Clustering and Constructing User Coresets to Accelerate Large-scale Top-K Recommender Systems

Jyun-Yu Jiang[*], Patrick H. Chen[*], Cho-Jui Hsieh and Wei Wang
Department of Computer Science, University of California, Los Angeles, CA, USA
{jyunyu,patrickchen,chohsieh,weiwang}@cs.ucla.edu

ABSTRACT

Top-K recommender systems aim to generate few but satisfactory personalized recommendations for various practical applications, such as item recommendation for e-commerce and link prediction for social networks. However, the numbers of users and items can be enormous, thereby leading to myriad potential recommendations as well as the bottleneck in evaluating and ranking all possibilities. Existing Maximum Inner Product Search (MIPS) based methods treat the item ranking problem for each user independently and the relationship between users has not been explored. In this paper, we propose a novel model for clustering and navigating for top-K recommenders (CANTOR) to expedite the computation of top-K recommendations based on latent factor models. A clustering-based framework is first presented to leverage user relationships to partition users into affinity groups, each of which contains users with similar preferences. CANTOR then derives a coreset of representative vectors for each affinity group by constructing a set cover with a theoretically guaranteed difference to user latent vectors. Using these representative vectors in the coreset, approximate nearest neighbor search is then applied to obtain a small set of candidate items for each affinity group to be used when computing recommendations for each user in the affinity group. This approach can significantly reduce the computation without compromising the quality of the recommendations. Extensive experiments are conducted on six publicly available large-scale real-world datasets for item recommendation and personalized link prediction. The experimental results demonstrate that CANTOR significantly speeds up matrix factorization models with high precision. For instance, CANTOR can achieve **355.1x** speedup for inferring recommendations in a million-user network with 99.5% precision@1 to the original system while the state-of-the-art method can only obtain **93.7x** speedup with 99.0% precision@1.

KEYWORDS

Large-scale top-K recommender systems; Latent factor models; Approximate nearest neighbor search

ACM Reference Format:
Jyun-Yu Jiang[*], Patrick H. Chen[*], Cho-Jui Hsieh and Wei Wang. 2020. Clustering and Constructing User Coresets to Accelerate Large-scale Top-K Recommender Systems. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380283

[*]Equal contribution.

1 INTRODUCTION

Building large-scale personalized recommender systems has already become a core problem in many online applications since the explosive growth of internet users in the recent decade. For example, user-item recommender systems achieve many successes in e-commerce markets [23] while link prediction in social networks can be treated as a variant of recommender systems [2, 33]. To establish recommender systems, latent factor models for collaborative filtering have become popular because of their effectiveness and simplicity. More precisely, each user or item can be represented as a low-dimensional vector in a latent space so that the inner products between user and item vectors are capable of indicating the user-item preferences. Furthermore, these latent vectors can then be learned by optimizing a loss function with sufficient training data. For instance, matrix factorization [19] has been empirically shown to outperform conventional nearest-neighbor based approaches in a wide range of application domains [11].

After obtaining user and item latent vectors, to make item recommendations for each user, recommender systems need to calculate the inner products for all user-item pairs. Although learning user and item latent vectors is efficient and scalable for most existing models, recommender systems can take an enormous amount of time in evaluating all user-item pairs. More specifically, the time complexity of learning latent vectors is only proportional to the number of user-item pairs in the training data which is a small subset of all possible user-item pairs, but finding the top recommendations entails examining all $O(mn)$ inner products between all m users and n items. As a result, the quadratic complexity becomes a hurdle for large-scale recommender systems. For example, it can take more than a day to compute and rank all preference scores, and consequently the systems cannot be updated on a daily basis [12]. In order to make large-scale recommender systems practical, it is critical to accelerate the process of computing and ranking the inner products of user and item latent vectors, in order to efficiently obtain the top-K recommendations for all users.

To accelerate the computation of inner products, the maximum inner product search (MIPS) [27, 31, 34] is one of the feasible approaches. Locality sensitive hashing (LSH) [16] and PCA tree [32] may be applied to solve MIPS after reducing the problem to nearest-neighbor search. To reduce the computation for making recommendations for a given user, one may find a small group of candidate items whose latent vectors have large inner products with the user's latent vector using clustering algorithms [7], or sort entries of each dimension in the latent vectors separately by some greedy algorithms [12, 34]. In essence, most of the existing MIPS algorithms adopt a two-stage strategy, decomposing the computation into a preparation process and a prediction process. In the preparation stage, these methods will construct suitable data structures [34] or reduce the number of ranking candidates [7], and these prepared data structures are used to conduct efficient maximum inner product search for query vectors in the inference stage. However, most of these existing MIPS algorithms have the following two weaknesses, making them often impractical for real applications: (1) they only focus on optimizing the inference speed for a given user at the cost of considerable preparation time, but for recommender systems, the overall execution time (including both preparation and inference time) matters more because the system needs to be re-trained frequently as new data arrive. (2) All the MIPS approaches aim to quickly identify the top item set for *any* query vector. However, in recommender systems queries are not arbitrary vectors. They are user latent factors and usually have very strong *clustering structure*, which is ignored in most of the MIPS algorithms.

In order to speed up the overall execution time, our main idea is to exploit the relationships between users. More precisely, users with similar latent factors are more likely to share similar item preferences which may be reflected by their high inner products. However, existing methods for accelerating recommender systems do not consider user relationships and the distribution of user latent vectors. For instance, existing greedy strategies [12, 34] only consider the values of item latent vectors. Some studies based on proximity graphs [26, 35] and clustering algorithms [7] also solely reduce the search space of items. In the inference stage, these approaches treat the recommendation to each user as an independent query to the data structures and algorithms. As a consequence, it can be extremely time-consuming, especially with myriad users and enormous spaces of candidate items.

In this paper, we propose a novel model for clustering and navigating for top-K recommenders (CANTOR) that leverages the knowledge of user relationships to accelerate the process of generating recommendations for all users with a given latent factor model. CANTOR consists of two stages: preparation and prediction. In the preparation stage, we aim to cluster users sharing similar interests into affinity groups and compute a small set of preferred items for each affinity group. More specifically, the user vectors (generated from a given latent factor model) are used in clustering affinity groups. To further accelerate the preparation time, a user coreset of few representative vectors are derived for each affinity group, and are used to obtain a small set of preferred items for users in this group by an efficient approximate nearest neighbor search algorithm. Finally, in the prediction stage, the top-K recommendations for each user can be retrieved by ranking these preferred items of the corresponding affinity group, which can be done much more efficiently than evaluating and ranking all items. We summarize our contributions as follows:

- To the best of our knowledge, this paper is the first work to focus on the preparation time and user relationships for accelerating the prediction process of large-scale top-K recommender systems. Enhancing preparation time is essential for recommender systems to accommodate massive incoming data in a timely manner, which has not been studied previously.
- Clustering users into affinity groups based on the distribution of user latent vectors provides significant speedup of the prediction process, compared to conventional approaches that independently deal with each user. The representative vectors of the affinity groups offer a theoretically guaranteed precision for users with similar preferences. Approximate nearest neighbor search is applied to efficiently retrieve the satisfactory recommendations for each user from a small set of candidate items.
- Experiments conducted on six publicly available datasets demonstrate that CANTOR can significantly accelerate large-scale top-K recommender systems for both item recommendation and personalized link prediction. An in-depth analysis then indicates the robustness and effectiveness of the proposed framework.

2 PROBLEM STATEMENT

In this section, we first introduce the notations and then formally define the objective of this paper. Suppose that we have an incomplete $m \times n$ one-class matrix $\boldsymbol{R} = \{R_{ij}\} \in \{0, 1\}^{m \times n}$, where m and n are the numbers of users and items in the system. $R_{ij} = 1$ if user i prefers item j in the training data; otherwise, $R_{ij} = 0$. Based on $\boldsymbol{R}$, a matrix factorization based algorithm learns d-dimensional user and item latent vectors, denoted by $\boldsymbol{P} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{Q} \in \mathbb{R}^{n \times d}$ respectively, where $\hat{\boldsymbol{R}} = \boldsymbol{P}\boldsymbol{Q}^T \in \mathbb{R}^{m \times n}$ reflects the underlying preferences. To compute top-K recommendations for each user, we need to find items with the K highest scores among $\hat{\boldsymbol{R}}(i) = \{\hat{R}_{ij'} \mid j' \in 1 \ldots m\}$. Note that $m = n$ for personalized link prediction in social networks, where the goal is to suggest other users as recommended items.

