient candidate ranking methods and provide comparative analyses on several aspects of quantization and discretization strategies for candidate ranking.

4.3.1 ACCURACY AND EFFICIENCY TRADE-OFF

For *vector quantization*, higher accuracy can be achieved by increasing the number of quantization vectors n_C, which yields better item vector approximation (Du & Wang, 2014). However, that is equivalent to increasing the number of inner product computations between the user query vector and the codebook vectors at the querying step. The computational complexity for each query is $\mathcal{O}(n_C \times d)$, which scales linearly with both the number of codebook vectors and the dimension of the latent space. *Product quantization* is a clever approach to reduce the cost caused by the dimension d of the latent space. However, to achieve good approximation resolution, product quantization methods still need to increase the number of decompositional subspaces M (Guo et al., 2016), which leads to the expensive cost at the querying step.

Discrete representation approaches may be limited by the representation capacity of the binary vectors. Particularly, $r-$dimensional real-valued vectors can represent an infinite number of users and items. Whereas in the binary setting, $r-$dimensional binary vectors can represent a most 2^r users/items. Discrete representation methods generally achieve better recommendation accuracy through longer binary codes, i.e., r. However, large code lengths, say more than 64, would require over $\mathcal{O}(2^{64})$ space to store the binary codes. To deal with such problem, one can use Multi-Index Hashing (MIH) (Norouzi et al., 2012), which builds multiple hash tables, each for each code sub-segment. Given the query vector b_u, candidate items are aggregated from all the tables and Hamming ranking is applied for re-ranking. MIH has also been used in (Kang & McAuley, 2019), in which the authors introduce a multi-step framework CIGAR, consisting of: (1)-learning preference-preserving binary codes B, D, (2)- constructing a multi-index hash tables to index all item vectors, and (3)-training a ranking model for candidate re-ranking step for more accurate final top-k recommendation.

4.3.2 POST-LEARNING QUANTIZATION VS. NATIVE DISCRETIZATION

From another perspective, *vector quantization* and *quantization-based discretization* are related as they both employ post-learning quantization of item vectors. This one-time cost of item preprocessing is independent of the learning algorithms, i.e., it can be flexibly applied to the output of any preference learning algorithms. This one-time processing is also beneficial for many future requests. However, the major drawback of this two-step solution is the substantial loss of information at the quantization step, which usually requires a large number of codebook vectors to preserved the accuracy achieved by the original algorithms.

Meanwhile, *optimization-based discretization* natively supports efficient inner product computations in the Hamming space, however at the cost of solving harder optimization during the preference elicitation phase. Since it is dependent on the training data, *optimization-based discretization* is less flexible as compared to using data-independent hashing methods such as Locality Sensitive Hashing or quantization-based discretization. The learning framework is also based on collaborative filtering idea, thus they still potentially suffer from the sparsity of user-item interactions, and the adoptions of new users and items. A promising solution is to exploit the user and item auxiliary features to resolve the data sparsity issue and the cold-start problem.

5. Conclusions and Open Questions

The current scale of e-commerce systems requires recommender systems to improve the efficiency of both the *preference elicitation* phase and the *recommendation retrieval* phase. While there have been numerous works on designing preference elicitation algorithms that can handle millions of users and items, efficient recommendation retrieval has only gained more attention recently due to the demand for online personalized recommendation of large scale systems.

This survey focuses on the retrieval phase of recommendation and provides an overview of recent advances for efficient retrieval of matrix factorization recommendation. Particularly, the two requisite steps for recommendation retrieval are *candidate generation*, which produces a set of candidate items for top-k given a user vector x_u and *candidate ranking*, which re-ranks the candidate items to extract the final top-k. In this paper, for each step, we present a comprehensive taxonomy that categorizes the relevant methods based on their pertinent strategies to improve the retrieval efficiency, while still maintain high recommendation accuracy. We also provide several qualitative analyses to better compare the surveyed methods along different dimensions and scenarios.

