Group Recommendation with Temporal Affinities

Sihem Amer-Yahia, Behrooz Omidvar-Tehrani, Senjuti Basu, Nafiseh Shabib

Group Recommendation with Temporal Affinities

Sihem Amer-Yahia[†1], Behrooz Omidvar-Tehrani[†‡1], Senjuti Basu Roy[‡2], Nafiseh Shabib[‡†3]

[†] *CNRS and LIG;* [†‡] *LIG;* [‡] *University of Washington Tacoma;* [‡†] *Norwegian University of Science and Technology*

[1]{sihem.ameryahia,behrooz.omidvar-tehrani}@imag.fr, [2]senjutib@uw.edu, [3]shabib@idi.ntnu.no

ABSTRACT

We examine the problem of recommending items to ad-hoc user groups. Group recommendation in collaborative rating datasets has received increased attention recently and has raised novel challenges. Different consensus functions that aggregate the ratings of group members with varying semantics ranging from least misery to pairwise disagreement, have been studied. In this paper, we explore a new dimension when computing group recommendations, that is, *affinity between group members* and *its evolution over time*. We extend existing group recommendation semantics to include temporal affinity in recommendations and design GRECA, an efficient algorithm that produces temporal affinity-aware recommendations for ad-hoc groups. We run extensive experiments that show substantial improvements in group recommendation quality when accounting for affinity while maintaining very good performance.

1. INTRODUCTION

Group recommendation refers to finding the best items that a set of users will appreciate together. It is an active research area as exemplified by numerous publications [3, 6, 13, 20, 23]. The main focus of existing work in group recommendation is the design of appropriate consensus functions that aggregate individual group members' preferences to reflect the group's preference for each item. A variety of functions have been used ranging from majority voting to least misery. In this paper, we are interested in exploring *how affinity between group members* and *its evolution over time* affect group recommendations. To the best of our knowledge, our work is the first to study *affinity and its evolution over time* in combination with existing group consensus functions.

The premise of this work relies on a simple conjecture that is, a user appreciates recommendations differently in the company of different people and at different times. When with girlfriends, a female user may want to watch a romantic movie that she may not want to watch with men. When with her parents, she may prefer to go to a nice Italian restaurant while she would prefer a burger joint with her kids. In addition, her appreciation of an item with the same group of people may change over time depending on how their connection and shared interests evolve. In other terms, *the affinity of a user with other group members* should be captured in how that user appreciates an item.

Previous studies on single user recommendation have shown that *contextual dimensions* such as a user's mood and company or time and place, may affect her preferences [1, 2]. Indeed, according to behavioral research studies [5, 14, 18], consumers use different decision-making strategies and favor different brands and products depending on their context. Such observations can be incorporated in different ways into single user recommendations. In [1], a multidimensional recommendation model is developed to account for contextual information into a user's recommendation, one of which could be her affinity with other users. However, to the best of our knowledge, multidimensional recommendation has not been applied to group recommendation. In group recommendation, we conjecture that each user will have a *relative preference for an item* depending on her affinity with other group members. Formalizing the semantics of relative preference raises two new challenges: *(i) how to account for user affinities in the definition of relative preference* and *(ii) how to combine relative preference with popular group recommendation consensus functions* [13].

A major difficulty when addressing *(i)* is to integrate the evolution of affinities between users over time. For example, interns at a research lab may subscribe to a Facebook group during their internship. When the internship period is over, the group becomes an alumni of the research lab and affinities between its members will likely change. Therefore, if events such as workshops or conferences are to be recommended to the alumni group in the future, affinities between its members should be accounted for, in order to decide which subgroup would be interested in which event. While numerous recent studies have shown the importance of accounting for time in recommender systems [8, 15, 17, 25], they have focused on user-item preferences and single-user recommendations. In this work, we propose two dynamic models to capture temporal affinities: a *discrete* model where time is discretized over a set of time periods and affinities computed for each sub-period, and a *continuous* model where time is represented as an exponential function that positively or negatively affects affinity over time. Both models have a static component that denotes how close two users are in a time-independent fashion and a dynamic component that captures the drift that the affinity of a user-pair exhibits compared to the overall user population. Finally, while the discrete model is an approximation of the continuous one, they are both used to capture increasing and decreasing affinities.

Clearly, combining user-item preferences of group members independently of each other to produce group recommendations is not enough to capture the impact of affinities on those recommendations. In other terms, applying the well-known group consensus functions such as aggregated voting, average preferences or least

10.5441/002/edbt.2015.37

misery on individual group members' preferences, does not capture a scenario where the same user appreciates the same item differently in different groups. Therefore, we propose a two-step approach to address *(ii)*. First, we modify individual user-item preferences on-the-fly to account for affinities and then we apply a group consensus function over the modified preferences. This approach has the benefit of dissociating recommendation computation from affinity computation and therefore being able to use relative preferences with any group recommendation consensus semantics.

No recommendation work would be complete without considering both recommendation effectiveness and efficiency. Those two dimensions raise new challenges when dealing with relative preference, namely, *(i) how to assess the quality of group recommendations?* and *(ii) how to efficiently compute affinity-aware recommendations on-the-fly for ad-hoc groups?* To address *(i)*, we build a Facebook application and generate movie recommendations using MovieLens dataset[1]. We leverage friendship and common page-likes to compute affinities and run an extensive set of experiments varying *group size*, *cohesiveness* (rating similarity between group members) and *affinity* between group members. To address *(ii)*, we develop GRECA, an algorithm that non-trivially adapts the family of threshold algorithms [10], to account for affinities between user pairs that evolve over time. GRECA leverages index structures that are extremely efficient with updates for maintaining time-variant affinities, and are used to efficiently produce the top-k recommended itemset for a group. In fact, as affinity between users evolves over time, GRECA does not need to recalculate any of the previously calculated affinities and just augments the index to account for the latest affinities. In addition to being instance optimal, the key novelty of GRECA is the use of a new buffer condition for termination, which constitutes a clear departure from traditional top-k style algorithms [10]. This condition simply implies that just by examining the items in the buffer, GRECA can terminate with the guarantee to have found the correct top-k itemset.

Our experiments consistently indicate that incorporating temporal affinities into group consensus functions is most effective for dissimilar user groups as well as low-affinity user groups whose preference significantly evolves over time. Prior work has shown that such groups generally take longer to reach consensus [20]. Our performance experiments demonstrate that GRECA achieves a save up of 75% or beyond in the number of accesses. These results strongly corroborate the effectiveness of our proposed solutions to include temporal affinities in group recommendation functions.

The paper makes the following technical contributions:

- We motivate the need to account for user affinities between group members when computing recommendations and propose to capture affinities in *the relative preference* of individual group members for each item. Relative preference modifies a user-item preference with the user's affinity with other group members.
- Since affinities may evolve over time, we propose two models, discrete and continuous, to represent (positive or negative) affinity drift of two users over time. This dynamic component is combined with a static component, that captures how close two users are in a time-independent fashion, in order to form temporal affinities.
- We extend group recommendation semantics, i.e., average preferences, least misery and pairwise disagreement, to include temporal affinities and design GRECA, an efficient algorithm that computes recommendations on-the-fly for ad-hoc groups. GRECA uses a new early termination condition to efficiently produce the top-k itemset for a group.
- We run extensive experiments using Facebook and MovieLens datasets and examine the impact of our temporal affinity model on group recommendation quality and efficiency.

[1] *http://movielens.umn.edu/*

The paper is organized as follows. Section 2 contains our formalism. GRECA, our recommendation algorithm is provided in Section 3. Extensive experimental evaluation is given in detail in Section 4. Related work is summarized in Section 5 and conclusion in Section 6.

2. DATA MODEL AND PROBLEM

We present a data model that captures temporal affinities and define our problem of recommending items to ad-hoc groups.

