The VLDB Journal (2010) 19:877–900
DOI 10.1007/s00778-010-0209-3

SPECIAL ISSUE PAPER

Space efficiency in group recommendation

**Senjuti Basu Roy · Sihem Amer-Yahia ·
Ashish Chawla · Gautam Das · Cong Yu**

Received: 15 August 2009 / Revised: 5 May 2010 / Accepted: 17 November 2010 / Published online: 12 December 2010

Abstract Imagine a system that gives you satisfying recommendations when you want to rent a movie with friends or find a restaurant to celebrate a colleague's farewell: at the core of such a system is what we call *group recommendation*. While computing individual recommendations have received lots of attention (e.g., Netflix prize), group recommendation has been confined to studying users' satisfaction with different aggregation strategies. In this paper (Some results are published in an earlier conference paper (Amer-Yahia et al. in VLDB, 2009). See Sect. "Paper contributions and outline" for details.), we describe the challenges and desiderata of group recommendation and formalize different *group consensus* semantics that account for both an item's *predicted ratings* to the group members and the *disagreements* among them. We focus on the design and implementation of efficient group recommendation algorithms that intelligently prune and merge per-user predicted rating lists and pairwise disagreement lists of items. We further explore the impact of space constraints on maintaining per-user and pairwise item lists and develop two complementary solutions that leverage shared user behavior to maintain the efficiency of our recommendation algorithms within a space budget. The first solution, *behavior factoring*, factors out user agreements from disagreement lists, while the second solution, *partial materialization*, selectively materializes a subset of disagreement lists. Finally, we demonstrate the usefulness of our group recommendations and the efficiency and scalability of our algorithms using an extensive set of experiments on the 10 M ratings MovieLens data set.

Keywords Group recommendation · Top-k algorithm

1 Introduction

Recommender systems have grown to become very effective in suggesting items of high relevance to individual users. Group recommendation, or the task of finding items that please a set of users, on the other hand, started to received attention relatively recently [3,4,6,11,13,17–19]. We envision a system that a community of users can consult when planning an activity together such as looking for a book for a reading club, finding a restaurant to celebrate a project milestone with colleagues, or renting a movie to watch at a girls' night out. In this paper, we study this problem with a focus on time and space efficiency.

Even more so than in traditional individual recommendation, identifying items of high relevance to a group is challenging: What if group members disagree on their favorite items (e.g., people who prefer non-fiction books vs those who like fiction, in a book reading club)? What if there is a group member whose tastes highly differ from all others (e.g., a vegetarian going to a restaurant with non-vegetarians)? At its core, group recommendation necessitates the modeling of disagreements between group members and aims to find items with high predicted rating that also minimize disagreements between group members. In other terms, it is more

S. Basu Roy (✉) · A. Chawla · G. Das
University of Texas at Arlington, Arlington, TX, USA
e-mail: senjuti.basuroy@mavs.uta.edu

A. Chawla
e-mail: achawla@uta.edu

G. Das
e-mail: gdas@uta.edu

S. Amer-Yahia
Yahoo! Research, Barcelona, Spain
e-mail: sihem@yahoo-inc.com

C. Yu
Google Research, New York, USA
e-mail: congyu@google.com

desirable to return an item that each group member is happy with than to return an item that polarizes group members even if the latter has higher average ratings among them. In this work, we formalize the notion of *consensus functions* that capture such real-world scenarios.

Intuitively, the general form of consensus functions is a weighted combination of predicted rating and pairwise disagreement. For a given user, her individual preference (i.e., predicted rating generated by an underlying recommender system) for items can be maintained in the so-called *predicted rating list*. We can then leverage Fagin-style merge algorithms [8] to generate items to be recommended to the group based on individual lists of items sorted by their predicted ratings to each group member. Unfortunately, while item disagreements between users can be computed from their predicted rating lists, they do not increase or decrease monotonically with the predicted ratings: two users who both think highly of an item may still disagree more on that item than on an item they both dislike. This drastically reduces the pruning power of the merge algorithms. To address this issue, we introduce *pairwise disagreement lists* which are precomputed from predicted rating lists and sorted in decreasing order of disagreements. Both predicted rating lists and pairwise disagreement lists can then be merged, using Fagin-style algorithms, to find items to recommend to a group.

Without prior knowledge of what groups can be formed between users, a disagreement list has to be created for every user pair. In practice, this introduces enormous space requirements. A back-of-the-envelope computation shows that with a modest 70K-user, 10K-item database, a total of about 2TB space is needed to store the 14 trillion list entries in pairwise disagreement lists. To address this concern, we develop space reduction strategies which exploit two key characteristics of disagreement lists. First, entries in those lists may be redundant due to shared user behavior. Our strategy that *factors out common entries* in disagreement lists without affecting I/O. Second, all lists do not contribute equally to processing time because of different rating distributions. We develop a *partial materialization* strategy that identifies which subset of lists to materialize in order to maximize space reduction and minimize processing time.

Intuitively, if two users (u and v) agree on many items, their disagreement lists with all other users will be the same for those items. In other terms, given any other user w, the entries corresponding to the items that u and v agree on in the (u, w) disagreement list will be the same as those in the (v, w) disagreement list. Hence, they can be stored only once, instead of being replicated in all lists. We call this *behavior factoring* in disagreement lists. We formalize the problem and devise an algorithm for efficiently factoring common entries in disagreement lists. This space-saving strategy requires changes to the group recommendation algorithm to process factored lists.

Factoring comes for free and always saves space when at least two users agree on some items. Unfortunately, if a space budget is imposed, factoring alone does not always guarantee to produce a set of lists within that budget. We further explore *partial materialization* as a complementary space reduction strategy which selectively materializes a subset of the disagreement lists. In a nutshell, a disagreement list that does not significantly affect processing time and consumes too much space should be dropped. Not surprisingly, partial materialization may negatively affect processing time since the benefit of non-materialized disagreement lists will be lost. We formulate partial materialization as a variant of the Knapsack problem and develop an algorithm which identifies the subset of lists to materialize.

1.1 Paper contributions and outline

We make the following contributions in this paper:

1. We formalize the problem of *top-k group recommendation* and use a model for group consensus similar to the social value functions developed in [18] to incorporate various predicted rating and disagreement models.
2. We propose the use of pairwise disagreement lists, and design and implement efficient group recommendation algorithms based on the merging and effective pruning of individual predicted rating lists and pairwise disagreement lists.
3. Given the potentially large number of disagreement lists, we exploit shared user behavior to reduce the space requirement of those lists. As a result, we extend the group recommendation algorithms to process factored lists. We show that factoring common entries in disagreement lists can drastically reduce storage space without incurring I/O overhead.
4. The factoring strategy does not always guarantee reaching a fixed space budget. To achieve a certain space budget, we develop a partial materialization strategy which exploits the size of each disagreement list and their impact on query processing: it skips disagreement lists in order to minimize space, while incurring small processing time overhead. We formalize this question as an adaptation of the Knapsack problem and develop an algorithm to solve it.
5. We run an extensive set of experiments with different group sizes on MovieLens data sets. We perform extensive user study in Amazon's Mechanical Turk to demonstrate the effectiveness of our group recommendation semantics and how satisfied users are with recommended group ratings compared to individual ones. Our elaborate performance experiments exhibit the efficiency of group recommendation computation. We also demonstrate the

benefit of behavior factoring and partial materialization on space.

We note here that the group recommendation problem definition and the basic group recommendation algorithms were first introduced in the conference version [3] of this paper. Furthermore, [3] also discussed partial materialization to a certain extent. However, the partial materialization algorithms described here address the problem in a more formal way. Behavioral factoring is introduced for the first time here. The rest of the paper is organized as follows. Section 2 provides some background and formalism. It describes the family of consensus functions we tackle in this paper and defines the group recommendation problem. Section 3 describes the group recommendation algorithm. Section 4 presents our behavior factoring strategy and a revision of the group recommendation algorithm to operate on factored lists. Section 5 discusses partial materialization in the presence of a space budget and develops our adaptation of the Knapsack problem to achieve partial materialization post factoring. Experiments are presented in Sect. 6. Section 7 contains the related work. We conclude in Sect. 8.

2 Background and data model

Let $\mathcal{U}$ denote the set of users and $\mathcal{I}$ denote the set of items (e.g., movies, travel destinations, restaurants) in the system. Each user u may have provided a rating for an item i in the range of 0 to 5, which is denoted as $\texttt{rating}(u, i)$. If the user has not provided a rating for an item, the $\texttt{rating}$ is set to $\perp$. We further generate *predicted ratings* for each pair of user and item, denoted as $\texttt{predictedrating}(u, i)$. This predicted rating comes from two sources. If the user has provided a rating for the item, then it is simply the user provided rating. Otherwise, it is generated by the system using a recommendation strategy as outlined next.

2.1 Individual recommendation model

We review the two most popular families of recommendation strategies. These strategies rely on finding items similar to the user's previously highly rated items (item-based), or on finding items liked by people who share the user's interests (user-based) [1].

2.1.1 Item-based strategies

These are the oldest recommendation strategies. They aim to recommend items similar to those the user preferred in the past. While different strategies use different approaches to compute the predicted rating, we present one common formulation. The rating of an item $i \in \mathcal{I}$ by a current user $u \in \mathcal{U}$ is estimated as follows:

$$\begin{aligned}&\texttt{predictedrating}(u, i)\\&\quad = Avg_{i' \in \mathcal{I}\ \&\ \texttt{rating}(u,i') \neq \perp} \texttt{ItemSim}(i, i')\\&\qquad \times \texttt{rating}(u, i').\end{aligned}$$

Here, $\texttt{ItemSim}(i, i')$ returns a measure of similarity between two items i and i'. Item-based strategies are very effective when the given user has a long history of rating activity. However, item-based strategy do not work well when a user first joins the system. To address that collaborative filtering strategies have been proposed, which we briefly describe next.

2.1.2 User-based strategies

These strategies aim to recommend items, which are highly rated by users who share similar interests with or have declared relationship with the given user. The key of this method is to find other users connected to the given user. The rating of an item i by a user u is estimated as follows:

$$\begin{aligned}&\texttt{predictedrating}(u, i)\\&\quad = Avg_{u' \in \mathcal{U}\ \&\ \texttt{rating}(u',i) \neq \perp} \texttt{UserSim}(u, u')\\&\qquad \times \texttt{rating}(u', i)\end{aligned}$$

Here, $\texttt{UserSim}(u, u')$ returns a measure of similarity or connectivity between two users u and u' (it is 0 if u and u' are not connected). Collaborative filtering strategies broaden the scope of items being recommended to the user and have become increasingly popular.

We note that there are also so-called fusion strategies [14], which combine ideas from item-based and collaborative filtering strategies, and model-based strategies, which leverage machine learning techniques. While we do not consider them in this paper, we note that group recommendation does not rely on one specific strategy to generate recommendations for individual group members.

2.2 Group recommendation model

The goal of group recommendation is to compute a recommendation score for each item to reflect the interests and preferences of all the group members. In general, group members may not always have the same tastes and a *consensus score* for each item needs to be carefully designed. Intuitively, there are two main aspects to the consensus score. First, the score needs to *reflect the degree to which the item is preferred by the members*. The more group members prefer an item, the higher its score should be for the group. Second, the score needs to *reflect the level at which members disagree or agree with each other*. All other conditions being equal, an item that members agree most about should have a higher score than an item with a lower overall group agreement. We call the first aspect *group predicted rating* and the second aspect *group disagreement*.

Definition 1 (*Group Predicted Rating*) The predicted rating of an item i by a group $\mathcal{G}$, denoted as $\texttt{grouppredicte-drating}(\mathcal{G}, \texttt{i})$, is an aggregation over the predicated rating of each group member, $\texttt{predictedrating}\ (u, i)$ where $u \in \mathcal{G}$. We consider two main aggregation strategies:

(1) *Average:*
$\texttt{grouppredictedrating}(\mathcal{G}, i)$
$= \frac{1}{|\mathcal{G}|} \sum (\texttt{predictedrating}(u, i))$

(2) *Least-Misery:*
$\texttt{grouppredictedrating}(\mathcal{G}, i)$
$= Min(\texttt{predictedrating}(u, i))$

Average and least-misery aggregation models are considered because they are the most prevalent mechanisms being employed currently [11]. Alternative aggregations (e.g. Most-Happiness, i.e., taking the maximum over all individual predicted ratings) are also possible.

Definition 2 (*Group Disagreement*) The disagreement of a group $\mathcal{G}$ over an item i, denoted $\texttt{dis}(\mathcal{G}, i)$, reflects the degree of consensus in the predicted ratings for i among group members. We consider the following two main disagreement computation methods:

(1) *Average Pairwise Disagreements:*
$\texttt{dis}(\mathcal{G}, i) = \frac{2}{|\mathcal{G}|(|\mathcal{G}|-1)} \sum (|\texttt{predictedrating}\ (u, i) - \texttt{predictedrating}(v, i)|)$,
where $u \neq v$ and $u, v \in \mathcal{G}$;
(2) *Disagreement Variance:*
$\texttt{dis}(\mathcal{G}, i) = \frac{1}{|\mathcal{G}|} \sum_{u \in \mathcal{G}} (\texttt{predictedrating}(u, i) - \texttt{mean}(\mathcal{G}, \texttt{i}))^2$, where $\texttt{mean}(\mathcal{G}, \texttt{i})$ is the mean of all the individual predicted ratings for the item i.

The average pairwise disagreement function computes the average of pairwise differences in predicted ratings for the item among group members, while the variance disagreement function computes the mathematical variance of the predicted ratings for the item among group members. Intuitively, the closer the predicted ratings for i between users u and v, the lower their disagreement for i. In Sect. 3.1, we will characterize the properties of both disagreement functions in detail.

Finally, we combine group predicted rating and group disagreement in the *consensus function*.

Definition 3 (*Consensus Function*) The consensus function, denoted $\texttt{F}(\mathcal{G}, i)$, combines the group predicted rating and the group disagreement of i for $\mathcal{G}$ into a single group recommendation score using the following formula: $\texttt{F}(\mathcal{G}, i) = w_1 \times \texttt{grouppredictedrating}(\mathcal{G}, i) + w_2 \times (1 - \texttt{dis}(\mathcal{G}, i))$, where $w_1 + w_2 = 1.0$ and each specifies the relative importance of predicted rating and disagreement in the overall recommendation score.

While one could design more sophisticated consensus functions (see [19] for an example), we adopt this general form of weighted summation of group predicted rating and group disagreement for its simplicity, and the fact that the family of threshold algorithms can be easily applied for the computation. We note here that the commonly used least-misery model maps to the case where $w_1 = 1.0$ and group predicted rating is aggregated using the least-misery function.

