Exploiting Group Recommendation Functions for Flexible Preferences

Senjuti Basu Roy [*1], Saravanan Thirumuruganathan [#2], Sihem Amer-Yahia [+3], Gautam Das [#4], Cong Yu [−5]

[*] *Institute of Technology, University of Washington Tacoma*
[1] senjutib@uw.edu

[#] *Computer Science Department, University of Texas at Arlington*
[2] saravanan.thirumuruganathan@mavs.uta.edu
[4] gdas@uta.edu

[+] *CNRS-LIG*
[3] sihem.amer-yahia@imag.fr

[−] *Google Research*
[5] congyu@google.com

***Abstract*—We examine the problem of enabling the flexibility of updating one's preferences in group recommendation. In our setting, any group member can provide *a vector of preferences* that, in addition to past preferences and other group members' preferences, will be accounted for in computing group recommendation. This functionality is essential in many group recommendation applications, such as travel planning, online games, book clubs, or strategic voting, as it has been previously shown that user preferences may vary depending on mood, context, and company (i.e., other people in the group). Preferences are enforced in an *feedback box* that replaces preferences provided by the users by a potentially different feedback vector that is better suited for *maximizing the individual satisfaction* when computing the group recommendation. The feedback box interacts with a *traditional recommendation box* that implements a group consensus semantics in the form of Aggregated Voting or Least Misery, two popular aggregation functions for group recommendation. We develop efficient algorithms to compute *robust* group recommendations that are appropriate in situations where users have changing preferences. Our extensive empirical study on real world data-sets validates our findings.**

I. Introduction

Group recommendation, i.e., recommending items to a group of users based on individual users' preferences, is an active research topic [1], [2], [3], [4]. It is usually based on aggregating individual users' preferred items into a single list of recommendations to a group using a consensus function. In this paper, we are interested in a specific scenario where users are provided the flexibility to update their preferences during recommendation time by choosing items they would like or not to see, and the system accounts for those newly provided preferences to compute recommendations to the group. This new feature is useful in a number of practical applications such as travel planning, online games, and book clubs, or strategic voting [5], where users are likely to be in a different mindset at recommendation time and do not want the system to *solely rely on their past preferences*. Enabling this flexibility gives rise to novel unexpected challenges.

In the case of trip planning, most tour buses impose a rigid itinerary with pre-determined destinations and risk alienating users. Consequently, tour organizers have expressed interest in adjusting trips by incorporating current preferences of the users, when traveling with others. As was shown in previous work [4], reaching consensus between group members is an important step in group recommendation. Previous studies have shown that users' mood, context and company, may affect their preferences [6], [7] Oftentimes, users will like to update their preferences *consciously* in an effort to maximize their individual satisfaction, by understanding the preferences expressed by other members in the group. For example, an individual may notice that her most desired item is not preferred by many of the other group members, and is thus unlikely to be recommended to the entire group. In such a situation, it may be worthwhile for the individual to abandon her original preference and provide a new preference for items that are also popular with the other users, which will result in generating a group recommendation that satisfies her the most. This seemingly intuitive idea - i.e., how to update one's preferences in order to maximize individual satisfaction in group recommendation - turns out to be quite challenging to solve, and our investigation of this problem is one of the main contributions of this paper.

Therefore, we introduce a **flexible feedback model** in the form of a vector of preferences that any group member could provide at recommendation time. Next, we propose a **feedback box** that outputs a *feedback vector* for each group member such that her *satisfaction* is *maximized* in the generated recommendation, given her current preferences, if the generated feedback vector is used in the group recommendation generation process.

Enabling preferences that can be updated poses unexpected challenges to popular group recommendation consensus func-

The work of Saravanan Thirumuruganathan and Gautam Das is partially supported by NSF grants 0812601, 0915834, 1018865, a NHARP grant from the Texas Higher Education Coordinating Board and grants from Microsoft Research and Nokia Research.

tions [3] - such as *Aggregated Voting* and *Least Misery*. In fact, it is not enough for the system to use collected preferences from group members (some members may choose not to provide any, and let the system only use their past preferences) and produce new recommendations. Our feedback box will replace the preference vectors provided by the users by a potentially different feedback vector that is better suited for *helping users enforce their new preferences*. As we shall see next, traditional group recommendation functions, when applied in conjunction with a feedback box, generate recommendations that dramatically maximize the satisfaction of an individual member (for whom the feedback box is invoked), given her current preferences.

Example 1. *Consider a 3-member group $\mathcal{G} = \{u_1, u_2, u_3\}$ and a set of 5 POIs in London: Buckingham Palace, London Eye, Tower of London, London Dungeon, British Museum, out of which an itinerary with 3 POIs are to be recommended to $\mathcal{G}$. Assume u_1 prefers Buckingham Palace, Tower of London, and London Dungeon, u_2 prefers London Eye, Tower of London, London Dungeon, but u_3 wants to visit Tower of London, London Dungeon, British Museum. We explain this example using Least Misery which maximizes the minimum satisfaction (see Section II for formal definition) of the users. User satisfaction is measured using a simple form of Jaccard Index [8], namely Overlap Similarity (see Section II-B2 for details), which is the size of the overlap between the preferred and the recommended POIs. A recommendation for $\mathcal{G}$ could be Buckingham Palace, Tower of London, London Dungeon, which contains 2 out of the 3-POIs u_3 wants to visit. If u_3 uses the feedback box instead, it internally converts u_3's preferences into British Museum only, which results in a recommended itinerary to $\mathcal{G}$ having Tower of London, London Dungeon, British Museum. It is easy to observe that with the help of feedback box u_3 is more satisfied.*

The above example shows that the preferences provided by users at recommendation time may need to be converted internally to maximize individual satisfaction when providing the group recommendation[1]. A major challenge is to determine the best feedback vector, given the current preference of a user, where "best" is defined as maximizing user's *satisfaction* in the generated recommendation. For the feedback box to work, it has to take into account both recommendation semantics (consensus function), and the latest user preferences.

While our proposed solutions are easily extensible to ordinal or numerical preference model (Section V contains the details), we present most of the results using the Boolean preference model that is natural and easy to use [9], where the preference of an user for an item is binary - either she likes it, or not. Surprisingly, under the Boolean preference model, the task of accounting for flexible preferences is technically challenging, since the optimal recommendation generation problem itself is *NP-hard* (as we prove) under most group recommendation and user satisfaction semantics. We study the complexity of this problem in-depth, and propose novel algorithms with provable approximation factors. While the feedback box problem in general is *NP-hard* (since it uses a recommendation subroutine which itself is *NP-hard*), by assuming an oracle that can compute group recommendations efficiently, we propose novel efficient algorithms including a linear time optimal algorithm.

We realize that it can be disconcerting for existing users to experience drastic changes to recommendations due to other group members' change in preferences. To alleviate that, we introduce *recommendation robustness*, a key notion to ensure that generated recommendations after preference updates overlap with the ones generated before. We formalize robustness in group recommendation under changing preferences as a soft constraint so that the recommendation system could tune that parameter at its convenience. Most importantly, we empirically demonstrate that when multiple users use feedback box, appropriately tuned recommendation robustness successfully counterbalances the effect of feedback box, and still ensures overall group satisfaction.

In summary, this paper makes the following technical contributions:

- We motivate the need for a feedback box that enables flexible user preferences at recommendation time.
- We prove that the usefulness of the feedback box is strictly related to specific group recommendation semantics and study Boolean preference model in depth.
- We develop efficient algorithms to enable online group recommendations for optimizing flexible user preferences and recommendation robustness.
- We run extensive offline and online quality experiments to validate the need for a feedback box, validate recommendation robustness, and study the performance of our algorithms using multiple real-world datasets.

II. Preliminaries

A. Interaction Model

To elicit preference, a user provides a Boolean vector (Section V discusses extension to other feedback model), where 1 corresponds to the items she prefers to consume, and 0 otherwise. *Feedback generation* phase is followed next, by invoking the feedback box for this user. During this phase, the system, through the *feedback box*, computes for the user a suggested Boolean feedback vector, such that if the suggested feedback is used *instead of her current preference she provided*, the recommendation generated by the group consensus function would also *maximize her satisfaction*. The idea of *replacing the provided preference with a suggested feedback vector* is the main distinction between our work and other feedback-based semantics [9]. Next, the system analyzes the feedback of each member of the group, and recommends at most k items (k is a given budget) to the group[2].