The underlying scenario that will be used to illustrate our model is a social network of individuals who have some intrinsic characteristics (e.g., birthplace, gender and age) and who express interests for items via likes and votes as in Facebook and Twitter. At any given point in time, we are interested in recommending content items (e.g., movies, books, conferences) to an ad-hoc group. Parts of this scenario will be used in this section and one instance will be described in specific details in Section 3.1.

In our model, we assume a set of m items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ and a set of n users $\mathcal{U} = \{u_1, \ldots, u_n\}$ out of which any ad-hoc group $\mathcal{G} \subseteq \mathcal{U}$ can be built. To simplify exposition, we will not formalize user or item attributes and will refer to them when needed in our example. We consider time as a set of consecutive timestamps that form periods. Each period p is a time interval of the form $[s, f]$ where s is its starting timestamp and f its ending timestamp.

2.1 Dynamic User Affinity Models

Affinity describes the *bonding* between a pair of users (u, u') and is denoted $aff(u, u')$. It could be as simple as explicit friendship or users in the same age group or more sophisticated such as users who like similar movies, have visited similar places and have friends who live in different parts of the world. For simplicity, we assume that affinity between a user pair is symmetric, i.e., $aff(u, u') = aff(u', u)$. More importantly, $aff(u, u')$ is dynamic and changes over time. We therefore compute affinity $aff(u, u', p)$ for a time period $p = [s, f]$. This dynamic affinity captures changes over time by combining its *static* and *dynamic* components defined below.

- *Static Affinity* - $aff_S(u, u')$: This is a time-agnostic affinity component and is used to capture how close two users are in a time-independent fashion. Stable factors such as birthplace, age, and education naturally contribute to this component. However, depending on the application, other dimensions could be accounted for. For example, Facebook friendship being stable, we use it to model static affinity in our experiments (Section 4.1.2).
- *Dynamic Affinity* - $aff_V(u, u', p)$: This is a time-variant component that captures affinity between two users u and u' during period p by considering how close they are during that period. For example, shared political interests, common likes, and shared interests for world events, vary over time and could contribute to formulating this component. Intuitively, the objective is to capture the *aggregated drift* that the affinity of a user pair exhibits for every time period from the beginning of time s_0 to the end of the current period $p = [s, f]$, compared to the overall user population.

More specifically, time starts at the beginning of time s_0 and is segmented into subsequent time periods $p_0, \ldots, p_{now}$ of varying lengths. Given two time periods $p_i = [s_i, f_i]$ and $p_j = [s_j, f_j]$, $p_i \leq p_j$ is used to denote that p_i precedes p_j, i.e., $s_i \leq s_j$ and $f_i \leq f_j$. Determining the right granularity of a time period depends on the application at hand and the frequency of user actions and is orthogonal to our model. For example, in a social network such as Facebook, and when affinities are computed using shared posts, granularity may vary from hours to days depending on the time of year. On Twitter, granularity is finer and may vary from minutes to hours since post frequency is higher. Not all time periods are of the same length.

Given a time period $p = [s, f]$, for every time period p' that is included in the interval starting at the beginning of time s_0 and ending at f, the end of p, the periodic affinity drift is calculated as a difference between the periodic affinity $\mathit{aff}^P(u, u', p')$ between users u and u' and the average periodic affinity $\mathit{Avgaff}^P(p')$ of the whole user population. These drifts are aggregated over all time periods included in the interval $[s_0, f]$ and normalized to generate $\mathit{aff}_V(u, u', p)$. Formally,

$$\mathit{aff}_V(u, u', p) = \frac{\Sigma_{p' \leq p}(\mathit{aff}^P(u, u', p') - \mathit{Avgaff}^P(p'))}{\Delta} \quad (1)$$

The exact formulation of Δ depends on how time is modeled (discrete or continuous) and is described. Interestingly, $\mathit{aff}_V(u, u', p)$ could be either positive or negative and depends on how the affinity of (u, u') evolves compared to the overall population.

The exact formulation of $\mathit{aff}^P(u, u', p')$ depends on the application. In our Facebook experiment in Section 4.1.2, we use common page likes between u and u' during period p'.

Finally, $\mathit{Avgaff}^P(p')$ is defined as follows:

$$\mathit{Avgaff}^P(p') = \frac{2 \times \Sigma_{(u,u') \in \mathcal{U}, u \neq u'} \mathit{aff}^P(u, u', p')}{|\mathcal{U}|^2 - |\mathcal{U}|}$$

We now describe our dynamic affinity models that use $\mathit{aff}_S(u, u')$ and $\mathit{aff}_V(u, u')$ as building blocks. The first model relies on discretized time periods to capture aff_V whereas the second represents time in a continuous fashion.

- *Discrete Dynamic Affinity Model:* In this model, the aff_S and aff_V affinity components are aggregated using a linear function over a set of discretized time periods. Therefore, Δ, the denominator in Equation 1, is simply the number of time periods between s_0, the beginning of time, and e, the end of p. This simple aggregation also allows us to design efficient algorithms.

  $\mathit{aff}^D(u, u', p) = \mathit{aff}_S(u, u') + \mathit{aff}_V(u, u', p)$

- *Continuous Dynamic Affinity Model:* For this model, time is considered in a continuous fashion. In this case, the denominator in Equation 1, $\Delta = f - s_0$ is the length of time between the the beginning of time s_0 and f, the end of p. As a natural representation to capture continuous time, we consider an exponential function, which is also supported in prior work [17]. Formally,

  $\mathit{aff}^C(u, u', p) = \mathit{aff}_S(u, u') \times e^{\lambda(f - s_0)}$

  Here λ is the rate of growth/decay of affinity and could simply be replaced by $\mathit{aff}_V(u, u', p)$ in Equation 1 to represent the cumulative effect of affinity drift over time.

Consequently, the discrete time model could be viewed as an approximation of the continuous one where time is discretized into sub-periods and each user pair's affinity is normalized over the number of periods (Equation 1). Alternatively, the continuous model treats time as a single interval $[f\text{-}s_0]$ and captures an exponential growth, resp., decay, affinity model when $\mathit{aff}_V(u, u', p)$ is positive, resp., negative.

In both $\mathit{aff}^D(u, u', p)$ and $\mathit{aff}^C(u, u', p)$ affinity drift could be negative or positive thereby capturing situations in practice where affinity between two users may increase or decrease over time. We believe that the ability to capture this varying rate of change is important in practice in particular for social networks where different users exhibit different interests over time.

2.2 User-Item Preference Models

We now show how affinities are accounted for in computing the preference of a user for an item in a group. We first describe how affinity is incorporated into user-item preferences without accounting for time then we show how to modify the formulation to compute time-aware user-item preferences.

Time-Agnostic User-Item Preference: Given a group $\mathcal{G}$, the preference of a user $u \in \mathcal{G}$ for an item $i \in \mathcal{I}$ is denoted $\mathit{pref}(u, i, \mathcal{G})$ and depends on two components:

- *Absolute preference* - $\mathit{apref}(u, i)$. This describes how much u likes item i akin to the predicted rating of u for i. Existing single-user recommendation algorithms, such as collaborative filtering, could be used to compute $\mathit{apref}(u, i)$.

