Fast Group Recommendations by Applying User Clustering

Eirini Ntoutsi[1], Kostas Stefanidis[2], Kjetil Nørvåg[2], and Hans-Peter Kriegel[1]

[1] Institute for Informatics, Ludwig Maximilian University, Munich
{ntoutsi, kriegel}@dbs.ifi.lmu.de
[2] Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim
{kstef,Kjetil.Norvag}@idi.ntnu.no

Abstract. Recommendation systems have received significant attention, with most of the proposed methods focusing on personal recommendations. However, there are contexts in which the items to be suggested are not intended for a single user but for a group of people. For example, assume a group of friends or a family that is planning to watch a movie or visit a restaurant. In this paper, we propose an extensive model for group recommendations that exploits recommendations for items that similar users to the group members liked in the past. We do not exhaustively search for similar users in the whole user base, but we pre-partition users into clusters of similar ones and use the cluster members for recommendations. We efficiently aggregate the single user recommendations into group recommendations by leveraging the power of a top-k algorithm. We evaluate our approach in a real dataset of movie ratings.

1 Introduction

Recommendation systems provide users with suggestions about products, movies, videos and many other items. Many systems, such as Amazon, NetFlix and MovieLens, are very popular. Collaborative recommendation systems (e.g., [14, 7]) try to predict the utility of items for a particular user based on the items previously rated by other users, that is, users similar to a target user are identified, and then items are recommended based on the preferences of the similar users. Users are considered similar if there is an overlap in the items consumed. The two types of entities that are dealt in recommendation systems, i.e., users and items, are represented as sets of ratings, preferences or features. Assume, for example, a restaurant recommendation application (e.g., ZAGAT.com). Users initially rate a subset of restaurants that they have already visited. Ratings are expressed in the form of preference scores. A recommendation engine estimates preference scores for the items, e.g., restaurants, that are not rated by a user and offers appropriate recommendations. Once the unknown scores are computed, the k items with the highest scores are recommended to the user.

Since recommendations are typically personalized, different users are presented with different suggestions. However, there are cases where a group of

people participates in a single activity. For instance, visiting a restaurant or a tourist attraction, watching a movie or a TV program and selecting a holiday destination are examples of recommendations well suited for groups of people. For this reason, recently, there are methods for group recommendations, trying to satisfy the preferences of all the group members. These methods can be classified into two approaches [12]. The first approach creates a joint profile for all users in the group and provides the group with recommendations computed with respect to this joint profile (e.g., [27]). The second approach aggregates the recommendations of all users in the group into a single recommendation list (e.g., [3, 5]). Our work follows the second approach, since it is more flexible [12, 19] and offers opportunities for improvements in terms of efficiency.

In this paper, we propose a model for group recommendations following the collaborative filtering approach. We are mainly motivated by the observation that similarity can be used to cluster users in small groups with strong similarity [21]. This way, our framework applies user clustering for organizing users into groups of users with similar preferences. To do this, we employ a hierarchical agglomerative clustering algorithm. Thereafter, we propose the use of these clusters to efficiently locate similar users to a given one; recommendations for users are produced with respect to the preferences of their cluster members without extensively searching for similar users in the whole database. The k most prominent items for the group are identified by exploiting a top-k algorithm. Group recommendations are presented to users along with explanations about the reasons that the particular items are being suggested. Finally, to deal with the sparsity of the explicitly defined user preferences, we introduce the notion of support in recommendations to model how confident the recommendation of an item for a user is.

To summarize, our contributions are as follows:

- We enhance recommendations with the notion of support to model the confidence of recommendations.
- We formulate the top-k group recommendations problem as a top-k query problem and leverage the power of a top-k algorithm to efficiently derive the most prominent items for the whole group.
- We present recommendations along with explanations about why the specific recommendations appear in the top-k list.
- We introduce user clustering for partitioning users into clusters and use these clusters to efficiently compute personal recommendations. Personal recommendations are aggregated to produce group recommendations.

We have also evaluated both the efficiency and effectiveness of our approach using a real dataset of movie ratings.

The rest of the paper is organized as follows. Section 2 introduces the personal and group recommendation model, and clarifies the problem statement. Section 3 proposes methods for locating users similar to a target one and identifying the top-k recommendations for a group. Section 4 contains extensive experimentation to evaluate the efficiency and effectiveness of our approach, while Section 5 describes related work. Finally, Section 6 concludes the paper with a summary of our findings.

2 Recommendation Model

Assume a set of items $\mathcal{I}$ and a set of users $\mathcal{U}$ interacting with a recommendation application. Each user $u \in \mathcal{U}$ may express a preference for an item $i \in \mathcal{I}$, $preference(u, i)$, in the range $[0.0, 1.0]$. We use $\mathcal{P}_i$ to denote the set of users in $\mathcal{U}$ that have expressed a preference for item i. The cardinality of the items set $\mathcal{I}$ is usually high and typically users rate only a few of these items. For the items unrated by the users, we estimate a *relevance score*, denoted as $relevance(u, i)$, where $u \in \mathcal{U}$, $i \in \mathcal{I}$. To do this, a recommendation strategy is invoked. We distinguish between personal recommendations referring to a single user (Section 2.1) and group recommendations referring to a set of users (Section 2.2).

