Group Recommendation: Semantics and Efficiency

Sihem Amer-Yahia[†], Senjuti Basu Roy[‡], Ashish Chawla[†‡], Gautam Das[‡], Cong Yu[†]
[†]Yahoo! Labs, {sihem,achawla,congyu}@yahoo-inc.com,
[‡]Univ. of Texas at Arlington, senjuti.basuroy@mavs.uta.edu, {achawla,gdas}@uta.edu

ABSTRACT

We study the problem of group recommendation. Recommendation is an important information exploration paradigm that retrieves interesting items for users based on their profiles and past activities. Single user recommendation has received significant attention in the past due to its extensive use in Amazon and Netflix. How to recommend to a group of users who may or may not share similar tastes, however, is still an open problem. The need for group recommendation arises in many scenarios: a movie for friends to watch together, a travel destination for a family to spend a holiday break, and a good restaurant for colleagues to have a working lunch. Intuitively, items that are ideal for recommendation to a group may be quite different from those for individual members. In this paper, we analyze the desiderata of group recommendation and propose a formal semantics that accounts for both item relevance to a group and disagreements among group members. We design and implement algorithms for efficiently computing group recommendations. We evaluate our group recommendation method through a comprehensive user study conducted on Amazon Mechanical Turk and demonstrate that incorporating disagreements is critical to the effectiveness of group recommendation. We further evaluate the efficiency and scalability of our algorithms on the MovieLens data set with 10M ratings.

1. INTRODUCTION

The social component of people's activities on the Web has seen unprecedented growth lately. In addition to social networking sites like Facebook, content sites such as Yahoo! Travel, which have traditionally focused on managing content only, are beginning to encourage people to form social ties and share content. While there has been a recent trend in developing techniques for finding relevant content on social content sites [3], very little has been done to *help socially acquainted individuals* find content of interest to all of them together. We refer to this as the *group recommendation* problem and study how to effectively and efficiently find such recommendations in this paper.

Defined simply as a set of users, a *user group* can be formed on a recurring basis, e.g., friends who meet regularly for dinner, or on an ephemeral basis, e.g., random users getting together for a weekend climbing trip organized by their sports club. Regardless how the group is formed, recommending to the group presents two major challenges in comparison to individual user recommendations [9, 11].

The first challenge is how to define the semantics of group recommendation. In individual user recommendation, the usual goal is to retrieve items with the highest scores, also referred to as relevance or user's expected rating, computed by a given recommendation strategy. For example, a system such as Netflix recommends movies which are more likely to be rented by a user by finding those with the highest expected rating by that user. In group recommendation, however, an item may have different relevances to different group members and this *disagreement* among members must be resolved. Indeed, despite being friends, people may not share the same tastes for all movies. The same observation can be made for colleagues going to a working lunch or family members traveling together.

Existing methods have focused mostly on aggregating individual group members' relevances to produce recommendations to a group [9]. In particular, those methods can be classified into two approaches: *preference aggregation*, which aggregates group members' prior ratings into a single virtual user profile and makes recommendations to that user; *score aggregation*, which computes each member's individual recommendations and merges them to produce a single list for the group, where the score of each item is aggregated from individual recommendations. The *score aggregation* approach typically offers better flexibility [9, 13] and more opportunities for efficiency improvement and is therefore the approach we take in this paper.

Two main score aggregation functions have been proposed thus far: *average* and *least misery*. Strategies incorporating the former aim to maximize the average of group members' scores for an item, while strategies incorporating the latter aim at maximizing the lowest score among all group members. However, neither models the resolution of disagreement among users in a formal way. Intuitively, in the movie domain, taking the average of expected ratings for a movie may result in a high score for movies which are highly liked by some group members and highly disliked by others. Similarly, optimizing for the lowest projected rating would miss a movie which is expected to be liked by every group member except one.

VLDB '09, August 24-28, 2009, Lyon, France

In this paper, we formalize group disagreement by making it an integral part of group recommendation semantics and study the various ways it can be resolved. Specifically, we introduce the notion of *consensus function*, which consists of two components, *relevance* and *disagreement*, and, for each candidate item, produce a single recommendation score that is a weighted summation of the two component scores. Relevance is modeled in a way which captures existing strategies, i.e., average or least misery. Disagreement is modeled using two alternative ways: *average of pairwise disagreements* or *score variance*. We formally introduce the two disagreement models in Section 2.

The second challenge is how to efficiently compute group recommendations given a consensus function, especially in the presence of complex disagreement models.[1] The weighted summation nature of the consensus function is well supported by the family of threshold algorithms first proposed in [6]. Furthermore, the relevance component of the consensus function lends itself well to these algorithms since it aggregates recommendation scores from individual *relevance lists* (i.e., a list of items sorted by their score for each user) in a monotonic fashion. Threshold-based pruning is hence straightforward. This is not the case for the disagreement component.