- *Relative preference* - $\mathit{rpref}(u, i, \mathcal{G})$. This component captures that *a user likes an item i if close members in the group $\mathcal{G}$ also like i* and similarly that *a user dislikes an item i if close members in the group $\mathcal{G}$ dislike i*. Affinity between group members is used to capture how close they are. More formally, $\mathit{rpref}(u, i, \mathcal{G})$ combines the affinity of a user u with other members $u' \in \mathcal{G}$, denoted $\mathit{aff}(u, u')$, with the preference of u' for item i, denoted $\mathit{apref}(u', i)$.

  $$\mathit{rpref}(u, i, \mathcal{G}) = \Sigma_{\forall u' \neq u \in \mathcal{G}} \mathit{aff}(u, u') \times \mathit{apref}(u', i)$$

The overall affinity-aware user-item preference is a simple combination of these two factors: $\mathit{pref}(u, i) = \mathit{apref}(u, i) + \mathit{rpref}(u, i, \mathcal{G})$

Time-Aware User-Item Preference: We now modify the definition of relative preference to capture temporal affinities:

$$\mathit{rpref}(u, i, \mathcal{G}, p) = \Sigma_{\forall u' \neq u \in \mathcal{G}} \mathit{aff}(u, u', p) \times \mathit{apref}(u', i).$$

Therefore, a user u's overall preference on item i during time period p can be simply formulated as:

$$\mathit{pref}(u, i, \mathcal{G}, p) = \mathit{apref}(u, i) + \mathit{rpref}(u, i, \mathcal{G}, p)$$

2.3 Group Consensus Models

Members of a group may not always have the same preferences for items and a *consensus function* needs to aggregate user-item preferences into a single group's preference for an item. Intuitively, there are two main aspects in a consensus function [20]. First, the preference of a group for an item needs to *reflect the degree to which the item is preferred by all group members*. The more group members prefer an item, the higher its group preference. Second,

the group preference needs to *capture the level at which members disagree or agree with each other*. All other conditions being equal, an item that draws high agreement should have a higher score than an item with a lower overall group agreement. We call the first aspect *group preference* and the second aspect *group disagreement*. We revisit the definitions we introduced in [3] to include a time component.

Group Preference: The preference of an item i by a group $\mathcal{G}$ during a time period p, denoted $gpref(\mathcal{G}, i, p)$, is an aggregation over the preferences of each group member for that item. We consider two commonly used aggregation strategies:
Average Preference: $\frac{1}{|\mathcal{G}|}\sum_{u\in\mathcal{G}}(pref(u,i,\mathcal{G},p))$
Least-Misery Preference: $min_{u\in\mathcal{G}}(pref(u,i,\mathcal{G},p))$

Group Disagreement: The disagreement of a group $\mathcal{G}$ over an item i during a time period p, denoted $dis(\mathcal{G}, i, p)$, reflects the degree of consensus in the user-item preferences for i among group members over time. We revisit the two most common disagreement computation methods:

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

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

The average pair-wise disagreement function computes the average of pair-wise differences in preferences for the item among group members, while the variance disagreement function computes the mathematical variance of the preferences for the item among group members. Intuitively, the closer the preferences for i between users u and v, the lower their disagreement for i.

Time-Aware Group Consensus: We combine group preference and group disagreement in a time-aware consensus function, denoted $\mathcal{F}(\mathcal{G}, i, p)$. The function combines group preference and disagreement for an item i and a group $\mathcal{G}$ into a single group consensus score using the following formula: $\mathcal{F}(\mathcal{G},i,p) = w_1 \times gpref(\mathcal{G},i,p) + w_2 \times (1 - dis(\mathcal{G},i,p))$ where $w_1 + w_2 = 1.0$ and each specifies the relative importance of preference and disagreement in the overall group consensus.

Note that the formulation of group consensus incorporates temporal affinities by aggregating the relative user-item preferences of its members with disagreement. The proposed formulation is orthogonal to how affinities are modeled and incorporated into individual relative preferences. This way accounting for temporal affinities in group recommendation is orthogonal to the consensus function used to aggregate group members.

2.4 Problem Definition

Given a group $\mathcal{G}$, a time-aware consensus function $\mathcal{F}$ and an integer k, the objective is to recommend to $\mathcal{G}$ the k best itemset $\mathcal{I}_G$ that accounts for its members' affinities during a period p, such that:

- $|\mathcal{I}_\mathcal{G}| = k$
- $\forall i \in \mathcal{I}_\mathcal{G}, u \in \mathcal{G}$, i is not individually recommended to u
- $\nexists j \in \mathcal{I}$, s.t. $\mathcal{F}(\mathcal{G},j,p) > \mathcal{F}(\mathcal{G},i,p)$, where $j \notin \mathcal{I}_\mathcal{G}, i \in \mathcal{I}_\mathcal{G}$, i.e., there does not exist any other item j in $\mathcal{I}$ whose consensus score is higher than any item in i in $\mathcal{I}_\mathcal{G}$.

3. INSTANCE OPTIMAL ALGORITHMS

In this section, we discuss how to efficiently compute k affinity-aware recommendations for ad-hoc groups, meaning for groups that are not known beforehand. Recall that given a group $\mathcal{G}$, the goal, stated in Section 2, is to find the k best items to recommend to $\mathcal{G}$ according to a consensus function $\mathcal{F}$.

We propose instance optimal algorithms to compute top-k items for a given group under different group consensus functions. The overall intuition of this algorithm is appropriately adapted from the family of Fagin-style top-k algorithms [10]. These algorithms, such as, Threshold algorithms `TA` or No Random Access Algorithm `NRA`, rely on a function that aggregates multiple score components into a single score for each item. Those algorithms are used in Web search to compute the score of each item (a document in that case) as a combination of its component scores (its scores for each keyword in the search query). These algorithms aim to find the k items that rank the highest (the ones with the highest aggregated scores) in as little time as possible. They take sorted item lists that correspond to each component and scan them using *sequential* and *random accesses* (`SA`s and `RA`s), and the computation can be terminated without scanning the input lists fully, using stopping conditions based on score bounds (thresholds). Early stopping is possible when the ranking function is *monotone* [10].

LEMMA 1. *The temporal affinity-aware consensus function $\mathcal{F}$ is monotonic w.r.t. absolute preference lists and user-affinity lists for the dynamic user-affinity model, and pair-wise disagreement lists.*

PROOF. (sketch): In a prior work [3], we showed that all three group consensus functions without considering time-agnostic affinity (average preference, least misery and pair-wise disagreement) are monotone. If all group members, except a user u, rate items i_1 and i_2 the same, i_1 will have at least the same group preference as i_2 if u rates i_1 no less than i_2. This holds for both the average and least-misery. For pair-wise disagreement, we showed that our group disagreement functions (pair-wise and variance) can be transformed into aggregations of individual pair-wise disagreements and become monotone.

Monotonicity remains true with the introduction of affinities and time. For an item i, if both users like i highly, higher affinity between them only improves i's overall preference. On the contrary, for an item j, if they like j as highly as they do i, lower affinity between them only decreases j's overall preference. Introduction of time in the affinity model only makes the affinity calculation time-dependent by changing the temporal granularity at which it is computed; however, the relationship between dynamic affinity and the group consensus of an item does not change. □

As a result, we can design instance optimal algorithms with the early stopping.

3.1 Running Example and Data Structures

We now describe the data structures necessary to run Fagin-style top-k processing algorithms via an example that will also be used to illustrate our algorithm, `GRECA`.