In the literature, there have been many proposed techniques to generate high quality recommendation for each individual user, which would require the recommendation retrieval process to be performed efficiently. This survey, however, is dedicated to the class of matrix factorization techniques for personalized recommendation. There are still many interesting open problems we would like to highlight as promising research directions:

1. Items or products can be represented by both sparse and dense vectors. For example, matrix factorization vectors can be used in conjunction with sparse tf-idf vectors from textual features. In some scenarios, the system may want to recommend items that are of interest to the target user and also relevant to past adopted items by the user. Efficient recommendation retrieval for such representation is challenging as current retrieval techniques are designed to handle either dense or sparse component only. An interesting problem that worth further investigation is to design an efficient data structure to solve these emerging hybrid scenarios (Wu et al., 2019).

2. In this survey, we discuss preference elicitation and recommendation retrieval as two separated phases, which could potentially hurt the final accuracy of top-k recommendation. Recent studies have shown that considering the two phases altogether is beneficial. For instance, (Kang & McAuley, 2019) proposes CIGAR, which jointly learns user/item binary codes with implicit feedback, generates candidate with multi-index hash tables, and re-ranks candidates with real-valued models. CIGAR exhibits both the efficiency of candidate generation with hash tables and the accuracy of real-valued models for candidate ranking. (Le & Lauw, 2020) introduces SRPR, which learns the user and item vectors that are robust to the stochasticity of randomly generated LSH hash functions. SRPR thus achieves both efficient retrieval and accurate top-k recommendation when LSH is the designated indexing structure. Hence, a promising research direction that worth further study is to derive unified frameworks for both accurate preference elicitation and efficient recommendation retrieval.

3. Another interesting problem is how to perform efficient recommendation retrieval for deep neural network-based recommendation models (He et al., 2017; Deng et al., 2019; Zhang et al., 2019). Classic matrix factorization models encode the user-item relationship through

the inner product kernel between the user vector and the item vector. For neural network-based models, this kernel is replaced by multi-layered neural networks to encode complex and non-linear relationships between users and items. Which of classic matrix factorization with inner product formulation or neural network-based approaches (Rendle et al., 2020) are superior is still actively researched. However, due to the increasing attention of the latter, it is worth investigating how to perform efficient recommendation retrieval for these models. Recent studies (Zhu et al., 2018; Tan et al., 2020) have been proposed to achieve fast ranking of items of neural network-based recommendation models.

4. Beyond accuracy and efficiency, the focus of recommender systems has gradually shifted to consider other recommendation objectives such as *novelty, coverage,* and *serendipity* (Kaminskas & Bridge, 2016; Cheng et al., 2017; Wang et al., 2018; Xu et al., 2020). In the literature, these objectives are mostly considered at the *preference elicitation* phase (Kaminskas & Bridge, 2016) instead of at the *recommendation retrieval* phase. As most of the efficient recommendation retrieval strategies are *approximate*, the lists of recommended items are not entirely based on *relevancy* to the user and could include *surprising, unexpected* items. Hence, an interesting direction is to investigate how these approximate recommendation retrieval methods affect the above-mentioned objectives. Whether there is any trade-off between *accuracy*, *efficiency*, *novelty, coverage*, and *serendipity* of top-k using the efficient recommendation retrieval methods in this survey.

As recommender system is a widely employed technology for real-world large-scale systems, scalable recommendation retrieval has become an interesting important problem that needs further studies and experimentation. Due to the ever-growing number of users and items, the optimal solution for this problem still remains a challenge for researchers and practitioners.

Acknowledgments

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its NRF Fellowship Programme (Award No. NRF-NRFF2016-07).

Appendix A

The ubiquity of high dimensional objects such as images, audios, videos, and documents in today applications makes the brute force top-k similarity search ($k-$SS) solution prohibitively expensive. This prohibitive cost has given the rise to approximate $k-$SS studies, seeking a balanced trade-off between the search quality and the search efficiency. To equip the readers with necessary background, we describe in the following popular data structures that are used extensively for the approximate $k-$SS problem in general and for efficient MF recommendation retrieval in specific.