Although matrix factorization models can be learned expeditiously when $\boldsymbol{R}$ is sparse, inferring the top-K recommendations requires computing and sorting the scores $\hat{R}_{ij}$ of all items j for each user i. As a result, the inference process can be time-consuming with an $O(nmd)$ time complexity which becomes intractable when n and m are large. To address this problem, the goal of this paper is to speed up the inference time of top-K recommenders with a high precision. More specifically, given the trained matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$, we aim to propose an efficient approach that approximates the top-K recommended items for each user.

3 CONSTRUCTING USER CORESETS FOR TOP-K RECOMMENDER SYSTEMS

In this section, we present CANTOR for accelerating top-K recommender systems, starting with several key preliminary ideas.

3.1 Preliminary

In order to leverage the relationship between users, we first formally define the *affinity groups* of users in recommender systems as follows:

Definition 1. (Affinity Group) An *affinity group* $\boldsymbol{A}_t$ is a set of users sharing similar interests in items. Even though any similarity metrics may be used, in this paper, we adopt cosine similarity as the metric to define the affinity groups.

By this definition, the sets of satisfactory recommendations should be similar for users in the same affinity group. This suggests that the top recommendations for all users in an affinity group are confined to a small subset of the items and such item subset can be learned by

Table 1: Summary of notations and their descriptions.

Notation	Descriptions
m, n	numbers of users and items
d	number of dimensions for latent vectors
K	number of top recommendations
$\boldsymbol{R} \in \mathbb{R}^{m \times n}$	one-class preference matrix
$\boldsymbol{P} \in \mathbb{R}^{m \times d}$	user latent vectors for all users
$\boldsymbol{Q} \in \mathbb{R}^{n \times d}$	item latent vectors for all items
$\hat{\boldsymbol{P}} \in \mathbb{R}^{u \times d}$	sampled user latent vectors
u	number of sub-sampled users
r	number of affinity groups for m users
$\boldsymbol{A}$	set of r affinity groups, where $\boldsymbol{A} = \{\boldsymbol{A}_t \mid t = 1 \dots r\}$
$\boldsymbol{v}_t$	centroid vectors for the affinity group $\boldsymbol{A}_t$
$z(p)$	affinity group indicator for the user vector p
$\boldsymbol{P}_t$	latent vectors of users in the affinity group $\boldsymbol{A}_t$
$C(p_i, K)$	indexes of top-K items with full $p_i^T\boldsymbol{Q}$ evaluation.
$\boldsymbol{s}_t$	the user coreset for the affinity group $\boldsymbol{A}_t$
$\mathcal{N}_{s_t}(p_i)$	the nearest coreset representative in $\boldsymbol{s}_t$ for p_i
$\boldsymbol{c}_t$	reduced item set of top-K items for the affinity group $\boldsymbol{A}_t$
ϵ	similarity threshold in adaptive representative selection
w	number of new representatives for outliers
G	proximity graph of the item vectors
efs	the size of dynamic lists of nearest neighbors

examining only a few carefully selected users in the group, leading to the following definition of the *preferred item set*.

Definition 2. (Preferred Item Set) A *preferred item set* $\boldsymbol{c}$ for an affinity group is a set of (potentially) satisfactory items for the users in the group, and the size of the preferred item set is usually much smaller than the total number of items, i.e., $|\boldsymbol{c}| \ll n$.

Therefore, we only need to examine the preferred item set to generate top recommendations, leading to significant time saving overs the alternative of examining all items.

In order to robustly generate the preferred item set for each affinity group, we generate a few representatives from the group to compute the preferred item set. This is statistically more robust than using only the "centroid" user in the latent space, and is more computationally efficient than using all users in the group.

Definition 3. (User Coreset of an Affinity Group) A δ-*user coreset* $\boldsymbol{s}_t$ of an affinity group $\boldsymbol{A}_t$ is a (small) set of latent representative vectors to preserve the item preference of the users in $\boldsymbol{A}_t$ such that $\forall q \in \boldsymbol{Q}, i \in \boldsymbol{A}_t$:

$$\left| p_i q^T - \mathcal{N}_{s_t}(p_i)\, q^T \right| \leq \delta,$$

where $\mathcal{N}_{s_t}(p_i) \in \boldsymbol{s}_t$ is the nearest coreset representative for p_i; $\delta > 0$ is a small enough constant.

The user interests in the affinity group can be well captured by the representative vectors in the user coreset. Note that the representative vectors do not have to be identical to actual user latent vectors in the group.

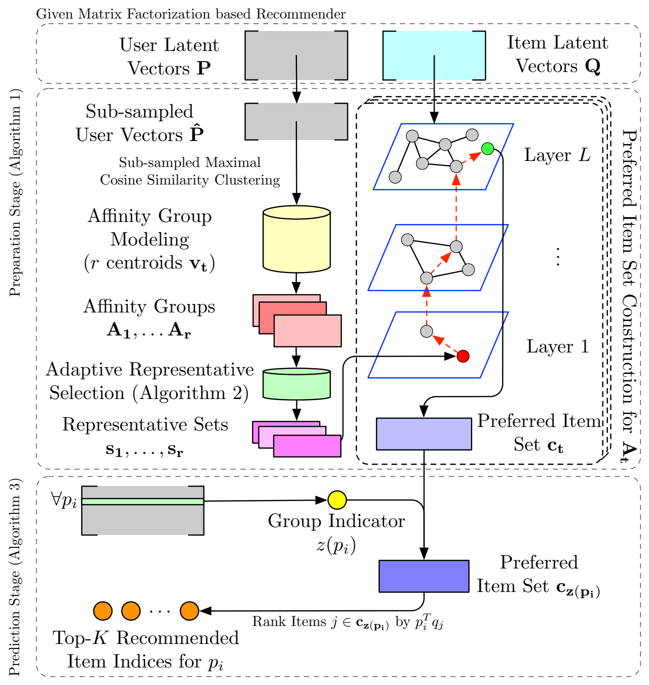

Given Matrix Factorization based Recommender
User Latent Vectors $\mathbf{P}$
Item Latent Vectors $\mathbf{Q}$
Preparation Stage (Algorithm 1)
Sub-sampled User Vectors $\hat{\mathbf{P}}$
Sub-sampled Maximal Cosine Similarity Clustering
Affinity Group Modeling (r centroids $\mathbf{v_t}$)
Affinity Groups $\mathbf{A_1}, \dots \mathbf{A_r}$
Adaptive Representative Selection (Algorithm 2)
Representative Sets $\mathbf{s_1}, \dots, \mathbf{s_r}$
Layer L
⋮
Layer 1
Preferred Item Set $\mathbf{c_t}$
Preferred Item Set Construction for $\mathbf{A_t}$
Prediction Stage (Algorithm 3)
$\forall p_i$
Group Indicator $z(p_i)$
Preferred Item Set $\mathbf{c_{z(p_i)}}$
Rank Items $j \in \mathbf{c_{z(p_i)}}$ by $p_i^T q_j$
Top-K Recommended Item Indices for p_i

Figure 1: The general framework of the proposed clustering and navigating for top-K recommenders (CANTOR).

3.2 Framework Overview

Figure 1 shows the general framework of CANTOR. The framework consists of two stages: preparation and prediction. In the preparation stage as shown in Algorithm 1, the m user latent vectors $\boldsymbol{P}$ are first sub-sampled as $\hat{\boldsymbol{P}}$ and clustered into r affinity groups $\boldsymbol{A}_t$ with a centroid vector $\boldsymbol{v}_t$ computed from the corresponding user vectors $\boldsymbol{P}_t$, where $t = 1 \dots r$. For each affinity group $\boldsymbol{A}_t$, we aim at deriving a small user coreset $\boldsymbol{s}_t$. To do so, we propose an adaptive representative selection method (Algorithm 2) to greedily construct a set cover of user latent vectors for each affinity group after mathematically proving that the set covers can be the coresets of affinity groups. Finally, a small preferred item set $\boldsymbol{c}_t$ can be obtained by approximate nearest neighbor search using its coreset $\boldsymbol{s}_t$ for each affinity group. In the prediction stage (Algorithm 4), CANTOR first locates the corresponding affinity group $\boldsymbol{A}_t$ for each user and then ranks the small number of items in the preferred item set $\boldsymbol{c}_t$, thereby efficiently providing satisfactory recommendations. In sum, Table 1 further summarizes all major notations in this paper.