Imagine a group $\mathcal{G}$ formed with three users u_1, u_2, u_3. Given an itemset $\mathcal{I} = \{i_1, i_2, i_3\}$, our objective is to identify the best item ($k = 1$) to recommend to the group at time period p (for example, January 2014). Also, assume that the system has information about

Algorithm 1 Group Recommendation with Temporal Affinities (GRECA)

Require: Group $\mathcal{G}$, k, consensus function $\mathcal{F}$;
1: Retrieve user preference lists $\mathcal{PL}_u$ for each user u in group $\mathcal{G}$;
2: Retrieve pair-wise affinities for aff_S, aff_V for each period;
3: $Sc_r = \{r_u\}$, the last user preference from $\mathcal{PL}_u, \forall u \in \mathcal{G}$
4: $Sc_{aff_S} = \{aff_{S\,u,v}\}$, the last pair-wise aff_S affinity values read $\forall u, v \in \mathcal{G}$
5: $Sc_{aff_V} = \{aff_{V\,u,v}\}$, the last pair-wise periodic affinity values read for each time period $p', \forall u, v \in \mathcal{G}$
6: Cursor cur = `getNext()` round-robin accesses to $\mathcal{PL}_u$, aff_S and aff_V lists
7: **while** (cur <> NULL) **do**
8: Get entry e at cur
9: **if** !(in $\mathcal{B}$(topKHeap, e)) **then**
10: ComputeUB(e, $\mathcal{F}$)
11: ComputeLB(e, $\mathcal{F}$)
12: Add e in $\mathcal{B}$
13: **else**
14: Update ComputeUB(e, $\mathcal{F}$) and ComputeLB(e, $\mathcal{F}$)
15: **end if**
16: Sc_{th} = ComputeTh({E}, $\mathcal{F}$) considering all current cursor positions
17: **if** $Sc_{th} \leq \mathcal{B}.k_{th}LB \& |\mathcal{B}| = k$ **then**
18: **return** topKList($\mathcal{B}$, ||);
19: Exit;
20: **else**
21: **if** CheckBuffer ($\mathcal{B}$) is satisfied **then**
22: **return** topKList($\mathcal{B}$, ||)
23: Exit;
24: **else**
25: cur = getNext()
26: **end if**
27: **end if**
28: **end while**
29: **return** topKList($\mathcal{B}$, ||)

group members u_1, u_2, u_3 for one year, i.e., January 2013 to January 2014. The user-item preference lists of those group members are provided in Table 1. Each list contains items preferred by each user sorted in decreasing order of preference.

The item preference of a member u of a group G is a combination $pref(u, i, \mathcal{G}, p) = apref(u, i) + rpref(u, i, \mathcal{G}, p)$, where $rpref$ is the relative preference that accounts for the temporal affinity of u with other group members.

u_1	u_2	u_3
i_1 5	i_1 5	i_3 2
i_2 1	i_2 1	i_1 2
i_3 1	i_3 0.5	i_2 1

Table 1: Absolute Preference Lists $\mathcal{PL}_u$ of u_1, u_2, u_3

Affinity between users consists of two components, static affinity aff_S and dynamic affinity aff_V. The detailed interpretations of these affinities and how they are calculated are given in Section 2. Out of the two aforementioned affinities, the latter is time-aware, where time is considered in a continuous fashion or over a discrete set of time periods (for example, two equal periods p_1 and p_2 each of six months in our case). For simplicity, we consider the discrete model in this example. The aff_S affinity involves all user-pairs thereby creating $3 \times (3-1)/2$ (i.e. $n(n-1)/2$ in general) entries. For every time period p', similarly, there is a periodic affinity list of the same size. Notice that each affinity list aff_V or aff_S with $n(n-1)/2$ entries could further be partitioned into a set of $n-1$ lists, where the i-th list stands for user u_i with $n-i$ entries. For example, we can have a $\mathcal{L}_{aff_S}(u_1)$ that stores u_1's static affinity with u_2 and with u_3, one for $\mathcal{L}_{aff_S}(u_2)$ with u_2's affinity with u_3 only (storing u_1's affinity here again is redundant), and no static affinity list needs to be created for user u_3. This partitioning allows us to design efficient algorithms, as we describe later in Section 3. Table 2 contains aff_S affinity lists of all users sorted in decreasing order and Tables 3 and 4 contain aff_V affinity of users in periods p_1 and p_2 respectively. Note that the temporal affinity of users u_1 and u_2 has decreased between periods p_1 and p_2.

In the above example, temporal affinity between user pairs (u, u') is modeled in a discrete manner as $aff^D(u, u', p)$. To facilitate efficient computation, it is easy to see that the different absolute preference lists and time-variant affinity lists are to be pre-computed. Even for a small group such as the one in the example with 3 users, there are 3 absolute preference lists. Furthermore, the all-pair user affinities for a given time period p' are to be stored as well, either as a single list with $n(n-1)/2$ entries, or decomposed over a set of $n-1$ lists, where the i-th list represent user u_i's affinities with $n-i$ other users. Since the period affinities are independent of each other, we must precompute such lists for every time period. For the example case, this requires either creating 2 periodic affinity lists to capture aff_V affinity and one static affinity list to capture aff_S. Each of these lists have $n(n-1)/2$ entries (as a single list or splitted in $n-1$ lists, as described in the example). The size of each list is quadratic in the number of users, but the number of such lists ($\mathcal{T}$) is a function of how time is discretized into periods. Even for a small group such as ours, many lists are to be used in the computation. Notice that all these user-affinity lists are required to compute the *complete score* of any item, because, the relative preference $rpref(u, i, \mathcal{G}, p)$ for every item requires accessing *all* $\mathcal{T} \times n(n-1)/2$ entries. An algorithm such as TA must read all those entries to compute the *complete score* of an item and will hence incur a large number of RAs.

We argue that all these accesses are not always necessary. For instance, based on preferences in Tables 1 to 4, we consider scanning item i_1 in $\mathcal{PL}_{u_1}$. If we were following TA method, to compute the complete score of this item, 21 RAs are needed, i.e., one RA for each $apref(u, i_1)$ component and 6 RAs for each $rpref(u, i_1, p)$ component where $u \in \{u_1, u_2, u_3\}$. Note that to compute the score of a single item i_1, we have accessed *all* entries in $aff_S(u_1)$, $aff_V(u_1, p_1)$ and $aff_V(u_1, p_2)$ lists. For instance, entries in the list $aff_S(u_1)$ is the static affinity scores between (u_1, u_2) and (u_1, u_3) where we have accessed both.

Instead, our instance optimal algorithm GRECA makes *only sequential accesses, i.e.,* SAs like NRA and *potentially avoids consuming all these* $\mathcal{T} \times n(n-1)/2$ *entries* to determine the top-k itemset. Following previous example, if for instance $aff_S(u_1, u_3)$ (in Table 2) is not yet scanned, we avoid making an RA to get this value, but based on NRA principle, we use the score under the cursor in the list of Table 2 (i.e., initially $aff_S(u_1, u_2)$) to compute a partial score for i_1. Details are mentioned in Section 3.2.

GRECA returns the top-k itemset which contains the best set of k-items, although the rank among the returned itemset may not be fully distinguishable (i.e. giving rise to a partial order). This is rather reasonable, because k is usually small, and the group is potentially interested in all of the k-items.

3.2 GRECA

For ease of exposition, we describe GRECA using the simplest group consensus function *Average Preference* considering time-aware affinity. The other group consensus functions mimic its behavior. The algorithm exploits the settings as is described in the example in Section 3.1.

u_1	u_2
u_1u_2 1	u_2u_3 0.3
u_1u_3 0.2	

Table 2: Static Affinity Lists $\mathcal{L}_{aff_S}$

u_1	u_2
u_1u_2 0.8	u_2u_3 0.2
u_1u_3 0.1	

Table 3: $\mathcal{L}_{aff_V}$ Lists for Period p_1

u_1	u_2
u_1u_2 0.7	u_2u_3 0.1
u_1u_3 0.1	

Table 4: $\mathcal{L}_{aff_V}$ Lists for Period p_2

Without loss of generality, for a given group with n users, GRECA uses n user-item preference lists, where each list $\mathcal{PL}_u$ for user u has m items that are sorted in decreasing user-item preference. Each $\mathcal{PL}$ can be obtained with any single user recommendation strategy (in our experiments in Section 4, we use collaborative filtering). In addition, GRECA uses $n-1$ static affinity lists, and another $n-1$ dynamic periodic affinity lists for each time period.

The algorithm runs in a round-robin fashion over the aforementioned lists by making only SAs. It reads an entry $e=(i,r)$, where i is the item-id and r is the user u's absolute preference score for i, or an entry $e'=(u',r')$, where r' is the pair-wise affinity of (u,u'). Affinity between a pair of users is either static or periodic (i.e. dynamic), and the computation does not distinguish between these two kinds. The algorithm invokes the following 3 different subroutines to determine whether to continue further or to safely terminate and return the top-k itemset during its execution:

(a) Compute Upper-Bound of an Item: `ComputeUB(i)`**:** UB_i computes the highest score that an item i can have in $\mathcal{G}$ based on so far accesses.