[1]if multiple users attempts to use the feedback box at the same time, they are arbitrarily sequentialized.

[2]Under certain Boolean satisfaction measures, such as Hamming Distance, it is possible that the group recommendation consensus function is optimized with less than k items, even though the budget is k. We explain these scenarios later on.

B. Data Model

A group $\mathcal{G}$ consists of a set of members $\{u_1, u_2, \ldots, u_n\}$. $\mathcal{I} = \{i_1, i_2 \ldots, i_m\}$ is the set of items from which the recommendation box has to suggest a set of at most k items.

1) User Preference Model: User preference is received in the form of a vector of length m (should be read from left-to-right) where the value at position j provides her user preference for the corresponding item i_j. We consider a Boolean preference model (where 1 stands for positive preference, and 0 for negative) that it is natural for many applications, especially when the user is not very knowledgeable about the domain (Section V contains extensions).

User Preference Vector: A preference vector under Boolean model defines a subset of items that a user would like to consume, and the ones she is not interested in. We denote the preference of a user u by $pref(u)$. As an example, the preference vectors of two users u and u' expressed over four items that are Points of Interests (POIs) in Paris: "Eiffel Tower", "Jardin des Plantes", "Musée d'Orsay" and "La Défense", in that order, could be $pref(u) = 0111$ and $pref(u') = 0011$, which convey the fact that the two users would like to visit "Musée d'Orsay" and "La Défense" and not "Eiffel Tower", and disagree on "Jardin des Plantes."

User Feedback Vector: Given $pref(u)$ of a user u, given an existing group $\mathcal{G}$, the feedback box generates a Boolean vector of length m for u, based on the optimization described in Sections III and IV. This vector is the generated feedback ($feedback(u)$) that the recommendation function uses as input to recommend items to the group that includes u.

2) Item Recommendation: The output of the recommendation function is a Boolean vector of length m that is constrained to have at most a budget of k 1's, where a 1 at the $j-th$ bit (should be read from left-to-right) corresponds to the positive recommendation of an item i_j. The output recommendation is denoted $\mathcal{I}^k$. The budget constraint of k applies to many practical scenarios - e.g., a group itinerary cannot visit more than k points of interest (items).

Robustness: By design, the generated recommendation in our system may change, whenever $pref(u)$ changes. As a result, the set of recommendations generated for two different *versions* (in terms of change in user preference) of the group often differ. Given two recommended item-sets (each is a Boolean vector of length m), $\mathcal{I}'^k$ and $\mathcal{I}''^k$, generated when some user updated her preference, the robustness quantifies the similarity between $\mathcal{I}'^k$ and $\mathcal{I}''^k$ using similarity measures. Smaller distance (or higher similarity) denotes higher robustness of recommendation to preference updates.

User Satisfaction: Given the preference vector $pref(u)$ of a user u and a recommended item vector $\mathcal{I}^k$, we define *user satisfaction*, $\mathcal{S}_u$, as the inverse of the distance between the two vectors, or proportional to the similarity between them. Traditional distance measures such as Hamming distance and Jaccard Similarity [8] are natural candidates for this purpose. Additionally, we consider a simpler notion of Jaccard Similarity in this work, namely *Overlap Similarity* that enables us to design efficient algorithms. More precisely, Overlap Similarity is defined as $\mathcal{S}_u = \Sigma_{\forall j \in m}(pref(u^j) \times \mathcal{I}^k_j)$, i.e., the number of positive bits that are shared between the user's preference and the recommended items.

3) Group Recommendation Functions: We consider two natural and popular group recommendation functions, denoted by $\mathcal{F}$. Those functions reflect different group consensus semantics [4]. Each group recommendation function needs to work in conjunction with a specific user satisfaction measure.

Aggregated Voting: Given $feedback(u)$, [3] $\forall u \in \mathcal{G}$, and k, the *Aggregated Voting Consensus* generates a set of items $\mathcal{I}^k$ such that $\Sigma_{\forall u \in \mathcal{G}} \mathcal{S}_u$ is maximized. For example, given two users with $feedback(u) = 1110$ and $feedback(v) = 0111$, and k set to 2 POIs, the resulting recommendation for the group of u and u' under the aggregated voting semantics will be 0110, which produces an overall aggregated user satisfaction of 4 (2 for u and u' respectively) under overlap similarity measure for user satisfaction.

Least Misery: Given $feedback(u)$, $\forall u \in \mathcal{G}$, and k, the *Least Misery Consensus* generates a set of items $\mathcal{I}^k$ such that $Min_{\forall u \in \mathcal{G}} \mathcal{S}_u$ is maximized. For example, given four feedbacks from the users in a group, 0101, 1101, 1001, 0010, and k set to 2 POIs, the resulting recommendation for the group under the least misery semantics will be 0011, which produces a minimum user satisfaction of 1 under overlap similarity measure for user satisfaction.

C. Problem Definitions

1) Recommendation Generation: Under a flexible user preference model, the recommendation generation task is executed whenever the preferences of a user are updated. Input to the recommendation function is the latest feedback vectors of all members in $\mathcal{G}$, and a budget k (i.e., the maximum number of items to be recommended). Output is the recommendation vector $\mathcal{I}^k$ with at most k 1-bits. The items that have 1 corresponding to their position are chosen and recommended to the entire group. The goal is to find $\mathcal{I}^k$ such that it optimizes user satisfaction based on the employed group consensus function $\mathcal{F}$.

2) Feedback Generation: The feedback box is executed whenever a member u updates her preferences. Input to the feedback box is the new preference of u, i.e., $pref(u)$, the feedback vectors $feedback(u')$ ($\forall u' \neq u \in \mathcal{G}$) of other existing members, the current budget k', [4] and the group recommendation objective $\mathcal{F}$ (see Section II-B3). The output is a Boolean feedback vector, $feedback(u)$, of length m for u. The goal is to compute $feedback(u)$, such that given $feedback(u)$, $feedback(u')$ ($\forall u' \neq u \in \mathcal{G}$), k', and $\mathcal{F}$, the recommendation box computes $\mathcal{I}'^k$ such that $\mathcal{S}_u$ is maximized.

[3] Note that the group recommendation functions can also admit $pref(u)$ instead of $feedback(u)$.

[4] As items are being consumed, e.g., a POI gets visited, the budget will gradually shrink.

D. Summary of Results

Among different Boolean satisfaction measures, only *Overlap Similarity* satisfies a *monotonicity* property (explained in Sections III-A and III-B) which allows us to design efficient algorithms with provable theoretical guarantees. For brevity, we report results on *Overlap Similarity* and *Hamming Distance* for two different group consensus functions in Sections III and IV. Unless otherwise stated, solutions for *Hamming Distance* can be trivially extended to *Jaccard Index* as well. We summarize our main technical contributions in Figure 1.

	Aggregated Voting	**Least Misery**
Overlap Similarity	**Recommendation Generation:** Optimal Algorithm **R-AGS** Complexity: O(mn)	**Recommendation Generation:** NP-hard Problem Greedy Approximation Algorithm **R-LMS** with approximation factor (1-1/e) Complexity: O(kn)
	Feedback Box: Not Useful	**Feedback Box: Useful** NP-hard Problem Optimal Algorithm **FB-LMS** (considering an oracle for recommendation computation) Complexity: O(m)
Hamming Distance	**Recommendation Generation:** NP-hard problem Algorithm **R-AGD**, based on centroid computation relaxing integrality constraint, followed by deterministic rounding. Complexity: O(mn)	**Recommendation Generation:** NP-hard problem Algorithm: **R-LMD**, based on Quadratic Programming formulation relaxing integrality constraint, followed by deterministic rounding. Complexity: polynomial
	Feedback Box: Useful NP-hard problem Algorithm: **FB-AGD,** based on Quadratic Programming formulation relaxing integrality constraint, followed by deterministic rounding. Complexity: polynomial	**Feedback Box: Useful** NP-hard problem Algorithm: **FB-LMD,** based on Quadratic Programming formulation relaxing integrality constraints, followed by deterministic rounding. Complexity: polynomial

Fig. 1. Summary of Results

III. Aggregated Voting Consensus

We discuss algorithms for computing recommendation and feedback vector after a user updated her preference for Aggregated Voting (described in Section II-B3), under Overlap Similarity and Hamming Distance. For each similarity measure, the recommendation algorithm is first described considering the updated preference of the user without feedback box, followed by the discussion considering feedback box in conjunction.