Since disagreements are derived from individual recommendation scores, any disagreement score can be computed on the fly based on the relevance scores.[2] However, given the sorted relevance lists of two users, the algorithm cannot predict their disagreement over unseen items. Indeed, two users may agree or disagree equally on movies they like or dislike. The fact that their individual movie lists are available in sorted order does not provide insights on how their disagreements over unseen movies may evolve. This is a key observation that led us to introduce *materialized pairwise disagreement lists* among group members. Such lists can be sorted on increasing disagreement thereby lending themselves to threshold-style processing. We prove that the two disagreement models we adopt in this paper can be decomposed into an aggregation of score disagreements between pairs of group members, and that this aggregation satisfies the monotonicity condition required by the threshold algorithms. As a result, we adopt threshold algorithms for the overall score computation with effective bounds for both relevance and disagreement.

Our final endeavor is to study optimization opportunities which exploit the dependencies between disagreement and relevance. The first optimization aims to reduce the number of disagreement lists that need to be maintained. Indeed, the number of pair-wise disagreement lists is quadratic in the number of users, which makes full materialization of all user pairs impractical. We further study the problem of selecting a subset of the disagreement lists in order to achieve the best runtime performance given a space budget. We investigate a second optimization which explores the use of relevance values to compute tighter disagreement bounds in the presence of (partially) materialized disagreement values.

In summary, we make the following contributions:

1. We formalize the problem of group recommendation and define its semantics as a consensus function that aims at maximizing item relevance and minimizing disagreements between group members (Section 2).
2. We prove that the two important disagreement models being proposed satisfy the conditions required by the family of top-k threshold algorithms (Section 3).
3. We design efficient algorithms based on one representative threshold algorithm, `TA`, to perform top-k group recommendation (Section 4).
4. We formalize two optimizations: the problem of which disagreement lists to materialize given a space budget (Section 4.2) and the refinement of score bounds (Section 4.3).
5. We conduct a comprehensive experimental evaluation (Section 5), including a user study on Amazon Mechanical Turk to demonstrate the benefits of incorporating disagreements into group recommendation, and an extensive performance experiment to demonstrate the efficiency of our top-k group recommendation algorithms.

Finally, we describe the related work and conclude in Sections 6 and 7, respectively.

[1] In this paper, we assume individual recommendation scores for items have been generated and can be fetched on a per user basis.

[2] We refer to this as the *relevance only* approach.

2. MODEL AND SEMANTICS

In this section, we formally define the problem of group recommendation and introduce the *consensus function* for estimating an item's worthiness for a group based on its relevance and disagreement among group members.

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 $\texttt{rating}(u, i)$. We further generate *relevance scores* for each pair of user and item, denoted as $\texttt{relevance}(u, i)$. This relevance score comes from two sources. If the user has provided a rating for the item, then it is simply the user provided rating. Otherwise, the system generates the relevance score using a recommendation strategy as outlined next.

2.1 Individual Recommendation

The most well-known recommendation 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 (collaborative filtering) [1].

In item-based strategies, the relevance of an item $i \in \mathcal{I}$ by a current user $u \in \mathcal{U}$ is commonly estimated as follows:

$$\texttt{relevance}(u, i) = \Sigma_{i' \in \mathcal{I}} \texttt{ItemSim}(i, i') \times \texttt{rating}(u, i')$$

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 activities.

The key of collaborative filtering is to find other users connected to the given user. The relevance of an item i by a user u is estimated as follows:

$$\texttt{relevance}(u, i) = \Sigma_{u' \in \mathcal{U}} \texttt{UserSim}(u, u') \times \texttt{rating}(u', i)$$

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).

2.2 Group Recommendation

The goal of group recommendation is to compute a recommendation score for each item that reflects the interests and preferences of all 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 with each other*. All other conditions being equal, an item that members agree more on should have a higher score than an item with a lower overall group agreement. We call the first aspect *group relevance* and the second aspect *group disagreement*.

DEFINITION 1 (GROUP RELEVANCE). *The relevance of an item i to a group $\mathcal{G}$, denoted $\texttt{rel}(\mathcal{G}, \texttt{i})$, is an aggregation over $\texttt{relevance}(u, i)$ where $u \in \mathcal{G}$. We consider two main aggregation strategies:*

1) Average: $\texttt{rel}(\mathcal{G}, i) = \frac{1}{|\mathcal{G}|} \sum (\texttt{relevance}(u, i))$

2) Least-Misery: $\texttt{rel}(\mathcal{G}, i) = Min(\texttt{relevance}(u, i))$

Average and Least-Misery aggregation models are considered because they are the most prevalent mechanisms being employed currently [9]. Least-Misery model captures cases where some user has a strong preference (e.g., a vegetarian who cannot go to a steakhouse) and that user's preference acts as a veto. Alternative aggregations (e.g., taking the maximum over all individual relevances to comply with the "most optimistic" user) 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 relevance scores for i among group members. We consider the following two main disagreement computation methods:*

1) Average Pair-wise Disagreements:
$\texttt{dis}(\mathcal{G}, i) = \frac{2}{|\mathcal{G}|(|\mathcal{G}|-1)} \sum (|\texttt{relevance}(u, i) - \texttt{relevance}(v, i)|)$, *where $u \neq v$ and $u, v \in \mathcal{G}$;*

2) Disagreement Variance:
$\texttt{dis}(\mathcal{G}, i) = \frac{1}{|\mathcal{G}|} \sum_{u \in \mathcal{G}} (\texttt{relevance}(u, i) - \texttt{mean})^2$, *where* $\texttt{mean}$ *is the mean of all the individual relevances for the item.*

The average pair-wise disagreement function computes the average of pair-wise relevance differences for the item among group members, while the variance disagreement function computes the mathematical variance of the relevances for the item among group members. Intuitively, the closer the relevance scores for i between users u and v, the lower their disagreement for i. In Section 3, we characterize the properties of both disagreement functions in detail.