2.3 Problem statement

Problem (*Top-k Group Recommendation*). Given a user group $\mathcal{G}$ and a consensus function $\mathcal{F}$, identify a list $\mathcal{I}_{\mathcal{G}}$ of items such that:

1. $|\mathcal{I}_{\mathcal{G}}| = k$
2. Items in $\mathcal{I}_{\mathcal{G}}$ are sorted on their decreasing group recommendation score as computed by the consensus function $\mathcal{F}$, and $\nexists j \in \mathcal{I}$ s.t. $\texttt{F}(\mathcal{G}, j) > \texttt{F}(\mathcal{G}, i)$, $j \notin \mathcal{I}_{\mathcal{G}}, i \in \mathcal{I}_{\mathcal{G}}$.

3 Efficient computation of group recommendation

In this section, we discuss efficient group recommendation algorithms. We first examine the applicability of existing top-k processing algorithms, then present our solution. We then discuss how to improve our algorithm with threshold tightening strategies that benefit from users' predicted rating lists.

3.1 Applicability of top-k threshold algorithms

Many of the best algorithms for computing top-k items belong to the family of threshold algorithms [8]. Given an overall scoring function that computes the score of an item by aggregating scores from individual components, threshold algorithms consume sorted item lists that correspond to each component. Those input lists are scanned using sequential or random accesses, and the computation can be terminated earlier using stopping conditions based on score bounds (thresholds). Early stopping is possible when the scoring function is *monotone*, i.e., if component c is the only component in the scoring function and items i_1 and i_2 differ in their scores, the overall score of i_1 is no less than i_2's if i_1's score on c is no less than i_2's score on c.

Recall from Definition 3 that our consensus function is a weight summation of two components, *group predicted rating* and *group disagreement*. It is clear that the consensus function itself is monotone in the two individual components.

(a)		**(b)**	
u_1	u_2	u_1	u_2
(i_1, 4)	(i_1, 3)	(i_2, 3)	(i_1, 4)
(i_2, 3)	(i_2, 3)	(i_1, 4)	(i_2, 4)

Fig. 1 Group disagreement is not monotonic w.r.t. predicted rating lists

In other words, if two items have the same group disagreement, the item with the higher group predicted rating will have at least the same group recommendation score, and vice versa.

It is also clear that the two group predicted rating functions proposed in Definition 1 are themselves monotone in the predicted ratings of individual members. If all group members, except u, rate items i_1 and i_2 the same, i_1 will have at least the same group predicted rating score as i_2 if u rates i_1 no less than i_2. This holds for both the average and the least-misery strategies.

It is, however, not clear whether the group disagreement functions proposed in Definition 2 are monotone. In this section, we prove that the two group disagreement functions proposed can be transformed into aggregations of individual pairwise disagreements and become monotone. This means we can apply threshold algorithms to compute the overall recommendation score with individual predicted rating lists and pairwise disagreement lists as inputs and take advantage of the pruning power that threshold algorithms give us.

3.2 Monotonicity of group disagreements

We use a simple example group of two users to show that computing group disagreement based on predicted ratings of individual members is not monotonic. Figure 1a illustrates the two sorted predicted rating lists for the two users (u_1 and u_2). It is clear that while i_1 has a higher predicted rating for u_1 than i_2 (4 vs. 3), the group disagreement score for i_2 is in fact higher (1 instead of 0). The same non-monotonicity can be encountered when predicted rating lists are sorted in decreasing order (as shown in the example in Fig. 1b). Hence, the problem of non-monotonicity of disagreement in predicted rating lists persists regardless of the order in which predicted rating lists are sorted.

To address this problem, we propose to maintain *pairwise disagreement lists* instead and prove their monotonicity properties for the two group disagreement functions in Definition 2.

A pairwise disagreement list (or simply disagreement list) for users u and v is a list of items that are sorted in the increasing order of the difference between their predicted rating scores for u and v. For an item i, we use $\Delta^{i}_{u,v} = |\mathtt{predictedrating}(u,i) - \mathtt{predictedrating}(v,i)|$ to denote this predicted rating difference.

Observation 1 *The average pairwise disagreement function in Definition 2 is monotonic w.r.t. pairwise disagreement lists.*

Proof Let us assume a group $\mathcal{G} = \{u_1, u_2, \ldots, u_p\}$ with all its $p(p-1)/2$ disagreement lists (one for each user pair). Also assume that there are a total of t items, $\mathcal{I} = \{i_1, i_2, \ldots, i_t\}$. Note that we want to retrieve items with minimum disagreements first. Consider two items i_r and i_s within $\mathcal{I}$.

The group disagreement for i_r and i_s can be written as: $f \times \Sigma_{\forall j,k=1,2,\ldots,p}(\Delta^{i_r}_{u_j,u_k})$ and $f \times \Sigma_{\forall j,k=1,2,\ldots,p}(\Delta^{i_s}_{u_j,u_k})$, respectively, where $f = \frac{2}{p(p-1)}$ (see Definition 2).

Without loss of generality, assume we have $\Delta^{i_r}_{u_x,u_y} < \Delta^{i_s}_{u_x,u_y}$ and $\forall j,k = 1,2,\ldots,p$, $\Delta^{i_r}_{u_j,u_k} = \Delta^{i_s}_{u_j,u_k}$, where $(j,k) \neq (x,y)$. It is easy to see that $f \times \Sigma_{\forall j,k=1,2,\ldots,p}(\Delta^{i_r}_{u_j,u_k}) < f \times \Sigma_{\forall j,k=1,2,\ldots,p}(\Delta^{i_s}_{u_j,u_k})$.

If the number of disagreement lists is restricted to m,[1] the monotonicity property can still be maintained by assuming the minimum disagreement values (0) for any unavailable user pairs during top-k computation. □

In the disagreement variance model in Definition 2, disagreement over an item is defined as the variance in predicted ratings among all group members. In other words, the predicted rating by each member is compared against the mean predicted rating of the group. We now show that this disagreement function can in fact be monotonically aggregated from pairwise disagreement lists.

Observation 2 *The disagreement variance function in Definition 2 is monotonic w.r.t. pairwise disagreement lists.*

Proof Let us consider the group $\mathcal{G}$ and set of items $\mathcal{I}$ in Lemma 1. Consider two items i_r and i_s. The group disagreement of i_r and i_s can be written as:

$$\frac{\Sigma_{\forall j \in p}[\mathtt{predictedrating}(u_j, i_r) - \frac{\Sigma_{\forall i \in p}\mathtt{predictedrating}(u_i, i_r)}{p}]^2}{p}$$

and

$$\frac{\Sigma_{\forall j \in p}[\mathtt{predictedrating}(u_j, i_s) - \frac{\Sigma_{\forall i \in p}\mathtt{predictedrating}(u_i, i_s)}{p}]^2}{p}$$

We can transform this disagreement variance formula for i_r into (ignoring p):

$$[\Delta^{i_r}_{12} + \Delta^{i_r}_{13} + \cdots + \Delta^{i_r}_{1p}]^2 + [\Delta^{i_r}_{21} + \Delta^{i_r}_{23} + \cdots + \Delta^{i_r}_{2p}]^2 + \cdots + [\Delta^{i_r}_{p1} + \Delta^{i_r}_{p2} + \cdots + \Delta^{i_r}_{p(p-1)}]^2$$

[1] We discuss partial materialization of disagreements lists in Sect. 5.

which can be further expressed as:

$$[\Delta_{12}^{i_r}]^2 + \cdots + [\Delta_{1p}^{i_r}]^2 + \cdots + 2 \times [\Delta_{12}^{i_r}][\Delta_{13}^{i_r}] + 2 \times [\Delta_{12}^{i_r}][\Delta_{14}^{i_r}] + \cdots$$

It is clear that the above-mentioned formula is a monotonic aggregation of $[\Delta_{jk}] \forall j, k \in p$. Without loss of generality, assume we have $\Delta_{u_x,u_y}^{i_r} < \Delta_{u_x,u_y}^{i_s}$ and $\forall j, k \in p$, $\Delta_{u_j,u_k}^{i_r} = \Delta_{u_j,u_k}^{i_s}$, where $(j, k) \neq (x, y)$. It is easy to see that the disagreement variance of i_r is less than the disagreement variance of i_s. Hence, we have proved that using pairwise disagreement lists is sufficient to compute disagreement variance in a monotonic fashion. □

Materializing all possible pairwise disagreement lists may not be practical since the number of such lists grows quadratically in the number of users. We discuss behavior factoring in Sect. 4 to save space and in Sect. 5, we discuss, given a fixed space constraint, which pairs to materialize in order to produce the best performance with threshold algorithms.

3.3 Group recommendation algorithms

Given a group $\mathcal{G}$, the goal, stated in Sect. 2.3, is to return the k best items according to a consensus function F (see Definition 3). We describe several algorithms for this problem; with each algorithm being a variant of the well-known TA [8] for top-k query processing.

The Full Materialization (FM) Algorithm: We start by describing Algorithm 1, which admits predicted rating lists $\mathcal{IL}$ of each user in the input group $\mathcal{G}$ and disagreement lists $\mathcal{DL}$ for every pair of users in $\mathcal{G}$. $\mathcal{IL}s$ are sorted in decreasing order of predicted rating, and $\mathcal{DL}s$ are sorted in increasing order of disagreement. These predicted rating lists and disagreement lists of a group are akin to attributes on which the algorithm TA [8] works. We refer to Algorithm 1 as FM (Full Materialization).

Each $\mathcal{IL}$ is obtained using an individual recommendation strategy (as described in Sect. 2.1). Each $\mathcal{DL}$ is generated for a user pair and records the difference in scores for all items in their respective $\mathcal{IL}s$.

Algorithm 1 Group Recommendation Algorithm with Fully Materialized Disagreement Lists (FM)

```
Require: Group $\mathcal{G}$, consensus function F;
1: Retrieve predicted rating lists $\mathcal{IL}_u$ for each user $u$ in group $\mathcal{G}$;
2: Retrieve disagreement lists $\mathcal{DL}_{(u,v)}$ for each user pair $(u, v)$ in group $\mathcal{G}$;
3: $S_r = \{\cup r_u\}$, the last predicted rating from $\mathcal{IL}_u, \forall u \in \mathcal{G}$
4: $S_\Delta = \{\cup \Delta_{u,v}\}$, the last pairwise disagreement value read on disagreement list, $\forall (u, v) \in \mathcal{G}$
5: Cursor $cur$ = getNext() accesses predicted rating lists and disagreement lists in round-robin;
6: while ($cur$ <> NULL) do
7:   Get entry $e = (i, r)$ at $cur$;
8:   if !(inHeap(topKHeap, $e$)) then
9:     if (ComputeMaxScore($S_r, S_\Delta$, F) > topKHeap.$k_{th}score$) then
10:      $score$ = ComputeExactScore(i, F) by performing random accesses to all $\mathcal{IL}_us$ for item $i$;
11:      if $score$ > topKHeap.$k_{th}score$ then
12:        topKHeap.addToHeap($e.i$, $score$);
13:      end if
14:    else
15:      return topKList(topKHeap);
16:      Exit;
17:    end if
18:  end if
19:  $cur$ = getNext();
20: end while
21: return topKList(topKHeap);
```

We showed in Sect. 3.1 that pairwise disagreement lists guarantee monotonicity for both pairwise and variance disagreements, thereby allowing FM to rely on a threshold for early stopping. Our algorithm makes sequential access (SA) on each input lists (predicted rating and disagreement) in a round-robin fashion (lines 3 and 12) and reads an entry $e = (i, r)$, where i is the item-id and r is the predicted rating or disagreement value associated with it. There are two routines: ComputeExactScore that computes the score of the current item, and ComputeMaxScore that produces a new threshold value at each round. During the execution of the algorithm, we also maintain a bounded buffer(heap) which stores the top-k elements encountered thus far and their corresponding exact scores using the input consensus function F. If a new item is encountered during a sequential access (SA), ComputeExactScore performs a random access (RA) on all other predicted rating lists to compute the score of that item using the input consensus function F. The main difference between FM and TA is that while SAs are done on $\mathcal{IL}s$ and $\mathcal{DL}s$ interchangeably, RAs are only done on $\mathcal{IL}s$ (since disagreement can be computed from predicted ratings). In fact, $\mathcal{DL}s$ are not necessary to compute the final result. They are only there to compute the threshold and enable early termination.

ComputeMaxScore produces a new threshold value at each round. Its basic purpose is to provide an upper bound of the score of any item that has not yet been seen by the algorithm. Thus, if r_u is the last predicted rating value read on list $\mathcal{IL}_u$ for all $u \in \mathcal{G}$, and $\Delta_{u,v}$ the last pairwise disagreement value read on disagreement list $\mathcal{DL}_{u,v}$ for all $u, v \in \mathcal{G}$, then the upper bound for the threshold (assuming the average pairwise disagreement model) is computed as follows:

$$\mathtt{F}(\mathcal{G}, i) \le w_1 \times \frac{1}{|\mathcal{G}|} \sum_{u \in \mathcal{G}} r_u + w_2 \times \left(1 - \frac{2}{|\mathcal{G}|(|\mathcal{G}| - 1)} \sum_{u,v \in \mathcal{G}} \Delta_{u,v}\right)$$

The Ratings Only (RO) Algorithm: We next describe another variation of the algorithm, called RO (Ratings Only), which applies when only the predicted rating lists are present and none of the $\mathcal{DL}s$ are available. RO has the obvious benefit of consuming less space. As discussed earlier, the lack of disagreement lists does not have any impact on ComputeExactScore. However, it has an impact on how the ComputeMaxScore has to be modified to produce a (somewhat less tight) threshold value. More precisely, since disagreement lists are not available, we assume that the pairwise disagreement between each pair of users for any unseen item is 0. Thus, the upper bound for the threshold value only comes from the last values read from each predicted rating list:

$$\mathtt{F}(\mathcal{G}, i) \leq w_1 \times \frac{1}{|\mathcal{G}|} \sum_{u \in \mathcal{G}} r_u$$

The Partial Materialization (PM) Algorithm: Finally, the most general variant is the case where only some disagreement lists are materialized, referred to as PM (Partial Materialization). As with RO, PM also has the obvious benefit of consuming less space than FM. In terms of processing, it differs from the others in how the threshold is computed. Let M be the set of all pairs of users for which disagreement lists have been materialized, the threshold can be computed as follows:

$$\mathtt{F}(\mathcal{G}, i) \leq w_1 \times \frac{1}{|\mathcal{G}|} \sum_{u \in \mathcal{G}} r_u + w_2 \times \left(1 - \frac{2}{|\mathcal{G}|(|\mathcal{G}| - 1)} \sum_{(u,v) \in M} \Delta_{u,v}\right)$$

Intuitively, one may think that the more $\mathcal{DL}s$ are materialized, the tighter the score bound and hence, the faster the algorithm terminates. It turns out that it is not always the case. The basic intuition is that overall performance is a balance between the total number of distinct items which need to be processed before finding the best k items, referred to as DIP (Distinct Items Processed), and the number of sequential accesses, SAs, that result from the proliferation of disagreement lists. Consider the case of a 3-member group. The question we ask ourselves is when does using two materialized lists, $\mathcal{DL}_1$ and $\mathcal{DL}_2$, perform worse than when only one materialized list, say $\mathcal{DL}_1$, is used? If none of top items in $\mathcal{DL}_2$ is in the final output, each SA on $\mathcal{DL}_2$ is pure overhead. This is exacerbated if the top items in $\mathcal{DL}_1$ and $\mathcal{DL}_2$, i.e., the ones with the least disagreement, are distinct. In both cases, if $\mathcal{DL}_2$ does not provide an opportunity to tighten the threshold, the number of SAs using $\mathcal{DL}_1$ and $\mathcal{DL}_2$ will be much higher than the number of SAs where only $\mathcal{DL}_1$ is used.