2.1 Personal Recommendations

There are different ways to estimate the relevance of an item for a user. In general, the recommendation methods are categorized into: (i) *content-based*, that recommend items similar to those the user has preferred in the past (e.g., [17]), (ii) *collaborative filtering*, that recommend items that similar users have liked in the past (e.g., [14, 7]) and (iii) *hybrid*, that combine content-based and collaborative ones (e.g., [4]). Our work falls into the *collaborative filtering* category. The key concept of collaborative filtering is to use preferences of other users that exhibit the most similar behavior to a given user in order to produce relevance scores for unrated items. Similar users are located via a *similarity function* $simU(u, u')$ that evaluates the proximity between u and u'.

We use $\mathcal{F}_u$ to denote the set of the most similar users to u. We refer to such users as the *friends* of u.

Definition 1 (Friends). *Let $\mathcal{U}$ be a set of users. The friends $\mathcal{F}_u$, $\mathcal{F}_u \subseteq \mathcal{U}$, of a user $u \in \mathcal{U}$ is a set of users, such that, $\forall u' \in \mathcal{F}_u$, $simU(u, u') \geq \delta$ and $\forall u'' \in \mathcal{U} \backslash \mathcal{F}_u$, $simU(u, u'') < \delta$, where δ is a threshold similarity value.*

Clearly, one could argue for other ways of selecting $\mathcal{F}_u$, for instance, by taking the k most similar users to u. Our main motivation is that we opt for selecting only highly connected users even if the resulting set of users $\mathcal{F}_u$ is small.

Given a user u and his friends $\mathcal{F}_u$, if u has expressed no preference for an item i, the relevance of i for u is estimated as follows:

$$relevance(u, i) = \frac{\sum_{u' \in (\mathcal{F}_u \cap \mathcal{P}_i)} simU(u, u') preference(u', i)}{\sum_{u' \in (\mathcal{F}_u \cap \mathcal{P}_i)} simU(u, u')}$$

However, since the number of items is huge and usually users rate only a few items, the following question usually arises: *How confident are the relevance scores associated with the recommended items?* Towards this direction, we introduce the notion of *support* for each candidate item i for user u, which defines the percentage of friends of u that have expressed preferences for i. Formally:

$$support(u, i) = |\mathcal{F}_u \cap \mathcal{P}_i| / |\mathcal{F}_u|$$

To estimate the worthiness of an item recommendation for a user, we propose to combine the *relevance* and *support* scores in terms of a *value* function.

Definition 2 (Personal Value). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. Let $w_1, w_2 \geq 0 : w_1 + w_2 = 1$. The personal value of an item $i \in \mathcal{I}$ for a user $u \in \mathcal{U}$ with friends $\mathcal{F}_u$, such that, $\nexists preference(u, i)$, is:*

$$value_{\mathcal{F}_u}(u, i) = w_1 \times relevance(u, i) + w_2 \times support(u, i)$$

Although more sophisticated functions can be designed, the weighted summation of the relevance and support scores is simple and intuitive. Moreover, when $w_2 = 0$, *value* maps to *relevance*, the typically used recommendation score.

2.2 Group Recommendations

The majority of recommendation systems are designed to make personal recommendations, i.e., recommendations for individual users. However, there are cases in which the items to be selected are not intended for personal usage but for a group of users. For example, assume a group of friends or a family that is planning to watch a movie, visit a restaurant or select a holiday destination. For this reason, some recent works have addressed the problem of identifying recommendations for a group of users, trying to satisfy the preferences of all the group members (e.g., [3, 5, 6]).

In our approach, we first compute the *personal value* scores for the unrated items for each user in the group, and then, based on these predictions, we compute the aggregated value scores for the group.

Definition 3 (Group Value). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. Given a group of users $\mathcal{G}$, $\mathcal{G} \subseteq \mathcal{U}$, the group value of an item $i \in \mathcal{I}$ for $\mathcal{G}$, such that, $\forall u \in \mathcal{G}$, $\nexists preference(u, i)$, is:*

$$value(\mathcal{G}, i) = Aggr_{u \in \mathcal{G}}(value_{\mathcal{F}_u}(u, i))$$

We employ three different designs regarding the aggregation method $Aggr$, each one carrying different semantics: (i) the *least misery design*, capturing cases where strong user preferences act as a veto (e.g., do not recommend steakhouses to a group when a vegetarian belongs to the group), (ii) the *fair design*, capturing more democratic cases where the majority of the group members is satisfied, and (iii) the *most optimistic design*, capturing cases where the more satisfied member of the group acts as the most influential member (e.g., recommend a movie to a group when a member is highly interested in it and the remaining members have reasonable satisfaction). In the least misery (respectively, most optimistic) design, the predicted value score of an item for the group is equal to the minimum (respectively, maximum) value score of the item scores of the members of the group, while the fair design, that assumes equal importance among all group members, returns the average score. Table 1 summarizes the aggregation methods.

2.3 Problem Statement

Given a group of users and a restriction k on the number of the recommended items, we would like to provide k suggestions for items that are highly relevant to the preferences of all the group members and, also, exhibit high support.