Finally, we combine the group relevance and group disagreement for an item in the *consensus function*.

DEFINITION 3 (CONSENSUS FUNCTION). *The consensus function, denoted $\mathcal{F}(\mathcal{G}, i)$, combines the group relevance and the group disagreement of i for $\mathcal{G}$ into a single group recommendation score using the following formula:*

$\mathcal{F}(\mathcal{G}, i) = w_1 \times \texttt{rel}(\mathcal{G}, i) + w_2 \times (1 - \texttt{dis}(\mathcal{G}, i))$, *where $w_1 + w_2 = 1.0$ and each specifies the relative importance of relevance and disagreement in the overall recommendation score.*

While one could design more sophisticated consensus functions, we adopt this general form of weighted summation of group relevance 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 relevance 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. $\forall i \in \mathcal{I}_\mathcal{G}, u \in \mathcal{G}$, u has not rated i before;
3. $\mathcal{I}_\mathcal{G}$ is sorted on decreasing group recommendation score as computed by the consensus function $\mathcal{F}$, and $\nexists j \in \mathcal{I}$ s.t. $\mathcal{F}(\mathcal{G}, j) > \mathcal{F}(\mathcal{G}, i), j \notin \mathcal{I}_\mathcal{G}, i \in \mathcal{I}_\mathcal{G}$.

3. APPLICABILITY OF TOP-K THRESHOLD ALGORITHMS

Many of the best algorithms for computing top-k items belong to the family of threshold algorithms [6]. 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 relevance* and *group disagreement*. It is clear that the consensus function itself is monotone in the two individual components. In other words, if two items have the same group disagreement, the item with the higher group relevance will have at least the same group recommendation score, and vice versa.

It is also clear that the two group relevance functions proposed in Definition 1 are themselves monotone in the relevances 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 relevance 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 relevance lists and pair-wise disagreement lists as inputs, and take advantage of the pruning power that threshold algorithms give us.

3.1 Monotonicity of Group Disagreements

We use a simple example group of two users to show that computing group disagreement based on relevances of individual members is not monotonic. Figure 1(a) illustrates

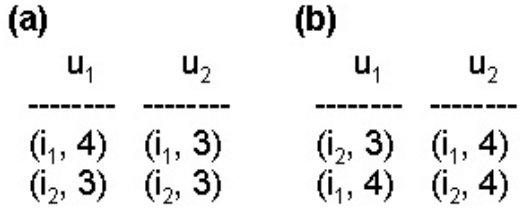

Figure 1: Group Disagreement is not monotonic w.r.t. relevance lists.

the two sorted relevance lists for the two users (u_1 and u_2). It is clear that while i_1 has a higher relevance for u_1 than i_2 (4 versus 3), the group disagreement score for i_2 is in fact higher (1 instead of 0). The same non-monotonicity can be encountered when relevance lists are sorted in decreasing order (as shown in the example in Figure 1(b)). Hence, regardless of the problem of non-monotonicity of disagreement in relebace lists persists regardless of the order in which relevance lists are sorted.

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

3.2 Disagreement Lists for Monotonicity

A pair-wise disagreement list (or simply disagreement list) for users u and v is a list of items which are sorted in the increasing order of the difference between their relevance scores for u and v. For an item i, we use $\Delta^i_{u,v} = |\texttt{relevance}(u,i) - \texttt{relevance}(v,i)|$ to denote this relevance difference.

LEMMA 1. *The average pair-wise disagreement function in Definition 2 is monotonic w.r.t. pair-wise disagreement lists.*

Proof: Let us assume a group $\mathcal{G} = \{u_1, u_2, ..., u_p\}$ with all its $p(p-1)/2$ disagreement lists (one for each user pair). Also assume that there are a total of t items, $\mathcal{I} = \{i_1, i_2, ..., i_t\}$. 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: $f \times \Sigma_{\forall j,k \in p}(\Delta^{i_r}_{u_j,u_k})$ and $f \times \Sigma_{\forall j,k \in p}(\Delta^{i_s}_{u_j,u_k})$, respectively, where $f = \frac{2}{p(p-1)}$ (see Definition 2).

Without loss of generality, assume i_r's pairwise disagreements are smaller than i_s's in one particular list $l = (u_x, u_y)$, and the same for all other lists. We have $\Delta^{i_r}_{u_x,u_y} < \Delta^{i_s}_{u_x,u_y}$, and $\forall j,k \in p, \Delta^{i_r}_{u_j,u_k} = \Delta^{i_s}_{u_j,u_k}$, where $(j,k) \neq (x,y)$. It is easy to see that $f \times \Sigma_{\forall j,k \in p}(\Delta^{i_r}_{u_j,u_k}) < f \times \Sigma_{\forall j,k \in p}(\Delta^{i_s}_{u_j,u_k})$.

If the number of disagreement lists is restricted to m,[3] 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 relevance scores among all group members. In other words, the relevance score by each member is compared against the mean relevance score of the group. We now show that this disagreement function can in fact be monotonically aggregated from pairwise disagreement lists.