(b) Compute Lower-Bound of an Item: `ComputeLB(i)`**:** LB_i computes the lowest score that an item i can have in $\mathcal{G}$ based on so far accesses.

(c) Compute Global Threshold: `ComputeTh(`$\{E\}$`)`**:** Input to this function is the current set $\{E\}$ of entries read from all the lists. The output is simply a numeric score that captures the highest score that an *unseen* item can have for group $\mathcal{G}$.

Subroutines can be invoked after reading one entry from each type of list (preference list, static affinity list or dynamic affinity list) to make sure all types of lists are visited before or after reading the j-th entry from all lists.

The first two subroutines return the latest bounds of an item i. Then, those updated bounds are pushed into an *item buffer* that is maintained throughout the execution of the algorithm. We describe our proposed buffer management strategy later on. Naturally, these two subroutines are to be invoked for all encountered items so far.

Illustration of the Subroutines: The upper-bound score of an item i is simply the highest score it can have on current accesses. It is computed by combining the actual encountered values for some of the entries and then assigning the current cursor readings to the rest. Consider our three-user group described in Section 3.1 and assume that `ComputeUB(`i_3`)` is invoked after the cursor reads the second entry at $\mathcal{PL}_{u_2}$. At that point, $apref(u_3,i_3)=2$ is encountered, but for the other two users, these are to be approximated based on the current cursor readings. For example, the highest score of $apref(u_1,i_3)=1$, $apref(u_2,i_3)=1$. Similarly, static and dynamic periodic affinities of users u_1,u_2 and u_2,u_3 are encountered, but those of u_1,u_3 are to be guessed based on the latest cursor reading from the respective lists. This gives rise to `ComputeUB(`i_3`)` = $\Sigma_{\forall i\in\{1,2,3\}}$`UB`$[apref(u_i,i_3)]$ + `UB`$[rpref(u_i,i_3,\mathcal{G},p)]$ = 13.02 (by ignoring normalization and final averaging).

The computation of the lower-bound of an item i is similar except that it replaces the unseen entries of the function with the lowest possible score. For example, instead of assigning $apref(u_1,i_3)=1$, $apref(u_2,i_3)=1$, it will consider those values to be 0 (assuming that the smallest absolute preference for an item could be 0). The same will happen in affinity calculation; as an example, it substitutes $aff_S(u1,u_3)=0$, instead of 0.8 in the upper-bound computation case. When invoked using item i_1, `ComputeLB(`i_1`)` returns a value of 14.2 (ignoring normalization and final averaging).

Computation of `ComputeTh(`$\{E\}$`)` is rather simple. It simply incorporates each of the entries in $\{E\}$ in the function and returns a numeric score.

Buffer Management Strategy: Once the upper-bound and lower-bound scores of each item are computed, they are pushed into a buffer $\mathcal{B}$ and are sorted in decreasing order of lower-bound score. The buffer is implemented as a heap data structure which allows efficient updates since it requires to maintain sorted lists of potential results and, in some cases, item lower-bounds and upper-bounds need to be updated (for example, when the item is encountered again in one of the lists).

Stopping Condition: The algorithm has both global threshold computation and buffer management strategies. We now show that *the buffer management* itself is sufficient to govern early stopping. More importantly, unlike traditional threshold algorithms, GRECA cannot terminate only based on the threshold condition in the cases, where the buffer contains more than k items.

- **Using the Global Threshold:** If the current global threshold is not larger than the lower-bound score of the k-th item in the buffer, GRECA will not find any item later on whose score is larger than the current threshold. On the other hand, if the current threshold is no larger than the lower-bound of the k-th item in the buffer, any unseen item can never be in the top-k itemset. This implies that a subset of the items in the current buffer is the actual top-k itemset. If the buffer contains k items only, then GRECA can safely terminate and return those items in the buffer as the answer. However, in general, when the buffer has more than k-items, to precisely determine the actual top-k itemset, it needs to apply the buffer management strategy that we describe now.

- **Using the Buffer:** A key novelty of GRECA is in using only the buffer condition for termination. This condition simply implies that just by looking into the items in the buffer, GRECA can terminate, as well as declare the partially ordered correct top-k itemset. The buffer stopping condition works as follows: the buffer contains k'-items ($k'>k$) such that the lower-bound of the k-th item score is no smaller than the upper-bound score of each of the remaining $k'-k$ items. In that case, those remaining $k'-k$ items could be safely pruned. Interestingly, satisfying this condition implies satisfying the threshold condition as well, as Theorem 1 states. The remaining k items are returned as answers.

- **Global Threshold and Buffer Management:** Global threshold can simply determine that the current buffer contains a subset of items which are the actual top-k itemset. In a general case, where the buffer has more than k items, GRECA applies the buffer stopping conditions to determine that subset. It is still possible that the buffer condition for stopping is not met. In that case, GRECA resumes computation until the buffer condition is satisfied or all lists are exhaustively scanned.

Theorem 1. Satisfying the buffer condition for termination implies that the global threshold condition for termination is met.

PROOF. (sketch): At a given snapshot during the execution of GRECA, the score returned by ComputeTh($\{E\}$) is strictly not greater than the upper-bound score of any item that is already seen and in the buffer, i.e., ComputeUB(i) $\geq$ ComputeTh($\{E\}$). Therefore, if the buffer condition is satisfied (meaning that the lower bound of the k-th item score in the buffer is not smaller than the upper-bound score of the remaining $k' - k$ items), this automatically implies, that the lower bound of the k-th item score is not smaller than the current global threshold. Hence the proof. □

For our running example in Section 3.1, this returns i_1 as the top-1 item to the group.

The pseudocode of GRECA is presented in Algorithm 1. In addition to the group $\mathcal{G}$ and k, it takes the preference and affinity lists of $\mathcal{G}$ as inputs as well as the consensus function $\mathcal{F}$. Lines $9 - 14$ either add a new item into the buffer $\mathcal{B}$ and compute its lower-bound and upper-bound scores, or update the latest lower-bound and upper-bound score of an existing item and reorganize the buffer. Line 16 computes the global threshold condition using the function ComputeTh(); lines $17 - 19$ checks if the threshold stopping condition is satisfied. Otherwise, the control goes on to line 21 on wards and CheckBuffer($\mathcal{B}$) checks whether the stopping condition is met using the buffer. The computation continues unless one of these conditions are satisfied, or all lists are exhaustively scanned. Of course, in the latter case, there is no save-up. However, as our experimental results exhibits, GRECA achieves speed-up, compared to its naive counterpart.

LEMMA 2. GRECA *returns correct top-k itemset.*

PROOF. Notice that GRECA returns from the buffer those k-items whose lower-bound scores are the highest and larger than the upper-bound score of any remaining item. As Theorem 1 proves that this also implies that the global threshold at that point cannot be larger than the lower-bound score of the k-th item in the buffer. Notice that the threshold captures the highest score that any unseen item can have. Due to the monotonicity property of the consensus function, global threshold decreases gradually, implying that the highest score of any item gets only smaller, as more entries are scanned from the lists. Therefore, when GRECA terminates and outputs the itemset with the highest top-k lower-bound scores, this implies that any other items that are discarded or unseen cannot have higher score than the returned itemset. Hence the proof. However, since the complete score of many of the items may not be computed upon termination, the output may give rise to a partial order among the top-k items. □

LEMMA 3. GRECA *is instance optimal.*

PROOF. (sketch): In [10], authors prove that NRA is instance optimal with optimality ratio m and no deterministic algorithm can perform any better. GRECA mimics the cursor movement of traditional NRA, however, it has a different stopping condition. Theorems 1 and 2 prove that our stopping condition implies both the threshold stopping condition and result correctness, therefore, the instance optimality of GRECA holds. A detailed proof is deferred to an extended version of the paper. □

4. EXPERIMENTS

We evaluate our group recommendation method from two major angles: effectiveness and efficiency. We conduct an extensive user study on Facebook to demonstrate that group recommendation with the consideration of temporal affinity is superior to solely relying on aggregating individual preferences (Section 4.1). We also run comprehensive experiments to show that GRECA achieves scalable performance when computing temporal affinity-aware recommendations for ad-hoc groups (Section 4.2).