Example 2. *Imagine that a group $\mathcal{G}$ of $3(n)$ travelers u_1, u_2, u_3 are going to visit at most $3(k)$ different places. Each place is an item, and an ordered list of possible $5(m)$ items are $\{i_1, i_2, i_3, i_4, i_5\}$. u_3 updates her preferences to $pref(u_3) = 00011$ (meaning, she is now interested in i_4, i_5). The feedback vectors of rest two travelers (obtained after running their individual preferences through feedback box) are as follows: $feedback(u_1) = 11000$, $feedback(u_2) = 10011$.*

A. Overlap Similarity

Recall that Overlap Similarity only considers the items for which user has expressed preference by setting the corresponding bit in the preference vector to 1. Under Aggregated Voting, the recommendation task becomes determining a *set* of items that maximizes the sum of Overlap Similarity for all group members (aggregated satisfaction). Overlap Similarity is *monotonic*, as adding an item to the recommended set only improves the Overlap Similarity value (thereby satisfaction). Due to this property, the optimal recommendation will always contain k items.

1) Generating Recommendations: Due to the monotonicity property, recommendation computation can progress in an iterative, item-by-item manner. We describe our proposed algorithm **R-AGS** that computes an optimal recommendation in $O(mn)$ time.

Algorithm R-AGS: Given the feedback vectors of all users in the group, **R-AGS** associates a score for each item $i_j \in \mathcal{I}$. The score of an item i_j is defined as the number of users in the group who have expressed preference for i_j. After this step, **R-AGS** chooses the k-items with the highest scores (i.e. the top-k items preferred by the most users) to be presented as recommendation $\mathcal{I}^k$. Notice that ties are broken arbitrarily as each choice still results in the same value of overall group satisfaction.

Consider the scenario described in Example 2. The set of items recommended by **R-AGS** (where $k = 3$) (considering $feedback(u_1), feedback(u_2), pref(u_3)$) are : $\{i_1, i_4, i_5\}$. As we can see, each of the chosen POIs is desired by more users than the ones not picked. In other words, **R-AGS** computes recommendation vector $\mathcal{I}^k$ as 10011 for the group.

2) Generating New Feedback Vector: We now describe the process for generating feedback vector using Example 2, where user u_3 uses the feedback box. The task for feedback box is to generate new $feedback(u_3)$ (potentially different from $pref(u_3)$) such that $\mathcal{S}_{u_3}(pref(u_3), \mathcal{I}''^k)$ is maximized, where

$$\mathcal{I}''^k = \mathcal{F}(feedback(u_1), feedback(u_2), feedback(u_3))$$

$\mathcal{I}''^k$ is 10011 when feedbacks of users u_1, u_2, u_3 is considered. Observe that any update of preferences of a user (u_3 here) only affects the possibility of a subset of these 5 items to be included (or excluded) to the new recommendation: items that have $j, j+1, j-1$ scores (where j is the aggregated score of the k-th item without the user who updated her preferences). Let $\mathcal{X}$ be that set. For example, i_1 will be present in the generated recommendation, irrespective of whether u_3 prefers it or not. Let the items that are set to 1 in $pref(u_3)$ be denoted by $\mathcal{Y}$. Observe that u_3's satisfaction ($\mathcal{S}_{u_3}$) could be maximized as long as the generated $feedback(u_3)$ contains those items that are present in $\{\mathcal{X} \cap \mathcal{Y}\}$. Thus, the following lemma holds:

Lemma 1. *Given a user u, items in $feedback(u)$ are always a subset of items in $pref(u)$.*

Furthermore, observe that the presence/absence of the additional items $\{\mathcal{Y} - \mathcal{X}\}$ in $feedback(u)$ does not change $\mathcal{S}_u$, simply because the feedback of u alone is insufficient to promote them to the generated recommendation. An exception to this happens when the lowest aggregated score of an item in $\{\mathcal{X} \cap \mathcal{Y}\}$ is 1, then the additional $\{\mathcal{Y} - \mathcal{X}\}$ items should

also be included in $feedback(u)$ (to exploit the benefit of ties) [5] . In either case, the following theorem holds:

Theorem 1. *$feedback(u)$ will be at most as useful[6] as $pref(u)$.*

Thus, we conclude that the feedback box is not useful when the consensus function is aggregated voting and satisfaction measure is Overlap Similarity.

B. Hamming Distance

Unlike Overlap Similarity, Hamming Distance is measured by considering both 0 and 1 preferences. Unlike Overlap Similarity, Hamming Distance does not satisfy monotonicity property: suppose there exists only one $pref(u)$ and any given $\mathcal{I}^k$. Based on that, $\mathcal{S}_u$ could be computed. Now, if we arbitrarily set one of the 0-bits in $\mathcal{I}^k$ to 1, that may decrease $\mathcal{S}_u$ (as the Hamming Distance may now be even larger). For the same reason, an optimal $\mathcal{I}^k$ may not always contain k 1's. This precludes the iterative approach and makes recommendation and feedback generation to be substantially challenging.

1) Generating Recommendations: The optimal recommendation vector maximizes the aggregated group satisfaction by minimizing the aggregated Hamming Distances between the respective feedback vectors and the generated recommendation. Since Hamming Distance could be expressed in L_2 measure, this definition is equivalent to the *Facility Location* [10] or *Geometric Median Finding* [11] problems. Unfortunately, this problem and several of its variants have shown to be NP-hard [12], [13].

Consider a slightly different variant of this classical definition, where the task is to minimize the aggregated square of the distance (as compared to the aggregate distance). The new objective function is still natural and minimizes the aggregate distance. This variant is equivalent to the geometric problem of finding the *centroid* of a set of points. Unlike geometric 1-median finding problem [14], finding centroid of a set of points is computationally much easier in geometric settings, as each coordinate of the centroid could simply be expressed as the average of the samples of that coordinate. We adopt this latter definition for our problem, as it enables us to design efficient algorithm while still producing a recommendation that is meaningful. Formally, the task is to,

$$\text{Compute } \mathcal{I}^k \text{ s.t. } \Sigma_{\forall u \in \mathcal{G}} Hamming(\mathcal{I}^k, feedback(u))^2 \text{ is minimized.}$$

Even after we adopt the centroid definition, the generated recommendation vector needs to be *Boolean* in our case. While efficient solutions could be designed when there are only 2 users, the integrality constraint gives rise to significant computational challenges, for general n and m.

Theorem 2. *The decision version of the $\mathcal{I}^k$ generation problem is NP-complete.*

[5]In the given example $feedback(u_3)$ is equal to $pref(u_3)$.

[6]Usefulness denotes the improvement in user satisfaction.

Algorithm 1 Subroutine Centroid

Input: $feedback(u_1), feedback(u_2) \dots feedback(u_n)$;
Output: Centroid $\mathcal{C}$: a vector of size m

1: $\mathcal{C}_j = \frac{\Sigma_{i=1}^{n}(feedback(u_i^j))}{n}, \forall j \in m$;
2: **return** $\mathcal{C}$;

We omit the details for brevity, and note that our proof uses a reduction from SAT [15], similar to Theorem 5 .

Algorithm R-AGD: The basic idea of this solution is to relax the integrality constraint, and then solve it as a centroid finding problem in geometric settings using Algorithm 1. For the j-th item, $\mathcal{C}_j$ is computed as the average of all the feedback vectors in the group. This task overall takes $O(m \times n)$ time. Once the centroid is obtained (of length m), we perform a deterministic rounding; In other words, recommendation bit for the j-th item is set to 1, if $\mathcal{C}_j >= 0.5$; otherwise, it is set to 0. The centroid $\mathcal{C}$ becomes the recommendation vector $\mathcal{I}^k$ after rounding. However, such a rounding that is oblivious to values of other items could result in a vector that has more than k ones. In such a case, we arbitrarily choose k of those 1's, and set the remaining bits to 0. This yields the final $\mathcal{I}^k$. We note that this rounding process may introduce an approximation in the computed $\mathcal{I}^k$. We leave the theoretical analysis of the approximation factor to future work, but evaluate it empirically in Section VII.