3.3 Preparation Stage

To overcome the hurdle of extremely long preparation time experienced by conventional methods, we propose to exploit similarities between user vectors in the latent space for acceleration as shown in Algorithm 1.

Affinity Group Modeling by User Clustering. Most of the conventional algorithms only rely on similarities of item latent vectors [30] and proximity graphs [14, 35] to accelerate the recommendation, and have not used the relationships between users and

Algorithm 1: Preparation Process for CANTOR

```
Input: User latent vectors $\boldsymbol{P}$; item latent vectors $\boldsymbol{Q}$; degree
       of each user $deg_{i=1}^{m}$; the number of desired
       recommendation $K$
Output: Centroid vectors $\boldsymbol{v}_t$ and preferred item sets $\boldsymbol{c}_t$ for
        each affinity cluster $\boldsymbol{A}_t$ for $t = 1 \dots r$.
1  Hyperparameter: Number of affinity groups $r$; small
   world graph search size $efs$.; number of sub-sampled users $u$;
2  $\hat{P}$ = Multinomial_Sampling($\boldsymbol{P}$, $deg_{i=1}^{m}$, $u$); $\hat{P} \in \mathbb{R}^{u \times d}$ ;
3  $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_r = 0$; $I = 0, I \in \mathbb{R}^{u \times 1}$ ;
4  repeat
5      for $i = 1, \cdots, r$ do
6          $\boldsymbol{v}_i = \sum_{j \in \{j | I[j] = i\}} \hat{\boldsymbol{P}}[j]$ ;
7          $\boldsymbol{v}_i = \boldsymbol{v}_i \,/\, \|\boldsymbol{v}_i\|_2$ ;
8      I = $\arg\max_t \boldsymbol{v}_t^T \hat{\boldsymbol{P}}$ ;
9  until Convergence;
10 G = CreateProximityGraph($\boldsymbol{Q}$, $efs$);
11 $\boldsymbol{c}_1, \dots, \boldsymbol{c}_r = \emptyset, \dots, \emptyset$ ;
12 I = $\arg\max_t \boldsymbol{v}_t^T \hat{\boldsymbol{P}}$ ;
13 for $i = 1, \cdots, r$ do
14     $\hat{\boldsymbol{P}}_i = \{p_j \mid p_j \in \hat{\boldsymbol{P}}, I[j] = i\}$ ;
15     $\boldsymbol{s}_i$ = AdaptiveClustering($\hat{\boldsymbol{P}}_i$, $\epsilon$, $w$) ;
16     for $q \in \boldsymbol{s}_i$ do
17         $\hat{I}_i$ = QueryProximityGraph(G, $\boldsymbol{s}$, $K$) ;
18         $\boldsymbol{c}_i = \boldsymbol{c}_t \cup \hat{I}_i$ ;
19 return $\boldsymbol{c}_t$,$\boldsymbol{v}_t$ for all $t = 1, \cdots, r$.
```

the distributions of user latent vectors in this endeavor. To exploit the knowledge of user relationships, we propose a clustering based framework to model user affinity groups.

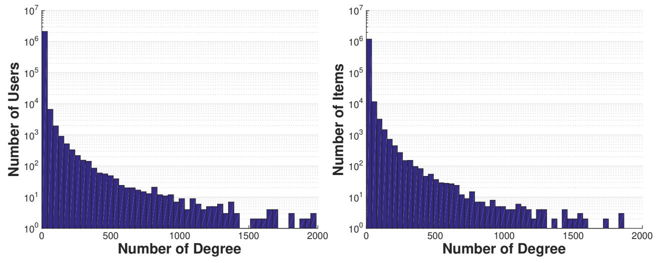
Let r be the number of affinity groups for all m users, where r is a hyperparameter in CANTOR. We partition all m users into r disjoint clusters as the affinity groups $\boldsymbol{A} = \{\boldsymbol{A}_t \mid t = 1 \dots r\}$ based on the user latent vectors $\boldsymbol{P} = \{p_i \mid i = 1 \dots m\}$. In addition, each affinity group $\boldsymbol{A}_t$ has a centroid vector $\boldsymbol{v}_t \in \mathbb{R}^d$ in the latent space. Each user i with the latent vector p_i belongs to $\boldsymbol{A}_{z(p_i)}$, where $z(p_i)$ is the affinity group indicator represented as:

$$z(p_i) = \arg\max_r \boldsymbol{v}_r^T p_i. \tag{1}$$

Let $C(p_i, K)$ be the top-K preferred items for user i which is defined as:

$$\{j \mid p_i^T q_j \geq p_i^T q_{j'}, \forall j' \notin C(p_i, K) \text{ and } |C(p_i, K)| = K\},$$

where $q_j \in \boldsymbol{Q}$ is the latent vector of item j. Intuitively, if users i and k are in the same affinity group, their preferred sets $C(p_i, K)$ and $C(p_k, K)$ may have substantial overlap because of their similar interests. This motivates us to compute a preferred item set $\boldsymbol{c}_t$ for users in the same affinity group $\boldsymbol{A}_t$ so that each $\boldsymbol{c}_t$ contains only a small subset of all n items, i.e., $|\boldsymbol{c}_t| \ll n$. Instead of computing the inner products between p_k and all item latent factors $q \in \boldsymbol{Q}$, we can narrow down the candidate set to be $\boldsymbol{c}_t$, and only evaluate the items in $\boldsymbol{c}_t$ to find the top-K predictions for user k.



(a) User Distribution **(b) Item Distribution**

Figure 2: The distributions of users and items over different degrees in the Amazon dataset.

Since our task is to accelerate the maximum inner product search, the centriod vector $\boldsymbol{v}_t$ for each affinity group $\boldsymbol{A}_t$ can then be updated by the maximum cosine similarity criteria as:

$$\boldsymbol{v}_t = \frac{\sum_{i=1}^{|\boldsymbol{P}_t|} \boldsymbol{P}_{ti}}{\|\sum_{i=1}^{|\boldsymbol{P}_t|} \boldsymbol{P}_{ti}\|_2}, \tag{2}$$

where $\boldsymbol{P}_t = \{p_i \mid z(p_i) = t\}$ contains the latent vectors of users that belong to the affinity group $\boldsymbol{A}_t$. Therefore, each affinity group $\boldsymbol{A}_t$ can obtain a centroid vector $\boldsymbol{v}_t$ by iteratively running Equations (1) and (2). However, iteratively performing Equations (1) and (2) can still cost a long computational time when the number of users m is large. To address this issue, we propose to sub-sample a portion of the m user latent vectors to learn the centroid vectors. Moreover, we sample the latent vectors based on the degree distribution in the one-class matrix $\boldsymbol{R}$. For example, Figure 2a shows that degree distribution of users usually follows a power-law distribution. Hence, instead of using a uniform sampling, we sample user i with a probability proportional to a log function of its degree as:

$$P(X = i) \propto \log \sum_{j=1}^{n} R_{ij}, \tag{3}$$

where X denotes the random variable of the target sampling process. We will later show in Theorem 2 that error of approximation based on sub-sampling will be asymptotically bounded.

After learning the centroids $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_r \in \mathbb{R}^d$ and the corresponding user latent vectors $\boldsymbol{P}_1, \cdots, \boldsymbol{P}_r$ for r affinity groups $\boldsymbol{A}_1, \cdots, \boldsymbol{A}_r$, the preferred item set $\boldsymbol{c}_t$ for each group $\boldsymbol{A}_t$ can be constructed so that user vectors $\boldsymbol{P}_t$ only need to search over this set of preferred items for top recommendations. However, the naïve approach to generate $\boldsymbol{c}_t$ would require $O(nd)$ operations to examine all n items in order to derive the top candidates for each user in $\boldsymbol{A}_t$. Each affinity group $\boldsymbol{A}_t$ would need $O(|\boldsymbol{P}_t|nd)$ operations for considering all $|\boldsymbol{P}_t|$ users in the group to construct the preferred item set $\boldsymbol{c}_t$.

Coreset Construction as Finding a Set Cover. To accelerate the process of constructing the preferred item set $\boldsymbol{c}_t$ for an affinity group A_t, we want to find a δ-user coreset of A_t, and use it only instead of whole $\boldsymbol{A}_t$ to construct $\boldsymbol{c}_t$. We achieve this by first defining the idea of ϵ-set cover, and then show that each ϵ-set cover corresponds to a δ-coreset.