We implement our prototype system using JDK 1.8.0. All scalability experiments are conducted on an 2.4 GHz Intel Core i5 with 8 GB of memory on OS X 10.9.5 operating system.

Dataset Description: We use the MovieLens 1M ratings dataset [2] for our evaluation. MovieLens is a collaborative rating database where users provide a rating ranging from 1 to 5 for movies (5 being the best). Table 5 contains the statistics of the 1M ratings dataset.

# users	# movies	# ratings
6,040	3,952	1,000,209

Table 5: The MovieLens 1M Dataset

Individual User Preferences: We use collaborative filtering [2] to generate individual user preferences where user similarity is computed with cosine similarity over $vec(u)$, i.e., the ratings of u for each movie.

$$\cos(\vec{u}, \vec{u'}) = \frac{\vec{u} \times \vec{u'}}{\|\vec{u}\|^2 \times \|\vec{u'}\|^2}$$

4.1 Quality Experiment

We exploit the availability of Facebook users for our user study which gives us the opportunity to obtain preferences of real users and leverage the social graph for affinities. Our aim is to compare our temporal affinity-aware group recommendation with naive methods without consideration of time or affinity. Our group recommendations are produced and compared using the following consensus functions (as discussed in Section 2).

- **Average Preference (AP)**, which computes the group preference for an item as the average of individual group members' preferences for that item.
- **Least-Misery Only (MO)**, which computes the group preference for an item as the minimum among individual group members' preferences for that item.
- **Pair-wise Disagreement (PD)**, which computes the group preference for an item as the combination of its average and its pair-wise disagreement between individual group members' preferences.

[2] *http://movielens.umn.edu*

For each of these functions, we incorporate time-aware affinity to compute the relative user preference to an item at a given time (see Section 2 for an exact definition of relative preference.)

We developed an application using the Facebook API[3] and recruited 72 Facebook users overall to rate movies from the MovieLens 1M dataset. We obtained 1981 ratings. Our Facebook application asks only for *public profiles* and *friend list access* permissions. Also, we anonymize the dataset by mapping Facebook IDs to a random 5-digit number. The study is conducted in two phases: *User Collection Phase* and *Quality Assessment*.

Summary of Results: In summary, we observe that including temporal affinity in group recommendation significantly improves user satisfaction. The amount of satisfaction is variable and is dependent on how groups are formed. In particular, dissimilar user groups as well as those with low-affinity users whose preference significantly evolves over time, are most satisfied. We found that prior work has indeed shown [20] that reaching consensus among such group members is indeed difficult. In addition, we found that **PD**, in general, is the method of choice and works best for dissimilar and high affinity groups. This observation is also in line with one of our prior results [22] where we showed that including disagreement in group consensus generates higher quality recommendations. We also observe that incorporating time models produces better results for high affinity groups suggesting that groups with high affinity are most sensitive to temporal affinities. Finally, the continuous time model is preferred by large groups of dissimilar members. That could be explained because it better captures variability for groups whose members are more sensitive to differences between them. The discrete model on the other hand, is a good approximation of the continuous one in the case of high affinity and high similarity groups.

4.1.1 User Collection Phase

In this phase, the goal is to recruit users and collect their data. Later, collected users are used to form different groups and perform judgments on group recommendations. For this aim, we start with 13 seed users (denoted S). Users in S have to complete two tasks: i. rate at least 30 movies in MovieLens, and ii. invite between 10 and 20 of their friends to participate in the study. The set of friends of a seed user $s \in S$ is denoted $friends(s)$. Note that we consider $\cup_{s \in S} friends(s) \cap S = \emptyset$. Friends are only asked to rate movies and not invite friends, i.e., we stop at the depth 1 of the social graph for this study.

We select a subset of MovieLens movies for participants to provide their preferences. We consider 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 for. On the other hand, we want to maximize our chances of capturing different tastes among movie-goers. Towards those two goals, we select two sets of movies. The first set is called the *popular set*, which contains the top-50 movies in MovieLens in term of popularity (i.e. the number of users who rated a movie in the set). The second set is called *diversity set*, which contains the 25 movies with the highest variance among their ratings and that are ranked in the top-200 in terms of popularity. Each participant rates movies in one of two pre-computed sets: the **Similar Set** which consists entirely of movies within the *popular set* and the **Dissimilar Set** which consists of the top-25 movies from the *popular set* and the 25 movies from the *diversity set*.

Users are instructed to provide a rating between 1 and 5 (5 being the best) for at least 30 movies listed in random order, according to their preferences.

4.1.2 Static and Dynamic Affinities

In addition to ratings, we store anonymized lists of *friends* and *page-likes* for each user. Since Facebook friendship is relatively stable over time, we use it to compute static affinity: $aff_S(u, u') = |friends(u) \cap friends(u')|$. We normalize all static affinity values in a group by the maximum pair-wise value in the group to obtain a number between 0 and 1.

Page likes are dynamic and are used to compute the time-varying component of affinity. To calculate dynamic affinity for each user, we store all pages (s)he has ever liked in Facebook and for each page, we record the timestamp of when the user liked it and the page category (music, movie, etc.). There exist 197 different page categories in Facebook. For privacy reasons, we do not record the name of the liked pages. Thus the periodic affinity between two users u and u' in time-period p is calculated as: $aff^P(u, u', p) = |page\_likes(u, p) \cap page\_likes(u', p)|$ where $page\_likes(u, p)$ is the set of page categories whose pages are liked by u in time-period p. Then we calculate $aff_V(u, u', p)$ using Equation 1. We also normalize dynamic affinity values to be between 0 and 1. We consider 6 different two-month consecutive periods (Section 4.2.1). Note that the average standard deviation over number of common page-likes for all user pairs during 6 periods is 0.42.

4.1.3 Group Formation

We consider three main factors in forming user groups, i.e., *group size*, *group cohesiveness* and *affinity strength*. Size and cohesiveness (i.e. how similar are group members in their movie tastes) are akin to prior work [22].

We hypothesize that varying group sizes will influence reaching consensus among the members and therefore to which degree members are satisfied with the group recommendation. We choose two group sizes, 3 and 6, representing *small* and *large* groups, respectively.

Similarly, we assume that group cohesiveness is also a significant factor in their satisfaction with group recommendation. As a result, we form two kinds of groups: *similar* and *dissimilar*. A similar group is formed by selecting users who i. have watched **Similar** movies and ii. have the maximum summation of pair-wise similarities (between group members based on their provided ratings) among all groups of the same size. A dissimilar group is formed by selecting users who i. have completed the **Dissimilar** movie set and ii. have the minimum summation of pair-wise similarities among all groups of the same size.

Finally, we consider groups with *low* and *high* affinity between members. We set affinity to be high if each pair-wise affinity in a group is equal to 0.4 or higher.

4.1.4 Quality Assessment

In the second phase of the study, users are instructed to decide which of the recommended movies they are satisfied with in a group. We form 8 groups out of Facebook users by considering different combinations of *group size*, *group cohesiveness* and *affinity strength*. Each user evaluates movies in two phases: *Independent* and *Comparative*.