[3]We discuss partial materialization of disagreements lists in Section 4.2.

LEMMA 2. *The disagreement variance function in Definition 2 is monotonic w.r.t. pair-wise 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:

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

and

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

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

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

which can be further expressed as:

$$[\Delta^{i_r}_{12}]^2 + ... + [\Delta^{i_r}_{1p}]^2 + ... + 2 \times [\Delta^{i_r}_{12}][\Delta^{i_r}_{13}] + 2 \times [\Delta^{i_r}_{12}][\Delta^{i_r}_{14}] + ...$$

It is clear that the above formula is a monotonic aggregation of $[\Delta_{jk}] \forall j,k \in p$. Without loss of generality, assume i_r's pairwise disagreements are smaller than i_s's in one particular list $l = (u_x, u_y)$, and the same for all other lists. We have $\Delta^{i_r}_{u_x,u_y} < \Delta^{i_s}_{u_x,u_y}$, and $\forall j,k \in p, \Delta^{i_r}_{u_j,u_k} = \Delta^{i_s}_{u_j,u_k}$, where $(j,k) \neq (x,y)$. It is easy to see based on the above formula that the disagreement variance of i_r is less than the disagreement variance of i_s. Hence, we have proved that using pair-wise disagreement lists is sufficient to compute disagreement variance in a monotonic fashion. □

Materializing all possible pair-wise disagreement lists may not be practical since the number of such lists grows quadratically in the number of users. In Section 4.2, we discuss, given a fixed space constraint, which pairs to materialize in order to produce the best performance with threshold algorithms.

4. EFFICIENT COMPUTATION OF GROUP RECOMMENDATION

In this section, we provide a brief description of efficient group recommendation algorithms and discuss two subequent optimizations: the effective materialization of disagreement lists in the presence of space constraints (Section 4.2); and exploiting dependencies between relevance and disagreement values to refine thresholds and enable early stopping in top-k processing (Section 4.3).

4.1 Group Recommendation Algorithms

Given a group $\mathcal{G}$, the goal, stated in Section 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 [7] for top-k query processing.

We start by describing Algorithm 1, which admits relevance 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 relevance and $\mathcal{DL}s$ are sorted

in increasing order of disagreement. We refer to Algorithm 1 as FM for Fully Materialized disagreement lists.

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

We showed in Section 3 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 (relevance and disagreement) in a round-robin fashion (`getNext` calls in lines 3 and 12). The algorithm uses two routines, `ComputeExactScore` which computes the score of the current item, and `ComputeMaxScore` which produces a new threshold value at each round.

`ComputeExactScore` performs a random access (RA) on all other relevance lists to compute the score of an 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 relevances). In fact, $\mathcal{DL}s$ are not necessary to compute the final result. They are only there to compute the threshold (using `ComputeMaxScore`) and hopefully, enable early termination.

`ComputeMaxScore` produces a new threshold value at each round. Its basic purpose is to provide an upper bound the score of any item that has not yet been seen by the algorithm. Thus, if r_u is the last relevance 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:

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

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

Require: Group $\mathcal{G}$, consensus function F;
1: Retrieve relevance 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: Cursor cur = `getNext`() moves across each relevance and disagreement lists;
4: **while** (cur <> NULL) **do**
5: Get entry $e = (i, r)$ at cur;
6: **if** not(`inHeap`(`topKHeap`, e)) **then**
7: **if** (`ComputeMaxScore`($e.i, e.r$, F) $\geq$ `topKHeap`.$k_{th}score$) **then**
8: `ComputeExactScore`; Probe $\mathcal{IL}s$ to compute exact score $score$ of e using F;
9: `topKHeap.addToHeap`($e.i, score$);
10: **end if**
11: **end if**
12: cur = `getNext`();
13: **end while**
14: **return** topKList(`topKHeap`);

We next describe another variation of the algorithm, called RO, for Relevance lists Only, which applies when only the relevance lists are present and none of the $\mathcal{DL}s$ are available. RO has the obvious benefit of consuming less space. As discussed above, 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 relevance list:

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

Finally, the most general variant is the case where only some disagreement lists are materialized, referred to as PM for Partial Materialization. As with relevance only, partial materialization also has the obvious benefit of consuming less space then 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. Then the threshold may be computed as follows:

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

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, 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 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 next subsection. Then in Section 4.3, we discuss interesting and novel techniques by which the threshold bounds (i.e., the `ComputeMaxScore` function) can be sharpened even further. The experiments section (Section 5.2) delves into the details of the algorithm's performance for different cases of disagreement lists materialization.

4.2 Partial Materialization

Given a set of n users, materializing all possible $n(n-1)/2$ pairwise disagreement lists may be prohibitive in applications where n is large or in applications where the number of groups is large. In such cases, it is more practical to materialize only a small but effective subset of the disagreement lists in a pre-processing step. The central problem that we consider in this subsection is thus: given a fixed space constraint m (i.e., only m out of $n(n-1)/2$ lists can be materialized), to determine which lists to materialize that will be of "maximum benefit" during query processing. Intuitively, a disagreement list should be materialized if (a) the corresponding users together are very likely to be a part of user groups seeking item recommendations, and (b) ma-

terializing the list significantly improves the running time of top-k recommendation algorithms. We make this problem definition more precise below.