Locality Sensitive Hashing

A.1 Locality Sensitive Hashing or LSH

(Indyk & Motwani, 1998; Datar et al., 2004a) is a probabilistic space-partitioning indexing technique. One important element of LSH is the hashing function $h(.)$, which maps a data point into a hash value. This hashing function is usually designed with the *locality-sensitive* property, which is: *two similar data points are more likely to get the same hash values as compared to two distant data points.* Mathematically, the above property can be expressed as follows:

$$\begin{aligned}&\mathrm{sim}(x, y_1) > \mathrm{sim}(x, y_2) \\ &\quad\Rightarrow \Pr\left(h(x) = h(y_1)\right) > \Pr\left(h(x) = h(y_2)\right),\end{aligned} \tag{23}$$

in which, $\Pr(.)$ denotes the collision probability function and $\mathrm{sim}(.,.)$ measures the similarity between two data points.

A set of *locality-sensitive* hashing functions effectively map a data point to a hashcode. Let $\mathbf{h}$ be a set of LSH hash functions i.e., $\mathbf{h} = (h_1, h_2, \ldots, h_b)$. $\mathbf{h}$ will assign each user u a binary code $\mathbf{h}(x_u)$, and each item i a binary hashcode $\mathbf{h}(y_i)$, all of length b. The length of hashcodes b is an important parameter to be specified at the index construction time. This parameter controls the trade-off between search accuracy and search speed. The longer the hashcode length b, the better the hashcodes will approximate the original similarities among data points, resulting in more accurate search results. However, longer hashcodes also slow down the other run-time components. The design of LSH hash function is highly sensitive to the similarity metric. There exists various LSH families to support different approximate similar search problems, e.g., *Euclidean LSH* (Datar et al., 2004b) for approximate *Nearest Neighbor Search* and *Sign Random Projection LSH* (Charikar, 2002) for approximate *Maximum Cosine Similarity Search.*

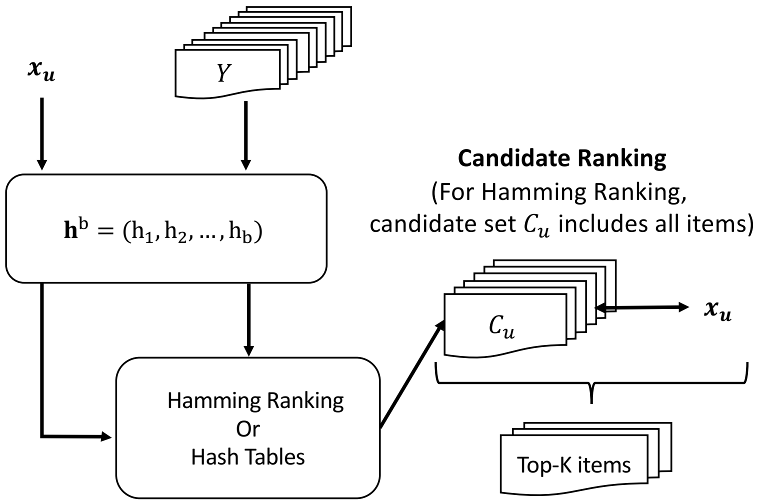
x_u
Y
$\mathbf{h}^b = (h_1, h_2, \ldots, h_b)$
Hamming Ranking Or Hash Tables
Candidate Ranking
(For Hamming Ranking, candidate set C_u includes all items)
C_u
x_u
Top-K items

Figure 8: Indexing and Searching with LSH

The most frequent search protocol using LSH is *Hamming Ranking* and *LSH Hashtable* (see Fig.8). The former strategy performs a linear scan during which the distance between the query hash-code and the hash-code of all items is computed. In the latter strategy, we store item hash-codes in hash tables, with items having the same hash-code in the same bucket. Given a query (user) code, we can determine the corresponding bucket in constant time. We search for the top-k only among items in that bucket, referred to as the candidate set C_u, reducing the number of items on which we need to perform exact similarity computations. Higher accuracy can be achieved by further searching in neighboring buckets or searching with many hash tables. However, this would increase the number of computations for inner product scores to arrive at the final top-k items.

A.2 Spatial Tree Indexing

Spatial Tree Indexing (McFee & Lanckriet, 2011) refers to a family of indexing methods that recursively partition the data space. The resulting data structure is a balanced binary search tree, in which each node encompasses a subset of the data points (Figure 9). A generic algorithm to construct the partition trees is described in (McFee & Lanckriet, 2011), in which one needs to specify the rule for generating space splitting directions. We refer the readers to (McFee & Lanckriet, 2011) for more details about different spatial tree indexing methods.