Definition 4. (ϵ-Set Cover) $\boldsymbol{s}_t$ is an ϵ-cover of $\boldsymbol{P}_t$ if $\exists \mathcal{N}_{s_t}(p) \in \boldsymbol{s}_t$ so that $\mathcal{N}_{s_t}(p)p^T \geq \epsilon$ for all $p \in \boldsymbol{P}_t$, where ϵ is a real number, and $\mathcal{N}_{s_t}(p_i) \in \boldsymbol{s}_t$ denotes the nearest vector in $\boldsymbol{s}_t$ of p_i.

Theorem 1. Given an ϵ-cover $\boldsymbol{s}_t$ of $\boldsymbol{A}_t$, there exists a δ such that ϵ-cover $\boldsymbol{s}_t$ is a δ-user coreset of the affinity group $\boldsymbol{A}_t$.

The proof is shown in Appendix A. Therefore, we could construct a user coreset with an arbitrarily small δ by finding a cover with a greater ϵ.

Another nice property is that we could find an ϵ-set cover on sampled subset of $\boldsymbol{P}$ and generalize asymptotically with bounded error. Denote $\boldsymbol{P}_{\boldsymbol{A}_t}$ to be same sampling process of $\boldsymbol{P}$ generating user vectors p_i belonging to $\boldsymbol{A}_t$. We will have following result:

Theorem 2. For an affinity group $\boldsymbol{A}_t$, given any query q, an ϵ-cover of k samples $\{p_i\}$ drawn from $\boldsymbol{P}_{\boldsymbol{A}_t}$ would satisfy following inequality with probability at least $1-\gamma$:

$$\min_i \left(\left| \mathcal{N}_{s_t}(p_i)\, q^T - p_t q^T \right| \right) \leq \delta + \sqrt{\frac{2\log(1/\gamma)}{k}}.$$

Note that we demonstrate the proof in Appendix B.

Theorem 2 indicates that we could construct an ϵ-cover of sub-sampled vectors to have an asymptotically guaranteed difference of inner-product values to true distributions within the same affinity group. Consequently, our task becomes finding an ϵ-cover of all $\boldsymbol{A}_t$s and constructing the preferred item set $\boldsymbol{c}_t$ of it. Hence, we propose a fast adaptive representative selection method to efficiently construct an ϵ-cover with sub-sampled user latent vectors for each affinity group as summarized in Algorithm 2. For each affinity group $\boldsymbol{A}_t$, the adaptive representative selection method is applied to obtain a few representative ϵ-cover $\boldsymbol{s}_t$. If there exists at least one user whose latent vector has cosine similarity lower than ϵ to all representative vectors, the algorithm iteratively generates more representatives until every user has high cosine similarity to at least one representative vector. As a result, the number of ϵ-cover $|\boldsymbol{s}_t|$ must be less than or equal to the number of user vectors in the cluster $|\boldsymbol{P}_t|$, and in practice, $|\boldsymbol{s}_t| \ll |\boldsymbol{P}_t|$ in most cases. Note that the adaptive representative selection method is applied on each affinity group $\boldsymbol{A}_t$ independently. Next, the ϵ-cover $\boldsymbol{s}_t$ will be utilized to construct the preferred item set to reduce complexity from $O(|\boldsymbol{P}_t|nd)$ to $O(|\boldsymbol{s}_t|nd)$.

Algorithm 2: Adaptive Representative Selection

```
Input: User latent vectors for an affinity group $P$, the
       number of iterations $T$, the threshold $\epsilon$, the number
       of new representatives $w$ ;
Output: Representative vectors $s$.
1  Initialize $s = \emptyset$ ;
2  I = $\arg\max_t s^T P$ ;
3  repeat
4      for $i = 1 \ldots |s|$ do
5          $s_i = \sum_{j \in \{j | I[j]=i\}} P[j]$ ;
6          $s_i = s_i \,/\, \|s_i\|_2$ ;
7      I = $\arg\max_t s^T P$ ;
8      Outliers = $\{j | s_{I[j]}^T P_j < \epsilon\}$ ;
9      for $j \in$ Outliers do
10         Draw $i$ from $1 \ldots w$ ;
11         I[$j$] = $|s|$ + i ;
12     if Outliers $\neq \emptyset$ then
13         Append $w$ vectors to $s$ ;
14 until Outliers = $\emptyset$;
15 Outliers = $\{j | s_{I[j]}^T P_j < \epsilon\}$ ;
16 Append $P_{\text{Outliers}}$ to $s$ ;
17 return $s$.
```

Proximity Graph Navigation for Preferred Item Set Construction. To avoid examining all n items ($O(nd)$ complexity) in preferred item set construction, we apply an approximate nearest neighbor search (ANNS) method to accelerate the computation. We adopt a model based on proximity graphs [26, 35] which has shown the state-of-the-art performance in ANNS. Specifically, a proximity graph is generated in which item vectors are nodes and nodes of similar item vectors are connected by edges. Since the item degree in recommender systems tends to follow a power-law distribution as illustrated in Figure 2b, this proximity graph has the small world properties [5] with sparse edges that offer high reachability between nodes. Hence, we apply the model of hierarchical navigable small world graphs [25, 26] to obtain the preferred item set $\boldsymbol{c}_t$ for each affinity group $\boldsymbol{A}_t$.

To construct the proximity graph of item vectors $\boldsymbol{Q}$ as a hierarchical small world graph G, we iteratively insert the item vectors into the graph, where each node q has a list $\boldsymbol{E}(q)$ of at most *efs* approximate nearest neighbors that could be dynamically updated when inserting other item vectors, where *efs* is a hyperparameter. In addition, the edges in the graph are organized as a hierarchy so that edges connecting items that have a high inner product value of their corresponding item vectors are at the bottom layers and edges connecting items whose vectors have low inner product values are at the top layers, thereby shrinking the search spaces for nearest neighbors. Let $L(e)$ denote the corresponding layer of edge e. Given two edges e_i and e_j, if $L(e_i) > L(e_j)$, then the nodes connected by edge e_i has a smaller inner product score than that of edge e_j. For simplicity, let $\boldsymbol{E}(q,l)$ denote the list of nodes connected to node q by edges in the l-th layer. Finally, the hierarchical small world graph G of item vectors $\boldsymbol{Q}$ can be constructed in $O(dn\log n)$ [26, 35], where n is the total number of items; *efs* is treated a constant hyperparameter. Note that *efs* controls the trade-off between efficiency and accuracy for searching nearest neighbors because it decides the size of search space and the potential coverage of real nearest neighbors.

The hierarchical small world graph G provides the capability of efficiently querying K nearest neighbors of a vector q with a hierarchical greedy search algorithm. More specifically, we can greedily traverse the graph G by navigating the query vector from the bottom layer to the top layer to derive K approximate nearest neighbors to q as shown in Algorithm 3 with a $O(d\log n)$ time complexity for each query. For each affinity group $\boldsymbol{A}_t$, we perform a small world graph query to approximate $C(\boldsymbol{s}_{t,i}, K)$ for each representative vector $\boldsymbol{s}_{t,i} \in \boldsymbol{s}_t$. The preferred item set $\boldsymbol{c}_t$ can then be constructed by taking the union operation to individual top-k sets

Algorithm 3: QueryProximityGraph

```
Input: Hierarchical small world graph $G$; the query vector
       $q$; the number of the output approximate nearest
       neighbors $K$
Output: $K$ nearest vectors in $G$
1 $p$ = Randomly select an entry node in $G$ ;
2 for $l = 1$ to $L$ do
3     $p = \arg\max_{r \in \{p' | p' E(p,l)\}} q^T \boldsymbol{r}$;
4 return $K$ Nearest Nodes in $\boldsymbol{E}(p, L)$ to $q$ ;
```

Algorithm 4: Prediction Process for CANTOR

```
Input: User latent vectors $\boldsymbol{p}_i$; item latent vectors $\boldsymbol{Q}$;
       Number of top recommendations $K$
Output: The indices of estimated top-$K$ recommendations
        for the user $i$.
1 $z(p_i) = \arg\max_t \boldsymbol{v}_t^T p_i$ ;
2 logits = $p_i^T \boldsymbol{Q} \left[\boldsymbol{c}_{z(p_i)}\right]$ ;
3 topIndices = argsort(logits, K) ;
4 return topIndices.
```

as

$$\boldsymbol{c}_t = \bigcup_{i=1}^{|s_t|} C(s_{t,i}, K). \tag{4}$$

3.4 Prediction Stage

To predict top recommendations for a user with the latent vector p_i, CANTOR relies on the clustering model parameterized by the centroid vector $\boldsymbol{v}_t \in \mathbb{R}^d$ and the preferred item set $\boldsymbol{c}_t$ for each affinity group $\boldsymbol{A}_t$. More precisely, we first compute the affinity group indicator $z(p)$ as:

$$z(p_i) = \arg\max_r \boldsymbol{v}_r^T p_i, \tag{5}$$

and evaluate full vector matrix product $p^T \boldsymbol{Q}_I$ over the corresponding item vectors of the preferred item set $\boldsymbol{Q}_I, I = \{j | j \in \boldsymbol{c}_{z(p_i)}\}$. The computed results are then sorted to provide the final top-K recommendations for the user. Algorithm 4 shows the procedure of the prediction process.