Let the set of users be $\mathcal{U} = u_1, \ldots, u_n$. Recall that $\mathcal{IL}_u$ is the relevant 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 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., $|M| = m$). 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 relevance 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. (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 relevance lists $\mathcal{IL}_u$ (for all $u \in \mathcal{G}$), i.e., without any disagreement lists.)

Our objective is to minimize the expected cost of executing the top-k algorithm on any user group query, using the relevance 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). *Determine the subset of pairs $M \subseteq S$ s.t. $|M| = m$ 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 pre-processing time.

In spite of these challenges, we have carefully investigated this problem, and proposed several principled and practical solutions. We discuss these next.

4.2.1 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 of users u and v, we temporarily materialize the disagreement list $\mathcal{DL}_{(u,v)}$, and compute $t_{\mathcal{DL}_{(u,v)}}(\{u,v\})$ 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 $t_{\mathcal{DL}_{(u,v)}}(\{u,v\}) \geq t_\phi((u,v))$

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 minimized:

$$\sum_{(u,v) \in M} p(\{u,v\}) \cdot (t_\phi(\{u,v\}) - t_{\{u,v\}}(\{u,v\})$$

Algorithm 2 Disagreement Lists Materialization Algorithm

Require: User pairs in S';
1: For each pair $(u,v) \in S'$, compute $p(\{u,v\}) \cdot (t_\phi(\{u,v\}) - t_{\{u,v\}}(\{u,v\}))$;
2: Return the m pairs with the largest values.

Algorithm 2 shows a very simple approach to compute M. The algorithm requires $O(n^2)$ executions of the top-k algorithm. Even though this is a pre-processing step, it may nevertheless be very time consuming. We discuss in Section 4.2.2 additional techniques by which this can be reduced. We also note that the space requirement of this algorithm is only m times the size of a single pairwise disagreement list, because while examining all user pairs, we can maintain the disagreement lists of only the current top-m most promising user pairs.

The disagreement lists materialization procedure discussed above assumed that the user groups are restricted to two members only. However, once the m 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 relevance 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.

4.2.2 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 save on pre-processing costs, it is critical that we identify

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

only those user pairs that have significant likelihood of occurring together, and only consider such pairs in the above 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 $\mathcal{G}_1, \ldots, \mathcal{G}_r$ be a query log of r user groups. Then for any user pair (u, v), we can compute

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

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.

In the case where such a query log is unavailable, we can utilize other sources of information to determine the likelihood that a pair of users are likely to occur together in a group. In related work [2, 14], it has been shown how *similarity* between a pair of users can be computed based on the tagging behavior of other users (e.g., friends, social acquaintances) in a social network. If we assume that two users are likely to be in the same group if they are very similar, we can can only restrict our attention to pairs of similar users, and set their probability to be proportional to their similarity value.

4.3 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.[5]

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 relevance 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 relevance values be i_u and i_v for user u and v respectively. The consensus function is defined as:

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

Since each list is sorted in decreasing order of relevance (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 Section 4.1, 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 can derive a sharper bound for $\mathtt{F}(\mathcal{G}, i)$ as follows:

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

This example illustrates that due to the dependencies between the disagreement lists and the relevance 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 relevance 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 pair-wise disagreement, as well as quadratic terms (e.g., of the form $(i_u - \mathtt{mean})^2$) in the case of variance based disagreement. 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 solvable by general purpose non-linear solvers with practically no overhead per iteration. Section 5.2.5 contains a preliminary experiment showing the benefit of threshold tightening.

5. EXPERIMENTS

We evaluate our group recommendation system from two major angles. First, from the *quality* perspective, we conduct an extensive user study through Amazon Mechanical Turk[6] to demonstrate that group recommendations with the consideration of disagreements are superior to those relying on aggregating individual relevance scores alone (Section 5.1). 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 (Section 5.2).

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

Data Set: We use the MovieLens [8] 10M ratings data set for evaluation purposes. The statistics of this data set is shown in Table 1.

[5]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.

[6]https://www.mturk.com/

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

Table 1: Statistics about the MovieLens Data Set.

Individual Recommendation: We adopt collaborative filtering [1] for generating individual recommendations. For a given user u, this method leverages other users who share similar interests (i.e., movies) with u to compute the scores of movies unknown to u. The basic formula being used is:

$$\texttt{relevance}(u,i) = \frac{1}{\Sigma \texttt{sim}(u,u')} \Sigma \texttt{sim}(u,u') \times \texttt{rating}(u',i)$$

where $\texttt{sim}(u,u')$ computes the similarity between two users using the following formula:

$$\texttt{sim}(u,u') = \frac{|\{i | i \in \mathcal{I}_u \ \wedge \ i \in \mathcal{I}_{u'} \ \wedge \ |\texttt{rating}(u,i) - \texttt{rating}(u,i)| \leq 2\}|}{|\{i | i \in \mathcal{I}_u \ \vee \ i \in \mathcal{I}_{u'}\}|}$$

where $\mathcal{I}_u$ denotes the set of items u has rated, and 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 to 5.