Table 1. Aggregation methods.

Design	Aggregation Method
Least misery	$value(\mathcal{G}, i) = \min_{u \in \mathcal{G}}(value_{\mathcal{F}_u}(u, i))$
Fair	$value(\mathcal{G}, i) = \left(\sum_{u \in \mathcal{G}} value_{\mathcal{F}_u}(u, i)\right) / \lvert\mathcal{G}\rvert$
Most optimistic	$value(\mathcal{G}, i) = \max_{u \in \mathcal{G}}(value_{\mathcal{F}_u}(u, i))$

Definition 4. (Top-k Group Recommendations). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. Given a group of users $\mathcal{G}$, $\mathcal{G} \subseteq \mathcal{U}$, and an aggregation method Aggr, recommend to $\mathcal{G}$ a list of items $\mathcal{I}_\mathcal{G} =< i_1, \ldots, i_k >$, $\mathcal{I}_\mathcal{G} \subseteq \mathcal{I}$, such that:*
(i) $\forall i_j \in \mathcal{I}_\mathcal{G}, u \in \mathcal{G}$, $\nexists preference(u, i_j)$,
(ii) $value(\mathcal{G}, i_j) \geq value(\mathcal{G}, i_{j+1})$, $1 \leq j \leq k-1$, $\forall i_j \in \mathcal{I}_\mathcal{G}$, and
(iii) $value(\mathcal{G}, i_j) \geq value(\mathcal{G}, x_y)$, $\forall i_j \in \mathcal{I}_\mathcal{G}$, $x_y \in \mathcal{I} \backslash \mathcal{I}_\mathcal{G}$.

The first condition ensures that the suggested items do not include already evaluated items by some users in the group (e.g., do not recommend a movie that a group member has already watched). The second condition ensures the descending ordering of the items with respect to their group value, while the third condition defines that every item in the result set has group value greater than or equal to the group value of any of the remaining items.

2.4 Group Recommendations Explanations

Recently, it has been shown that the success of recommendations relies on explaining the cause behind them [25]. To this end, except for the suggested items, we also provide the group with an explanation for each suggested item, i.e., why the specific item appears in the top-k list.

Although our explanations depend on the employed design, they have the following general form: "item i has group value score $value(\mathcal{G}, i)$ because of user(s) $\{u_x, \ldots, u_y\}$". For instance, for a movie recommendation system, an example explanation is: "Movie *Dracula* has group value score 0.9 because of user Jeffrey". More specifically, for the least misery design, we report with each suggested item its group value score and the member of the group with the minimum personal value score for the item, i.e., the member that is responsible for this selection. Similarly, for the most optimistic design, we report the member of the group with the maximum personal value score for the item. Finally, for the fair design, we report with each item the members of the group with personal value scores for the item close to its group value score (up to a distance p), i.e., the members that are highly satisfied, and hence, direct towards this selection.

3 Group Recommendations Computation

Our approach for suggesting items for a group of users $\mathcal{G}$, consists of the following steps: (i) locate the set of users $\mathcal{F}_u, \forall u \in \mathcal{G}$, (ii) compute the personal value scores $value_{\mathcal{F}_u}(u, i), \forall u \in \mathcal{G}, i \in \mathcal{I}$, (iii) combine the independent scores according to an aggregation method *Aggr* to derive the group value scores $value(\mathcal{G}, i), \forall i \in \mathcal{I}$,

and (iv) present the k items with the highest group value scores along with explanations.

For computing the scores of all items for the group, a solution that involves no pre-computation is to first find the friends of each user in $\mathcal{G}$, by computing the similarity measures between each user in $\mathcal{G}$ and each user in $\mathcal{U}$, then produce the value scores of the candidate items and finally rank them based on these scores. We refer to this approach as the *baseline approach*. Performance can be improved by performing some preprocessing steps offline. In particular, in this work, we propose building clusters of similar users, considering as similar those users that have similar preferences. Ideally, the friends of each user are the members of the cluster that the user belongs to. Based on the preferences of these cluster members, the group value scores are computed. We refer to this approach as the *user clustering approach*.

Next, we present our method for locating the friends of a user in the baseline and user clustering approach (Section 3.1). Then, we focus on how to identify the top-k group recommendations (Section 3.2).

3.1 Finding User Friends

Baseline Approach. To find the friends set $\mathcal{F}_u$ for a specific user $u \in \mathcal{U}$, the baseline approach would need to calculate all similarity measures $simU(u, u')$, $\forall u' \in \mathcal{U}$ and select those with $simU(u, u') \geq \delta$. We refer to this algorithm, as the *Baseline Friends Finder* algorithm. Such an approach though, would be extremely inefficient in large systems, since it requires the online computation of the set of friends for each user of the query group.

User Clustering Approach. Since the baseline approach is expensive for a real recommendation application where the response time is a critical parameter, we propose to organize the users into groups of users with similar preferences and employ these pre-computed groups to speed up the recommendation process.