4 EXPERIMENTS

In this section, we conduct extensive experiments and in-depth analysis to demonstrate the performance of CANTOR.

4.1 Experimental Settings

Experimental Datasets. We evaluate the performance in two common tasks: item recommendation and personalized link prediction, using six publicly available real-world large-scale datasets as shown in Table 2. For the task of item recommendation, the MovieLens 20M dataset (MovieLens) [15] consists of 20-million ratings between users and movies; the Last.fm 360K dataset (Last.fm) [6] contains the preferred artists of about 360K users; the dataset of Amazon ratings (Amazon) includes ratings between millions of users and items [20]. For the task of personalized link prediction, we follow the previous study [12] to construct three social networks among users: YouTube, Flickr, and Wikipedia [20]. Note that four of the six experimental datasets, Amazon, YouTube, Flickr, and Wikipedia, are available in the Koblenz Network Collection [20].

Table 2: The statistics of six experimental datasets. Note that the personalized link prediction problem can be mapped to an item recommendation problem by treating each user as an item and recommending other users to a user in a similar way to that of recommending items to a user, and in this case the numbers of users and items are equal.

Task	Item Recommendation		
Dataset	MovieLens	Last.fm	Amazon
#(Users)	138,493	359,293	2,146,057
#(Items)	26,744	160,153	1,230,915
Task	**Personalized Link Prediction**		
Dataset	YouTube	Flickr	Wikipedia
#(Users)	1,503,841	1,580,291	1,682,759
#(Items)	1,503,841	1,580,291	1,682,759

Evaluation Metrics. To measure the quality of an approximate algorithm for top-K recommendation we evaluate the top-K approximated recommendations with Precision@K (P@K), which is defined by

$$\frac{1}{m} \sum_i \frac{\left|Y_K^i \cap S_K^i\right|}{K},$$

where Y_K^i and S_K^i are the top-K items computed by the approximate algorithm and full inner-product computations for user i; m is the number of users. To measure the speed of each algorithm, we report the speedup defined by the ratio of wall clock time consumed by the full set of $O(mn)$ inner products to find the top-K recommendations divided by the wall clock time of the approximate algorithm.

Baseline Methods. To evaluate our proposed CANTOR, we consider the following five algorithms as the baseline methods for comparison.

- ϵ-approximate link prediction (ϵ-Approx) [12] sorts entries of the latent factor for each dimension to construct a guaranteed approximation of full inner products.
- Greedy-MIPS (GMIPS) [34] is a greedy algorithm for solving the MIPS problem with a trade-off controlled by varying a computational budget parameter in the algorithm.
- SVD-softmax (SVDS) [30] is a low-rank approximation approach for fast softmax computation. We vary the rank of SVD to control the trade-off between prediction speed and accuracy.
- Fast Graph Decoder (FGD) [35] directly applies small world graph on all items $\boldsymbol{Q}$ and navigates to derive recommended items with user latent vectors as queries on the proximity graph. It also serves a direct baseline of only using proximity graph navigation.
- Learning to Screen (L2S) [7] is the first clustering-based method on fast prediction in NLP tasks with the state-of-the-art results on inference time but suffers from long preparation time. CANTOR is inspired by the clustering step in L2S, thus L2S serves as a

Table 3: Comparisons of top-K recommendation results on six datasets in two tasks. Note that P@K measures the precision of approximating the top-K recommendations of full inner-product computations. SU indicates the ratio of speedup based on the original full inner product time of inferring top-K recommendations. For example, 9.4x means the computation time of the method is 9.4 times faster than the full inner product computation time. PT means the preparation time and IT represents the inference time in prediction process. The time units of *seconds*, *minutes*, and *hours* are represented as s, m, and h, respectively. Computation time of the full inner product method for each dataset is 71s (MovieLens), 1,017s (Last.fm), 92,828s (Amazon), 56,824s (Youtube), 71,653s (Flickr), and 72,723s (Wikipedia).

Task	Item Recommendation														
Dataset	MovieLens					Last.fm					Amazon				
Method	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5
ϵ-Approx	0.7x	0.19s	99.00s	0.753	0.671	0.5x	1.40s	36.78m	0.378	0.467	0.2x	23.42s	107.34h	0.529	0.559
GMIPS	3.9x	N/A	18.41s	1.000	0.972	2.3x	N/A	7.55m	0.997	0.966	1.8x	N/A	14.57h	0.993	0.952
SVDS	1.0x	0.10s	69.00s	1.000	1.000	0.9x	0.10s	19.25m	0.984	0.984	1.3x	5.32s	19.46h	0.952	0.953
FGD	2.8x	4.94s	20.10s	1.000	0.999	10.9x	0.49m	1.07m	0.997	0.988	19.7x	42.76m	35.83m	0.986	0.977
L2S	3.0x	22.15s	1.72s	1.000	1.000	9.0x	1.77m	0.12m	0.993	0.980	21.2x	71.02m	1.86m	0.988	0.979
CANTOR	**9.4x**	6.17s	1.36s	1.000	0.999	**37.1x**	0.37m	0.09m	0.999	0.998	**29.0x**	52.13m	1.26m	0.994	0.991
Task	Personalized Link Prediction														
Dataset	YouTube					Flickr					Wikipedia				
Method	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5	SU	PT	IT	P@1	P@5
ϵ-Approx	0.1x	0.3m	129.2h	0.364	0.432	0.4x	0.29m	53.44h	0.545	0.581	0.2x	0.39m	130.61h	0.374	0.480
GMIPS	1.4x	N/A	11.12h	0.987	0.965	2.0x	N/A	10.10h	0.987	0.962	3.6x	N/A	5.64h	0.991	0.974
SVDS	1.0x	0.03m	15.30h	0.965	0.963	1.4x	0.03m	14.00h	0.952	0.946	1.4x	0.03m	14.83h	0.949	0.944
FGD	44.8x	10.28m	10.85m	0.989	0.981	37.5x	17.61m	14.25m	0.985	0.980	93.7x	4.18m	8.76m	0.990	0.985
L2S	6.9x	135.93m	0.79m	0.984	0.968	8.3x	142.84m	0.58m	0.989	0.980	22.4x	53.38m	0.84m	0.988	0.968
CANTOR	**112.7x**	7.75m	0.65m	0.993	0.985	**54.7x**	21.31m	0.53m	0.994	0.990	**355.1x**	2.45m	0.97m	0.995	0.991

direct baseline. In our implementation, random sub-sampling is applied to choose a subset of users for training L2S.

Note that [34] has shown that Greedy-MIPS outperforms other MIPS algorithms including LSH-MIPS [27, 31], Sampling MIPS [3] and PCA-MIPS [1], so we omit those other MIPS algorithms in our comparisons. Although bandit-based methods [13, 21, 22] have elegant mathematical properties and theoretical bounds, we did not include them originally because they generally perform worse than other methods in practical cases. For example, SCLUB [21], which is one of the state-of-the-art bandit-based approaches, only achieves 0.81x and 0.62x speedups on the Amazon and Wikipedia datasets with the official implementations. This is because bandit-based methods independently manipulate each dimension and cannot benefit from low-level optimization for linear algebra operations.

Implementation Details. For each dataset, the LIBMF library [8] is used to train a non-negative MF (NMF) model. More specifically, the number of dimensions for latent vectors is 10 while the models are trained with all data for 100 iterations. Note that we adopt NMF models because of the restrictions of ϵ-Approx, but CANTOR does not have any limitation on matrix types. We implement CANTOR in Python with NumPy optimized by BLAS [4]. For the baseline methods, the implementations of GMIPS, SVDS, FGD, and L2S are provided by the original authors and highly-optimized while we utilize an efficient C++ implementation of ϵ-Approx. All experiments were run on a 64-bit Linux Ubuntu 16.04 server with 512 GB memory and single thread regime on an Intel® Xeon® CPU E5-2698 v4 2.2 GHz.