Considering $feedback(u_1), feedback(u_2), pref(u_3)$ of Example 2, Algorithm **R-AGD** turns the following 2 items to 1 for the group in the generated $\mathcal{I}^k$: $\{i_1, i_4\}$; remaining 3 items are all set to 0.

2) Generating New Feedback Vector: Recall from the previous subsection, that the optimal recommendation is ($\mathcal{I}^k$) 10010, when the feedback box is not used. $\mathcal{S}_{u_3}$ is 2 here, as the Hamming Distance between u_3 and $\mathcal{I}^k$ is 2. But if $feedback(u_3) = 01111$ is used instead, the generated recommendation $\mathcal{I}''^k$ will be forced to include one of the items that u_3 prefers; i.e., one optimal generated recommendation $\mathcal{I}''^k$ will be 01011. Note that now $\mathcal{S}_{u_3}$ is improved further, since the Hamming Distance between $pref(u_3)$ and $\mathcal{I}''^k$ is just 1 now.

Lemma 2. *Given a user u, there exists a $feedback(u)$ that will be at least as useful as $pref(u)$.*

Algorithm FB-AGD: Computing an optimal $feedback(u)$ for the user who has updated her profile is NP-hard as the recommendation subroutine it uses is itself NP-hard (see Section III-B1). Our proposed solution **FB-AGD** works as follows: Let u be the user who has a new preference. Then we formulate $feedback(u)$ computation as an optimization problem, such that the objective function maximizes $\mathcal{S}_u$ (by minimizing the Hamming Distance between $pref(u)$ and the generated recommendation). Each of the variables (items) in $feedback(u)$ are required to be between 0 and 1 (but not

necessarily integers) [7]. After that, it performs deterministic rounding, and obtain a Boolean $feedback(u_4)$. Formally, the task is to

$$\begin{aligned} \text{minimize} \quad & Hamming(pref(u), \mathcal{I}^k) \\ \text{subject to} \quad & |feedback(u)| = m \\ & 0 \leq \forall i \in feedback(u) \leq 1 \\ & \mathcal{I}_k = \mathcal{F}(feedback(u_1), \ldots, feedback(u)) \end{aligned}$$

During the computation of $feedback(u)$, Algorithm **FB-AGD** uses Subroutine 1 for generating $\mathcal{I}^k$'s.

Output of this quadratic optimization problem assigns a value (between 0 and 1) to each of the m variables (items) of $feedback(u)$; [8]. Since the final output needs to be Boolean, deterministic rounding is performed that transforms any item with fractional value ≥ 0.5 to 1, and the remaining to 0. This yields our final $feedback(u)$. Note that this rounding process may introduce approximations in the final $feedback(u)$; The theoretical analysis of the approximation factor is left to future work, while we experimentally evaluate these factors in Section VII.

IV. Least Misery Consensus

We next discuss algorithms for computing recommendation and feedback vector for Least Misery (described in Section II-B3), under Overlap Similarity and Hamming Distance. Recommendation robustness could be ensured following techniques described in subsection VI.

We describe our running example in Example 3 for this section. For each similarity measure, the recommendation algorithm is first described considering the updated preference of the user without feedback box, followed by the discussion considering feedback box in conjunction.

Example 3. *Group $\mathcal{G}$ consists of 3(n) travelers u_1, u_2, u_3 to visit at most 3(k) different places. An ordered list of possible 5(m) items (i.e., places) are $\{i_1, i_2, i_3, i_4, i_5\}$. u_3 updates her preferences to $pref(u_3) = 00111$ (meaning, she is now interested in i_3, i_4, i_5). The feedback vectors of rest two travelers (obtained after running their individual preferences through feedback box) are as follows: $feedback(u_1) = 10110$, $feedback(u_2) = 01110$.*

A. Overlap Similarity

1) Generating Recommendations: We are interested to compute $\mathcal{I}^k$ such that the minimum satisfaction of the group is maximized (i.e., the minimum Overlap Similarity between the generated recommendation and the individual feedback vector (or preference vector) is maximized).

When there are only two users in the group, computing $\mathcal{I}^k$ is simple. Intuitively, the task is to consider the items where the user's respective feedback vector differs. The algorithm splits those items in two equal halves, and the generated recommendation sets the items of each of these halves according to the feedback vector of one user. This simple computation could be done in $O(m)$ time. However, this simple process fails to extend for a general n and m.

Theorem 3. *The decision version of the $\mathcal{I}^k$ generation problem is NP-complete.*

Proof: The decision version of the problem of recommendation generation is as follows: For a given set of feedback vectors $(feedback(u_1), feedback(u_2) \ldots feedback(u_n))$, defined over a set of m items, is there a recommendation vector $\mathcal{I}^k$ such that the Least Misery value is 1. The membership of the decision version of the recommendation generation problem in NP is obvious. To verify its NP-completeness, we reduce the Hitting Set(U, S) [15] problem to an instance of our problem.

We consider an instance of Hitting Set(U, S) [15]; we are required to construct an instance of recommendation problem, where each set represents a feedback vector $feedback(u_i)$, and the task is to compute $\mathcal{I}^k$ such that the Least Misery is 1; such that there exists a Hitting Set of size k, covering each set in S, if and only if, a solution to our instance of recommendation problem exists (each 1-bit in $\mathcal{I}^k$ corresponds to an element of the Hitting Set). ■

Approximation Algorithm R-LMS: As the problem is NP-hard, we propose an efficient yet effective approximate solution that incrementally composes the recommendation vector in a greedy manner. Algorithm **R-LMS** initializes each item of $\mathcal{I}^k$ to 0 in the beginning. After that, it operates in k-iterations, where a single item is selected in each iteration, and the corresponding bit is set to 1 in the partially computed $\mathcal{I}^k$ according to one rule: at each iteration, it selects an item that is not yet set in $\mathcal{I}^k$ from that user who has the current Least Misery value (i.e., least overlap with the current $\mathcal{I}^k$). The algorithm terminates when exactly k bits of $\mathcal{I}^k$ are set to 1 in this process.

R-LMS generates $\mathcal{I}^k$ that sets the following 3 items to 1 in Example 3 for $feedback(u_1), feedback(u_2), pref(u_3)$: $\{i_1, i_3, i_4\}$.

Compared to its optimal counterpart, **R-LMS** is efficient, as $\mathcal{I}^k$ could be generated in $O(k \times n)$ time. Additionally, it has a provable approximation factor, as we discuss next.

Approximation Factor: We begin by describing our strategy for proving approximation guarantees. Consider an arbitrary function $f()$ that maps the subsets of a finite ground set U to non-negative real numbers[9] . We say that $f()$ is submodular [16], [17] if it satisfies the "diminishing-returns" property: the marginal gain from adding an item to a set S is at least as high as the marginal gain from adding the same element to a superset of S. Formally, a submodular function satisfies

[7]That task could be formulated as a convex optimization problem, where the objective function as well as the constraints could be expressed as positive definite matrices; this form of quadratic programming admits a polynomial time solution.

[8]Integrality constraints are relaxed because integer programming is NP-hard.

[9]Overlap Similarity satisfies this form: It maps each subset of the item-set $\mathcal{I}$ to a real-number in $\mathcal{I}^k$ denoting the Overlap Similarity if that subset is included in $\mathcal{I}^k$.

$$f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$$

for all elements v, and all pairs of set $S \subseteq T$.

Lemma 3. *Algorithm* **R-LMS** *approximates the Least Misery of the generated $\mathcal{I}^k$ within $(1 - \frac{1}{e})$ factor of the optimal solution.*

Proof: Overlap Similarity is monotonic, and could be proved to satisfy the "diminishing-returns" property. We omit the details for brevity. Algorithm **R-LMS** is akin to the greedy strategy described above [17]. Hence, it admits the above approximation factor. ∎

2) Generating New Feedback Vector: With updated $pref(u_3)$ of Example 3, recall a possible $\mathcal{I}^k = 10110$. Note that, $\mathcal{S}_{u_3}$ is 2 (as u_3's desired i_3 and i_4 items are present in $\mathcal{I}^k$). Instead of that, if $feedback(u_3) = 00001$ is used as an input along with $feedback(u_1), feedback(u_2)$, then a possible $\mathcal{I}''^k = 00111$. Note, now, u_3 is fully satisfied with the generated $\mathcal{I}''^k$ (it contains all 3 of her preferred items). Therefore, the following lemma holds.