We use clustering for partitioning users into groups[3]. In particular, we use a bottom up hierarchical agglomerative clustering algorithm. Initially, the *Clustering Friends Finder* algorithm (Algorithm 1) places each user in a cluster of his own. Then, at each step, it merges the two most similar clusters. The similarity between two clusters is defined as the minimum similarity between any two users that belong to these clusters (max linkage). The algorithm terminates when the similarity of the closest pair of clusters violates the user similarity threshold δ.

Property 1. Let δ be a threshold similarity value. Each cluster produced by the Clustering Friends Finder algorithm contains users, such that, for each pair of users u, u' in the cluster $simU(u, u') \geq \delta$.

Proof: From lines $3-5$ of the Clustering Friends Finder algorithm, we merge the two most similar clusters, if their similarity is greater than or equal to δ.

[3] Hereafter, we refer to the groups of users with similar preferences as *clusters*, to remove any ambiguity with the term *group* referring to the group of users asking for recommendations (query group).

Algorithm 1 Clustering Friends Finder Algorithm

Input: A set of user $\mathcal{U}$ and a threshold similarity value δ.
Output: A set of clusters of users with similar preferences.

```
1: create a cluster for each user u ∈ U;
2: repeat
3:   if the max similarity between any pair of clusters is greater or equal to δ then
4:     merge these two clusters;
5:   else end loop;
```

This similarity score represents the minimum similarity between a user u of the first cluster and a user u' of the second cluster. Therefore, any two users u, u' that belong to the same cluster, have similarity $simU(u, u') \geq \delta$. □

This means that, if for a user u, we employ for estimating personal value scores the users in the cluster of u, we may lose some users similar to u, but we never consider as similar, users that are not, and so, our method does not result in some form of false positives.

In general, following this clustering approach, we consider as friends of each user u the members of the cluster $\mathcal{C}_u$ that the user u belongs to. We use $value_{\mathcal{C}_u}(u, i)$ to denote the personal value score of item i for u computed taking into account the users in $\mathcal{C}_u$. It is easy to show that:

Property 2. Let $\mathcal{G}$ be a group of users and i be an item in $\mathcal{I}$.
(i) If the resulting score of both $\max_{u \in \mathcal{G}}(value_{\mathcal{F}_u}(u, i))$ and $\max_{u \in \mathcal{G}}(value_{\mathcal{C}_u}(u, i))$ is associated with a user $u' \in \mathcal{G}$, and the most optimistic design is applied, both the baseline and the user clustering approaches employ the personal value score of u' for determining $value(\mathcal{G}, i)$.
(ii) If the resulting score of both $\min_{u \in \mathcal{G}}(value_{\mathcal{F}_u}(u, i))$ and $\min_{u \in \mathcal{G}}(value_{\mathcal{C}_u}(u, i))$ is associated with a user $u' \in \mathcal{G}$, and the least misery design is applied, both the baseline and the user clustering approaches employ the personal value score of u' for determining $value(\mathcal{G}, i)$.

Note that we cannot have a counterpart observation for the fair design, since in the fair design the personal value scores of sets of users are taken into account for computing group value scores, in both approaches.

3.2 Identifying Top-k Group Recommendations

Having established the methodology for finding the friends of a single user, we focus next on how to generate *valued recommendations* for a group of users. The first step towards this direction is to compute the personal value scores of each item for each user in the group. The next step is to combine the scores of each item in order to select, based on the group value scores, the items to be suggested.

Given a user $u \in \mathcal{U}$ and his similar users $\mathcal{C}_u$, the procedure for estimating the personal value score of each item i in $\mathcal{I}$ for u requires the computation of its *relevance* and *support*. Pairs of the form $(i, value_{\mathcal{C}_u}(u, i))$ are maintained in a set $\mathcal{V}_u$. As a post-processing step, we rank all pairs in $\mathcal{V}_u$ on the basis of

their personal value scores. The set $\mathcal{C}_u$ refers to the clustering approach. For the baseline approach, we use, for each user u, the set $\mathcal{F}_u$, instead of $\mathcal{C}_u$.

For a group of users $\mathcal{G} = \{u_1, \ldots, u_n\}$, a common way to provide k recommendations to $\mathcal{G}$ is assigning to all items their group value scores and reporting the k items with the highest scores. Instead of following the naive approach of computing group value scores of items and ranking them based on these scores, we employ a variation of a threshold-based algorithm, called TA, first proposed in [10]. Given a group $\mathcal{G} = \{u_1, \ldots, u_n\}$, the *Top-k Group Recommendations* algorithm uses as input the ranked sets $\mathcal{V}_{u_1}, \ldots, \mathcal{V}_{u_n}$ and it considers two types of available item accesses: the *sorted access* and the *random access*. Sorted access enables item retrieval in a descending order of their scores, while random access enables retrieving the score of a specific item in one access.

The algorithm's main steps are (Algorithm 2):

(i) Do sorted access to each ranked set $\mathcal{V}_{u_j}$. For each item seen, do random accesses to the other ranked sets to retrieve the missing item personal value scores.

(ii) Compute the group value score of each item that has been seen. Rank the items based on their group value scores and select the top-k ones.

(iii) Stop to do sorted accesses when the group value scores of the k items are at least equal to a threshold value that is defined as the aggregation score of the scores of the last items seen in each ranked set.