Hyperparameters in CANTOR. For the general settings of hyperparameters in CANTOR, we fix the number of sub-sampled user latent vectors u as 50,000 and the number of clusters r as 8. For adaptive representative selection, we set the number of iterations in the adaptive selection T as 10 and the similarity threshold ϵ as 0.99. The number of new representatives w in adaptive representative selection algorithm is set as 8. We also tune the size of dynamic nearest neighbor lists *efs* in the construction of hierarchical small world graphs for each dataset to achieve acceptable accuracy scores. As a result, the selections of *efs* are 200, 200, 1,500, 500, 1,500, and 100 for the datasets MovieLens, Last.fm, Amazon, YouTube, Flickr, and Wikipedia, respectively.

4.2 Performance Comparison

To fairly compare the performance, for each dataset, we tune the parameters such that each method can roughly achieve 0.99 P@1 accuracy. Table 3 shows the efficiency and the precision scores of CANTOR and all baseline methods on six datasets. Note that since the open-sourced library of GMIPS does not provide the breakdown of execution time into preparation and prediction time, the reported time includes both preparation and prediction processes. Among the baseline methods, FGD performs the best because it exploits the state-of-the-art algorithm for approximate nearest neighbor search to retrieve recommendations for each user. Although L2S is the most efficient baseline in the inference process, its preparation process is slow so that the overall speedup is further degraded. SVDS can efficiently decompose the preference matrix as its preparation process, but it still requires to examine all items many times to achieve sufficient accuracy so that the acceleration is unsatisfactory. In addition, it is worth noting that, although ϵ-Approx theoretically needs fewer multiplications than the full evaluation, it actually does not provide any acceleration in practice. Similar to bandit-based

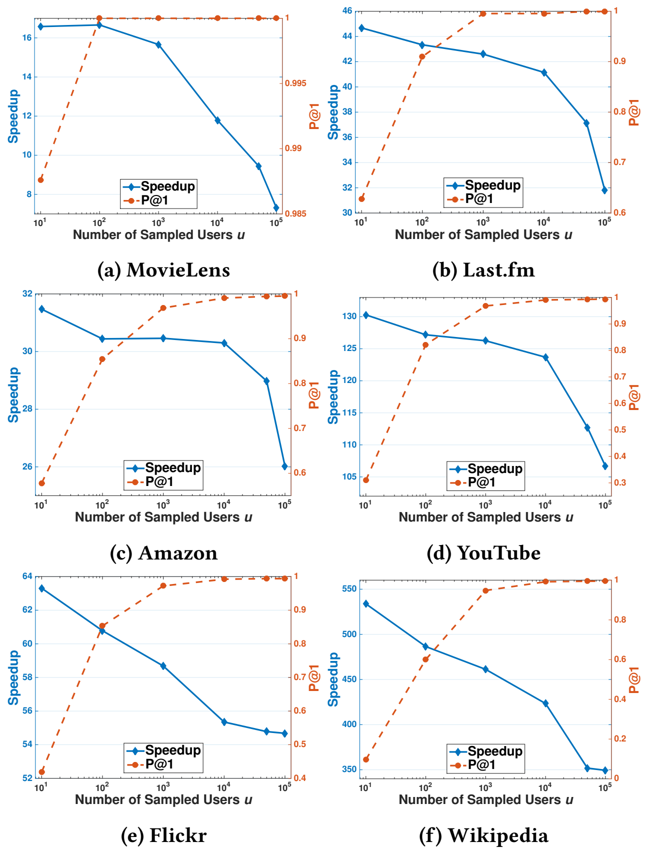


Figure 3: The ratios of speedup and the P@1 scores of CANTOR over different numbers of sampled users u in six datasets.

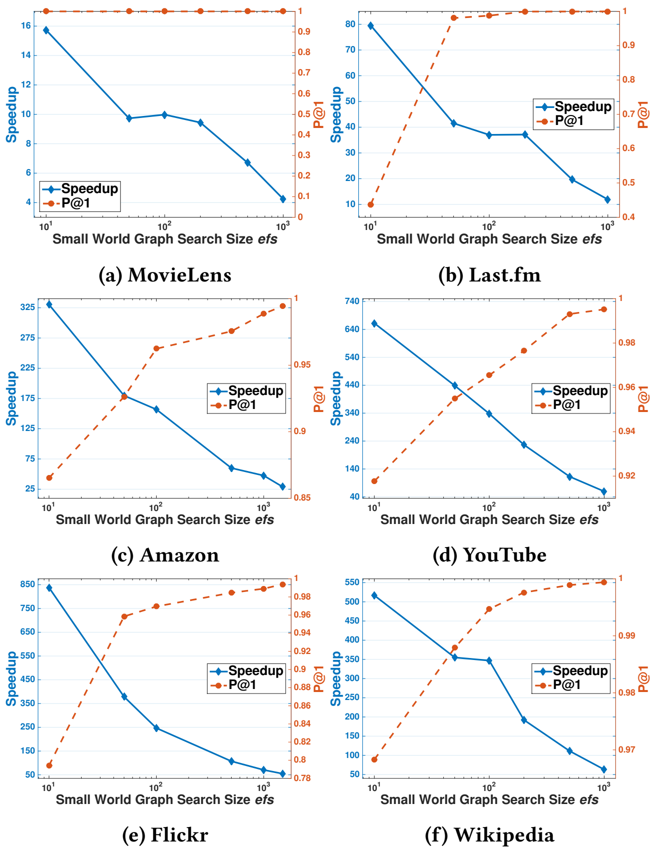


Figure 4: The ratios of speedup and the P@1 scores of CANTOR over different sizes of the hyperparameter *efs* in six datasets.

methods, this is because each dimension is independently processed so that the model cannot benefit from any low-level optimization for linear algebra operations.

Our approach CANTOR significantly outperforms all of the baseline methods in accelerating the overall execution time to provide top-K recommendations in all datasets. More specifically, CANTOR has similar inference time for the prediction process to that of L2S (that also reduces the candidate item sets for less computation) but the preparation process of CANTOR is much faster. This is because similarities between user latent vectors are well leveraged to avoid unnecessary and redundant computation.

4.3 Number of Sub-Sampled User Latent Vectors u

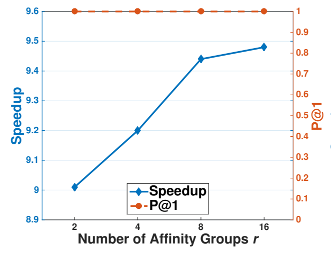
Since only a small subset of user latent vectors will be sub-sampled for user clustering, we verify how the number of sub-sampled users affects the performance in both efficiency and accuracy. Figure 3 illustrates the P@1 scores and the ratios of speedup of CANTOR for different numbers of sampled users in six datasets. It is obvious that smaller number of sampled user latent vectors is accompanied with greater speedup and lower P@1 score. However, CANTOR can generally achieve 99% P@1 scores after sampling more than around 10^4 users in all datasets. In other words, the distribution of the whole dataset can be preserved by sampling a small portion of users. For example, the Wikipedia dataset needs to sample only 5% of all users for high accuracy of recommendations.

4.4 Trade-off in Proximity Graph Construction

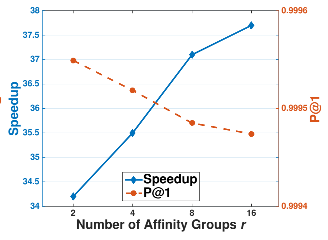
Proximity graph plays an important rule in CANTOR while the hyperparameter *efs* controls a trade-off between the efficiency and accuracy for generating the preferred item sets. Figure 4 depicts the P@1 scores and the speedup ratios of CANTOR for different *efs* in six datasets. Obviously, a too-small *efs* leads to unsatisfactory approximation and low accuracy scores. More precisely, the $P@1$ scores considerably drops when *efs* is below 10^2. On the other hand, CANTOR works well when *efs* is greater than 10^3 in all datasets. Hence, we suggest to tune *efs* in the range between 10^2 and 10^3 to reach a balance between efficiency and accuracy.