The PM variant raises an interesting question—which pairwise disagreement lists should be materialized as a preprocessing step? This partial list materialization problem is discussed in the Sect. 5. But first, in Sect. 3.4, we discuss interesting and novel techniques by which the threshold bounds can be sharpened even further.

3.4 Sharpening thresholds

In this subsection, we examine the different variants of the TA algorithm that we have developed thus far—FM, RO and PM—and suggest techniques by which their performance can be further improved, mainly by modifying the ComputeMaxScore function to compute sharper thresholds that enable earlier termination.[2]

Our approach is best illustrated by the following simple example. Consider a group consisting of two users $\mathcal{G} = \{u, v\}$. Recall that $\mathcal{IL}_u$ (resp. $\mathcal{IL}_u$) is the relevant list for user u (resp. v), and $\mathcal{DL}_{(u,v)}$ is the disagreement list of user pair u and v. Assume that the disagreement list has been materialized.

Consider a snapshot of the FM algorithm after a certain number of iterations. Let $r_u = 0.5$, $r_v = 0.5$ and $\Delta_{u,v} = 0.2$ be the last predicted rating and disagreement values read from each list, respectively. The task of the ComputeMaxScore function is to provide an upper bound on the maximum possible value of the consensus function $\mathtt{F}(\mathcal{G}, i)$ for any item i that has not yet been seen in any of the lists. Let the unseen item i's unknown predicted rating values be i_u and i_v for user u and v, respectively. The consensus function is defined as:

$$\mathtt{F}(\mathcal{G}, i) = (i_u + i_v)/2 + (1 - |i_u - i_v|/1) \tag{1}$$

Since each list is sorted in decreasing order of predicted rating (increasing order of disagreement), it should be clear that the following inequalities hold:

$$0 \leq i_u \leq 0.5$$
$$0 \leq i_v \leq 0.5$$
$$0.2 \leq |i_u - i_v| \leq 1$$

As described in Sect. 3.3, our current approach provides a simple upper bound for $\mathtt{F}(\mathcal{G}, i)$ by substituting the upper bounds for i_u and i_v (and the lower bound for $|i_u - i_v|$) from the above inequalities, to arrive at the following threshold:

$$\mathtt{F}(\mathcal{G}, i) \leq (0.5 + 0.5)/2 + (1 - 0.2/1) = 0.5 + 0.8 = 1.3$$

However, a more careful examination of the inequalities reveals that this bound is not tight. Notice that i_u and i_v should be at least 0.2 units apart, thus both cannot be at 0.5. Since the upper bound of i_u is 0.5, i_v can be at most 0.3. Thus, we

[2] While these techniques appear very promising, we note that they are the subject of our ongoing investigations—we discuss them in this version of the paper primarily to illustrate their potential.

can derive a sharper bound for $\texttt{F}(\mathcal{G}, i)$ as follows:

$$\texttt{F}(\mathcal{G}, i) \leq (0.5 + 0.3)/2 + (1 - 0.2/1) = 0.4 + 0.8 = 1.2$$

This example illustrates that due to the dependencies between the disagreement lists and the predicted rating lists, there are opportunities for deriving sharper thresholds for early termination after each iteration of the algorithm. More generally, after every iteration, we are faced with a formal *optimization problem* where we seek to maximize the consensus function over $|\mathcal{G}|$ real-valued variables, subject to various constraints on their values arising from the cursor positions on the predicted rating and disagreement lists. These optimization problems have seemingly complex formulations, because the consensus function as well the inequalities arising from disagreement lists are non-linear, involving absolute terms (e.g., of the form $|i_u - i_v|$) in the case of average pairwise disagreement, as well as quadratic terms (e.g., of the form $(i_u - \texttt{mean})^2$) in the case of variance-based disagreement.

In this paper, we conduct a detailed investigation of the optimization problem involving the pairwise disagreement model. Presence of absolute terms in the inequalities and consensus function makes the optimization problem non-linear; however, we realize that the non-linear optimization problem can be reformulated as multiple linear optimization problems. Solution to this non-linear optimization can be achieved by solving each linear optimization problems individually and finally selecting the linear optimization solution that offers the maximum objective value.

Using LP-based reformulation technique, optimization problem in Eq. 1 can be reformulated as two linear optimization problems:

(a) Maximize

$$\texttt{F}(\mathcal{G}, i) = (i_u + i_v)/2 + (1 - (i_u - i_v)/1)$$

s.t.

$$0 \leq i_u \leq 0.5$$
$$0 \leq i_v \leq 0.5$$
$$0.2 \leq (i_u - i_v) \leq 1$$

and

(b) Maximize

$$\texttt{F}(\mathcal{G}, i) = (i_u + i_v)/2 + (1 - (i_v - i_u)/1)$$

s.t.

$$0 \leq i_u \leq 0.5$$
$$0 \leq i_v \leq 0.5$$
$$0.2 \leq (i_v - i_u) \leq 1$$

Solution to problem 1 is the maximum of the objective values that linear optimization problems (a) and (b) take. In general, consensus function involving n variables requires $n!$ linear reformulations and solving each of them individually for obtaining the correct optimization value. However, at the same time, the sizes of the problems themselves are very small, consisting of only a few variables and constraints (assuming user group sizes are small), and thus are likely to be efficiently solvable by reformulating the problem into multiple linear optimization problems with practically no overhead per iteration. Note that this reformulation only works for the absolute operator in the consensus function (pairwise disagreement model) and not for the quadratic operator (variance-based disagreement model).

4 Behavior factoring

In this section, we explore our first space-saving strategy, which relies on factoring shared behavior from disagreement lists. The intuition is that if two users have the same rating on a subset of the items, they can be treated as a single virtual user whose disagreement lists with other users should only be stored once. More precisely, if two users u and v agree on a set of items $\mathcal{S}$, their disagreement lists $\mathcal{DL}_{(u,w)}$ and $\mathcal{DL}_{(v,w)}$ with any other user v share the same disagreement values for items in $\mathcal{S}$. An extreme case is when u_1 and u_2 agree on every single item, the two lists $\mathcal{DL}_{(u,w)}$ and $\mathcal{DL}_{(v,w)}$ are the same. We begin by defining the *factoring set* of a pair of users.

Definition 4 (*Factoring Set*) A factoring set for a pair of users u and v is the largest set of items in which u and v agree. This set is referred to as $\mathcal{S}_{(u,v)} \subseteq \mathcal{I}$ and is defined as $\forall i \in \mathcal{S}_{(u,v)}$, $\Delta^i_{u,v} = 0$, where $\Delta^i_{u,v} = |\texttt{predictedrating}(u, i) - \texttt{predictedrating}(v, i)|$

Given a pair of users, (u, v), $\forall w \in \mathcal{U}$ s.t., w is different from u and v, the disagreement lists $\mathcal{DL}_{(u,w)}$ and $\mathcal{DL}_{(v,w)}$, share the same values for items in $\mathcal{S}_{(u,v)}$.

We define a configuration $\mathcal{C}$ as the set of disagreement lists materialized for a user base $\mathcal{U}$. The algorithms developed in Sect. 3 admit different configurations as input. FM accepts a configuration where a disagreement list is created for every user pair in $\mathcal{U}$. RO accepts an empty configuration (since it only processes predicted rating lists).

Given a space constraint m (number of entries for storing materialized disagreement lists) and a configuration

$\mathcal{C}$, factoring aims to output a configuration ($\mathcal{C}'$) such that $\texttt{size}(\mathcal{C}') \leq m$, where $\texttt{size}(\mathcal{C}') = \sum_{\mathcal{DL}_{(u,v)} \in \mathcal{C}'}(|\mathcal{DL}_{(u,v)}|)$. The size of predicted rating lists $\mathcal{IL}_{(u,v)}$ is ignored since they are not affected by factoring. We next describe the factoring algorithm in Sect. 4.1 and the modification to query processing in the presence of factored lists in Sect. 4.2.

4.1 Factoring algorithm

The outline of the algorithm is as follows: factoring begins by deciding the user pair which has the largest factored set (say user pair (u, v)). The factored set is removed from the original disagreement list of (u, v). That set is also removed from every other original disagreement list that is shared by either u (or v), and a third user (say w) and their disagreements on those items are stored only once in a common list (note that in the original case, items of the factored set are present in both (u, w) and (v, w)'s disagreement lists). This step overall achieves a space reduction. However, if the space budget (m) is not satisfied yet, same factoring strategy is repeated on the user base which has all other users except u and v. This factoring process is reiterated unless one of these two conditions are satisfied: a) overall space is reduced under m or b) no more factoring is possible.

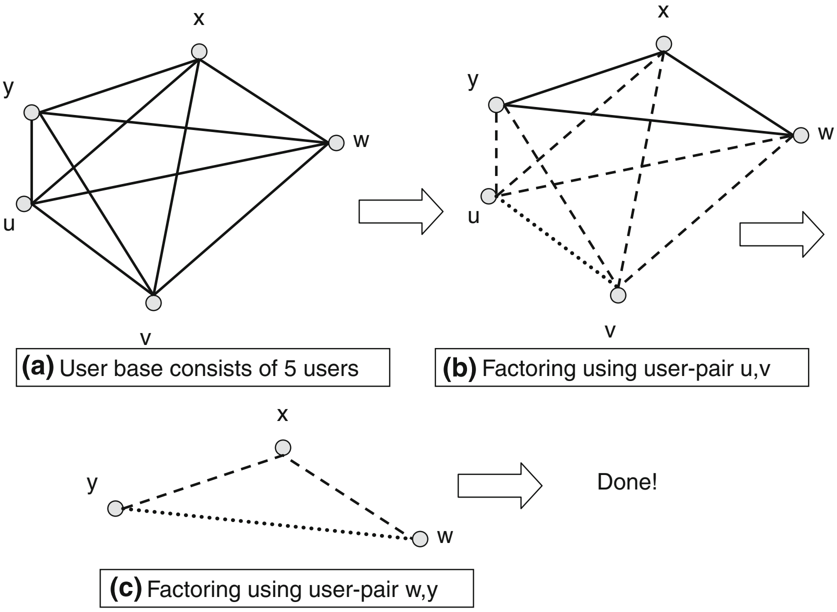
Consider Fig. 2 that illustrates one complete run of the proposed factoring algorithm on an example user base of size 5, $\{u, v, w, x, y\}$. Inputs to the factoring algorithm are a space budget (m, total no of entries in all pairwise disagreement lists in the user base), and the set of all possible pairwise disagreement lists of the user base. Figure 2a models the user base in form of a 5-node clique, where each user contributes one node in that clique. An edge between a pair of nodes is the pairwise disagreement list between them. Note that, initially, the presence of all pairwise disagreement lists make this graph complete, as shown in Fig. 2a. Next, it aims to compute factoring sets for every user pair and identify the user pair which has the largest factoring set. (Since all disagreement lists are of same size, largest factored set attains the highest space reduction.) Let that user pair be (u, v), as shown in Fig. 2b. Once (u, v) is identified, the disagreement between u and any other user, and disagreement between v and the same user, over items in their factoring set, are factored out and only stored once. The core primitive in the algorithm is to consider one triangle of users at a time involving edge (u, v) and perform factoring. Note that Fig. 2b explains this step where $\mathcal{S}_{(u,v)}$ is factored out from disagreement list $\mathcal{DL}_{(u,v)}$. Next, $\mathcal{S}_{(u,v)}$ is factored out from $\mathcal{DL}_{(u,w)}$ and $\mathcal{DL}_{(v,w)}$ and is stored only once in $\mathcal{DL}_{\mathcal{S}_{(u,v)},w}$. Similarly, $\mathcal{S}_{(u,v)}$ is factored out from $\mathcal{DL}_{(u,x)}$, $\mathcal{DL}_{(v,x)}$ and $\mathcal{DL}_{(u,y)}$,$\mathcal{DL}_{(v,y)}$ and stored once in $\mathcal{DL}_{\mathcal{S}_{(u,v)},x}$ and $\mathcal{DL}_{\mathcal{S}_{(u,v)},z}$, respectively. Conceptually, this step involves modifications of 3 triangles involving edge (u, v) for the given user base that consists of 5 users. For each triangle, overall space reduction is $\{2 \times |\mathcal{S}_{(u,v)}|\}$ after factoring. Note that, user pairs that do not involve either u or v are not affected so far in this factoring step. We show (w, x), (w, y) and (x, y) in solid lines in Fig. 2b which remain unaffected after factoring w.r.t. user pair (u, v).

Next, the factoring algorithm checks if the overall space now satisfies the specified space budget. It stops immediately if that condition is satisfied. Otherwise, it continues to the next step where it considers the largest complete graph (of size >= 3) which is not yet affected by factoring (the size 3 clique in the example). Note that the clique size gets reduced by 2 in two successive steps. Therefore, the algorithm computes the factored sets of the user pairs (w, x), (w, y), (x, y) and selects the one which has the largest factored set (say (w, y) as shown in Fig. 2c). It adheres to the same factoring strategy as earlier by factoring out $\mathcal{S}_{(w,y)}$ from $\mathcal{DL}_{(w,y)}$, $\mathcal{DL}_{(w,x)}$, $\mathcal{DL}_{(y,x)}$ and storing it only once in $\mathcal{DL}_{\mathcal{S}_{(w,y)},x}$. The overall space reduction in this step is $\{2 \times |\mathcal{S}_{(w,y)|}\}$. Note that after this step, all disagreement lists are affected by factoring. Hence, the algorithm stops and outputs the factored disagreement lists.