Independent Evaluation: In the independent evaluation, a user, who is a member of a group, observes a single recommendation list at each time and is asked to say how satisfied she is with watching those movies with other group members using a scale between 0 and 5 (5 being the best). Figure 1 illustrates the results of this

[3] *https://developers.facebook.com*

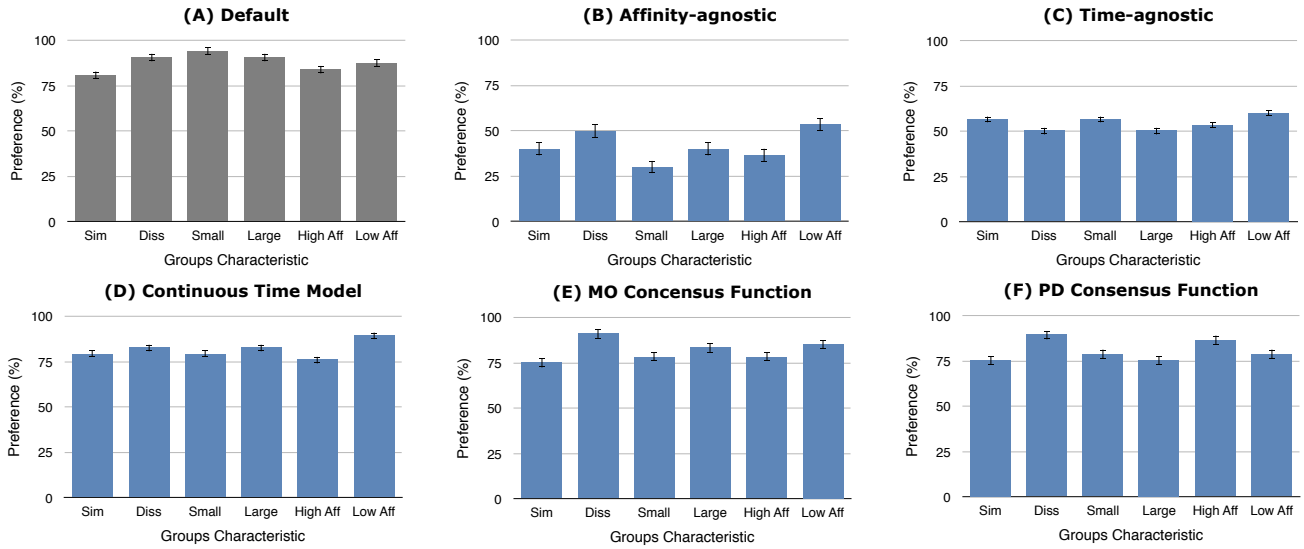

Figure 1: Independent Evaluation

evaluation phase. The score is reported as a percentage, i.e., a result with an average score of 5 gets 100%. Four parameters play a role in generating different recommendation lists in Figure 1, i.e., affinity awareness, time model (discrete vs. continuous), temporal awareness and consensus function. Figure 1. A illustrates results with default values, i.e., affinity-aware, discrete, time-aware and **AP** consensus function. In all other figures, only one parameter value changes, i.e., affinity-agnostic in B, time-agnostic in C, continuous time model in D, **MO** function in E and finally **PD** function in F. That parameter is mentioned in the title of each chart in Figure 1.

We observe that in general, participants give a score of at least 80% to A, which is the default case with discrete temporal affinity. Participants in dissimilar groups have scored A with 90.66% preference while it is 10% lower for similar groups. This could be interpreted as: averaging individual ratings and using a discrete time model works well for groups formed by users who like different movies. Interestingly, the same result holds for low affinity and high affinity groups. This potentially shows that our model is robust to time-varying tastes. On the other hand, the low preference of high affinity groups show that members of those groups benefit from another consensus function, i.e., **PD** (chart F).

Lists without affinity (chart B) and time awareness (chart C) have at most 55% and 60% overall preference respectively. This margin of 20% difference in preference with the temporal affinity case (chart A) shows explicitly the importance of affinity and temporal affinity in group recommendation. In B, worst results are obtained for small (30.08%), high affinity (36.66%) and similar groups (40%) where we observe a decrease in satisfaction. This potentially shows those are the groups that would best benefit from using affinity in computing their recommendations. In C the worst results are for dissimilar and large groups (both 50.19%). One explanation is that dissimilar large groups, i.e., those who differ in their movie tastes among many members, prefer temporal recommendations, i.e., movies that are generated by taking into account their friendship and page-like differences over time.

Groups with different tastes (dissimilar, large and low affinity) prefer the continuous time model (chart D). This is potentially because of a higher precision in capturing time. Considering the time as a whole from the beginning of time is needed to deliver recommendations that satisfy all members of those heterogeneous groups. In case of **MO** (chart E), we observe a superior satisfaction for dissimilar and low affinity groups as *increase in uncertainty in large groups* leads members to like **MO** better.

Comparative Evaluation: In the comparative evaluation, users are asked to compare two lists l_1 and l_2 at a time and pick the list they prefer. Following the closed world assumption, when a list is not chosen by a user, it means that it is not preferred. A user has to choose one and only one of the proposed lists. Figure 3 illustrates the preferences of l_1 over l_2.

First, a user is asked to compare affinity-aware (l_1) vs. affinity-agnostic (l_2) recommendations. In A, we observe that in general, in 75% of the cases, affinity-aware recommendations are preferred. They are mostly appreciated by small groups followed by high affinity groups. Larger groups have less preference for affinity-aware results. A large group potentially leads to higher variability of preference and weaker affinity among its members, thus naturally prohibiting an early agreement.

In the second comparative study, we examine the effect of temporal affinity by comparing time-aware (l_1) vs. time-agnostic (l_2) recommendations. In B, we observe that in most groups, temporal recommendations are preferred in over 80% of the cases. This leaves no doubt that participants like better results obtained based on time. It also shows that high affinity groups prefer not only affinity-based results, but also its temporal version. Small groups have also exhibited a high preference. This is because in groups with fewer members or groups whose participants deeply know each other, the effect of time manifests itself more strongly. Finally, high preference for large groups show that the temporal dimension of affinity is a useful component for such groups to obtain higher quality results, because group members potentially observe that their common affinity history plays a role in recommendations.

We now examine which of the discrete or the continuous temporal affinity models is better and in which case. In C, we observe that in general, the discrete time model is preferred for groups with strong connections between members (high affinity and high similarity). In the case of dissimilar and large groups, it is the continuous model that is preferred. The continuous nature of the latter is certainly better to capture variability for groups whose members

are more sensitive to differences between them while the discrete one is a good approximation of the continuous model in the case of high affinity and high similarity groups.

Finally, we compare different group consensus functions. This time, we compare 3 different lists together which are results of **AP**, **MO** and **PD** consensus functions. We are interested to discover which function delivers more satisfactory results when we account for temporal affinities. Figure 2 illustrates this comparison. In short, while the choice of which consensus function to apply heavily depends on group characteristics, there exists a general preference for **PD** especially in the case of loosely connected groups (low affinity and dissimilar groups). That could be explained by the fact that **PD** favors items that minimize disagreement between group members which is more appropriate for dissimilar group members.

In summary, it is shown that **AP** is highly preferred in small and high affinity groups. Whenever **AP** has a high preference, **PD** is also highly preferred. **MO** provides higher quality results for larger groups (this is consistent with our findings in [22]) and for groups with loose connections.

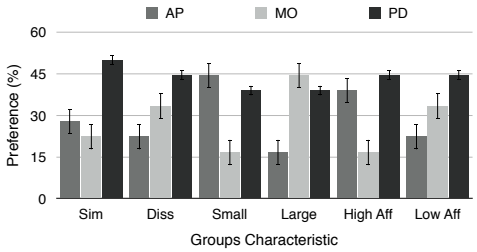

Figure 2: Qualitative Evaluation of Consensus Functions

4.2 Scalability Experiment

Experiment Settings: Unless otherwise stated, we form 20 different random groups by selecting a subset of users who participated in our quality experiment. The default settings of the rest of the parameters are, group size = 6, k = 10, number of items = 3900, consensus function = **AP**. Unless otherwise stated, affinity is computed using the discrete time model. For each scalability experiment, we compute the average percentage of SAs needed by GRECA in different settings. The percentage of SAs represents the computational cost that GRECA incurs, compared to a naive algorithm which entirely scans all lists. A smaller percentage exhibits higher scalability.