During the search process, the tree locates the nodes that the query belongs to, and exact similarity computation is performed only on the points indexed by those nodes. It is easy to implement and requires minimal parameter tuning, but may not be suitable for high dimensional data (i.e., the curse of dimensionality). Efficient search with spatial trees can be achieved in many ways such as limiting the number of visited leaf nodes or the depth level of the tree that the search process are allowed to reach. While originally developed for metric spaces, the framework has been recently extended to support efficient retrieval for non-metric similarity.

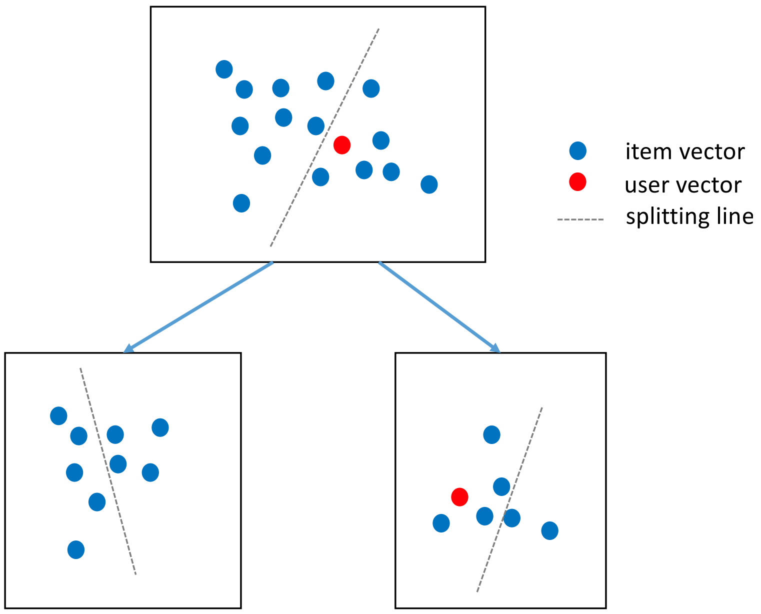

Figure 9: Indexing and Searching with Spatial Tree

For recommendation algorithms that model the user-item association through L_2 distance, i.e., $r_{ui} \propto -||x_u - y_i||^2$, spatial trees can be used to index the item vectors $\{y_i\}_i \in \mathcal{I}$. Top-$k$ recommendation is thus equivalent to nearest neighbor search for the query (user) vector. For MF-based models, as inner product is used as the predictor, a post-learning vector augmentation process should be conducted before constructing the spatial tree index of the item vectors (See Section 3.2).

A.3 Inverted Indexing

Inverted index (Zobel & Moffat, 2006) is designed for top-K relevant documents retrieval with sparse TF-IDF vector representation. Inverted index exploits the sparsity structure of TF-IDF vectors to quickly filter out potentially irrelevant documents. However, MF-based methods usually derive dense latent user and item vectors from user-item interactions. Using inverted indexing for matrix factorization-based recommendation retrieval could be counter-intuitive. (Bhowmik et al., 2016) recently presents a post-MF-learning processing with sparsity mapping scheme that derives sparse representation for each user and item from their respective dense real-valued latent vectors. Two close points are mapped to sparse vectors with similar sparsity pattern (i.e., significant overlap of non-zero indices). Two distant points are mapped to vectors with different sparsity pattern. It then uses inverted indexing with the resulting sparse vectors for efficient top-k retrieval.

A.4 Similarity Graph

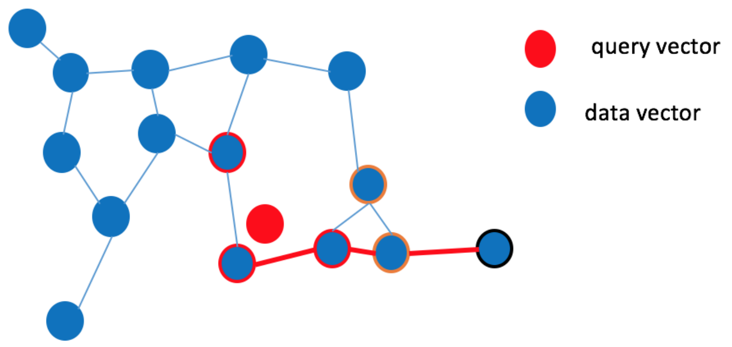

Figure 10: Indexing and Searching with Similarity Graph:
Starting at a random vertex (black circle), the search traverses over the graph by moving to the neighbors of visited vertices (orange circle) to arrive at the true top closet neighbors (red circles).