Algorithm 2 summarizes the factoring strategy. One artifact of this factoring algorithm is it requires at least 3 user pairs to be effective. Note that, any disagreement list is factored out into at most two parts using our factoring strategy. We intend to explore more complex factoring techniques in the future.

Algorithm 2 Factoring

```
Require: Configuration $\mathcal{C}$, space budget $m$
Compute space requirement of userbase ProcessSpace($\mathcal{G}$) as $StorageR$.
ProcessedPair = null;
if (($|\mathcal{C}|$ $mod$ 2 = 0) and ($|ProcessedPair| = |\mathcal{G}| - 2$)) then
    Exit;
end if
if (($|\mathcal{G}|$ $mod$ 2 $\neq$ 0) and ($|ProcessedPair| = |\mathcal{G}| - 1$)) then
    Exit;
end if
while StorageR > $m$ do
    for each user pair (($u, v$) $\in \mathcal{G}$) do
        if (u $\notin$ ProcessedPair) and (v $\notin$ ProcessedPair) then
            Configuration $\mathcal{C}_{(u,v)}$ = Factor($\mathcal{G}$, u, v);
            Compute storage requirement of $\mathcal{C}_{(u,v)}$ as Process-
            Space($u, v$);
            Compute $\Delta S_{(u,v)} = StorageR - \mathcal{C}_{(u,v)}$;
            Store Configuration $\mathcal{C}_{(u,v)}$, $\Delta S$ and ProcessSpace($u, v$) in
            CompProcessList;
        end if
    end for
    Select Configuration($\mathcal{C}_{(x,y)}$) such that $\Delta S_{(x,y)}$ is maximum.
    Set StorageR = ProcessSpace($x, y$);
    Set ProcessedPair = {$x, y$};
end while
return $\mathcal{C}_{ProcessedPair}(StorageR)$;
```



(a) User base consists of 5 users

(b) Factoring using user-pair u,v

(c) Factoring using user-pair w,y

Fig. 2 Factoring steps

```
Algorithm 3 Subroutine—Factor
Require: Configuration $\mathcal{C}$, user pair $u, v$
{Perform factoring of a Configuration wrt a particular user pair.}
Modify $\mathcal{DL}_{(u,v)}$ into $\mathcal{DL}_{\mathcal{S}_{(u,v)}}$ such that any item $\in \mathcal{DL}_{\mathcal{S}_{(u,v)}}$ is sorted in increasing disagreement value and $> 0$;
Add $\mathcal{DL}_{\mathcal{S}_{(u,v)}}$ in the Configuration($\mathcal{C}_{(u,v)}$);
Create list $\mathcal{DL}_{\mathcal{C}_{(u,v)}}$ from $\mathcal{DL}_{(u,v)}$ such that all items in $\mathcal{DL}_{\mathcal{C}_{(u,v)}}$ are 0.
for each $x \in \mathcal{G}$ and $(x \neq u, v)$ do
    Decompose $\mathcal{DL}_{(x,u)}$ and $\mathcal{DL}_{(x,v)}$ in three lists
    Create disagreement list $\mathcal{DL}_{\mathcal{S}_{(u,v)},x}$ for items present in $\mathcal{DL}_{\mathcal{C}_{(u,v)}}$
    Create $\mathcal{DL}_{\mathcal{S}_{(x,u)}}$ from $\mathcal{DL}_{(x,u)}$, $\mathcal{DL}_{\mathcal{S}_{(x,v)}}$ from $\mathcal{DL}_{(x,v)}$ such that an item $\in$ ($\mathcal{DL}_{\mathcal{S}_{(x,u)}}$ or $\mathcal{DL}_{\mathcal{S}_{(x,v)}}$) is not in $\mathcal{DL}_{\mathcal{S}_{(u,v)},x)}$
    Add $\mathcal{DL}_{\mathcal{S}_{(u,v)},x)}$, $\mathcal{DL}_{\mathcal{S}_{(x,u)}}$ and $\mathcal{DL}_{\mathcal{S}_{(x,v)}}$ in Configuration($\mathcal{C}_{uv}$);
end for
return Configuration($\mathcal{C}_{(u,v)}$);
```

```
Algorithm 4 Subroutine—ProcessSpace
{Computes the space (number of entries) required to store a particular configuration}
Require: Configuration $\mathcal{C}$;
for each list $\mathcal{DL}_{\mathcal{S}_{(i)}} \in \mathcal{C}$ do
    Compute $TotalSpace = TotalSpace + \mathcal{DL}_{\mathcal{S}_{(i)}}$;
end for
return $TotalSpace$;
```

Factoring $\mathcal{S}_{(u,v)}$ from the list $\mathcal{DL}_{(u,w)}$ (resp., $\mathcal{DL}_{(v,w)}$) results in converting $\mathcal{DL}_{(u,w)}$ (resp., $\mathcal{DL}_{(v,w)}$) into two lists: a *factored list* $\mathcal{DL}_{(u-\mathcal{S}_{(u,v)},w)}$ (resp., $\mathcal{DL}_{(v-\mathcal{S}_{(u,v)},w)}$), and a *common list*, $\mathcal{DL}_{\mathcal{S}_{(u,v)},w}$. In this case, the space saving is proportional to the size of the factoring set, $|\mathcal{S}_{(u,v)}|$. Hence, the larger the factoring set the higher the saving. Note that, factoring may fail to reach the specified budget (m) if factored sets are not large enough to reduce the overall space consumption to that extent. In fact, at the worst case, factoring fails to reduce any space if all factored sets are of length 0. However, factoring preserves all information of the original pairwise disagreement lists and thus achieves space reduction without impacting performance.

4.2 Impact of factoring on query processing

Algorithm 1 (FM) in Sect. 3.3 admits a group and a configuration containing all disagreement lists and outputs the best recommendations to the group given a consensus function. Here, we discuss how to adapt the algorithm to the case of a factored configuration where at least one disagreement list is factored out.

It turns out all is needed is to redefine `getNext()` to adapt query processing to work on factored disagreement lists. The main algorithm (Algorithm 1) does not need to be aware of such lists. Given a disagreement list $\mathcal{DL}_{(v,w)}$ which has been factored into two lists $\mathcal{DL}_{(v-\mathcal{S}_{(u,v)},w)}$ and $\mathcal{DL}_{\mathcal{S}_{(u,v)},w)}$, the `getNext()` routine on $\mathcal{DL}_{(v,w)}$ decides whether to advance the cursor on one list or the other. The decision is simply based on choosing the entry with the highest agreement value (lowest disagreement) among those two lists.

A consequence of confining the implementation to `getNext()` is that factoring does not modify the number of I/Os which makes it an appealing space-saving strategy.

5 Partial materialization

In the previous section, we discussed the preprocessing technique of factoring that reduces the space required to store all the pairwise disagreement lists between users. However, if the set of n users is large, since the number of user pairs is quadratic in n, factoring alone may not be enough to reduce the space to manageable proportions. In such cases, it is more practical to materialize (i.e., retain) only a small but effective subset of the disagreement lists. The central problem that we consider in this section is thus: given a fixed space constraint m, to determine (after factoring) which lists to materialize such that the total space consumed by these lists is at most m, and these lists are of "maximum benefit" during recommendation processing.[3]

Intuitively, a (factored) disagreement list should be materialized if (a) the corresponding users together are more likely to be a part of the same group, and (b) materializing the list significantly improves the running time of top-k recommendation algorithms. In the following subsections, we formalize this problem and develops algorithms to address it.

Our discussion will proceed in two stages. We shall first consider the simple scenario, where factoring has not been applied to a configuration C. In this scenario, all disagreement lists in C are equal in size, and each contains r entries where r is the total number of items. In Sect. 5.1, we discuss a simple algorithm that materializes a subset (at most m/r) of these disagreement lists that are of maximum benefit during recommendation processing, i.e., such that the average processing time is least affected.

However, once factoring has been performed, each original disagreement list may be composed of up to two factored lists of varying sizes. For example, using the factoring set $\mathcal{S}_{(u,v)}$, a disagreement list $\mathcal{DL}_{(v,w)}$ will be decomposed into two lists $\mathcal{DL}_{(v-\mathcal{S}_{(u,v)},w)}$ and $\mathcal{DL}_{\mathcal{S}_{(u,v)},w)}$. The sum of those lists' sizes is the same as that of $\mathcal{S}_{(u,v)}$. We discuss this more general situation in Sect. 5.2, where the task is to materialize a subset of the (factored) disagreement lists such that the total number of entries in all the materialized lists is m, and the average processing time is least affected. We formalize this new problem as an adaptation of the well-known NP-hard Knapsack Problem [16] and develop an approximation algorithm to address it.

5.1 Partial materialization without factoring

Let the set of users be $\mathcal{U} = u_1, \ldots, u_n$. Recall that $\mathcal{IL}_u$ is the predicted rating list for user u, and $\mathcal{DL}_{(u,v)}$ is the disagreement list of user pair u and v. Let the set of all possible user pairs in $\mathcal{U}$ be $S = \{(u, v)|u, v \in \mathcal{U}\}$. Let $M \subset S$ be the (unknown) subset of user pairs whose corresponding disagreement lists we wish to materialize (i.e., $|M| = m/r$). Let $\mathcal{G} \subseteq \mathcal{U}$ be any user group. Let $p(\mathcal{G})$ be the probability (or likelihood) that $\mathcal{G}$ will be the next "query", i.e., the next group that will seek item recommendations. Let $t_M(\mathcal{G})$ be the execution time of the top-k algorithm on user group $\mathcal{G}$ when run using the predicted rating lists $\mathcal{IL}_u$ (for all $u \in \mathcal{G}$) *as well as* the disagreement lists $\mathcal{DL}_{(u,v)}$ (for all $u, v \in \mathcal{G}$) that have been materialized in M, i.e., using algorithm FM. (Note that therefore $t_\phi(\mathcal{G})$ denotes the execution time of the top-k algorithm on user group $\mathcal{G}$ when run using only the predicted rating lists $\mathcal{IL}_u$ (for all $u \in \mathcal{G}$), i.e., using algorithm RO.

Our objective is to minimize the expected cost of executing the top-k algorithm on any user group query, using the predicted rating lists as well the disagreement lists. Let the expected cost be denoted as t_M. The partial materialization of disagreements list problem may now be formally defined as follows.

Problem (*Partial Materialization Without Factoring*). Determine the subset of pairs $M \subseteq S$ s.t. $|M| = m/r$ and $t_M = \sum_{\mathcal{G} \subseteq \mathcal{U}} p(\mathcal{G}) t_M(\mathcal{G})$ is minimized.

Although clearly very important and practical, the partial materialization problem is unfortunately quite hard to solve optimally. There are several reasons for this. First, it is very difficult to get reliable and accurate estimates for the distribution $p(\mathcal{G})$, i.e., the probability that a given user group $\mathcal{G}$ will be queried next. Moreover, the set of possible user groups is exponential in n, so it is not clear how such information can be compactly represented, even if it were reliably available. Next, due to the complex dependencies involved, it is very hard to estimate the impact of a materialized disagreement list in improving the running time of a top-k algorithm, without actually materializing candidate disagreement lists and running the top-k algorithms with and without the lists to determine their benefit. Finally, an important parameter of a top-k algorithm is the value of k, which is usually unknown at preprocessing time. As a first step toward addressing these challenges, we propose several principled and practical solutions.

5.1.1 *A simplifying assumption, and a simple lists materialization algorithm*

In order to make the problem more tractable, we make the following simplifying assumption. We assume that each future user group query $\mathcal{G}$ will only contain exactly two users, and moreover, $p(\mathcal{G})$ is reliably known for all pairs of users $\mathcal{G}$. This assumption is of course patently false, but we emphasize here that we use it only for simplifying the computation of M. Once M has been computed and the corresponding disagreement lists materialized, we shall later show that they can be used at query time for answering any user group $\mathcal{G}$, even groups containing more than two users.

This assumption considerably simplifies the computation of M, which can now proceed as follows. Recall that S is the set of all $n(n-1)/2$ pairs of users. For every pair

[3] We assume that m represents a user-specified threshold on the total number of entries in all the materialized disagreement lists.

of users u and v, we temporarily materialize the disagreement list $\mathcal{DL}_{(u,v)}$ and compute $t_{\{(u,v)\}}(\{u, v\})$ as well as $t_{\phi}(\{u, v\})$ by running the top-k algorithm twice, once with the disagreement list and once without the disagreement list, respectively.[4]

We can then eliminate from S those pairs $\{u, v\}$ where $t_{\{(u,v)\}}(\{u, v\}) \geq t_{\phi}((u, v))$

Although situations where the additional use of a disagreement list actually hurts the top-k execution may appear counter-intuitive, they can occur. For example, consider two users that are very similar to each other (e.g., they agree on most items) or are very dissimilar to each other (e.g., they disagree on most items). In both cases, their disagreement list contains very similar disagreement values (mostly 0's, or mostly 1's, respectively), and consequently is of no help in forcing early termination of the top-k algorithm, and in fact hurts the execution because of the extra sequential list accesses incurred. A disagreement list is useful for forcing early termination *only if there is significant skew in its disagreement scores,* i.e, at the top of the list the users agree on most items, whereas their disagreement is more pronounced as we go deeper into the list.

Let the remaining set of pairs be S'. Then, we should select M from S' such that following expression is maximized: $\sum_{(u,v)\in M} p(\{u, v\}) \cdot (t_{\phi}(\{u, v\}) - t_{\{(u,v)\}}(\{u, v\}))$

Algorithm 5 Partial Materialization Without Factoring

```
Require: User pairs in $S'$;
1: Sort the pairs $(u, v) \in S'$ by decreasing $p(\{u, v\}) \cdot (t_{\phi}(\{u, v\}) - t_{\{(u,v)\}}(\{u, v\}))$;
2: Return the $m/r$ pairs with the largest values.
```

Algorithm 5 shows a very simple approach to compute M optimally. The algorithm requires $O(n^2)$ executions of the top-k algorithm. Even though this is a preprocessing step, it may nevertheless be very time consuming. We discuss in Sect. 5.1.2 additional techniques by which this can be reduced.