Lemma 4. *Given a user u, there exists a $feedback(u)$ that will be at least as useful as $pref(u)$.*

The above example leaves some intuition for designing $feedback(u_3)$. Observe that even when u_3 is not considered at all, any recommendation considering users u_1, u_2 has to have items i_3 and i_4 in it. Intuitively, these items should not be present in $feedback(u_4)$. The idea is to set only those items that are exclusively preferred by u_3, and compose $feedback(u_3)$ such that the recommendation function is forced to select some of those items. This intuition is formalized further in our proposed algorithm **FB-LMS**.

Optimal Algorithm FB-LMS: Without loss of generality, let $u_1, u_2, u_3, \ldots, u_n$ be the users in a group, where u_i has a preference update. Let $\mathcal{X}$ be the set of k items that are set to 1 in the generated recommendation (i.e., in $\mathcal{I}'^k$) for $\mathcal{G}$, without considering u_i. Let $\mathcal{Y}$ be the items that are present in u_i's latest preference. Algorithm **FB-LMS** generates that $feedback(u_i)$ which only sets $\{\mathcal{Y} - \mathcal{X}\}$ items to 1.

Note that the feedback box is an NP-hard problem, since its recommendation counterpart is shown to be NP-hard above. However, if an oracle for recommendation computation is assumed, we claim that this surprisingly simple set difference based algorithm generates optimal feedback vector for the latest preference of the user, as we shall prove next.

Lemma 5. *Let $\{\mathcal{Y} \cap \mathcal{X}\}$ be the subset of the items that are present in one of the optimal recommendation vectors ($\mathcal{I}'^k$) for $\mathcal{G}$ (without u_i). With an updated $pref(u_i)$, $\{\mathcal{Y} \cap \mathcal{X}\}$ items will still be present in one of the new optimal recommendation vector ($\mathcal{I}^k$) .*

Proof: We prove the above lemma by contradiction. Let us assume that such a recommendation vector $\mathcal{I}^k$ does not exist. This means that the set of $\{\mathcal{Y} \cap \mathcal{X}\}$ items are now replaced by a new set of $|\mathcal{Y} \cap \mathcal{X}|$ items. We note that such a substitution will only happen if $\{\mathcal{Y} \cap \mathcal{X}\}$ are not present in the updated $pref(u_i)$. That is a contradiction; since $\{\mathcal{Y} \cap \mathcal{X}\} \subset \mathcal{Y}$, and $\mathcal{Y}$ is present in the updated $pref(u_i)$. Therefore, one optimal $\mathcal{I}^k$ sets all the items in $\{\mathcal{Y} \cap \mathcal{X}\}$ to 1 . ∎

Given the above lemma, we can additionally prove:

- It is not useful to set the items in $\{\mathcal{Y} \cap \mathcal{X}\}$ to 1 in $feedback(u_i)$.
- $\mathcal{S}_{u_i}$ can only increase if the items in $\{\mathcal{Y} - \mathcal{X}\}$ are included in $feedback(u_i)$.
- $\mathcal{S}_{u_i}$ is only higher for $feedback(u_i)$ that contains the items in $\{\mathcal{Y} - \mathcal{X}\}$, compared to any other feedback that contains its proper subset ($S \subseteq \{\mathcal{Y} - \mathcal{X}\}$).

Therefore, the following theorem holds.

Theorem 4. *Given a user u,* **FB-LMS** *generates optimal $feedback(u)$.*

Algorithm **FB-LMS** generates $feedback(u)$ in $O(m)$ time, if an oracle for recommendation computation is assumed.

B. Hamming Distance

1) Generating Recommendations: We are interested to compute $\mathcal{I}^k$ such that the minimum satisfaction of the group is maximized (i.e., the maximum Hamming Distance between the generated recommendation and the individual feedback vector (or preference vector) is minimized).

As Hamming Distance could be expressed in L_2 measure, under Least Misery, the task of recommendation computation is analogous to finding the *smallest enclosing ball* [18] in Computational Geometry. Similar to the case of Aggregated Voting, the geometric exact and approximate solutions [19] are inappropriate, as we study the problem in very high dimension [18], and intend to generate a *Boolean* vector. Again, the task is easy to solve for 2 users, but remains extremely complex for general n and m. Moreover, unlike the problem in Section IV-A1, it does not admit a greedy solution, as Hamming Distance does not satisfy the monotonicity property.

Theorem 5. *The decision version of the $\mathcal{I}^k$ generation problem is NP-complete.*

Proof: The decision version of the recommendation generation problem is as follows: For a given set of feedback vectors ($feedback(u_1), feedback(u_2) \ldots feedback(u_n)$), defined over a set of m items, is there a recommendation vector $\mathcal{I}^k$ such that the Least Misery (based on Hamming Distance) is $m - 1$. The membership of the decision version of the recommendation generation problem in NP is obvious. To verify its NP-completeness, we reduce SAT [15] to an instance of our problem.

We consider an instance of SAT [15] with m variables, where each clause contains all m variables, and the Boolean expression is a conjunction of n clauses. We are required to construct an instance of our recommendation problem where each clause represents a feedback vector $feedback(u_i)$ of one of the n users and the task is to generate an $\mathcal{I}^k$ that has the Least Misery (maximum Hamming Distance between $\mathcal{I}^k$ and the feedback vectors) value of $m - 1$; such that there

exists an assignment of the m variables which satisfies the Boolean expression, if and only if, a solution to our instance of recommendation problem exists. ■

Algorithm R-LMD: We design our proposed Algorithm **R-LMD** as an optimization problem, as the task could be translated in the form that admits polynomial time solution (convex optimization problem on positive definite matrices). Given n feedback vectors defined over m Boolean variables (items), the task is to compute $\mathcal{I}^k$, such that each variable in $\mathcal{I}^k$ is between $0-1$ (but not necessarily integers). Formally, the task is to,

$$\begin{aligned} \text{minimize} \quad & Max(Hamming_{\forall u \in \mathcal{G}}(\mathcal{I}^k, feedback(u)) \\ \text{subject to} \quad & 0 \leq i \ (\forall i \in \mathcal{I}^k) \ \leq 1 \\ & |\mathcal{I}^k| = m \\ & \mathcal{I}_k = \mathcal{F}(feedback(u_1), \ldots, feedback(u_n)); \end{aligned}$$

This quadratic equation is solved by a general purpose solver to obtain a $\mathcal{I}^k$, where each $i \in \mathcal{I}^k$ is a fractional value between $(0, 1)$. After that, we perform a deterministic rounding as explained in Section III-B1 such that the resultant $\mathcal{I}^k$ may at the most have k-1's.

Algorithm **R-LMD** sets the following 3 POIs to 1 in Example 3 for the group considering $feedback(u_1), feedback(u_2), pref(u_3) : \{i_1, i_3, i_4\}$. The rest of the items are all set to 0.

2) Generating New Feedback Vector: Recall if $pref(u_3)$ of Example 3 is used, an $\mathcal{I}_k$ will be 10110. If $feedback(u_3) = 00001$ is instead used, generated $\mathcal{I}''_k$ will be 00111. With $\mathcal{I}''_k$, u_3 is more satisfied.

The optimal feedback generation is an NP-hard problem, since its recommendation counterpart is proved to be NP-hard. Similar to the recommendation computation algorithm, we propose algorithm **FB-LMD** based on quadratic optimization (relaxing integrality constraints) which admits polynomial time solution. However, there are non-trivial challenges to do so, as we describe next:

Algorithm FB-LMD: Without loss of generality, given the updated preference of user a u, the task is to generate $feedback(u)$ that maximizes $\mathcal{S}_u$ (by minimizing Hamming Distance). Note that the optimizer may need to compute multiple $\mathcal{I}^k$ in this process, where computing $\mathcal{I}^k$ itself is an optimization problem. The objective function admits only one optimization task, and the rest is expressed as constraints. This requires careful formulation, as the recommendation computation now needs to be expressed as constraints, without compromising its correctness. The problem is formalized as follows:

$$\begin{aligned} \text{minimize} \quad & d = Hamming(\mathcal{I}^k, pref(u)) \\ \text{subject to} \quad & feedback(u, \mathcal{I}^k) \leq d, \forall u \in \mathcal{G} \\ & feedback(u, \mathcal{I}^k) \leq d \\ & 0 \leq i \leq 1, (\forall i \in feedback(u)) \\ & |feedback(u)| = m \end{aligned}$$

Note that, although the recommendation task is expressed as a set of constraints, with this careful formulation, the optimizer always produces a correct result; as the task now is to minimize the Hamming distance (corresponds to d in the formulation), and in turn generate recommendation where the Least Misery is not larger than that distance.