Similarity graph approaches (Malkov & Yashunin, 2019; Morozov & Babenko, 2018) have shown a superior performance over other approaches in solving $k-$SS problem in terms of accuracy. The key step of most methods under this category lies in the process of *constructing the similarity graph* from the data points. In this similarity graph, each vertex represents a data point. Two vertices are connected if the two corresponding data points are sufficiently close to one another according to a similarity measurement. The *search* process starts from a group of random seed vertices and traverses iteratively over the graph. Guided by the neighbors of visited vertices, the search procedure descends closer to the true nearest neighbor in each round until no better candidates could be found. Figure 10 illustrates the greedy search process on the constructed similarity graph.

In the literature, various constraints have been proposed on the edges to make the graph more suitable for approximate nearest neighbor search. For example, Delaunay Graph (Aurenhammer, 1991) or Monotonic Search Networks (Dearholt et al., 1988) ensure that from any node p to another the node q, there exists a path on which the intermediate nodes are closer towards q (Dearholt et al., 1988). Other work includes Randomized Neighborhood Graphs (Arya & Mount, 1993) and Navigable Small-World Networks (Malkov & Yashunin, 2019). However, these approaches still face a challenge that the time complexity of building the similarity graphs is still impractical for many real-world applications.

Some recent approaches try to address this problem by designing approximations for the similarity graphs. For example, GNNS (Hajebi et al., 2011), IEH (Jin et al., 2014), and Efanna (Fu &

Cai, 2016) are based on the $k-$NN graph, which is an approximation of the Delaunay Graph. NSW (Malkov & Yashunin, 2019) is proposed to take advantage of the Delaunay Graph, the NSWN, and the Relative Neighborhood Graphs, enabling multi-scale hopping on different layers of the graph.

References

Abdi, M. H., Okeyo, G., & Mwangi, R. W. (2018). Matrix factorization techniques for context-aware collaborative filtering recommender systems: A survey. *Comput. Inf. Sci.*, *11*, 1–10.

Abuzaid, F., Sethi, G., Bailis, P., & Zaharia, M. A. (2019). To index or not to index: Optimizing exact maximum inner product search. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 1250–1261.

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, *17*(6), 734–749.

Aggarwal, C. C., Wolf, J. L., Wu, K.-L., & Yu, P. S. (1999). Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 201–212. ACM.

Arya, S., & Mount, D. M. (1993). Approximate nearest neighbor queries in fixed dimensions.. In *SODA*, Vol. 93, pp. 271–280.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, *23*(3), 345–405.

Auvolat, A., & Vincent, P. (2015). Clustering is efficient for approximate maximum inner product search. *CoRR*, *abs/1507.05910*.

Aytekin, A. M., & Aytekin, T. (2019). Real-time recommendation with locality sensitive hashing. *Journal of Intelligent Information Systems*, *53*(1), 1–26.

Bachrach, Y., Finkelstein, Y., Gilad-Bachrach, R., Katzir, L., Koenigstein, N., Nice, N., & Paquet, U. (2014). Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*, pp. 257–264. ACM.

Ballard, G., Kolda, T. G., Pinar, A., & Comandur, S. (2015). Diamond sampling for approximate maximum all-pairs dot-product (mad) search. *2015 IEEE International Conference on Data Mining*, 11–20.

Baranchuk, D., Persiyanov, D., Sinitsin, A., & Babenko, A. (2019). Learning to route in similarity graphs. In *ICML*.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517.

Bhowmik, A., Liu, N., Zhong, E., Bhaskar, B. N., & Rajan, S. (2016). Geometry aware mappings for high dimensional sparse factors. In *AISTATS*.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pp. 380–388, New York, NY, USA. ACM.

Cheng, P., Wang, S., Ma, J., Sun, J., & Xiong, H. (2017). Learning to recommend accurate and diverse items. In *Proceedings of the 26th international conference on World Wide Web*, pp. 183–192.