5.1 User Study

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

Average Relevance Only (AR), which computes an item score as its average relevance among group members. The disagreement weight is set to zero.

Least-Misery Relevance Only (MO), which computes an item score as its minimum relevance among all group members. The disagreement weight is set to zero.

Consensus with Pair-wise Disagreement (RP), which computes an item score as a weighted summation of its average relevance and its average pair-wise disagreements between all group members.

Consensus with Disagreement Variance (RV), which computes an item score as a weighted summation of its average relevance and the variance of its relevance between 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.

5.1.1 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 moviegoers. Towards these two goals, we select two set 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.

	Small Group	Large Group
Similar Group	0.89	0.90
Dissimilar Group	0.29	0.27
Random Group	0.69	0.73

Table 2: Similarities of User Study Groups.

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 above; 2) combined with having the maximum summation of pair-wise 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 above; 2) combined with having the minimum summation of pair-wise 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.

5.1.2 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 com-

bined 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 small-group HITs and 10 users participated in the three large-group HITs, for a total of 45 users.

5.1.3 Result Interpretation

Given a Mechanical Turk user's ground truth evaluation of the candidate movies, we adopt the Discounted Cumulative Gain (DCG) [10] measure to evaluate each of the following six group recommendation strategies:

AR, MO: these two are group recommendation lists generated based on average relevance and least-misery, respectively.

RP20, RP80: these two are group recommendation lists generated by combining group relevance (average relevance) and pair-wise disagreements. 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 group relevance (average relevance) and disagreement variance. RV20 sets w_2 in Definition 3 to 0.2, while RV80 sets it to 0.8.

We do not consider **MP** (least-misery relevance model combined with pair-wise disagreement), **MV** (least-misery relevance model combined with variance) strategies because the least-misery model is by definition about the rating of one group member.

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

$$DCG_{10} = rel_1 + \sum_{i=2}^{10} \frac{rel_i}{\log_2(i)}$$

where rel_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 re-sorting the movies in the list in the order of their relevances.)

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 Figure 2.

The top-left chart in Figure 2 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 Figure 2 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 relevance 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 Figure 2 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 Figure 2 reports the differences in our disagreement models (notice the different weights) for dissimilar user groups. It is interesting to notice that, for small 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 [9]. Such feedback would help draw a stronger connection between group size and overall group dynamics in group recommendation.

5.2 Performance Evaluation

In this section, we analyze the performance of the three group recommendation algorithms described in Section 4: Dynamic Computation with Relevance 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 [6], 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

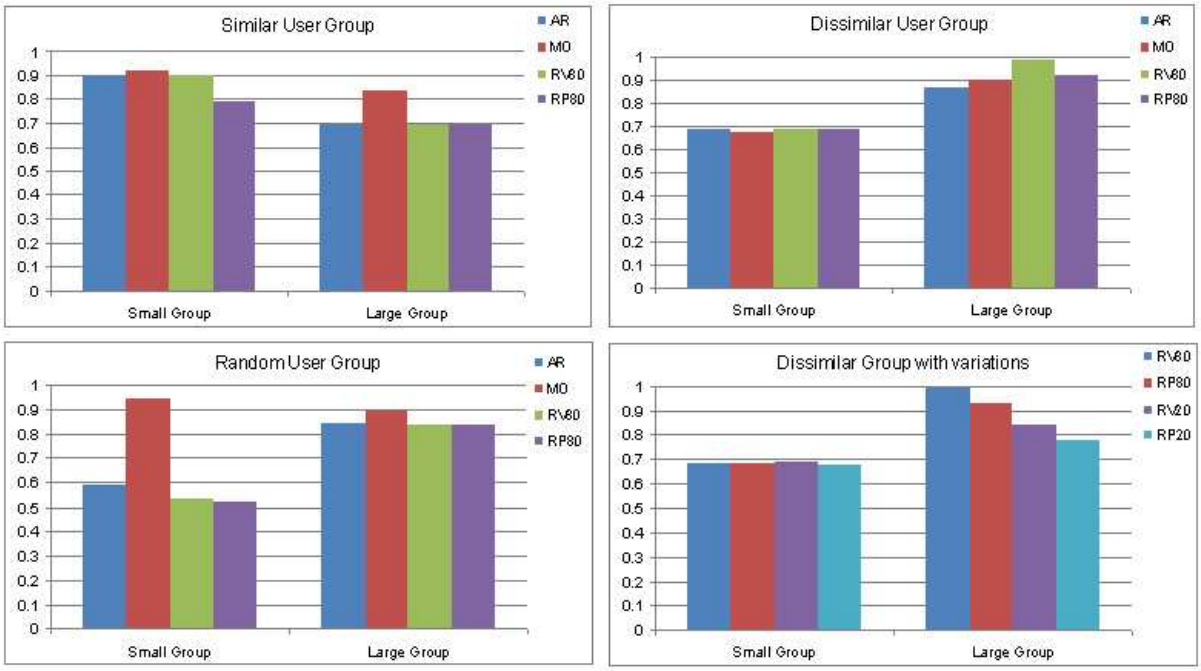

Figure 2: Comparison of User Relevances among Different Group Recommendation Lists.

number of random accesses is proportional to the *number of distinct items processed*. We adopt the bounded buffer version of TA and therefore mostly measure the number of sequential accesses to compare the performance between various algorithms.