The optimizer generates $feedback(u)$ with m variables, where each variable may be a fraction between 0 and 1. Deterministic rounding is then performed as we have done in Section III-B1. This yields the final $feedback(u)$. We note that the rounding process may introduce approximation in the final result. The empirical analysis of the approximation factor is studied in Section VII, while the theoretical analysis is deferred to future work.

V. Non-Boolean Preference Models

In this section, we generalize the user preference model and describe how our algorithms can be adapted. Thus far, user preferences were expressed as a Boolean vector where 1 corresponds to the items she prefers to consume. Two natural alternatives are numeric or *ordinal* preference models. Under numeric preference model, the user expresses her preference for an item as a real number between 0 and 1, where 1 represents the highest preference. Under an ordinal preference model, user expresses her preference over items through a discrete set of values (such as not liked, neutral, liked, very liked). However, in contrast to categorical values, there exist an ordering between preferences. A user would prefer an item with neutral preference over one with not liked preference.

Both the user preference vector and the feedback vector are expressed using the non-Boolean preference model. For both cases, user satisfaction with the generated recommendation needs to be measured considering generalized distance measures (e.g., Euclidean distance). The group recommendation functions, aggregated voting and least misery, can naturally be extended to this setting.

Generating group recommendation for numeric user preferences have been studied previously [3], [1]. Unfortunately, when the user preferences are expressed in ordinal scale, the problem becomes NP-hard. However, our recommendation algorithms for Boolean preferences can be easily extended to generate group recommendations. Similarly, the feedback generation algorithms can easily be extended to non-Boolean preference models. For numeric preferences, our optimization formulation is still valid. However, the integrality constraints that necessitated user preferences to be Boolean could be relaxed. The objective function with the altered constraints defines a convex quadratic optimization problem for which solutions can be computed efficiently in polynomial time. However, the computational complexity remains unchanged for ordinal preference model, since the generated feedback vector needs to select one of the discrete values from its ordinal domain. Our proposed algorithms can be adapted by treating the ordinal scale as numeric and post processing the solution back to ordinal scale.

VI. Recommendation Robustness

After a dynamic update of user preference, the feedback box followed by recommendation generation process needs to be re-invoked, which may generate a different recommendation vector. In this section, we discuss recommendation *robustness* to ensure that the generated recommendations are not *too different* before and after one (or a small number of) user preference updates.

Robustness is considered as a soft constraint that could further be tuned. Alternatively, one may also think of it as a hard constraint, where the task would be to generate that recommendation, which not only optimizes group satisfaction, but also contains the highest similarity with the previous recommendation. Expressing robustness as a hard-constraint has several shortcomings: first, one has to enumerate all possible optimal recommendations (as there could be multiple optimal solutions), which is prohibitively expensive to compute; second, the proposed techniques fall short, and the theoretical results do not hold under such settings. Conversely, we easily adapt robustness to our solution framework as a soft tunable constraint. The primary idea is to add previously generated recommendation(s) as new feedback vector(s) (as pseudo-users) to the recommendation function. Not only that applications that desire high degree of robustness, may potentially replicate the same recommendation vector several times, and input all of them as multiple feedback vectors to the recommendation function. This should achieve higher degree of robustness, as we shall see in our experimental study. All our theoretical results are extended under this setting.

VII. Experiments

We conduct a comprehensive set of performance and quality experiments using real world datasets extracted from Lonely Planet, Flickr, and MovieLens[10]. Our prototype system is implemented in C++ using IBM CPLEX as the solver for ILP and QP formulations. All experiments are conducted on an AMD machine quad-core 2.0GHz CPUs, 8GB Memory, and 1TB HDD, running Ubuntu 12.10. All numbers are obtained as the average of five runs.

A. Summary of Experimental Results

For brevity, we present scalability results conducted on the large dataset (MovieLens, 10M data corpus), and quality results conducted on the relatively smaller dataset (Lonely Planet and Flickr). The omitted results are similar to the ones depicted. The usefulness of the feedback box is validated using extensive quality and user-study experiments where we record the instances where one of the group members updates her preferences necessitating the invocation of feedback box and subsequent recommendation generation.

Our primary observations are as follows: a) the presence of a feedback box is deemed *always useful*. The degree of usefulness varies; it is most useful for Least Misery recommendation. Furthermore, the presence of a feedback box is critically important for groups formed by users with similar preferences. b) Our group consensus functions and Boolean preference measures are enthusiastically preferred by human evaluators. c) Finally, our proposed robustness technique is both effective and efficient in handling changing preferences in group recommendation.

[10]http://www.lonelyplanet.com/, http://www.flickr.com/services/api/, http://www.grouplens.org/node/73

B. Data Preparation

City Names and POI Generation: We consider 12 popular tourist destinations (cities) and their POIs (Points of Interest). Popular POIs of those cities are extracted using *Lonely Planet* dataset. *London, New York, Barcelona, Bangkok, Amsterdam* are some example cities we consider. The number of POIs per city varies between 35 and 163.

User Preferences for Travel dataset: We use publicly available Flickr photos to simulate Boolean preference vectors for users. They are tagged with corresponding POI names, and the respective date/time associated with the photos define the itineraries (such as, a set of POIs visited on the same day). Given a Flickr log of a city, each row in that log corresponds to a user itinerary that is visited in a 12-hour window. The set of POIs present in each itinerary constitutes a $pref(u)$.

User Preferences for MovieLens dataset: The MovieLens dataset contains 10 million movie ratings (from 0.5 to 5) provided by 71567 users, over 10681 movies. We adopt a simple procedure to convert numerical ratings to Boolean preferences: a rating smaller than 3 is transformed to a 0, or transformed to a 1, otherwise. A rating of 3 is considered neutral, and is excluded from further considerations. Given a set of movies $\mathcal{I}$ (i.e., items), the Boolean preference vector $pref(u)$ is thus generated for each user. A movie not rated by user u is treated as 0 in $pref(u)$.

C. Performance Experiments

We report efficiency results for the recommendation algorithm and feedback boxes under different user satisfaction measures and group consensus functions using MovieLens dataset. Performance is recorded by mainly varying 3 parameters - number of items in the generated recommendation (k), group size (n), and total number of items (m). The total running time of the system is the aggregated running time of the recommendation and feedback box.

1) Recommendation Generation: As a straw-man competitor, a brute-force algorithm (referred to as Brute-Force) is also implemented. This enumerates and evaluates all possible combinations before selecting the best answer.

Varying Budget k: We study the running time of different recommendation algorithms by varying k, the number of recommended items at each iteration. We fix n at 5000, and m at 5000. Figure 2 shows the output of this experiment. Brute-Force quickly becomes very expensive even for small values of $k(< 5)$, therefore, we use secondary Y-axis for that in minute scale. Our algorithms are very efficient (scale linearly with k) and are measured in seconds using the primary Y-axis.

Hamming Distance-based algorithms, exhibit higher running times than their Overlap Similarity counterparts.

Varying Group Size n: In this experiment we set $k = 100$, $m = 5000$ and compute recommendation for different group sizes. Figure 3 shows the result. While the running time increases with group size, this increase is much slower with increasing values of n. This is due to the fact that algorithm complexity depends on m.

Varying Total Number of Items m : Here, we set k = 100, $n = 5000$. Figure 4 shows the result. We fail to report the performance of the brute force algorithm as it takes hours to execute. The running time of **R-AGD** stays almost constant. The performance of other recommendation algorithms show a steeper increase in comparison, as additional items exponentially increase the number of potential candidate solutions.