Cohen, E., & Lewis, D. D. (1997). Approximating matrix multiplication for pattern recognition tasks. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, pp. 682–691, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Covington, P., Adams, J., & Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pp. 191–198.

Dai, X., Yan, X., Ng, K. K., Liu, J., & Cheng, J. (2019). Norm-explicit quantization: Improving vector quantization for maximum inner product search. *arXiv preprint arXiv:1911.04654*.

Das, D., Sahoo, L., & Datta, S. (2017). A survey on recommendation system. *International Journal of Computer Applications*, *160*, 6–10.

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004a). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262. ACM.

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004b). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pp. 253–262, New York, NY, USA. ACM.

Dearholt, D., Gonzales, N., & Kurup, G. (1988). Monotonic search networks for computer vision databases. In *Twenty-Second Asilomar Conference on Signals, Systems and Computers*, Vol. 2, pp. 548–553. IEEE.

Deng, Z.-H., Huang, L., Wang, C.-D., Lai, J.-H., & Philip, S. Y. (2019). Deepcf: A unified framework of representation learning and matching function learning in recommender system. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 61–68.

Dong, Y., Indyk, P., Razenshteyn, I., & Wagner, T. (2020). Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations*.

Drineas, P., Kannan, R., & Mahoney, M. W. (2006). Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*, *36*(1), 132–157.

Du, C., & Wang, J. (2014). Inner product similarity search using compositional codes. *CoRR*, *abs/1406.4966*.

Fraccaro, M., Paquet, U., & Winther, O. (2016). Indexable probabilistic matrix factorization for maximum inner product search.. In *AAAI*, pp. 1554–1560.

Fu, C., & Cai, D. (2016). Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228*.

Fu, C., Xiang, C., Wang, C., & Cai, D. (2019). Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment*, *12*(5), 461–474.

Guo, G., Yang, E., Shen, L., Yang, X., & He, X. (2019). Discrete trust-aware matrix factorization for fast recommendation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 1380–1386. AAAI Press.

Guo, R., Kumar, S., Choromanski, K., & Simcha, D. (2016). Quantization based fast inner product search. In *Artificial Intelligence and Statistics*, pp. 482–490.

Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., & Kumar, S. (2020). Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pp. 3887–3896. PMLR.

Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., & Zhang, H. (2011). Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI*.

He, C., Parra, D., & Verbert, K. (2016). Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. *Expert Systems with Applications*, *56*, 9–27.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pp. 173–182.

He, X., Zhang, H., Kan, M.-Y., & Chua, T.-S. (2016). Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 549–558.

Hsieh, C.-K., Yang, L., Cui, Y., Lin, T.-Y., Belongie, S., & Estrin, D. (2017). Collaborative metric learning. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 193–201. International World Wide Web Conferences Steering Committee.

Huang, Q., Ma, G., Feng, J., Fang, Q., & Tung, A. K. H. (2018). Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*, KDD ’18, pp. 1561–1570, New York, NY, USA. ACM.

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC ’98, pp. 604–613, New York, NY, USA. ACM.

Jiang, J.-Y., Chen, P. H., Hsieh, C.-J., & Wang, W. (2020). Clustering and constructing user coresets to accelerate large-scale top-k recommender systems. *Proceedings of The Web Conference 2020*.

Jin, Z., Zhang, D., Hu, Y., Lin, S., Cai, D., & He, X. (2014). Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics*, *44*(11), 2167–2177.

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

Kaminskas, M., & Bridge, D. (2016). Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Trans. Interact. Intell. Syst.*, *7*, 2:1–2:42.

Kang, W.-C., & McAuley, J. (2019). Candidate generation with binary codes for large-scale top-n recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1523–1532.

Keivani, O., Sinha, K., & Ram, P. (2018). Improved maximum inner product search with better theoretical guarantee using randomized partition trees. *Machine Learning*, 1–26.

Khoshneshin, M., & Street, W. N. (2010). Collaborative filtering via euclidean embedding. In *Proceedings of the fourth ACM conference on Recommender systems*, pp. 87–94. ACM.

Koenigstein, N., Ram, P., & Shavitt, Y. (2012). Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pp. 535–544, New York, NY, USA. ACM.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37.