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 relevance lists of similar users tend to contain similar items at similar positions, including those with high relevances. 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. Our final experiment shows that thereshold tightening, described in Section 4.3, always benefits TA's performance, and is worth further exploration.

5.2.1 Varying Group Similarity

Figure 3(a)(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 Figure 3(a)(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.

5.2.2 Varying K

Figure 3(c) 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.

5.2.3 Varying Group Size

We examine the effect of different group sizes in Figure 3(d). The group similarity is fixed at 0.5 and the number of recommended items is 10. For PM, the number of mate-

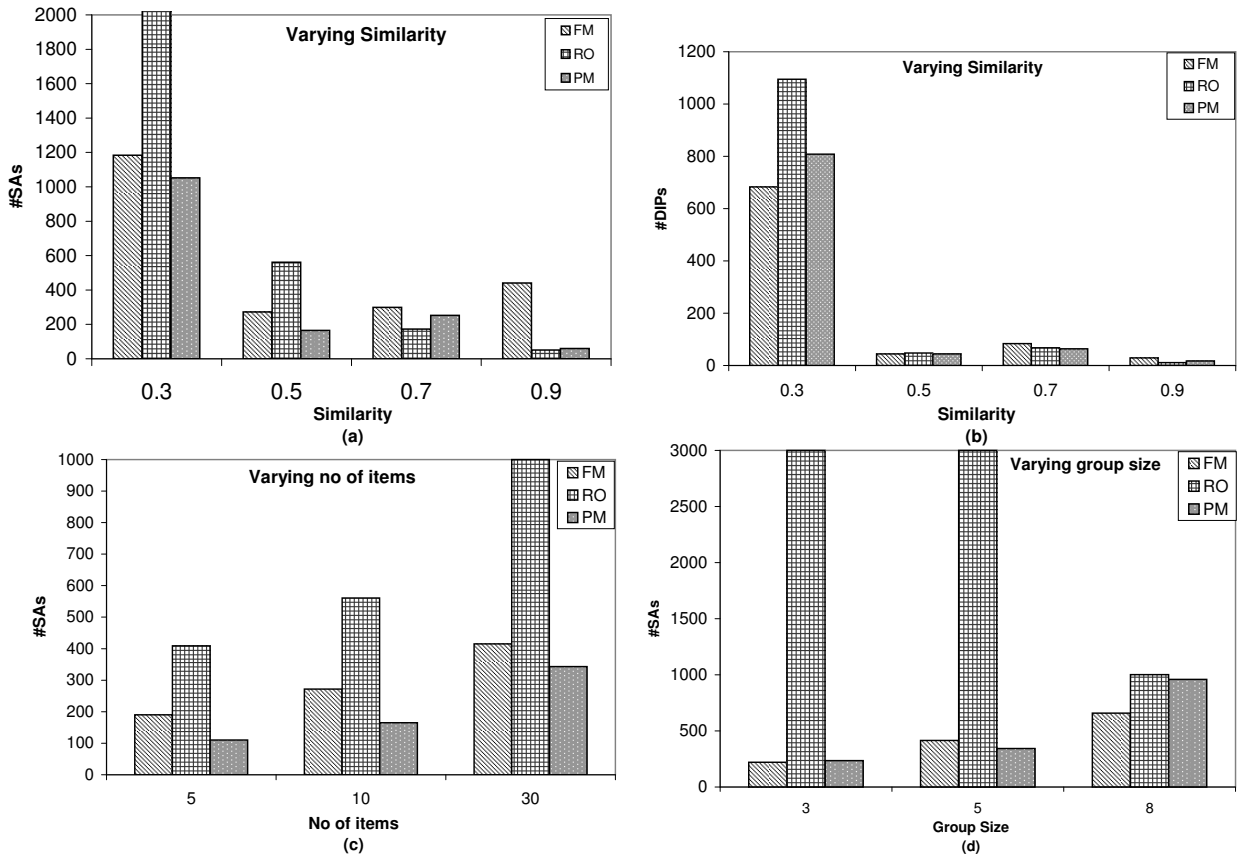

Figure 3: Performance Comparison among Algorithms RO, FM, PM.

rialized 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 out-perform 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 relevance 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.

5.2.4 *Effect of Partial Materialization*

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, the number of recommended items is 5. We report SAs and DIP by varying the number of materialized disagreement lists. As shown in Figure 4, 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 4^{th} to the 10^{th} list, the number of DIP remains almost the same. By examining the 4^{th} 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 4^{th} 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.

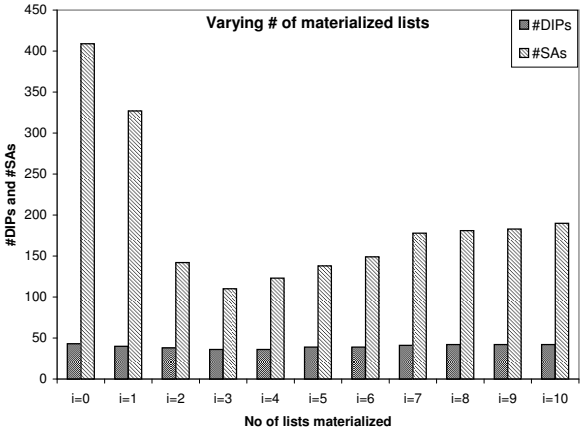

Figure 4: Effect of the number of $\mathcal{DL}$s.