2) Feedback Box: **Varying Budget k:** Our empirical study shows that the running time of the feedback box remains almost unchanged for all the algorithms with varying k. For brevity, we omit those results.

Varying Group Size n : In this experiment, k is set to 100, and $m = 5000$. Figure 5 shows the result. Run-time of **FB-LMS** remains constant with varying n because this algorithm relies on set difference that depends only on m. Both **FB-LMD** and **FB-AGD** are based on Quadratic Programming, and demonstrate similar performance. Note that group size bears significant impact on the running time of the recommendation algorithms, unlike its feedback box counterpart, where the impact is minimal.

Varying Total Number of Items m : Here $k = 100$, and $n = 5000$. Figure 6 shows that while **FB-LMS** scales linearly with increasing m, QP based algorithms **FB-LMD** and **FB-AGD** have a steeper increase in running time.

D. Quality Experiments

1) Offline Quality Experiments: We qualitatively evaluate how feedback box improves the satisfaction of the user (with updated preference) in the generated recommendation, compared to her (latest) preference.

User Satisfaction with feedback box: The objective is to verify if $\mathcal{S}_u$ is improved in the presence of a feedback box. Parameters are set at $m = 125, k = 20, n = 75$. The usefulness of a feedback box is evaluated by varying pairwise similarity between users in a group. The similarity of a group is computed as the average pairwise Jaccard Index [8] of the preference vectors. The group similarity varies between 0.1 and 0.5. Given a u, $\Delta\mathcal{S}_u$ is measured in Y-axis with and without a feedback box. Figure 8 shows the result.

Under both preference measures, $\mathcal{S}_u$ is higher when a feedback box is used. However, the degree of usefulness is maximum for groups with similar members and Least Misery consensus function; this observation is intuitive, as it is expected that the feedback box would suggest a more effective $feedback(u)$, when existing group members are similar to each other. The usefulness of the feedback box however is minimal under Aggregated Voting; this is also intuitive, as Aggregated Voting is less sensitive to individual user satisfactions compared to Least Misery.

Group Size Vs User Satisfaction: In this experiment, we evaluate the change in $\mathcal{S}_u$ with increasing n. Figure 7 shows the result, where $k = 20$, and $m = 75$. Overall, we observe that user satisfaction decreases as the group size increases. The drop is mildest under Aggregated Voting. This is due to the fact that in order to replace an item in the recommendation with a different one (chosen from the items in user's new preference), a *majority* of the existing users would have to prefer it. The decrease is steepest for Least Misery, as it tries to satisfy these new users with increasing group size, causing $\mathcal{S}_u$ to further decrease.

Empirical Evaluations of Approximation Factor: Recall that algorithms **R-AGD** and **R-LMD** for recommendation generation and **FB-AGD** and **FB-LMD** for feedback box generation are approximate in nature. Each of them involves solving the optimization problem by relaxing integrality constraints and then performing deterministic rounding. We compare the quality of results generated by the optimal brute-force algorithm with that of our approximate algorithms. Due to very high time complexity of brute-force, we are only able to run these algorithms for small values of k and m. We notice that our designed approximate algorithms routinely achieve approximation factor as high as 90%.

2) Online User Study: We now describe our user study performed through the crowd-sourcing platform Amazon Mechanical Turk using Flickr dataset.The goal of this study is twofold: first, investigate improvement in user satisfaction in presence of a feedback box; second: analyze user's affinity to different group recommendation functions.

TABLE I
USER STUDY STATISTICS

#Cities	#Total POIs	#Users	#HITs	#Worker/HIT
3	439	270	54	5

TABLE II
USER STUDY: USER SATISFACTION IN GENERATED RECOMMENDATION WITH AND WITHOUT RESPONSE BOX

	FB-LMD	R-LMD	FB-LMS	R-LMS	R-AGD	R-AGS
Si(S)	5%	0%	65%	30%	0%	0%
D(S)	0%	0%	40%	35%	0%	25%
R(S)	10%	5%	62%	10%	3%	10%
Si(M)	20%	0%	40%	0%	5%	35%
D(M)	11%	2%	35%	7%	3%	42%
R(M)	1%	1%	38%	20%	0%	40%

Required statistics are described in Table I. We only consider reliable user feedback, i.e., those users who satisfactorily pass qualifying test to check if they are familiar with a city. We choose 3 cities (London, San Francisco, and Barcelona) and their associated POIs for this study. We consider two different group sizes - small(S) groups with 5 users, and medium(M) groups with 15 users. k is set to 10. For each city and group size combination, we created a HIT (Human Intelligence Task) for similar(Si) (aggregated pair-wise similarity ≥ 0.4), dissimilar(D) (≤ 0.06), and random(R) user groups.

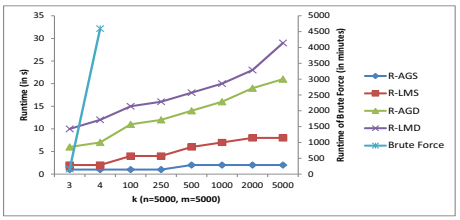

Fig. 2. Recommendation: k Vs Runtime

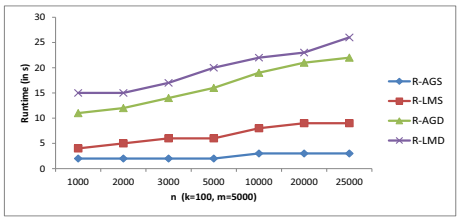

Fig. 3. Recommendation: n Vs Runtime

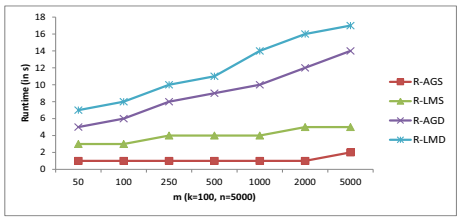

Fig. 4. Recommendation: m Vs Runtime

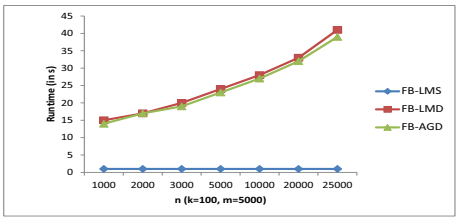

Fig. 5. Feedback box : n Vs Runtime

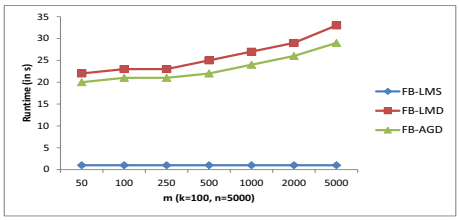

Fig. 6. Feedback box : m Vs Runtime

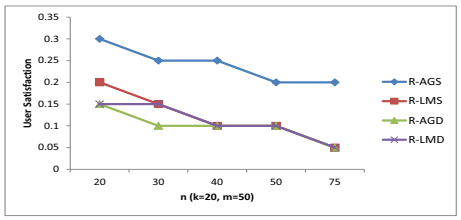

Fig. 7. Feedback box: n Vs User Satisfaction for Similar groups

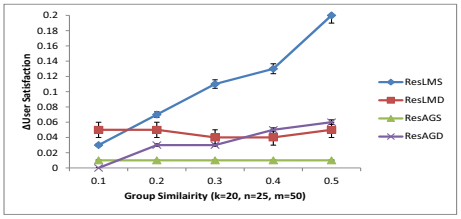

Fig. 8. Feedback box: Group Similarity Vs User Satisfaction

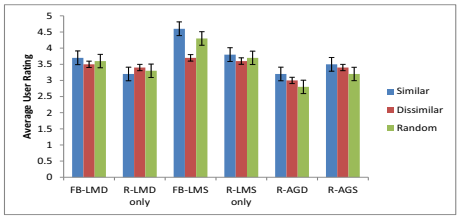

Fig. 9. User Study: User Rating for recommendations (Small group)

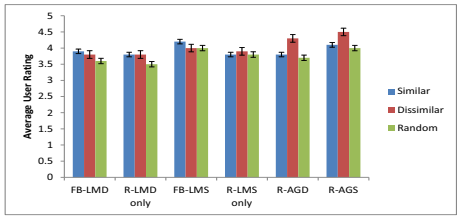

Fig. 10. User Study: User Rating for recommendations (Medium group)