The disagreement lists materialization procedure discussed earlier assumed that the user groups are restricted to two members only. However, once the m/r lists have been materialized, they can be used at query processing time for user groups of any size in a straightforward manner. Consider any arbitrary user group $\mathcal{G}$. In executing the top-k recommendation algorithm for this group, we use the predicted rating lists $\mathcal{IL}_u$ (for all $u \in \mathcal{G}$) as well as all disagreement lists $\mathcal{DL}_{(u,v)}$ (for all $u, v \in \mathcal{G}$) that have been materialized in M.

5.1.2 Avoiding examining all user pairs

In a large user base, it is very likely that many user pairs are almost never going to occur in query groups. In order to reduce preprocessing costs, it is critical that we identify only those user pairs that have significant likelihood of occurring together, and only consider such pairs in the above-mentioned algorithm.

If we have a rich *query log* (or workload) of past user groups, then it is possible to analyze the query log in determining this information. For example, let $\mathcal{G}_1, \ldots, \mathcal{G}_q$ be a query log of q user groups. Then, for any user pair (u, v), we can compute

$$p(\{u, v\}) = \frac{|\{\mathcal{G}_i | u, v \in \mathcal{G}_i\}|}{q}$$

This computation can be carefully done to ensure that we only compute the probabilities for those user pairs that occur in the query log, thus avoiding having to examine a vast majority of the user pairs that never occur together. Moreover, even for user pairs that occur together in the query log, we can eliminate those that have extremely low probabilities.

5.2 Partial materialization after factoring

We next consider the more complex case when the disagreement lists have already been factored. Recall that given a factoring set $\mathcal{S}_{(u,v)}$, each original disagreement list $\mathcal{DL}_{(u,w)}$ is now factored into a possibly smaller list $\mathcal{DL}_{u-\mathcal{S}(u,v),w}$ such that the original list is the union of the factored list $\mathcal{DL}_{u-\mathcal{S}(u,v),w}$ and a common list $\mathcal{DL}_{\mathcal{S}_{(u,v)},w)}$ for some other user w.

Our partial materialization goal will be to identify the subset of pairs $M \subseteq S$ such that both the factored as well as common component of the original disagreement list for each such pair is materialized. Using notation similar to Sect. 5.1, let $t_M(\mathcal{G})$ be the execution time of the top-k algorithm on user group $\mathcal{G}$ when run using the predicted rating lists $\mathcal{IL}_u$ (for all $u \in \mathcal{G}$) *as well as* the materialized (factored as well as common) disagreement lists corresponding to all user pairs (u, v) that appear in both M and $\mathcal{G}$. Our objective is to minimize the expected cost of executing the top-k algorithm on any user group query, using the predicted rating lists as well the materialized factored and common disagreement lists. Let the expected cost be denoted as t_M. Given a space budget m, the partial materialization problem after factoring problem may be formally defined as follows.

Problem (*Partial Materialization After Factoring*). Determine the subset of pairs $M \subseteq S$ s.t. the space required by all factored and common lists corresponding to all pairs in M is at most m, and $t_M = \sum_{\mathcal{G}\subseteq\mathcal{U}} p(\mathcal{G})t_M(\mathcal{G})$ is minimized.

[4] Performance numbers are obtained for a fixed k, specifically set for each application. E.g., in a movie recommendation, 10 movies is typical.

As before, we will make the simplifying assumption that each future user group query $\mathcal{G}$ will only contain exactly two users, and $p(\mathcal{G})$ is reliably known for all pairs of users $\mathcal{G}$. We also reduce the set of user pairs from S to S', eliminating those pairs for which the availability of the disagreement list does not improve the query processing time.

Let $\mathcal{DL}_{\mathcal{S}(P_i)}$ be the factored list corresponding to any user pair $P_i \in S'$. Since common lists are shared, let $C(S')$ represent the set of all common lists corresponding to S'. Then, the space consumed by all factored as well as common lists is

$$Space(S') = \sum_{P_i \in S'} |\mathcal{DL}_{\mathcal{S}(P_i)}| + \sum_{\mathcal{DL}_C \in C(S')} |\mathcal{DL}_C|$$

It may be that this space is still greater than the space constraint m. In this case, we will have to remove a few more user pairs from S', eliminating those pairs for which the availability of the disagreement list adversely impacts query processing time the least.

For user pair $(u, v) = P_i$, let the *benefit* B_i be defined as

$$B_i = p(\{u, v\}) \cdot (t_\phi(\{u, v\}) - t_{\{(u,v)\}}(\{u, v\}))$$

The residual problem can be formally defined as follows.

Problem (*0/1 Knapsack-Based Formulation of Partial Materialization After Factoring*). Determine the subset of pairs $M \subseteq S'$ s.t.

$$\sum_{P_i \in M} B_i$$

is maximized, subject to

$$Space(M) = \sum_{P_i \in M} |\mathcal{DL}_{\mathcal{S}(P_i)}| + \sum_{\mathcal{DL}_C \in C(M)} |\mathcal{DL}_C| \leq m$$

We note that this problem is similar, but not identical, to the classical NP-Hard 0/1 Knapsack Problem [16]. This is because the space constraint contains a term that represents the space consumed by the common lists of M. If this term were not there, then the formulation can be easily seen to be identical to 0/1 Knapsack.

In solving this problem, we leverage the well-known greedy 1/2-approx algorithm for 0/1 Knapsack, suitably modified to account for the extra complexity of having to consider the materialization of common lists.

Algorithm 6 essentially orders the pairs in S' by decreasing "benefit density", except that in the calculation of this density, the common lists are not considered. The common lists are only considered in the space calculation of M. The returned user pairs are either (a) the largest prefix of this ordered list that can fit within the space budget or (b) the very last user pair that causes the space to exceed the budget.

While Algorithm 6 is not an optimal algorithm for the problem, it is adapted along the lines of the 1/2-approx algorithm for the classical 0/1 Knapsack problem, and our experiments indicate it is both efficient and provides solutions of good quality. More interestingly, when run on un-factored disagreement lists, it is *identical* to Algorithm 5 which is optimal for that case. As shown in our experiments, for the same space constraint, factoring followed by partial materialization is always better than partial materialization alone.

Algorithm 6 Partial Materialization After Factoring

```
Require: User pairs in $S'$;
Sort the pairs $P_i \in S'$ by decreasing $B_i/|\mathcal{DL}_{\mathcal{S}(P_i)}|$
$M = \{\}$
$i = 1$
while $Space(M) + |\mathcal{DL}_{\mathcal{S}(P_i)}| \leq m$ do
    $M = M \cup P_i; i++;$
end while
if $\sum_{P_j \in M} B_j \geq B_i$ then
    return $M$
else
    return $P_i$
end if
return
```

Table 1 Statistics about the MovieLens data set

# users	# movies	# ratings
71,567	10,681	10,000,054

6 Experiments

We evaluate our group recommendation system from three major angles. First, from the *quality* perspective, we conduct an extensive user study through Amazon Mechanical Turk[5] to demonstrate that group recommendations with the consideration of disagreements are superior to those relying on aggregating individual predicted rating scores alone (Sect. 6.1). Second, from the *performance* perspective, we conduct a comprehensive set of experiments to show that our materialization algorithms can achieve better pruning than alternative algorithms (Sect. 6.2). Third, we investigate the performance of our space-saving strategies with respect to both space and time.

We implemented our prototype system using JDK 5.0. All performance experiments were conducted on an Intel machine with dual-core 3.2 GHz CPUs, 4 GB Memory, and 500 GB HDD, running Windows XP. The Java Virtual Memory size is set to 256 MB. All numbers are obtained as the average of three runs.

Data Set: We use the MovieLens [10] 10 M ratings data set for evaluation purposes. The statistics of this data set is shown in Table 1.

Individual Predicted Ratings: We adopt collaborative filtering [1] for generating individual predicted ratings as described in Sect. 2.1.2, where the user-user similarity,

[5] http://www.mturk.com/.

$\texttt{UserSim}(u, u')$, is computed as follows: $\texttt{sim}(u, u') = \frac{|\{i|i\in\mathcal{I}_u \wedge i\in\mathcal{I}_{u'} \wedge |\texttt{rating}(u,i)-\texttt{rating}(u',i)|\leq 2\}|}{|\{i|i\in\mathcal{I}_u \vee i\in\mathcal{I}_{u'}\}|}$ where $\mathcal{I}_u$ denotes the set of items u has rated. We consider a movie to be shared between two users if they both rated it within 2 of each other on the scale of 0–5.

6.1 User study

We conduct an extensive user study through Amazon Mechanical Turk to compare our proposed group recommendation consensus functions with prior group recommendation mechanisms, which rely solely on rating aggregations. In particular, we compare four group recommendation mechanisms:

Average Rating (AR), which computes the group recommendation score as the average of individual predicted ratings. The disagreement weight is set to zero.

Least-Misery Only (MO), which computes the group recommendation score as the minimum individual predicted rating among all group members. Again, the disagreement weight is set to zero.

Consensus with Pairwise Disagreement (RP), which computes the group recommendation score as a weighted summation of the average predicted rating and the average pairwise disagreements between all group members.

Consensus with Disagreement Variance (RV), which computes the group recommendation score as a weighted summation of the average predicted rating and the variance of individual predicted ratings among all group members.

The user study is conducted in two phases: *User Collection Phase* and *Group Judgment Phase*. At each phase, a series of HITs (Human Intelligence Tasks) are generated and posted on Mechanical Turk, Amazon users are invited to complete those tasks.

6.1.1 User collection phase

The goal of the User Collection Phase is to recruit users and obtain their movie preferences. Those users will later form groups and perform judgments on group recommendations.

Preferences Collection: Asking a user to go through all ten thousand movies in our system and give ratings as they go is clearly not practical. Therefore, we selected a subset of the movies for users to provide their preferences. We considered two factors in selecting those movies: *familiarity* and *diversity*. On one hand, we want to present users with a set of movies that they do know about and therefore can provide ratings. On the other hand, we want to maximize our chances of capturing the different tastes among movie-goers. Toward these two goals, we select two sets of movies. The first set is called the *popular set*, which contains the top-40 movies in MovieLens in terms of popularity (i.e., the number of users who rated a movie in the set). The second set is called the *diversity set*, which contains the 20 movies in MovieLens that have the highest variance among their user ratings and that are ranked in the top-200 in terms of popularity. We created two HITs with 40 movies each. The **Similar HIT** consisted entirely of the movies within the *popular set*, and the **Dissimilar HIT** consisted of the top-20 movies from the *popular set* and the 20 movies from the *diversity set*. Fifty users were recruited to participate in each HIT. Users are instructed to provide a rating between 0 and 5 (5 being the best) for at least 30 of the 40 movies listed (in random order) according to their preferences. In addition to their ratings, we also record their Mechanical Turk IDs for future reference.

Table 2 Similarities of user study groups

	Size = 3	Size = 8
Similar group	0.89	0.90
Dissimilar group	0.29	0.27
Random group	0.69	0.73

Group Formation: We consider two main factors in forming user groups: *group size* and *group cohesiveness*. We hypothesize that varying group sizes will impact the difficulties in reaching consensus among the members and therefore affect to which degree members are satisfied with the group recommendation. We chose two group sizes, 3 and 8, representing small and large groups, respectively. Similarly, we hypothesize that group cohesiveness (i.e., how similar are group members in their movie tastes) is also a significant factor in the satisfaction with group recommendation. As a result, we chose to form three kinds of groups: *similar, dissimilar, random.* A similar group is formed by selecting users who: (1) have completed the **Similar HIT** described earlier; (2) combined with having the maximum summation of pairwise similarities (between group members) among all groups of the same size. A dissimilar group is formed by selecting users who: (1) have completed the **Dissimilar HIT** described earlier; (2) combined with having the minimum summation of pairwise similarities (between group members—based on the provided ratings) among all groups of the same size. Finally, a random group is formed by randomly selecting users from all the pool of available users. Table 2 illustrates the average similarity between group members of the six groups formed.

6.1.2 Group judgment phase

The goal of the Group Judgment Phase is to obtain ground truth judgments on movies by users in a group setting. Those judgments can then be used to compare group recommendation generated by the four different mechanisms **AR**, **MO**, **RP** and **RV**.

Individual Recommendation: For each user in one of the six groups in Table 2, we generated and materialized a list of individual recommendations against the MovieLens database using collaborative filtering.

Group Recommendation Candidates: For each group, we generated group recommendations using all of our four strategies. The resulting recommendation lists were combined into a single set of distinct movies, called *group candidate set*. This ensures that we obtain ground truth judgments on all the movies we will encounter using any of the four strategies.

For each group, a **Group HIT** was generated and contained the following group context: for each movie in the group candidate set, the individual recommendation score of each member. The users are then instructed to decide *whether a movie in the group candidate set is suitable for recommendation given its group context*. Users from the previous phase were invited back (with a higher payout) to participate in the HITs which correspond to a group to which they belong. Additional users were also recruited to participate in the HITs to complement the set of prior users, and they were instructed to pretend themselves to be one of the group members in the HIT. At the conclusion of the user study, on average 5 users participated in the three 3-member-group HITs and 10 users participated in the three 8-member-group HITs, for a total of 45 users.

6.1.3 Result interpretation

Given a Mechanical Turk user's ground truth evaluation of the candidate movies, we adopt the Discounted Cumulative Gain (DCG) [12] measure to evaluate each of the following six group recommendation strategies (note that the least-misery model by definition considers only one member of the group and therefore cannot be combined with either of the disagreement models):

AR, MO: these two are group recommendation lists generated based on average and least-misery models, respectively, without the disagreement component.

RP20, RP80: these two are group recommendation lists generated by combining the average predicted ratings model with the pairwise disagreement model. RP20 sets w_2 in Definition 3 to 0.2, while RP80 sets it to 0.8.

RV20, RV80: these two are group recommendation lists generated by combining the average predicted ratings model and the disagreement variance model. RV20 sets w_2 in Definition 3 to 0.2, while RV80 sets it to 0.8.

Each strategy generates a 10-movie recommendation list and for a given list, its DCG value is calculated as follows:

$$\mathrm{DCG}_{10} = rating_1 + \sum_{i=2}^{10} \frac{rating_i}{\log_2(i)}$$

where $rating_i$ is the ground truth (provided by the Mechanical Turk user) of the movie at position i and is either 1 (the user considers this movie suitable for the group setting) or 0 (otherwise). We further normalize the DCG value into a range between 0 and 1 by dividing it by the DCG value of the ideal list to produce the nDCG value. (The ideal list is obtained by resorting the movies in the list in the order of their predicted ratings.)