4.5 Number of Affinity Groups r

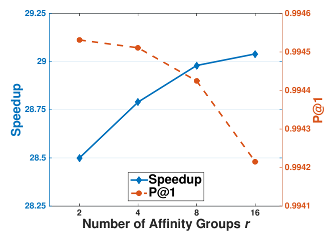
Figure 5 shows the performance of CANTOR with different numbers of user affinity groups r in six datasets. When there are more affinity groups, the sizes of preferred item sets shrink because of fewer representative vectors for each cluster. As a consequence,



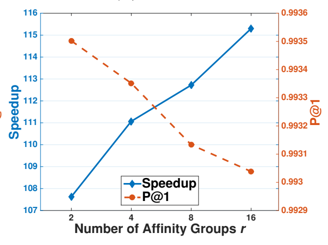
(a) MovieLens



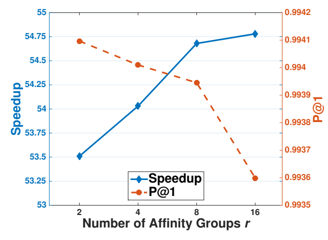
(b) Last.fm



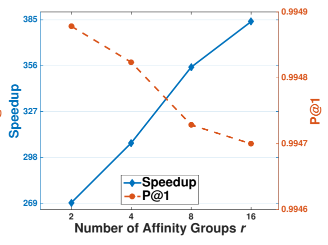
(c) Amazon



(d) YouTube

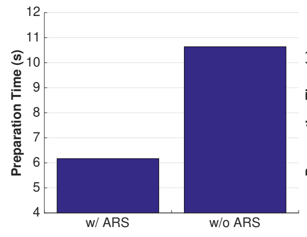


(e) Flickr

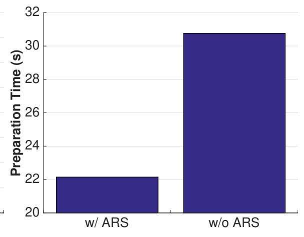


(f) Wikipedia

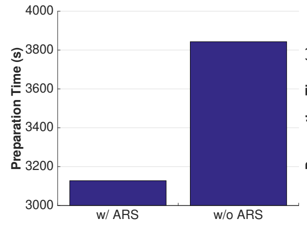
Figure 5: The ratios of speedup and the P@1 scores of CANTOR over different numbers of affinity groups r in six datasets.



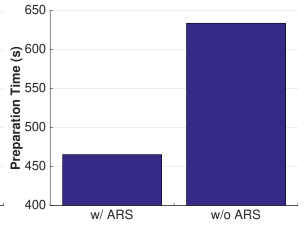
(a) MovieLens



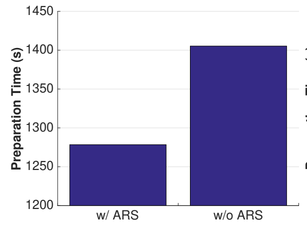
(b) Last.fm



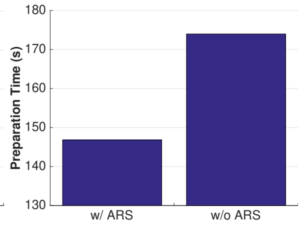
(c) Amazon



(d) YouTube



(e) Flickr



(f) Wikipedia

Figure 6: The preparation time of CANTOR with (w/) and without (w/o) the adaptive representative selection method (ARS) in six datasets.

CANTOR with larger group numbers considers fewer items in each affinity group, thereby achieving greater speedup. It is also noted that the great speedup comes with slight drop in accuracy. For example, there is only a 0.1% drop from $r = 2$ to $r = 16$. From these results, we suggest to set the number of user clusters r as a reasonable large number.

4.6 Effectiveness of Adaptive Representative Selection

The adaptive representative selection (ARS) method as shown in Algorithm 2 is important for CANTOR to accelerate the preparation process, so we also evaluate its effectiveness and robustness. Figure 6 illustrates the preparation time of CANTOR in six datasets with and without applying ARS. As a result, CANTOR using adaptive representative selection significantly outperforms the one without using ARS across all datasets when achieving similar accuracy. This further demonstrates the effectiveness and robustness of the adaptive representative selection method.

5 RELATED WORK

In this section, we discuss the related work in collaborative filtering for recommender systems and maximum inner product search.

5.1 Collaborative Filtering for Recommender Systems

Collaborative filtering (CF) [9] is one of the most popular solutions for recommendation problems, including the task of top-K recommender systems in this paper. Moreover, the low-rank assumption in CF further leads to the prominence of latent factor models or matrix factorization (MF) [19]. For example, Kang et al. [17] exploited MF models to optimize numerical ratings for top-K recommenders while Rendle et al. [29] observed pairwise implicit feedback in a one-class preference matrix and enhanced the personalized ranking performance of MF models. However, MF models can be time-consuming in inferring recommendations. More specifically, although MF models can be trained efficiently with sparse preference matrices, the number of possible recommendations can be enormous when the numbers of users and items are massive. To tackle this problem, Duan et al. [12] proposed to separately compute dot-product results in each dimension so that some items can be discarded if their dot-product values are below a threshold for specific dimensions. However, separately processing different dimensions and discarding certain entries not only lead to inaccuracy, but also give up the opportunity to take advantage of low-level runtime optimization like BLAS [4] as shown in our experiments. Moreover, this approach does not reduce the number of possible recommendations. On the other hand, although some of the previous studies [18, 28] achieve acceleration by group recommendation, users in a certain group would receive identical recommendations that can be unsatisfactory for individual users. To the best of our knowledge, this paper is the first work leveraging each user information to accelerate the inference process for top-K recommender systems without accuracy loss.

5.2 Maximum Inner Product Search

Maximum inner product search (MIPS) can be treated as a closely related problem to MF based top-K recommender systems. Shrivastava and Li [31] and Neyshabur and Srebro [27] proposed to reduce MIPS to nearest neighbor search (NNS) and then solve NNS by Locality Sensitive Hashing (LSH) [16]. PCA tree [32] partitions the space according to the directions of principal components and shows better performance in practice. Bachrach et al. [1] showed tree-based approaches can be used for solving MIPS but the performance is poor for high dimensional data. Malkov et al. [25], Malkov and Yashunin [26] recently developed an NNS algorithm based on small world graph. Zhang et al. [35] applied the MIPS-to-NNS reduction and showed that graph-based approach performs well on neural language model prediction. Some algorithms were proposed to directly tackle MIPS problem instead of transforming to NNS. For example, Yu et al. [34] proposed Greedy-MIPS and showed a significant improvement over LSH and tree-based approaches. Another branch of research exploited sampling techniques with guaranteed approximation precision. Liu et al. [24] applied a bandit framework to iteratively query each dimension of the item vector; Ding et al. [10] proposed a 2-stage entry-wise sampling scheme and constructed an alias table to accelerate the sampling process. Despite having theoretical guarantee of approximation precision, in practice these methods suffer from slow entry-wise computation and the speedup is thus limited or even worse than the naive computation. Among all previous works, learning to screen (L2S) proposed by Chen et al. [7] is most similar to our method. L2S also leverages the clustering architecture to accelerate MIPS computation of multiple NLP tasks. However, L2S takes a long preparation time as it finds the clustering by end-to-end training and constructs a reduced search space by naive computation. In addition, L2S does not use representative vectors which differs from our proposed method. In our experiments, hierarchical graphical models [35], Greedy-MIPS [34] and L2S [7] are selected as the state-of-the-art MIPS methods for comparison.

Another related problem is the Maximum All-pair Dot-product (MAD) problem discussed in [3]. However, their goal is to find top-K user-item pairs among all mn pairs (K largest elements in the m-by-n preference matrix), while our goal is to find top-K items for each user (top-K elements in each row).

6 CONCLUSIONS

In this paper, we propose a novel framework for accelerating large-scale top-K recommender systems by exploiting user relationships and the redundancy of user vectors in the latent space. Our model, CANTOR, first clusters users into affinity groups, thereby determining as a user coreset of representative vectors for each group so that only a limited number of preferred items need to be examined for the users in the affinity group. Moreover, we mathematically prove that user coresets can be efficiently constructed by set covers of sub-sampled user latent vectors with an asymptotically guaranteed bound. Experimental results demonstrate that CANTOR significantly outperforms existing MIPS and approximate MF algorithms for accelerating top-K recommender systems. In particular, CANTOR achieves 355x and 29x speedup on the largest Wikipedia and Amazon datasets in two tasks while the accuracy scores still remain to be 99% for both P@1 and P@5. Moreover, the results of analysis also show the effectiveness and robustness of CANTOR.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their helpful comments. The work was partially supported by NSF DGE-1829071, NSF IIS-1719097, Intel, and Facebook.