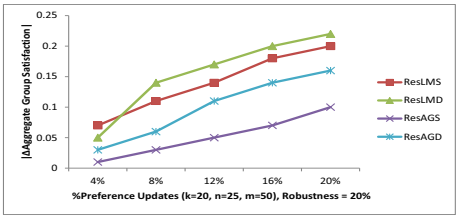

Fig. 11. Robustness: Percentage of Preference Updates Vs Group Satisfaction

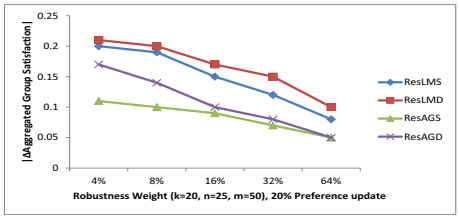

Fig. 12. Robustness: Weight Vs Group Satisfaction

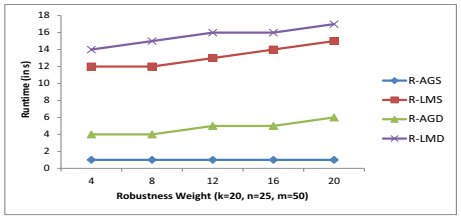

Fig. 13. Robustness: Weight Vs Runtime

HIT Design: We designate one of the users in the group as the user who updates her preferences and use her recent preferences for the experiment. We generate group recommendation for the entire group for each combination of recommendation function and user satisfaction, with and without the presence of feedback box;. For Aggregated Voting, we only consider the recommendation generation task, as Figure 8 clearly shows that the feedback box has the least effect in improving user satisfaction. A worker of the HIT is asked to evaluate the group recommendation as if she were the designated user who updates her preference. In addition to her own preference, she has also access to the preferences of other members of the group. We ask the worker to perform two tasks. In the first, she is asked to *independently* evaluate the output of each recommendation function (Figures 9,10) with ratings $1-5$ (5- most satisfied, 1-least satisfied) . In the second, she is asked to choose which recommendation is preferred by her in *comparison* with the rest. The worker is not told which method is used to generate which list. Percentage breakdown of worker satisfaction is shown in Table II.

Result Interpretation: For the first task (Figures 9,10), we observe that the workers in each of three different groups (similar, dissimilar, random) clearly prefer the recommendation generated using the feedback box, rather than the one without it. This behavior is consistent for both small and medium sized groups, but to different degrees. It could also be seen that feedback box seems to be more useful for similar and random groups, than dissimilar groups. This is consistent with our offline quality evaluations in Figure 8. For the second task, we observe that workers overwhelmingly prefer Overlap

Similarity with and without feedback box. It can be explained by the fact that it is a simpler satisfaction measure for users to understand than Hamming distance.

Satisfaction is higher in smaller groups. For medium groups, the two most popular recommendation functions are Least Misery under Overlap Similarity and Aggregated Voting under Overlap Similarity. This behavior can be explained as follows: Figure 7 shows that regardless of the recommendation function, user satisfaction decreases when group size increases. Furthermore, for a constant k, the effectiveness of the feedback box decreases with increased group size. Hence when the group size increases, the workers fall back to simpler recommendation functions such as Least Misery with Overlap Similarity and Aggregate Voting with Overlap Similarity.

E. Recommendation Robustness: Effectiveness and Runtime

Recommendation robustness is set as a soft constraint, which could further be tuned (see Section VI). For example, to achieve a robustness weight of 20% after the preference update of a user with $k = 20, m = 125, n = 75$, the previous recommendation needs to be added 15 times.

We experimentally demonstrate the effectiveness of robustness to counterbalance the effect of feedback box and ensure overall group satisfaction, when multiple users use feedback box. In Figure 11, X-axis varies the percentage of preference updates, with a fixed robustness weight of 20%, whereas, in Figure 12, we vary the robustness weight in X-axis with a fixed preference update of 20%. The Y-axis measures the difference in group satisfaction before and after the preference updates. Even though feedback box maximizes the preference of individuals who use it, but with appropriately designed robustness weights, the overall group satisfaction could be preserved significantly (the y-axis measures the absolute difference in group satisfaction between generated recommendation without feedback box, and recommendation with feedback box and robustness weight). Finally, as Figure 13 demonstrates, robustness is scalable as increasing robustness weight results in very small increase in runtime.

VIII. Related Work

Our work is the first to study flexible Boolean preferences in group recommendations with a novel *feedback box* idea.

Context Aware Recommendation: [6], [7] have shown that users' mood, context and company (other users) may affect their preferences. These works present a multidimensional approach to recommender systems that can provide recommendations based on additional contextual information (such as time, location and accompanying-people) besides the typical information on users and items. These works also support multiple dimensions, extensive profiling, and hierarchical aggregation of recommendations [20]. In contrast, we propose techniques to accommodate dynamic updates in an effective and efficient manner.

Group Recommendation: The task of group recommendation [4], [1], [21] has attained significant research attention in recent past. There are two prevalent approaches in group recommendation computation [3]: *virtual user* and *recommendation aggregation*. We adopt the latter for its flexibility. It has been also argued that the presence of appropriate group recommendation semantics is of paramount importance in successful group recommendation process. The objective of this research is not proposing new group recommendation semantics, rather, we leverage existing semantics in the analysis of the feedback box.

IX. Conclusion

We motivate the need for flexible user preferences in group recommendation and develop a *feedback box* that computes for each user with evolving preferences the best feedback to provide to maximize her satisfaction in the generated recommendation. We present robustness to counterbalance the effect of feedback box and ensure group satisfaction. We present a rigorous theoretical and empirical study that corroborates the usefulness of this box.

References

[1] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu, "Group recommendation: Semantics and efficiency," *PVLDB*, 2009.

[2] L. Boratto, S. Carta, A. Chessa, M. Agelli, and M. L. Clemente, "Group recommendation with automatic identification of users communities," in *Web Intelligence/IAT Workshops*, 2009, pp. 547–550.

[3] A. Jameson and B. Smyth, "Recommendation to groups," P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds., 2007.

[4] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl, "Polylens: a recommender system for groups of users," in *ECSCW*, 2001.

[5] D. P. Myatt, "On the theory of strategic voting," University of Oxford, Department of Economics, Economics Series Working Papers 186, 2004.

[6] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *TKDE*, vol. 17, no. 6, pp. 734–749, Jun. 2005.

[7] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin, "Incorporating contextual information in recommender systems using a multidimensional approach," *ACM Trans. Inf. Syst.*, Jan. 2005.

[8] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley, 1999.

[9] S. B. Roy, G. Das, S. Amer-Yahia, and C. Yu, "Interactive itinerary planning," in *ICDE*, 2011.

[10] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*, 1985.

[11] J.-H. Lin and J. S. Vitter, "Approximation algorithms for geometric median problems," 1992.

[12] D. Hochbaum, "Heuristics for the fixed cost median problem," *Math Programming*, vol. 22, pp. 148–162, 1982.

[13] G. Cornuejols, G. Nemhauser, and L. Wolsey, "The uncapacitated facility location problem," in *Discrete Location Theory*, 1990, pp. 119–171.

[14] R. Chandrasekaran and A. Tamir, "Algebraic optimization: The fermat-weber location problem," *Math. Program.*, vol. 46, pp. 219–224, 1990.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, 1990.

[16] S. Fujishige, *Submodular Functions and Optimization*. Elsevier, 2005.

[17] Nemhauser and et. al., "An analysis of approximations for maximizing submodular set functions–i," *Mathematical Programming*, 1978.

[18] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *Maurer, H., Ed., Cambridge, MA, USA*. Springer-Verlag, 1991, pp. 185–208.

[19] P. Kumar, J. S. B. Mitchell, and E. A. Yildirim, "Approximate minimum enclosing balls in high dimensions using core-sets," *ACM JEA*, 2003.

[20] K. Stefanidis, N. Shabib, K. Nørvåg, and J. Krogstie, "Contextual recommendations for groups," in *ER Workshops*, 2012, pp. 89–97.

[21] K. McCarthy, M. Salamó, L. Coyle, L. McGinty, B. Smyth, and P. Nixon, "Cats: A synchronous approach to collaborative group recommendation," in *FLAIRS Conference*, 2006, pp. 86–91.