For each group with a given size and cohesiveness, the nDCG values of each recommendation list are computed as the average of all the users who participated in the group HIT. The results are shown in Fig. 3. We note that our observation is anecdotal and our finding in this experiment is not necessarily statistically significant.

The top-left chart in Fig. 3 reports the nDCG for small and large groups of similar users. In a real-world setting, a group of friends can be thought of as such a group. According to this chart, **MO** results in the best performance for both small and large groups. This can be explained as a group activity of similar users, where the objective is to agree with the person who has the harshest opinion. **MO** is most practical for this setting since agreeing upon the worst opinion results in the least disagreement from a user's personal opinion. It is also interesting to notice, that for large groups, **MO** performs very well. The next best strategy is **AR**, which is intuitively true for any set of similar users—people with very high similarity have no difference in their opinion. **RV80** and **RP80** perform worst since there is hardly any scope of difference in opinion in a group of similar users.

The top-right chart in Fig. 3 reports the nDCG for small and large groups of dissimilar users. In a practical setting, a group of family members, whose tastes typically differ is a good example here. For dissimilar users, differences in opinion is conspicuous hence needs to be captured carefully. Indeed, we can see that, our disagreement-based models **RV80, RP80** start performing better than other two models. Specifically, for large groups, **RV80** results in the best value of nDCG, while the predicted rating-based models are useless. This observation corroborates our initial claim that formalizing disagreement as a component of the consensus function is important for group recommendation.

The bottom-left chart in Fig. 3 reports the nDCG for small and large groups of random users. A random group can consist of both similar and dissimilar users. For small groups, **MO** works best, whereas for large groups, there is no significant difference between all four strategies.

The bottom-right chart in Fig. 3 reports the differences in our disagreement models (notice the different weights) for dissimilar user groups. It is interesting to notice that, for small

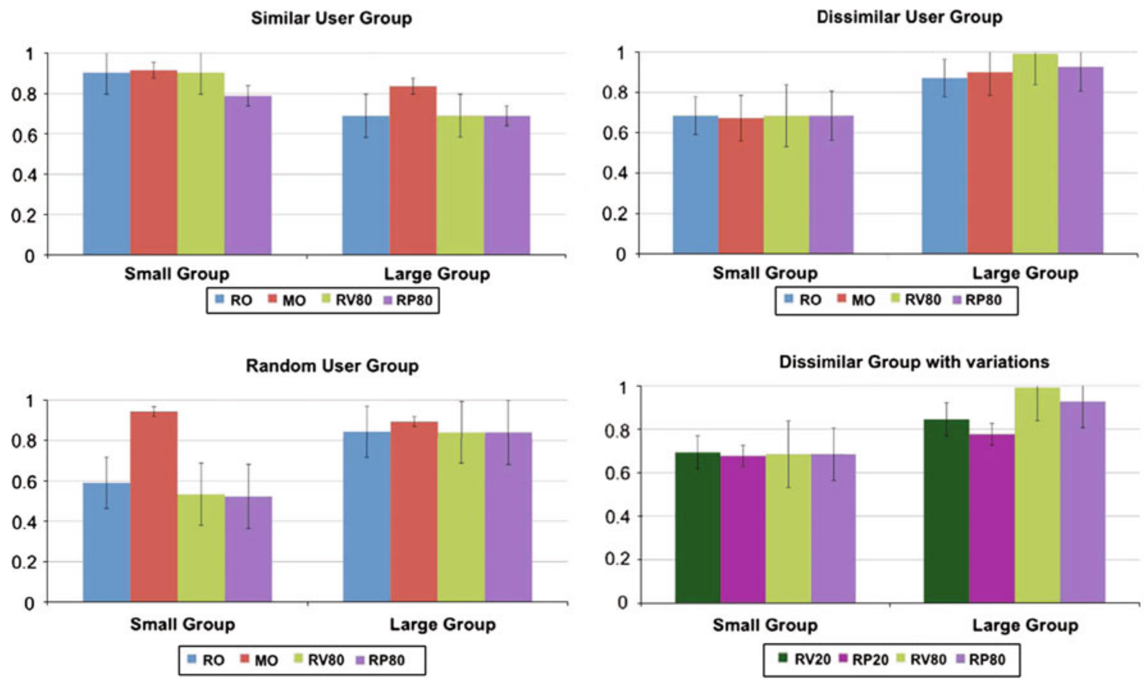


Fig. 3 Comparison of user predicted ratings (using NDCG) among different group recommendation lists

groups, all four disagreement models perform equally well in general. However, for large groups, disagreement becomes a conspicuous part in decision-making. Consequently, the disagreement strategies **RV80, RP80** outweigh the other two models **RV20, RP20**.

To summarize, we can say that user similarity in a group as well as group size should be accounted in modeling disagreement in the consensus function. One of our planned experiment is to involve users more actively in the final judgment by letting group members consult with each other and reach consensus in an iterative manner as described in [11]. Such feedback would help draw a stronger connection between group size and overall group dynamics in group recommendation.

6.1.4 Effectiveness of group ratings

We next perform user studies to validate the effectiveness of the group ratings. More precisely, we ask users to compare group ratings generated by our group recommendation strategies with the individual ratings obtained directly from the underlying recommendation system. Again, **Group HITs** are generated based on similar and dissimilar groups. Additional users were recruited to participate in the HITs to complement the set of prior users, and they were instructed to pretend themselves to be one of the group members in the HIT. Within each HIT, a 10-movie recommendation list is presented to each user within the context of a group. Each movie comes with the individual predicted rating and the group rating generated by one of our group recommendations strategies (**RP80** and **RV80**). The users were then instructed to give their *preference for either the group or the individual rating for each movie*, although the explicit model name was kept hidden from them. Additionally, they were also required to *describe their satisfaction level for the group ratings overall*, in the scale of 1–5. In this new user study, on average 15 users participated in the each of the two (similar and dissimilar) 3-member-group HITs and 8 users participated in each of the two (similar and dissimilar) 8-member-group HITs, for a total of 218 users.

Result Interpretation: For each group with a given size and cohesiveness, we calculate the percentage of user's preference for group ratings corresponding to a strategy and compare that with the percentage of user's preference for individual ratings. The results are listed in Fig. 4. In all cases, group ratings are preferred by more than 50% of users. It is also easy to observe that the group ratings are more preferred over individual ratings for similar user groups when compared to dissimilar user groups. The reason is intuitive and can be explained as follows: similar users have similar ratings for movies; hence, with a small compensation, they can match their individual preference with the group preference. However, for dissimilar user groups, preference varies widely among group members—hence dissimilar users are more reluctant to adopt group ratings. Another interesting observation is, irrespective of group cohesiveness, members in large groups prefer group ratings more than members in small groups do. This corroborates the efficacy of our group recommendation strategies which are designed to

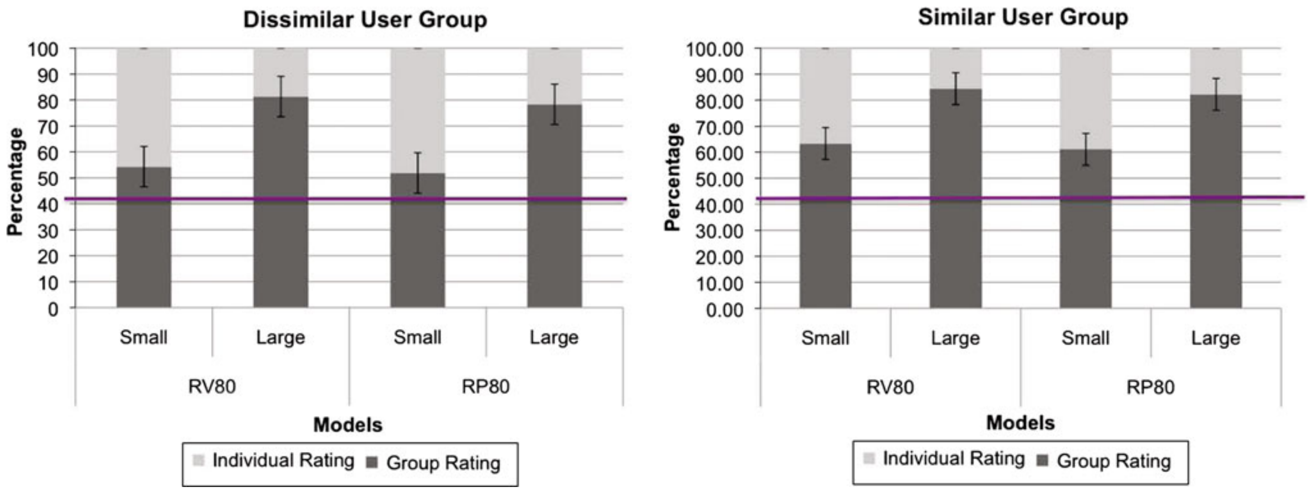


Fig. 4 Comparison of percentage of user preference for group ratings and individual predicted ratings among different group recommendation lists

Table 3 Dissimilar user group—overall model ratings

Rating	RP80		RV80	
	Small (%)	Large (%)	Small (%)	Large (%)
1	0	0	0	0
2	5	0	8	3
3	31	20	28	17
4	42	44	60	36
5	22	36	4	44

Table 4 Similar user group—overall model ratings

Rating	RP80		RV80	
	Small (%)	Large (%)	Small (%)	Large (%)
1	3	0	5	0
2	14	0	8	0
3	14	14	20	11
4	52	30	40	41
5	17	56	27	48

minimize the difference in opinions between group members individual preference and are more conspicuous for larger groups.

Tables 3 and 4 record the percentage of overall group ratings (in the scale of 1–5) of different group recommendation strategies for different group cohesiveness and group size. It can be easily observed from the tables that proposed group recommendation strategies are highly rated (mostly 3 and above) always, irrespective of the size and cohesiveness of the group under consideration.

6.2 Performance evaluation

In this section, we analyze the performance of the three group recommendation algorithms described in Sect. 3: Dynamic Computation with Predicted Rating List Only (`RO`), Full Materialization (`FM`) and Partial Materialization with a given budget on number of lists (`PM`). At the core of all three algorithms is the top-k TA algorithm [8], which scans down the input lists and stops processing when score bounds indicate that no more items qualify. The cost of `TA` is determined by two factors: the number of *sequential accesses*, which corresponds to the number of `next()` calls made during the scan of each list, and the number of *random accesses*, which corresponds to the number of calls made to each list for score retrieval given an item. During the processing, when the buffer is bounded and only the top-k items are kept, the number of random accesses is proportional to the number of sequential accesses. When the buffer is unbounded, the number of random accesses is proportional to the *number of distinct items processed*. We adopt the bounded buffer version of the TA algorithm and therefore mostly measure the number of sequential accesses to compare the performance between various algorithms.

In addition to that we also compare our proposed group recommendation algorithms with a very simple baseline approach—`Without-Fagin RO`. This algorithm works as follows: It works only with the set of lists relevant to a specific group. This algorithm does not work in Fagin(top-k) style; i.e., it cannot acquire any early stopping using upper bound value of thresholds. In order to compute the top-k group ratings, it maintains a heap and stores the top-k ratings encountered thus far. However, the algorithm can only terminate once the entire database is scanned and outputs the top-k best ratings thereafter.

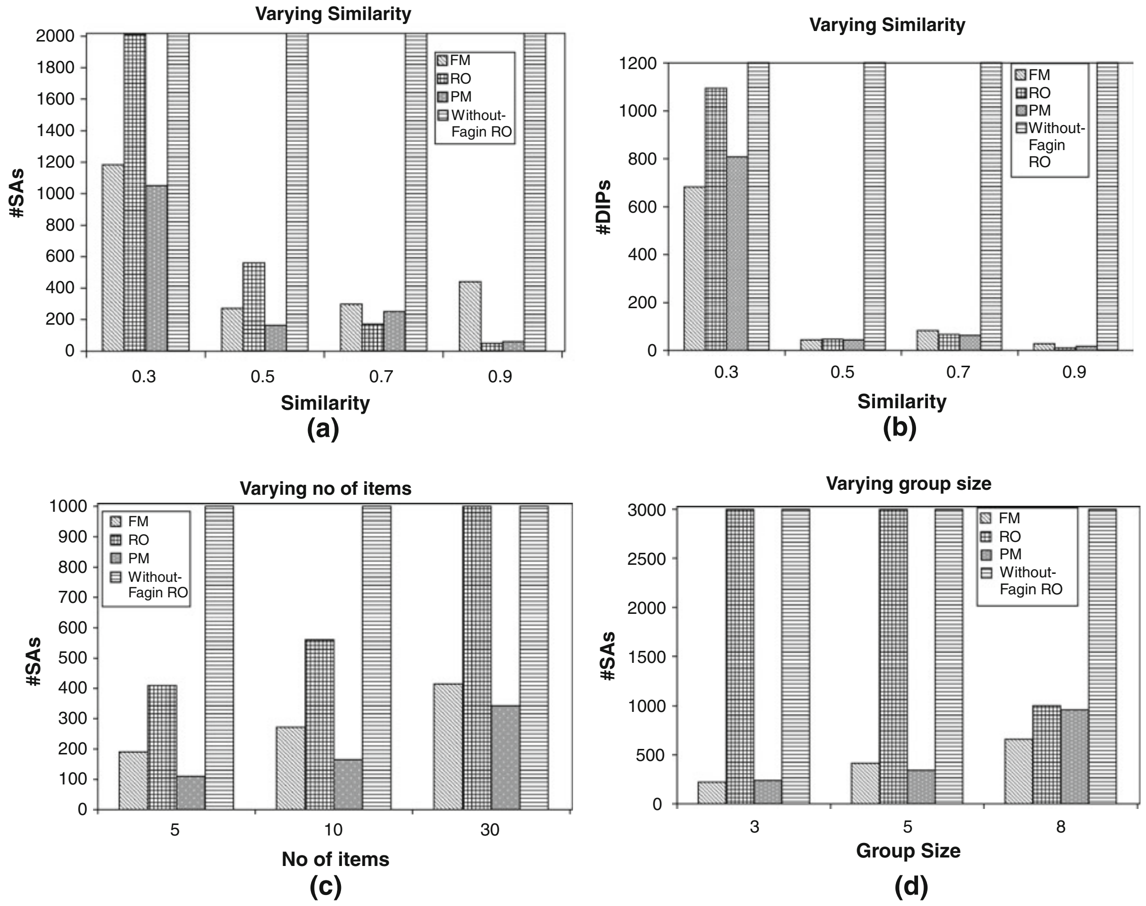


Fig. 5 Performance comparison among algorithms RO, FM, PM and without-Fagin RO

Group Formation: Groups are formed by selecting users from the MovieLens database. The key factor we consider is group cohesiveness (or similarity). We defined four group similarity levels: 0.3, 0.5, 0.7, 0.9, with a margin of ±0.05. To form a group of 3 with similarity 0.3, we select three users u_1, u_2, u_3 from the database, such that $\forall i, j, 0.25 < \texttt{sim}(u_i, u_j) < 0.35$, where $1 \leq i, j \leq 3, i \neq j$. The other factors we consider are number of recommendations being produced (small = 5, medium = 10, large = 30) and the size of groups (small = 3, medium = 5, large = 8).