The *Top-k Group Recommendations* algorithm is correct when the item group value scores are obtained by combining their individual scores using a monotone function [10]. In our approach, aggregations are performed in a monotonic fashion, hence the applicability of the algorithm is straightforward.

Algorithm 2 Top-k Group Recommendations Algorithm

Input: A group of users $\mathcal{G} = \{u_1, \ldots, u_n\}$ with ranked sets $\mathcal{V}_{u_1}, \ldots, \mathcal{V}_{u_n}$ and an aggregation method $Aggr$.

Output: k pairs $(i_j, value(\mathcal{G}, i_j))$.

1: $topRecs = \emptyset$;
2: $scoreK = 0$;
3: $thresholdK = \infty$;
4: $t = 1$;
5: **while** scoreK < thresholdK **do**
6: retrieve the score of the t^{th} item from $\mathcal{V}_{u_1}, \ldots, \mathcal{V}_{u_n}$, $value_{\mathcal{C}_{u_1}}(u_1, i_1)$, ..., $value_{\mathcal{C}_{u_n}}(u_n, i_n)$;
7: $thresholdK = Aggr(value_{\mathcal{C}_{u_1}}(u_1, i_1), \ldots, value_{\mathcal{C}_{u_n}}(u_n, i_n))$;
8: retrieve the missing personal value scores for $i_1, \ldots, i_n$ from $\mathcal{V}_{u_1}, \ldots, \mathcal{V}_{u_n}$;
9: compute the group value scores for $i_1, \ldots, i_n$;
10: add, in decreasing order, the pairs of the form $(i_j, value(\mathcal{G}, i_j))$ to $topRecs$;
11: make $scoreK$ equal to the k^{th} group value score in $topRecs$;
12: t++;
13: **end while**
14: **return** the k first pairs $(i_j, value(\mathcal{G}, i_j))$ in $topRecs$;

4 Experiments

For the evaluation, we used the MovieLens dataset [1], which consists of 100,000 user ratings given by 1,000 users for 1,700 items. We show that the user clustering method is both efficient and effective comparing to the baseline approach.

To illustrate the efficiency of the user clustering approach versus the baseline alternative, we measure the execution time for computing the personal recommendations of the members of a query group. We omit the aggregation time for computing the top-k recommendations, since this time is the same in both cases.

We also evaluate the quality of our approach against the baseline method. We employ the *recommendations agreements* as a quality measure, which is defined as the number of items recommended by both approaches. That is, we compute the top-k group recommendations for both approaches and we find their intersection, which stands for the common suggested items. We denote this measure by *commonRecs*. The recommendations agreement measure computes the common items in both recommendation lists, but does not consider the actual ordering of the recommendations. To this end, we also employ a variation of the Kendall tau distance for partial rankings that computes the distance between two partial rankings based on the number of pairwise disagreements between them [9]. We denote this measure by $rankRecsDist$.

We use distance instead of user similarity. We define the distance between two users as the Euclidean distance over the items rated by both. Let $u, u' \in \mathcal{U}$ be two users, $\mathcal{I}_u$ be the set of items for which $\exists preference(u, i)$, $i \in \mathcal{I}_u$, and $\mathcal{I}_{u'}$ be the set of items for which $\exists preference(u', i)$, $i \in \mathcal{I}_{u'}$. We denote by $\mathcal{I}_u \cap \mathcal{I}_{u'}$ the set of items for which both users have expressed preferences. Then, the distance between u, u' is defined as:

$$distU(u, u') = \left(\sqrt{\sum_{i \in \mathcal{I}_u \cap \mathcal{I}_{u'}} (preference(u, i) - preference(u', i))^2}\right) / |\mathcal{I}_u \cap \mathcal{I}_{u'}|.$$

To set up a query group, we randomly select the members of the group from the user base. Group recommendations are extracted using both approaches and the execution time, the *commonRecs* and the $rankRecsDist$ scores are computed. We run each experiment 100 times and report the average values.

4.1 Performance Evaluation

The main difference between the two approaches lies on the friends set computation for the users of the query group. In the baseline approach, for each user of the query group, we compute his distance to all database users and the users within distance δ are selected. So, for each user of the query group, a total number of $|\mathcal{U}|$ distance computations is required, where $|\mathcal{U}|$ is the number of users in the database. If $|\mathcal{G}|$ is the number of users in the query group, the total number of distance computations is $|\mathcal{G}| \times |\mathcal{U}|$. In the user clustering approach, the friends of a user are taken directly from his cluster, so no further computations are required. There is though an initialization cost for creating the clusters ($\approx 2h$ for $\delta = 0.15$, $\approx 2.5h$ for $\delta = 0.3$), however this is done once and it can be reused afterwards for answering different query groups requests.

In Figure 1, we display the time complexity for the two approaches for different query group sizes under different distance thresholds δ. The user clustering

approach requires around 25% of the time required by the baseline approach. As $|\mathcal{G}|$ increases, the reduction becomes more evident, since the larger the query group size, the more the distance computations of the baseline approach. Note that for bigger threshold values, the execution time increases in both approaches, since each user of the query group has more friends/cluster members.