Le, D. D., & Lauw, H. W. (2016). Euclidean co-embedding of ordinal data for multi-type visualization. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pp. 396–404. SIAM.

Le, D. D., & Lauw, H. W. (2017). Indexable bayesian personalized ranking for efficient top-k recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1389–1398. ACM.

Le, D. D., & Lauw, H. W. (2020). Stochastically robust personalized ranking for lsh recommendation retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, pp. 4594–4601.

Li, H., Chan, T. N., Yiu, M. L., & Mamoulis, N. (2017). Fexipro: Fast and exact inner product retrieval in recommender systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 835–850. ACM.

Li, Y., Wang, S., Pan, Q., Peng, H., Yang, T., & Cambria, E. (2019). Learning binary codes with neural collaborative filtering for efficient recommendation systems. *Knowledge-Based Systems*, *172*, 64–75.

Lian, D., Liu, R., Ge, Y., Zheng, K., Xie, X., & Cao, L. (2017). Discrete content-aware matrix factorization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pp. 325–334, New York, NY, USA. ACM.

Liu, C., Lu, T., Wang, X., Cheng, Z., Sun, J., & Hoi, S. C. (2019a). Compositional coding for collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 145–154.

Liu, C., Wang, X., Lu, T., Zhu, W., Sun, J., & Hoi, S. (2019b). Discrete social recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 208–215.

Liu, H., He, X., Feng, F., Nie, L., Liu, R., & Zhang, H. (2018). Discrete factorization machines for fast feature-based recommendation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018 (IJCAI-18)*.

Liu, J., Yan, X., DAI, X., Li, Z., Cheng, J., & Yang, M.-C. (2020). Understanding and improving proximity graph based maximum inner product search. In *AAAI*.

Liu, R., Wu, T., & Mozafari, B. (2019). A bandit approach to maximum inner product search. *2019 AAAI Conference on Artificial Intelligence*.

Lorenzen, S. S., & Pham, N. (2020). Revisiting wedge sampling for budgeted maximum inner product search. In Hutter, F., Kersting, K., Lijffijt, J., & Valera, I. (Eds.), *Machine Learning and*

Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I, Vol. 12457 of *Lecture Notes in Computer Science*, pp. 439–455. Springer.

Malkov, Y. A., & Yashunin, D. A. (2019). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1.

McFee, B., & Lanckriet, G. R. G. (2011). Large-scale music similarity search with spatial trees. In *ISMIR*.

Morozov, S., & Babenko, A. (2018). Non-metric similarity graphs for maximum inner product search. In *Advances in Neural Information Processing Systems*, pp. 4722–4731.

Neyshabur, B., & Srebro, N. (2015). On symmetric and asymmetric lshs for inner product search. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1926–1934. JMLR.org.

Norouzi, M., Punjani, A., & Fleet, D. J. (2012). Fast search in hamming space with multi-index hashing. In *CVPR 2012*, pp. 3108–3115. IEEE.

Pan, W. (2016). A survey of transfer learning for collaborative recommendation with auxiliary data. *Neurocomputing*, *177*, 447–453.

Qin Ding, Hsiang-Fu Yu, C.-J. H. (2019). A fast sampling algorithm for maximum inner product search. *2019 International Conference on Artificial Intelligence and Statistics*, 11–20.

Qu, S., Guo, G., Liu, Y., Yao, Y., & Wei, W. (2020). Fast discrete factorization machine for personalized item recommendation. *Knowledge-Based Systems*, 105470.

Ram, P., & Gray, A. G. (2012). Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pp. 931–939, New York, NY, USA. ACM.

Ramlatchan, A., Yang, M., Liu, Q., Li, M., Wang, J., & Li, Y. (2018). A survey of matrix completion methods for recommendation systems. *Big Data Min. Anal.*, *1*, 308–323.

Rendle, S., Krichene, W., Zhang, L., & Anderson, J. (2020). Neural collaborative filtering vs. matrix factorization revisited. *Fourteenth ACM Conference on Recommender Systems*.

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pp. 452–461. AUAI Press.

Salakhutdinov, R., & Mnih, A. (2008). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pp. 880–887. ACM.

Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J., et al. (2001). Item-based collaborative filtering recommendation algorithms.. *Www*, *1*, 285–295.

Shrivastava, A., & Li, P. (2014). Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pp. 2321–2329.

Shrivastava, A., & Li, P. (2015). Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pp. 812–821, Arlington, Virginia, United States. AUAI Press.

Tan, S., Zhou, Z., Xu, Z., & Li, P. (2020). Fast item ranking under neural network based measures. In *WSDM*, pp. 591–599.

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

Teflioudi, C., Gemulla, R., & Mykytiuk, O. (2015). Lemp: Fast retrieval of large entries in a matrix product. In *SIGMOD*.

Vucetic, S., & Obradovic, Z. (2005). Collaborative filtering using a regression-based approach. *Knowledge and Information Systems*, *7*(1), 1–22.

Wang, C.-D., Deng, Z.-H., Lai, J.-H., & Philip, S. Y. (2018). Serendipitous recommendation in e-commerce using innovator-based collaborative filtering. *IEEE transactions on cybernetics*, *49*(7), 2678–2692.

Wang, H., Lian, D., & Ge, Y. (2019). Binarized collaborative filtering with distilling graph convolutional networks. In *IJCAI*.

Wu, X., Guo, R., Simcha, D., Dopson, D., & Kumar, S. (2019). Efficient inner product approximation in hybrid spaces..

Xiang, L., Tang, B., & Yang, C. (2019). Accelerating exact inner product retrieval by cpu-gpu systems. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1277–1280.

Xu, Y., Yang, Y., Wang, E., Han, J., Zhuang, F., Yu, Z., & Xiong, H. (2020). Neural serendipity recommendation: Exploring the balance between accuracy and novelty with sparse explicit feedback. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *14*(4), 1–25.

Yan, X., Li, J., Dai, X., Chen, H., & Cheng, J. (2018). Norm-ranging lsh for maximum inner product search. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 2956–2965, USA. Curran Associates Inc.

Yang, X., Guo, Y., Liu, Y., & Steck, H. (2014). A survey of collaborative filtering based social recommender systems. *Computer communications*, *41*, 1–10.

Yin, J., Liu, C., Li, J., Dai, B., Chen, Y.-c., Wu, M., & Sun, J. (2019). Online collaborative filtering with implicit feedback. In *International Conference on Database Systems for Advanced Applications*, pp. 433–448. Springer.

Yu, H.-F., Hsieh, C.-J., Lei, Q., & Dhillon, I. S. (2017). A greedy approach for budgeted maximum inner product search. In *Advances in Neural Information Processing Systems*, pp. 5453–5462.

Zhang, H., Shen, F., Liu, W., He, X., Luan, H., & Chua, T.-S. (2016). Discrete collaborative filtering. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 325–334. ACM.

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, *52*(1), 5.

Zhang, Y., Lian, D., & Yang, G. (2017). Discrete personalized ranking for fast collaborative filtering from implicit feedback.. In *AAAI*, pp. 1669–1675.

Zhang, Y., Wang, H., Lian, D., Tsang, I. W., Yin, H., & Yang, G. (2018a). Discrete ranking-based matrix factorization with self-paced learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*, KDD '18, pp. 2758–2767, New York, NY, USA. ACM.

Zhang, Y., Yin, H., Huang, Z., Du, X., Yang, G., & Lian, D. (2018b). Discrete deep learning for fast content-aware recommendation. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pp. 717–726, New York, NY, USA. ACM.

Zhang, Y., & Chen, X. (2020). Explainable recommendation: A survey and new perspectives. *Found. Trends Inf. Retr.*, *14*, 1–101.

Zhang, Z., Wang, Q., Ruan, L., & Si, L. (2014). Preference preserving hashing for efficient recommendation. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pp. 183–192. ACM.

Zhou, K., & Zha, H. (2012). Learning binary codes for collaborative filtering. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 498–506. ACM.

Zhou, Z., Tan, S., Xu, Z., & Li, P. (2019). Möbius transformation for fast inner product search on graph. In *Advances in Neural Information Processing Systems*, pp. 8216–8227.

Zhu, H., Li, X., Zhang, P., Li, G., He, J., Li, H., & Gai, K. (2018). Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1079–1088.

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Survey*, *38*(2).