Summary of Results: Our first observation is that group similarity has a direct impact on the number of sequential accesses (SAs). This is not surprising: the predicted rating lists of similar users tend to contain similar items at similar positions, including those with high predicted ratings. Our second observation is that some Disagreement Lists ($\mathcal{DL}$s) almost always guarantee earlier stopping. Hence, RO wins in very few cases. However, the presence of $\mathcal{DL}$s is not always beneficial and can sometimes become *redundant*. In fact, the results show that for different user groups, different strategies (RO, FM or PM) will win. In particular, a higher number of $\mathcal{DL}$s does not guarantee earlier stopping. The proliferation of lists may increase the number of SAs and also the number of distinct items seen unnecessarily, thereby hurting the performance in the end. In addition to that we compare our three algorithms, with the baseline approach Without-Fagin RO. Eventually, as shown in Fig. 5, this algorithm always scans the entire database and encounters all items in the database before producing the output. Consequently, it attains the worst performance among all. We provide detailed descriptions on our experiments below.

6.2.1 Varying group similarity

Figure 5a and b illustrate the performance of RO, FM and PM with different group similarities in terms of both SAs and DIP. The group size is fixed at 5, and the number of recommended items is 10. For PM, the number of materialized lists is 3. As the group similarity increases, the effectiveness of our materialization algorithms gradually decrease. This is not surprising since the more similar the members are with each other, the more likely their agreements on the top items are close to the upper bounds that are estimated in the RO algorithms. As a result, RO can reach stopping conditions as early as PM and FM do. This observation is also corroborated by the similar numbers of DIP between RO and the other two algorithms for high similarity values. Furthermore, FM forces the system to scan unnecessarily large number of lists and results in poor performances instead. In fact, it can be easily observed from Fig. 5a and b, for very high similarity, RO results in the best performances, whereas for very low similarity, FM is the winner in most of the cases. The performance of PM can be observed to be in between. An interesting observation in this case is, for average similarity, PM results in the best performances for both SAs and DIP. This corroborates the fact that in certain cases partial materialization can be the best option.

6.2.2 *Varying k*

Figure 5c illustrates the performance comparison of RO, FM and PM with different numbers of items recommended. The group size is fixed at 5, and the group similarity is fixed at 0.5. Algorithm PM uses three materialized lists for $k = 5$, 10 and five lists for $k = 30$. As expected, the number of SAs increases with the increasing number of recommended items. For all three cases, algorithm PM out-performs both RO and FM significantly.

6.2.3 *Varying group size*

We examine the effect of different group sizes in Fig. 5d. The group similarity is fixed at 0.5, and the number of recommended items is 10. For PM, the number of materialized lists is 3. As expected, the number of SAs increases as the group size increases. When the group sizes are small and medium, both materialization algorithms significantly outperform RO. It is counter-intuitive to see that when the group size is large, the benefit of materialization decreases. After some investigation, we discovered that when the group is large, it is easy to have a predicted rating list that can provide enough pruning power to trigger the early stopping conditions. As a result, pruning through the disagreement lists is no longer as effective.

6.2.4 *Effect of disagreement lists in query processing*

We study the impact of materializing different numbers of disagreement lists ($\mathcal{DL}$s). The group size is fixed at 5 and its similarity is fixed at 0.5, and the number of recommended items is 5. We report SAs and DIP by varying the number of materialized disagreement lists. As shown in Fig. 6, the performance is at its worst when the number of $\mathcal{DL}$s is 0, which corresponds to RO. It starts getting better as more $\mathcal{DL}$s are added, and the performance is best when the number of $\mathcal{DL}$s reaches 3. Then, it starts degrading and never gets better. However, from the 4th to the 10th list, the number of DIP remains almost the same. By examining the 4th list, we noticed that many top items in that list are not present in the final result, and, as a result, the number of SAs increases unnecessarily. We also noticed that the top items in the 4th list are shared by all subsequent lists (which explains the close-to-constant performance). This situation can arise when a subset of the group dislikes the same set of movies equally.

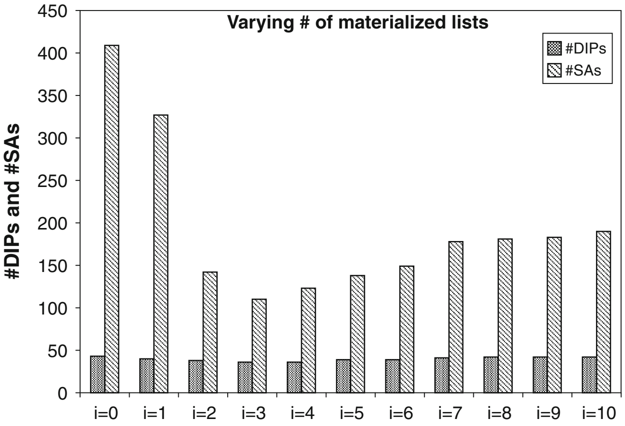


Fig. 6 Effect of the number of $\mathcal{DL}$s

6.3 Space reduction techniques and their impact on query processing

The main focus of this subsection is to analyze and compare the query processing performance of PM algorithm under space constraints. Recall that the PM algorithm is designed when a space budget is enforced and a subset of possible set of pairwise disagreement lists can be materialized. We proposed to combine factoring and disagreement lists materialization to satisfy such hard space constraints. Here, we experimentally evaluate query processing performance attained by PM using configurations offered by these different space reduction techniques.

Summary of Results: Our first observation is PM is never worse than RO. For some groups, the best performance can be attained by using PM algorithm. In general, FM gets better with bigger group sizes. However, the difference in performance between FM and PM is not noteworthy as the group size is increased. Hence, under a space constraint, PM is an acceptable solution. Next, we observe that our proposed behavior factoring algorithm performs well in reducing space. Finally, we experimentally demonstrate that factoring is always beneficial from performance perspective since it aids to preserve more disagreement lists in a lossless way. Consequentially, factoring followed by Knapsack based PM is better than

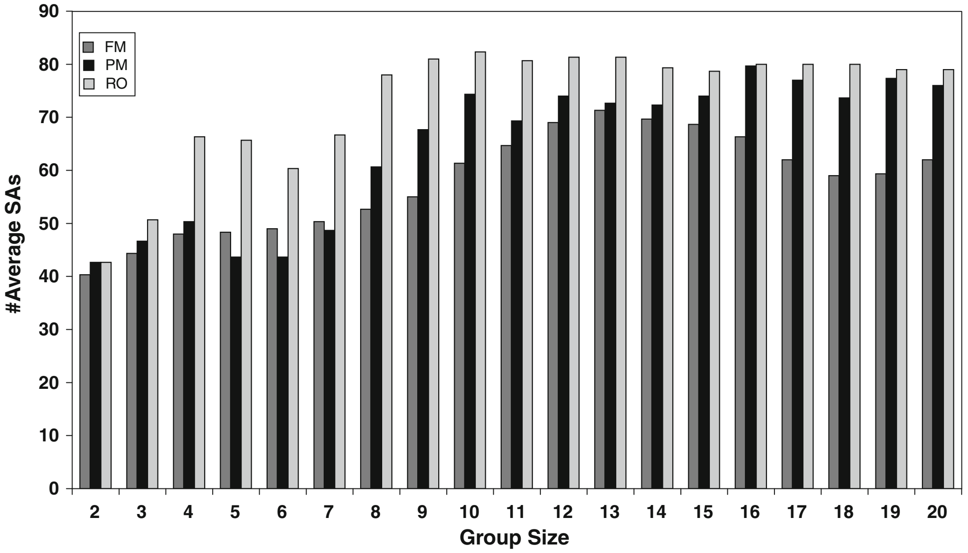


Fig. 7 Query processing performance of different group recommendation algorithms

PM-Only even when a small fraction of space is offered to materialize disagreement lists.

In these experiments, the space required by a configuration is interpreted as the total number of entries in the disagreement lists (as defined in Sect. 4.) That is because predicted rating lists being necessary, they are not affected by our space reduction strategies, factoring and partial materialization.

6.3.1 Effect of partial materialization (PM-Only) on query processing

First, we perform a comparative performance study of query processing of different group recommendation algorithms (RO, FM and PM). In these experiments, we set the available space to materialize disagreement lists to 50% of the total space consumed by all possible pairwise disagreement lists in the user base. We vary the query size from 2 to 20 (recall that a query is a group that is seeking recommendations) and measure the number of sequential accesses (SAs) required to compute top-k ($k = 30$) recommended items to the group. Each performance number of a particular query size is obtained by averaging the number of sequential accesses required to compute top-k recommendations of three different groups of that particular size. For a particular query, its size is increased by adding one random new user from the user base.

Figure 7 illustrates the performance comparison of different group recommendation algorithms. As expected, the average number of SAs increases with the increasing group size in general. In general, FM gets better as group size is increased. RO performs the worst among all three in all cases. For groups 5, 6 and 7, PM is the best solution. By examining group 5 in one individual run, we noticed that PM uses only 4 disagreement lists at that step, whereas FM uses all 10 disagreement lists. These extra disagreement lists incur unnecessary sorted accesses in FM. Also, PM gets better from group 4 to group 5. Our analysis reveals that PM uses only 1 disagreement list in group 4, whereas in group 5, it uses 3 new disagreement lists. We further investigate that behavior and notice that the new disagreement lists play crucial role in reaching the threshold fast during top-k computation. Consequently, the overall number of accesses drops from group 4 to group 5. This experiment also reinforces the intuition that different disagreement lists have varying impacts on performance.

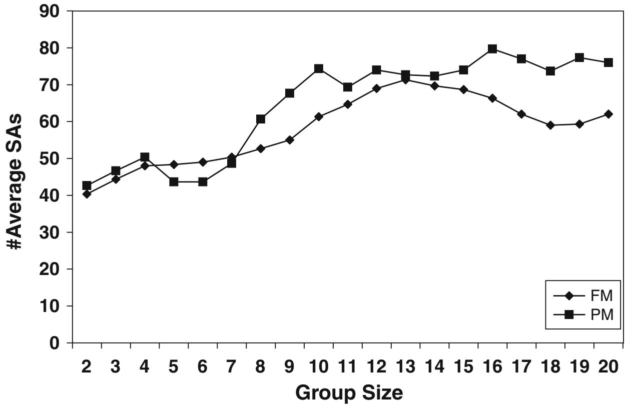


Fig. 8 Difference in performance between PM and FM

Next, we study the difference in performance between PM and FM (the better one between FM and RO) in the same settings in Fig. 8. Although FM outperforms PM with the increase in group size; however, the difference is not significant. These two experiments corroborate our initial claims: even when full materialization is acceptable, partial materialization is

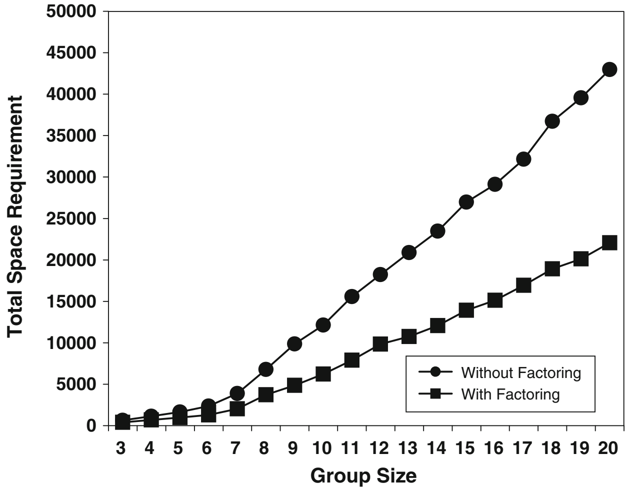


Fig. 9 Space savings of factoring algorithm for similar userbase

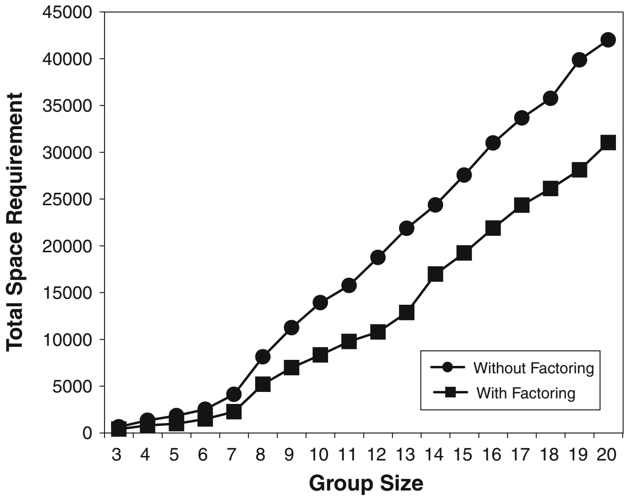


Fig. 10 Space savings of factoring algorithm for random userbase

important since that can attain the best performance sometime. Also, under a space constraint, PM is a satisfactory solution since its performance is reasonably close to the best solution.

6.3.2 Benefit of factoring algorithm in space saving

Next, we evaluate the space-saving benefit of behavior factoring. We increase the size of the user base (from 3 to 20) and measure the space requirement (i.e., no of entries) to store all pairwise disagreement lists for that user base *with and* without factoring. Recall that the benefit of factoring can only be achieved for groups with size 3 and beyond. In particular, we consider two different cases: in one case, a new user is added into the existing user base at random, whereas in other case, a new user is only selected for addition into the existing user base when it is highly similar (50% or more) to at least one existing user (henceforth referred to as Random Userbase and Similar Userbase, respectively, in this section.)

Figure 9 demonstrates the benefit of space saving for Similar Userbase. The user group of size 20 has 190 disagreement lists that contain 42978 entries (space) originally. Upon factoring, the total size of these lists is reduced to 22063 entries, thus achieving a space saving of 48.66% in a lossless manner.