If the support is not taken into account (i.e., $w_1 = 1, w_2 = 0$), the running time slightly reduces for both approaches because less computations are required.

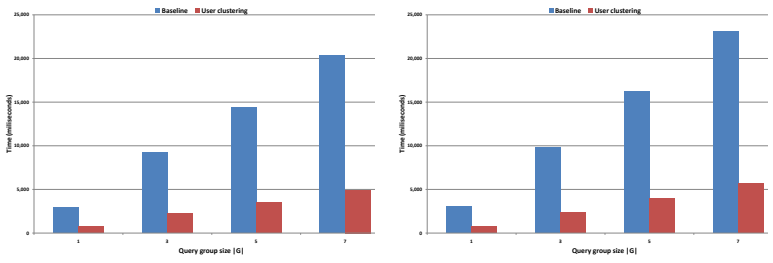
4.2 Quality Evaluation

In this set of experiments, we demonstrate the effectiveness of the user clustering approach. We compute the *commonRecs* and the *rankRecsDist* scores varying the query group size $|\mathcal{G}|$, the number k of the recommended items and the distance threshold δ.

Figure 2 depicts the *commonRecs* score for $\delta = 0.15$ and $\delta = 0.3$, for the three available designs, while Figure 3 depicts the corresponding *rankRecsDist* scores. $G1$ stands for groups with 1 member. Similarly, $G3$, $G5$ and $G7$ stand for groups with 3, 5 and 7 members, respectively. As the query group size $|\mathcal{G}|$ increases, the *commonRecs* score decreases, since the group recommendations rely in a more diverse set of users and personal values. Based on the same rationale, *rankRecsDist* increases as $|\mathcal{G}|$ increases. Also, we experimentally confirm that *commonRecs* increases with k, while *rankRecsDist* decreases.

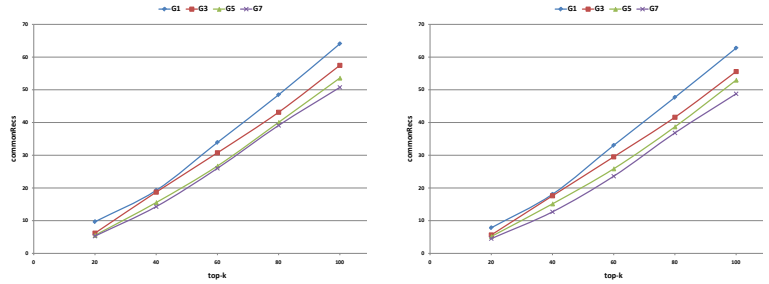
Regarding the different designs, we observe that the fair and the least misery designs achieve better results when compared to the most optimistic design. In our scenario, where the members of the query group are selected randomly, this is expected, since it is more difficult to find agreements for max personal values. When δ increases, the *commonRecs* decreases for the fair and least misery designs and slightly increases for the most optimistic design. Corresponding findings also hold for *rankRecsDist*.

Finally, we consider the case where only relevance is taken into consideration, that is, $w_1 = 1$ and $w_2 = 0$. In Figure 4, we show the results for the fair design and $\delta = 0.3$. *commonRecs* and *rankRecsDist* behave worst comparing to the equal importance case, showing that support improves the quality of recommendations.

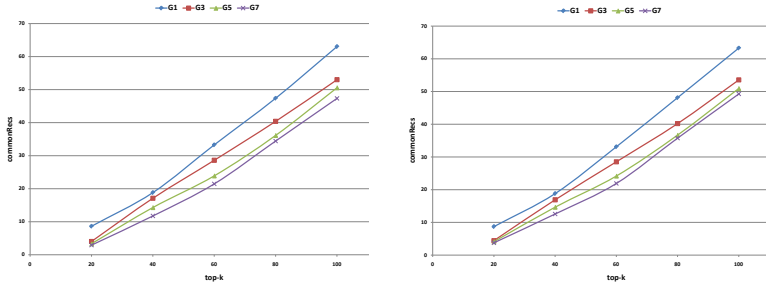


(a) $\delta = 0.15$ (b) $\delta = 0.3$

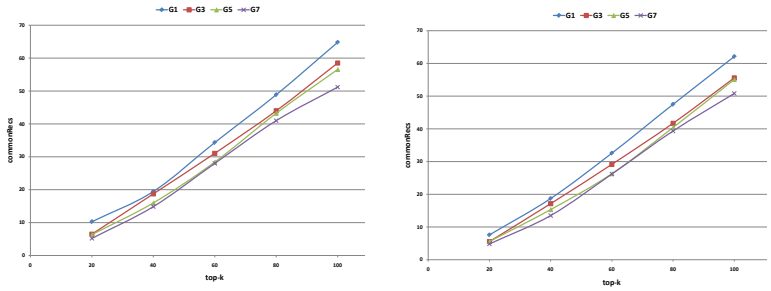
Fig. 1. Time complexity for the fair design with $w_1 = 0.5$, $w_2 = 0.5$.