APPENDIX

A PROOF OF THEOREM 1

Proof. Without loss of generality, we assume that vectors in $\boldsymbol{A_t}$, $\boldsymbol{Q}$, and $\boldsymbol{s_t}$ have unit norms. $\forall q \in \boldsymbol{Q}, i \in \boldsymbol{A_t}$, we have:

$$\begin{aligned} |p_i q^T - \mathcal{N}_{s_t}(p_i)\, q^T| &= \left|(p_i - \mathcal{N}_{s_t}(p))q^T\right| \\ &\overset{(a)}{\leq} \sqrt{d}\|p_i - \mathcal{N}_{s_t}(p_i)q^T\|_2 \leq \sqrt{d}\|p_i - \mathcal{N}_{s_t}(p_i)\|_2 \leq \sqrt{d}\|p_i - \mathcal{N}_{s_t}(p_i)\|_2^2 \\ &= \sqrt{d}\left(\|p_i\|_2^2 + \|\mathcal{N}_{s_t}(p_{i_j})\|_2^2 - 2\mathcal{N}_{s_t}(p_i)p_i^T\right) \overset{(b)}{\leq} \sqrt{d}\,[2 - 2\epsilon] = \delta, \end{aligned}$$

where we define $\delta = \sqrt{d}\,[2 - 2\epsilon]$. (a) follows from the fact that $\|\cdot\|_1 \leq \sqrt{d}\|\cdot\|_2$, where d is the dimension of the vector. (b) follows from the condition of theorem. □

B PROOF OF THEOREM 2

Proof. Since $\boldsymbol{s_t}$ is a ϵ set cover of p_is, there exist a δ such that $\boldsymbol{s_t}$ is a δ-user coreset of p_is. Therefore, for any given query q and vector p_t sampled from $\boldsymbol{P_{A_t}}$, we have

$$\begin{aligned} |\mathcal{N}_{s_t}(p_i)q^T - p_t q^T| &= |\mathcal{N}_{s_t}(p_i)q^T - p_i q^T + p_i q^T - p_t q^T| \\ &\leq |\mathcal{N}_{s_t}(p_i)q^T - p_i q^T| + |p_i q^T - p_t q^T| \leq \delta + |p_i q^T - p_t q^T| \end{aligned}$$

Since p_i and p_t follow the same distribution, p_i and p_t will have same expectation value and we have:

$$\begin{aligned} \mathbb{E}[|\mathcal{N}_{s_t}(p_i)q^T - p_t q^T|] &\leq \mathbb{E}[\delta + |p_i q^T - p_t q^T|] \\ &= \delta + \mathbb{E}[|p_i q^T - p_t q^T|] \\ &\overset{(a)}{\leq} \delta + |\mathbb{E}[p_i q^T] - \mathbb{E}[p_t q^T]| \\ &= \delta, \end{aligned}$$

where (a) follows the Jensen's inequality. Therefore, by Hoeffding's inequality, with probability at least 1 - γ,

$$\frac{1}{k}\sum_{i=1}^{k}\left|\mathcal{N}_{s_t}(p_i)\, q^T - p_t q^T\right| \leq \delta + \sqrt{\frac{2\log(1/\gamma)}{k}}.$$

By the fact that for any set S, $\min(S) \leq \text{mean}(S)$, we will have:

$$\begin{aligned} \min_i\left(\left|\mathcal{N}_{s_t}(p_i)q^T - p_t q^T\right|\right) &\leq \frac{1}{k}\sum_{i=1}^{k}|\mathcal{N}_{s_t}(p_i)q^T - p_t q^T| \\ &\leq \delta + \sqrt{\frac{2\log(1/\gamma)}{k}}, \end{aligned}$$

□

REFERENCES

[1] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 257–264.

[2] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 635–644.

[3] Grey Ballard, Tamara G Kolda, Ali Pinar, and C Seshadhri. 2015. Diamond sampling for approximate maximum all-pairs dot-product (MAD) search. In *2015 IEEE International Conference on Data Mining*. IEEE, 11–20.

[4] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. 2002. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Software* 28, 2 (2002), 135–151.

[5] Peer Bork, Lars J Jensen, Christian Von Mering, Arun K Ramani, Insuk Lee, and Edward M Marcotte. 2004. Protein interaction networks from yeast to human. *Current opinion in structural biology* 14, 3 (2004), 292–299.

[6] O. Celma. 2010. *Music Recommendation and Discovery in the Long Tail*. Springer.

[7] Patrick Chen, Si Si, Sanjiv Kumar, Yang Li, and Cho-Jui Hsieh. 2019. Learning to Screen for Fast Softmax Inference on Large Vocabulary Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ByeMB3Act7

[8] Wei-Sheng Chin, Bo-Wen Yuan, Meng-Yuan Yang, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. 2016. LIBMF: a library for parallel matrix factorization in shared-memory systems. *JMLR* 17, 1 (2016), 2971–2975.

[9] Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 143–177.

[10] Qin Ding, Hsiang-Fu Yu, and Cho-Jui Hsieh. 2019. A Fast Sampling Algorithm for Maximum Inner Product Search. In *The 22nd International Conference on Artificial Intelligence and Statistics*. 3004–3012.

[11] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. 2011. The yahoo! music dataset and kdd-cup'11. In *Proceedings of the 2011 International Conference on KDD Cup 2011-Volume 18*. JMLR. org, 3–18.

[12] Liang Duan, Charu Aggarwal, Shuai Ma, Renjun Hu, and Jinpeng Huai. 2016. Scaling up link prediction with ensembles. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 367–376.

[13] Claudio Gentile, Shuai Li, and Giovanni Zappella. 2014. Online clustering of bandits. In *International Conference on Machine Learning*. 757–765.

[14] Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for GPUs. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 1302–1310.

[15] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.

[16] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

[17] Zhao Kang, Chong Peng, and Qiang Cheng. 2016. Top-n recommender system via matrix completion. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[18] Ondrej Kaššák, Michal Kompan, and Mária Bieliková. 2016. Personalized hybrid recommendation for group of users: Top-N multimedia recommender. *Information Processing & Management* 52, 3 (2016), 459–477.

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[20] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1343–1350.

[21] Shuai Li, Wei Chen, Shuai Li, and Kwong-Sak Leung. 2019. Improved algorithm on online clustering of bandits. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2923–2929.

[22] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. 2016. Collaborative filtering bandits. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 539–548.

[23] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 1 (2003), 76–80.

[24] Rui Liu, Tianyi Wu, and Barzan Mozafari. 2019. A Bandit Approach to Maximum Inner Product Search. *CoRR* abs/1812.06360 (2019).

[25] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[26] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[27] Behnam Neyshabur and Nathan Srebro. 2015. On symmetric and asymmetric LSHs for inner product search. In *ICML*.

[28] Eirini Ntoutsi, Kostas Stefanidis, Kjetil Nørvåg, and Hans-Peter Kriegel. 2012. Fast group recommendations by applying user clustering. In *International Conference on Conceptual Modeling*. Springer, 126–140.

[29] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[30] Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. 2017. SVD-Softmax: Fast Softmax Approximation on Large Vocabulary Neural Networks. In *Advances in Neural Information Processing Systems 30*. 5463–5473.

[31] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*. 2321–2329.

[32] Robert F Sproull. 1991. Refinements to nearest-neighbor searching inkdimensional trees. *Algorithmica* 6, 1-6 (1991), 579–589.

[33] Jiliang Tang, Shiyu Chang, Charu Aggarwal, and Huan Liu. 2015. Negative link prediction in social media. In *Proceedings of the eighth ACM international conference on web search and data mining*. ACM, 87–96.

[34] Hsiang-Fu Yu, Cho-Jui Hsieh, Qi Lei, and Inderjit S Dhillon. 2017. A greedy approach for budgeted maximum inner product search. In *Advances in Neural Information Processing Systems*. 5453–5462.

[35] Minjia Zhang, Xiaodong Liu, Wenhan Wang, Jianfeng Gao, and Yuxiong He. 2018. Navigating with Graph Representations for Fast and Scalable Decoding of Neural Language Models. In *NIPS*.