Figure 10 demonstrates the benefit of space saving for Random Userbase. The user group of size 20 has 190 disagreement lists that consumes 42012 entries (space) originally. Upon factoring, the total size of these lists is reduced to 31023 entries, thus achieving a space saving of 26.15% in a lossless manner. It is easy to observe that space reduction is very significant for Similar Userbase, however, even for Random Userbase the reduction achieves good performance. This demonstrates that the factoring algorithm is effective and performs well in practice, thereby, reinforcing the idea that unless no two users agree on any item, factoring is always beneficial.

6.3.3 Impact of different space reduction strategies on query processing

Finally, we investigate different space reduction strategies and their comparative effectiveness in query processing. Recall that given a space budget (i.e., number of entries) for materializing disagreement lists, behavior factoring may fail to reduce the original pairwise disagreement lists of the user base to that extent. Effectiveness of space saving solely depends on similarity between users in the user base under consideration. Therefore, it may be necessary to apply techniques to drop some disagreement lists on the factored user base (refer to Algorithm in Sect. 5) to satisfy the hard space constraint. On the other hand, the hard space constraint can also be guaranteed by applying partial materialization only (refer to Sect. 5) on the original (not factored) pairwise disagreement lists.

In these experiments, we intend to evaluate the impact of space reduction techniques from two major angles: first, given different space budgets, we evaluate the impact on query processing of factoring followed by disagreement lists materialization (henceforth referred to as `Factoring followed by Knapsack-based PM` in this section) and compare that with the performance attained by applying partial materialization only (henceforth referred to as `PM-Only` in this section).

Figure 11 shows the comparative study of the query processing performance of `Factoring followed by Knapsack-based PM` and `PM-Only` on Random Userbase. The group size is fixed at 10. Performance numbers are obtained by averaging the number of sequential accesses required to compute top-k ($k = 30$) recommendations

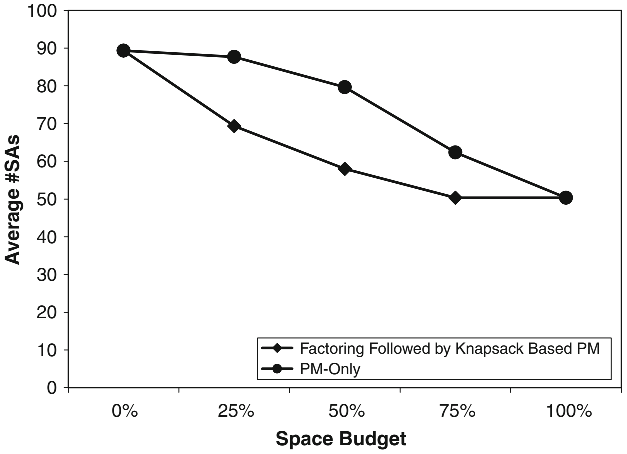


Fig. 11 Performance of different space-saving strategies under different space budgets

of three different groups of size 10 chosen randomly from the Random Userbase. Space constraints are varied by 5 different numbers, in an equi-spaced manner, ranging from (0–100%). Recall that this hard space constraint allows only certain amount of space (# of entries) for materializing disagreement lists. Note that 0% space means no disagreement list can be materialized (Algorithm `RO`) and 100% space budget allows all disagreement lists to be materialized (Algorithm `FM`.)

Figure 11 demonstrates one such case, where a higher space budget results in better performance. Therefore, performance is the worst for 0% space and the best for 100% space. It also corroborates the fact that factoring is always beneficial, since it conserves more information in a lossless way under the same space constraint. Consequently, `Factoring followed by Knapsack-based PM` performs better than `PM-Only` in all three intermediate space constraints, 25, 50 and 75%. The most interesting observation is `Factoring followed by Knapsack-based PM` attains the same performance in 75 and 100% space constraints. Recall that we use Random Userbase in this experiment which achieves a 26.15% overall space saving, i.e., factoring stores all disagreement lists in 26.15% less space, while guaranteeing the same processing performance as `FM`.

Finally, we investigate the comparative performance of the two space reduction strategies discussed above at a fixed space constraint (50%). We profile the performance of query processing by varying query size (i.e., group size from 3 to 20) there. Each performance number is presented after averaging the individual performance numbers as discussed earlier.

Figure 12 summarizes the result of this experiment. As expected, the average number of `SAs` increases with the increasing group size in general. However, `Factoring followed by Knapsack-based PM` outperforms `PM-Only` significantly in all group sizes. This result corroborates the effectiveness of the proposed `Factoring followed by Knapsack-based PM` algorithm.

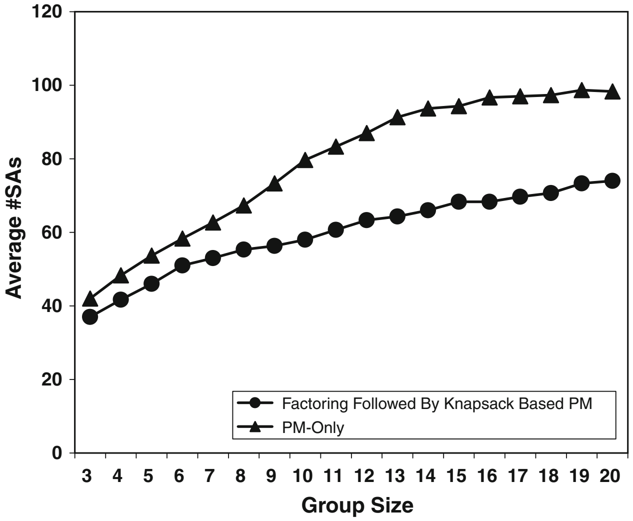


Fig. 12 Performance of different space-saving strategies with a space budget of 50%

7 Related work

We organized our related work section into two subsections: recommendations and query processing.

7.1 Recommendations

Two good surveys of recommendations can be found in [1] and [14]. Briefly, the goal of a recommendation strategy is to estimate a user's rating for items he has not rated before, and return *k* items with highest estimated ratings. The two most popular families of recommendation strategies are item-based and user-based strategies. The former leverages items similar to the user's previously highly rated items and the latter leverages users who share the user's interests. In this paper, we use collaborative filtering to generate individual recommendations.

A survey on group recommendations is given in [11]. It describes the two prevalent approaches: *virtual user* and *recommendation aggregation*. The former combines existing ratings of each group member to create a virtual user to whom conventional recommendation strategies are applied, whereas the latter creates individual recommendation lists for each member and consolidates those lists to form the group's list. In this paper, we adopt the latter approach for its flexibility as described in [11].

Existing research on group recommendations mainly focuses upon group formation and evolution, privacy concerns and interfaces for supporting group recommendations.

To the best of our knowledge, we have not encountered any related work that emphasizes on performance aspect of group recommendation computation, nor do they provide a theoretical and empirical study of different consensus functions, as we have done in this work. A few conducted user studies to evaluate the benefits of group recommendations. Those are summarized later.

PolyLens [18] is a group recommender extension to the MovieLens recommender system. The authors report a user study where existing MovieLens users were allowed to form groups of their preference(e.g., by inviting each other) and the system studies the impact of group behavior on the recommender system MovieLens. In order to produce group recommendations, individual groups members' recommendations were merged using the least-misery model. User satisfaction was measured using following different criteria: how easy the process of creating groups was; how easy it was to add members into a group; how useful group recommendations were; and the overall satisfaction. The study concluded, among other findings, that users in a group prefer group recommendations than individual ones. This inspired our group vs individual recommendation comparison in Sect. 6.1.4.

In [6], the authors develop a genetic algorithm-based collaborative filtering strategy to infer interactions between group members to compute the predicted rating of an item for a group. Even here, their experimental evaluation validates the quality of group recommendations and users satisfaction.

In [13], the authors distinguish between group recommendations in online communities and in non-online ones. They propose a two-phase approach, where first a set of recommendations are generated for a group using collaborative filtering, and then items are filtered from that set in order to improve satisfaction of individual members preferences. Their experiments show that the proposed method has consistently higher precision and individual members are more satisfied.

AHP (Analytic Hierarchy Process) of multi-criteria decision-making is used in [19] to model group preferences using the preferences of individuals. The authors also use a Bayesian network to model uncertainty in an individual user's preference. Their evaluation on 10 different situations assesses the high usability of their system and a comparison with both random and rule-based recommendation is also provided.

The authors in [17] develop 3 different aggregation policies of individual user models into a group model and for the purpose of biasing recommendations in a critiquing-based, case-based recommender. They conduct experiments to highlight the benefits of group recommendation using live-user preference data.

Finally, in [4], the authors use hierarchical clustering and decision trees to generate recommendations of user groups in Facebook. This work differs from ours because it focuses on recommending friends groups instead of recommending items to groups. The experiments show that a large number of groups in Facebook (73%) are accurately predicted using members's profiles.

Factoring Lists: In [2], the authors developed space-saving strategies on keyword inverted lists using shared user behavior. Their approach is based on clustering users first and then building per-cluster keyword indices instead of individual users' indices. The experiments show that such clustering saves space and that processing keyword queries on cluster-based indices has acceptable time overheads. There are two key differences between our factoring strategy and this work. First, factoring is explored in a pairwise fashion (and not for an entire user cluster). Second, factoring does not incur additional I/O. One extension of our work is to explore factoring for a cluster of users.

Top-K Processing: The family of top-k threshold algorithms [7,8] aims to reduce the amount of processing required to compute top-ranked answers and have been used in the relational [5], XML [15] and many other settings. Monotonic score aggregation functions, which operate on sorted input, enable the early pruning of low-rank answers. In this work, we apply these algorithms on user's predicted rating lists and introduce pairwise disagreement lists to improve performance.

Knapsack Problem: This combinatorial optimization problem [9,20] arises whenever resource allocation is required between many contenders under budgetary constraints. Each resource has a cost and a value, and the total allocated resource cost is restricted under a hard constraint, so it aims to allocate resources such that it gathers maximum value for a given cost. Two main variants of this problem are *Bounded Knapsack* and *Unbounded Knapsack*. Bounded Knapsack assumes limited availability of each resource type, whereas each resource may have infinite no of copies in Unbounded Knapsack problem. We adapt a *special* case of Bounded Knapsack known as 0/1 *Knapsack* for modeling disagreement lists materialization problem. Each disagreement list is selected for materialization under overall space constraints (space budget) based on how much benefit it offers in speeding up query processing (value) by consuming how much space (cost).

8 Conclusion

Group recommendations are becoming of central importance as people engage in online social activities together. In this paper, we define the semantics and study the efficiency of delivering recommendations to groups of users. We introduce the notion of a consensus function which aims to achieve a balance between an item's aggregate predicted rating in the group and individual member's disagreements over the item. We design and implement efficient threshold algorithms to compute group recommendations. We report on a user study conducted on the MovieLens data sets using Amazon's Mechanical Turk and a comprehensive performance study of our algorithms. We established that similarity between group members impacts both quality and efficiency.

In the absence of any information about what groups could be formed, pairwise user disagreement lists need to be maintained in order to efficiently process recommendations to randomly formed groups. Hence, we developed two complementary space reduction strategies and studied their impact on space and time. In particular, our experiments showed that behavior factoring, a space-saving strategy where items two users agree on are stored only once, achieves considerable space reduction. That strategy combined with selectively materializing disagreement lists successfully addresses applications where a space budget is enforced.

There are many avenues we would like to explore in the future. One extension to this work is to devise a query optimization algorithm which takes a group and a configuration (a set of materialized and possibly factored disagreement lists) and determines which lists to use for that group. The experiment in Sect. 6.2.4 showed that it is sometimes beneficial to merge a subset of the disagreement lists for some groups, even if they are materialized. Another avenue for improvement is the implementation of threshold sharpening as described in Sect. 3.4 for the pairwise disagreement model. We believe this will have drastic improvements on processing recommendations.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. Knowl. Data Eng. **17**(6), 734–749 (2005)
2. Amer-Yahia, S., Benedikt, M., Lakshmanan, L., Stoyanovich, J.: Efficient network-aware search in collaborative tagging sites. In: VLDB (2008)
3. Amer-Yahia, S., Roy, S.B., Chawla, A., Das, G., Yu, C.: Group recommendation: Semantics and efficiency. In: VLDB (2009)
4. Baatarjav, E.-A., Phithakkitnukoon, S., Dantu, R.: Group recommendation system for facebook. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM Workshops, vol. 5333 of Lecture Notes in Computer Science, pp. 211–219. Springer, Berlin (2008)
5. Carey, M.J., Kossmann, D.: On saying "enough already!" in sql. In: SIGMOD (1997)
6. Chen, Y.-L., Cheng, L.-C., Chuang, C.-N.: A group recommendation system with consideration of interactions among group members. Expert Syst. Appl. **34**(3), 2082–2090 (2008)
7. Fagin, R.: Combining fuzzy information: An overview. SIGMOD Rec. **32**(2), 109–118 (2002)
8. Fagin, R. et al.: Optimal aggregation algorithms for middleware. In: PODS (2001)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco (1979)
10. GroupLens at University of Minnesota. http://www.grouplens.org/node/73
11. Jameson, A., Smyth, B.: The Adaptive Web, vol. 4321 of LNCS, Chapter Recommendation to Groups, p. 596. Springer, Berlin (2007)
12. Jarvelin, K., Kekalainen, K.: Cumulated gain-based evaluation of ir techniques. ACM TOIS **20**(4), 422–446 (2002)
13. Kim, H.O.J.K., Kim, H.K., Ryu, Y.: A group recommendation system for online communities. Int. J. Inf. Manage. (2009)
14. Konstan, J.A.: Introduction to recommender systems. In: SIGIR (2007)
15. Marian, A., Amer-Yahia, S., Koudas, N., Srivastava, D.: Adaptive processing of top-k queries in xml. In: ICDE (2005)
16. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations (Wiley-Interscience Series in Discrete Mathematics and Optimization). Wiley, London (1990)
17. McCarthy, K., McGinty, L., Smyth, B.: Case-based group recommendation: Compromising for success. In: Weber, R., Richter, M.M. (eds.) ICCBR, vol. 4626 of Lecture Notes in Computer Science, pp. 299–313. Springer, Berlin (2007)
18. O'Connor, M., Cosley, D., Konstan, J.A., Riedl, J.: Polylens: A Recommender System for Groups of User. In: ECSCW, pp. 199–218. (2001)
19. Park, M.-H., Park, H.-S., Cho, S.-B.: Restaurant recommendation for group of people in mobile environments using probabilistic multi-criteria decision making. In: Lee, S., Choo, H., Ha, S., Shin, I.C. (eds.) APCHI, vol. 5068 of Lecture Notes in Computer Science, pp. 114–122. Springer, Berlin (2008)
20. Sahni, S.: Approximate algorithms for the 01 knapsack problem. In: Journal of the ACM. ACM (1975)