(a) Fair design (δ=0.15) (b) Fair design (δ=0.3)



(c) Most optimistic design (δ=0.15) (d) Most optimistic design (δ=0.3)



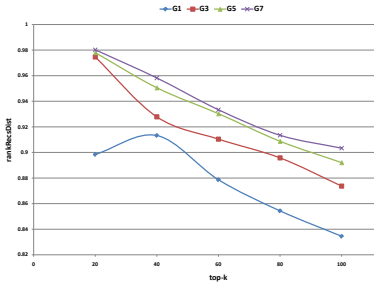
(e) Least misery design (δ=0.15) (f) Least misery design (δ=0.3)

Fig. 2. *commonRecs* for $\delta = 0.15$, $\delta = 0.3$ with $w_1 = 0.5$, $w_2 = 0.5$.

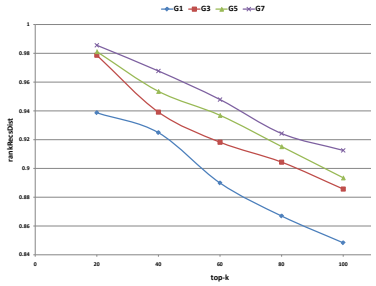
5 Related work

The research literature on recommendations is extensive. Typically, recommendation approaches are distinguished between: content-based, that recommend items similar to those the user previously preferred [17], collaborative filtering, that recommend items that users with similar preferences liked [14, 7] and hybrid ones [4]. Several extensions have been proposed, such as employing multi-criteria ratings [2] or further contextual information [20], and providing time-aware recommendations [26, 23]. Recently, there are also approaches focusing on extending database queries with recommendations [15, 22].

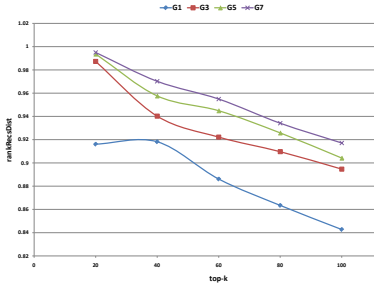
Moreover, there has been some recent work on group recommendations. PolyLens [19], a group recommendation system for movies, performed a user study to evaluate the usefulness of group recommendations. This approach em-



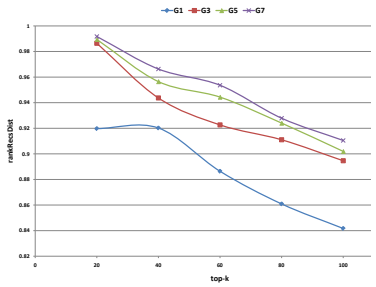
(a) Fair design



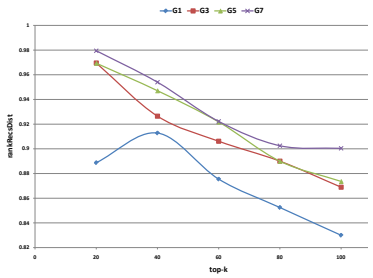
(b) Fair design



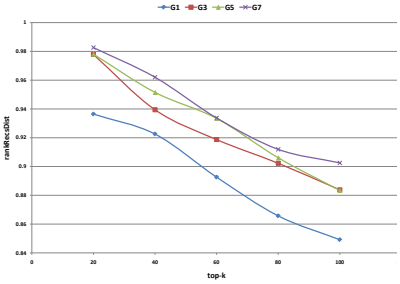
(c) Most optimistic design



(d) Most optimistic design

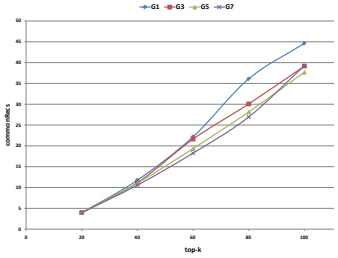


(e) Least misery design

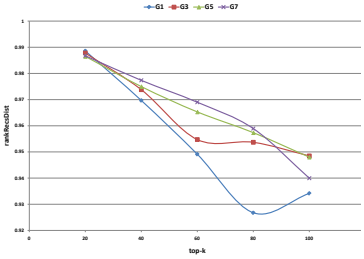


(f) Least misery design

Fig. 3. *rankRecsDist* for $\delta = 0.15$, $\delta = 0.3$ with $w_1 = 0.5$, $w_2 = 0.5$.



(a) *commonRecs*



(b) *rankRecsDist*

Fig. 4. Fair design for $\delta = 0.3$ with $w_1 = 1$, $w_2 = 0$.

ploys the least misery design to aggregate personal recommendations. [3] produces group recommendations by aggregating the personal ones using a consensus function that takes into account both the relevance of the items to the users and the level at which users disagree with each other. [11] proposes a group recommendation method that employs both the social and content interests of the users of a group, while [13] provides a two-step approach: first recommendations for groups are generated and then items are filtered to increase the individual users satisfaction. [5] compares the effectiveness of personal and group recommendations. To our knowledge, none of the proposed approaches perform user clustering for finding similar users. As part of our effort, we have also designed a user interface for group recommendations based on clustering [18].