5.2.5 *Effect of Threshold Sharpening*

Figure 5 illustrates the effect of Threshold Sharpening during TA. More specifically, we investigated the impact of optimization for FM by recording average number of SAs with different numbers of items recommended for 10 small user groups. Figure 5 records average number of SAs for FM with optimization and FM without optimization for k=5, 10 and 30. This result corroborates our theoretical analysis, i.e., FM with optimization is never worse than FM without optimization. In fact, it is easy to observe that the effect of optimization becomes significant with higher values of k. We realized that our data set is responsible for such phe-

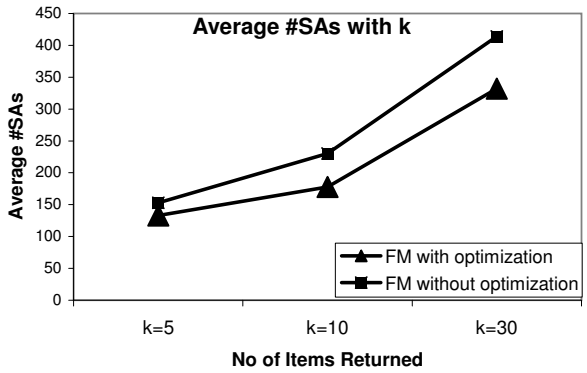

Figure 5: Effect of Threshold Sharpening on TA.

nomenon, where disagreement is noteworthy only after first few items between user pairs and becomes more significant as we scan deep into the lists.

6. RELATED WORK

Recommendations: The goal of a recommendation strategy [1] and [11] 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 collaborative filtering. 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.

Group Recommendations: [9] 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. The latter creates individual recommendation lists for each member and consolidate those lists to form the group's list. We adopt the latter approach for its flexibility as described in [9]. Most of the studies on group recommendations focus on group formation and evolution, privacy concerns and interfaces for supporting group recommendations. Furthermore, with the exception of PolyLens [13], only few conducted user studies to evaluate the benefits of group recommendations. PolyLens 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 (e.g., by inviting each other) and the system studied how groups affected the way they used MovieLens. In order to produce group recommendations, individual groups members' recommendations were merged using the least misery model. User satisfaction was measured in terms of different criteria: how easy the process of creating groups was; how easy it is to add members to a group; how useful group recommendations were; and overall satisfaction. The study concluded, among other findings, that users in a group prefer group recommendations than the individual recommendations.

We note here that none of the group recommendation studies we have encountered provides any performance experiments or a theoretical and empirical study of different consensus functions, as it is done in this paper.

Top-K Processing: The family of top-k threshold algorithms [5, 7] aim to reduce the amount of processing required to compute top-ranked answers, and have been used in the relational [4], XML [12], 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 relevance lists and introduce pair-wise disagreement lists to improve performance.

7. CONCLUSION

Group recommendation is becoming increasingly important as people engage in social activities on the Web. This is the first attempt to define the semantics and study the efficiency of delivering recommendations to groups of users. We formalized the notion of a consensus function which achieves a balance between an item's aggregate relevance to the group and individual member's disagreements over the item. We designed and implemented efficient threshold algorithms to compute group recommendations. We report on a user study on MovieLens data using Amazon's Mechanical Turk and a comprehensive performance study of our algorithms. We established that disagreements between group members impacts both quality and efficiency, and can be exploited to increase the effectiveness of group recommendation.

8. REFERENCES

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

[2] S. Amer-Yahia, M. Benedikt, L. Lakshmanan, and J. Stoyanovich. Efficient Network-Aware Search in Collaborative Tagging Sites. In *VLDB*, 2008.

[3] S. Amer-Yahia, L. Lakshmanan, and C. Yu. SocialScope: Enabling Information Discovery on Social Content Sites. In *CIDR*, 2009.

[4] M. J. Carey and D. Kossmann. On saying "enough already!" in sql. In *SIGMOD*, 1997.

[5] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 32(2):109–118, 2002.

[6] R. Fagin and et. al. Optimal Aggregation Algorithms for Middleware. In *PODS*, 2001.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[8] GroupLens at University of Minnesota. http://www.grouplens.org/node/73.

[9] A. Jameson and B. Smyth. *The Adaptive Web*, volume 4321 of *LNCS*, chapter Recommendation to Groups, page 596. Springer-Verlag, 2007.

[10] K. Jarvelin and K. Kekalainen. Cumulated gain-based evaluation of ir techniques. *ACM TOIS*, 20(4), 2002.

[11] J. A. Konstan. Introduction to recommender systems. In *SIGIR*, 2007.

[12] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive processing of top-k queries in xml. In *ICDE*, 2005.

[13] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl. Polylens: A recommender system for groups of user. In *ECSCW*, pages 199–218, 2001.

[14] J. Stoyanovich, S. Amer-Yahia, C. Marlow, and C. Yu. A Study of the Benefit of Leveraging Tagging Behavior to Model UsersÍnterests in del.icio.us. In *AAAI Spring Symposium on Social Information Processing*, 2008.