A topic related to group recommendations is *rank aggregation*, that is, given a set of different rankings of items, produce a single ranking for the items. An instance of rank aggregation known as social choice studies the problem of determining the ranking of alternatives that is best for a group given the individual opinions of its members. Social choice has been studied extensively in economics, politics, sociology, and mathematics (e.g., [24]). Rank aggregation is also studied in the web, for example, in meta-search, where ranked lists of web pages produced by different search engines need to be combined into a single one (e.g., [8]). In database middleware, a related problem refers to finding the most preferred items when there are multiple rankings based on preferences defined on different attributes or dimensions of these items (e.g., [10]). [16] reviews different aggregation strategies or functions for group modeling.

6 Conclusions

We proposed an efficient framework for group recommendations, by organizing users into clusters of users with similar preferences. These clusters are used to efficiently locate similar users to a given one; this way, recommendations for users are produced with respect to the preferences of their cluster members without extensively searching for similar users in the whole database. Top-k group recommendations are computed by aggregating the personal recommendations of the individual users, while they are presented along with explanations on the reasons that the particular items are being suggested to the group. Our results show that employing user clustering considerably improves the execution time, while preserves a satisfactory quality of recommendations.

References

1. Movielens data sets, available online at: http://www.grouplens.org/node/12; visited on Nov. 2011.
2. Adomavicius, G., Kwon, Y.: New recommendation techniques for multicriteria rating systems. IEEE Intelligent Systems 22(3), 48–55 (2007)
3. Amer-Yahia, S., Roy, S.B., Chawla, A., Das, G., Yu, C.: Group recommendation: Semantics and efficiency. PVLDB 2(1), 754–765 (2009)
4. Balabanovic, M., Shoham, Y.: Content-based, collaborative recommendation. Commun. ACM 40(3), 66–72 (1997)
5. Baltrunas, L., Makcinskas, T., Ricci, F.: Group recommendations with rank aggregation and collaborative filtering. In: RecSys. pp. 119–126 (2010)

6. Berkovsky, S., Freyne, J.: Group-based recipe recommendations: analysis of data aggregation strategies. In: RecSys. pp. 111–118 (2010)
7. Breese, J.S., Heckerman, D., Kadie, C.M.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI. pp. 43–52 (1998)
8. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: WWW10 (2001)
9. Fagin, R., Kumar, R., Sivakumar, D., Sivakumar, D.: Comparing top k lists. In: SODA. pp. 28–36 (2003)
10. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS (2001)
11. Gartrell, M., Xing, X., Lv, Q., Beach, A., Han, R., Mishra, S., Seada, K.: Enhancing group recommendation by incorporating social relationship interactions. In: GROUP. pp. 97–106 (2010)
12. Jameson, A., Smyth, B.: Recommendation to groups. In: The Adaptive Web. pp. 596–627 (2007)
13. Kim, J.K., Kim, H.K., Oh, H.Y., Ryu, Y.U.: A group recommendation system for online communities. International Journal of Information Management 30(3), 212 – 219 (2010)
14. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: Grouplens: Applying collaborative filtering to usenet news. Commun. ACM 40(3), 77–87 (1997)
15. Koutrika, G., Bercovitz, B., Garcia-Molina, H.: Flexrecs: expressing and combining flexible recommendations. In: SIGMOD Conference. pp. 745–758 (2009)
16. Masthoff, J.: Group modeling: Selecting a sequence of television items to suit a group of viewers. User Modeling and User-Adapted Interaction 14(1), 37–85 (2004)
17. Mooney, R.J., Roy, L.: Content-based book recommending using learning for text categorization. In: ACM DL. pp. 195–204 (2000)
18. Ntoutsi, I., Stefanidis, K., Nørvåg, K., Kriegel, H.P.: grecs: A group recommendation system based on user clustering. In: DASFAA (2). pp. 299–303 (2012)
19. O'Connor, M., Cosley, D., Konstan, J.A., Riedl, J.: Polylens: A recommender system for groups of user. In: ECSCW. pp. 199–218 (2001)
20. Palmisano, C., Tuzhilin, A., Gorgoglione, M.: Using context to improve predictive modeling of customers in personalization applications. IEEE Trans. Knowl. Data Eng. 20(11), 1535–1549 (2008)
21. Rajaraman, A., Ullman, J.D.: Mining of massive datasets. Lecture Notes for Stanford CS345A Web Mining p. 328 (2010), `http://infolab.stanford.edu/~ullman/mmds.html`
22. Stefanidis, K., Drosou, M., Pitoura, E.: *You May Also Like* results in relational databases. In: PersDB. pp. 37–42 (2009)
23. Stefanidis, K., Ntoutsi, I., Nørvåg, K., Kriegel, H.P.: A framework for time-aware recommendations. In: DEXA (2012)
24. Taylor, A.: Mathematics and politics: Strategy, voting, power and proof. New York: Springer Verlag (1995)
25. Tintarev, N., Masthoff, J.: Designing and evaluating explanations for recommender systems. In: Recommender Systems Handbook, pp. 479–510. Springer (2011)
26. Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., Sun, J., Sun, J.: Temporal recommendation on graphs via long- and short-term preference fusion. In: KDD. pp. 723–732 (2010)
27. Yu, Z., Zhou, X., Hao, Y., Gu, J.: Tv program recommendation for multiple viewers based on user profile merging. User Model. User-Adapt. Interact. 16(1), 63–82 (2006)