We conduct experiments by varying time periods, result size (k), group size, number of input items, similarity and dissimilarity among items and users in the group, and considering the discrete and continuous affinity models. Our results illustrate the scalability of GRECA with different group consensus functions. We only present a subset of these results. The omitted results are similar to the ones presented. All results are presented with standard error bars, wherever applicable.

Summary of Results: First and foremost, we observe that GRECA is highly scalable with varying k, group size, number of items and enables a significant saveup in the number of accesses (almost always, more than 75% accesses are avoided) with early termination. Then, we observe that the pruning ability is highest for similar user groups. We observe that the score distribution of top-k itemsets for such groups is different from the rest of items, therefore, the stopping condition in the buffer is satisfied early. Third, we observe that GRECA is effective across all group consensus functions. In fact, for some of the complex group consensus functions that consider user disagreement, GRECA incurs the smallest percentage of accesses ensuring the highest saveup in computation cost. Fourth, GRECA scales linearly with an increasing number of periods. Finally, we observe that GRECA is effective both for discrete and continuous models.

4.2.1 Varying Time Period

We explore discretizing time into periods of different lengths: week, month, two-month, season and half-year. Since dynamic affinity relies on user page-likes in Facebook and liking a page is not a frequent action, many time segments were empty after discretization (Figure 4). The length of a time period should be chosen in such a way that each period contains enough data to compute affinities. Figure 4 shows that two-month periods achieve a good balance between the percentage of non-emptiness (65%) and the number of periods (6). We hence pick a two-month discretization for the rest of our experiments.

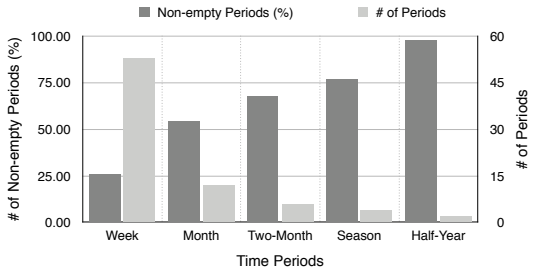

Figure 4: Different Time Periods

Figure 6 illustrates the average number of accesses in each period. As expected, this figure shows a linear behavior in general, as going to subsequent periods increases the number of lists. An exception happens in period 5 where its average #SA is very close to its next period. By looking more carefully at the underlying data distribution, we noticed that the number of common page-likes between user pairs in period 5 is very low. Therefore, scanning this period does not help to update bounds in order to have early termination.

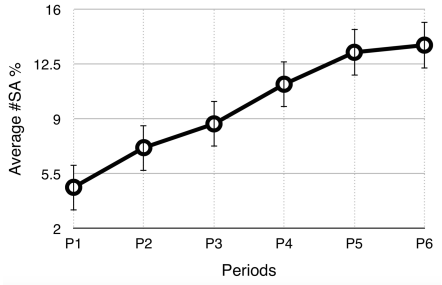

Figure 6: Average Percentage of SAs for Different Periods in Discrete Time Model

4.2.2 Varying k, Group Size and Number of Items

In Figure 5, we illustrate the scalability of GRECA by varying result size, group size and number of items. In A, we vary k from 5 to 30 and run GRECA with the **AP** consensus function for 20 different groups with 6 members. We observe that GRECA scales linearly with varying k. The algorithm always produces a saveup of 81% or higher.

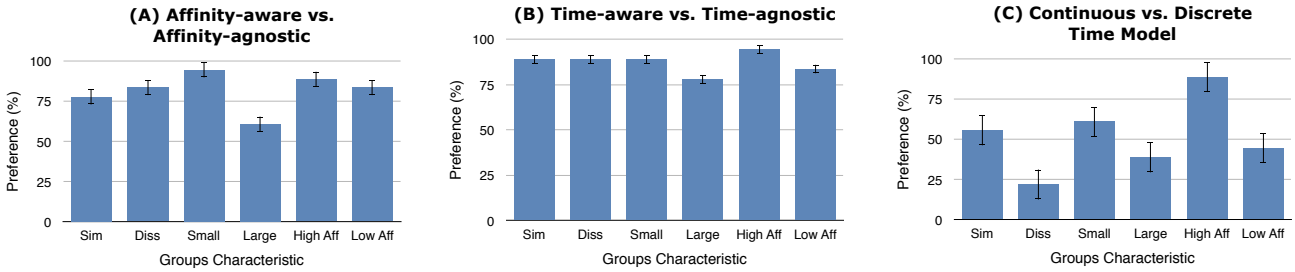

Figure 3: Comparative Evaluation

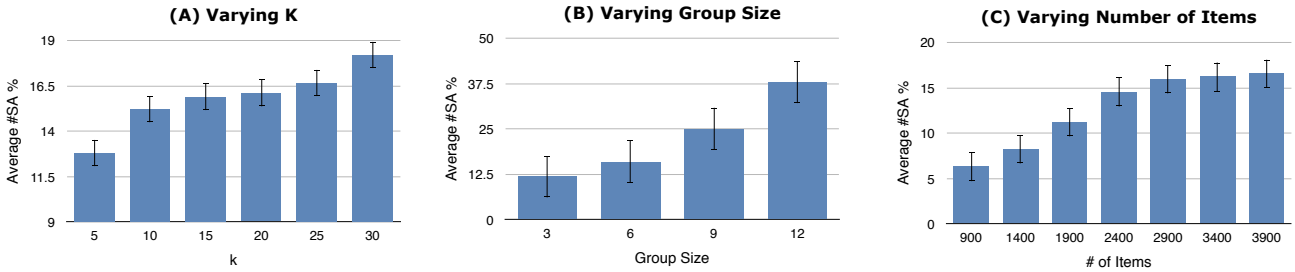

Figure 5: Average Percentage of SAs by Varying Result Size, Group Size and Number of Items

In B, we examine the effect of different group sizes on performance. The results clearly demonstrate that GRECA scales well with varying group sizes. The average saveup is greater than 77%.

In C, we vary the number of available items for group recommendation from 900 to 3900. The results demonstrate that the number of accesses does not necessarily increase with that. This observation is unsurprising as the number of accesses depends on the score distribution of the item preferences and user affinities. GRECA saves more than 83% accesses even in the worst case.

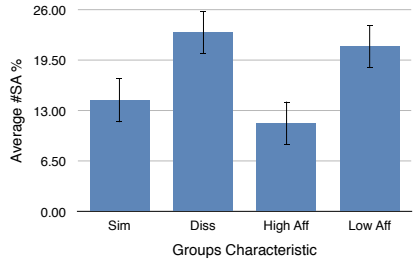

Figure 7: Average Percentage of SAs for Similar, Dissimilar, High Affinity and Low Affinity Groups

4.2.3 *Similarity/Dissimilarity*

We examine the effect of similarity on GRECA in two ways: first, we compare the number of accesses between groups with similar and dissimilar ratings; then, we compare groups with high and low affinities. Figure 7 contains the result. The results demonstrate that the effectiveness is higher for similar groups in both cases (item based similarity and high affinity).

4.2.4 *Time Models*

We examine the effect of continuous and discrete time models on GRECA. The average number of SAs for the continuous model is 16.32% and 16.6% for the discrete one. This means that in both cases, we obtain a saveup greater than 83%. The number of accesses for both methods are very similar with a slight superiority for the discrete model.

4.2.5 *Consensus Functions*

In this last performance study, we compare different consensus functions. Figure 8 contains the results. We introduce two different versions of **PD** based on [22] by varying the weights used in the linear combination of rating aggregation (w_1) and disagreement (w_2) s.t. $w_1 + w_2 = 1$. In **PD V1**, we consider $w_1 = 0.8$ and in **PD V2**, $w_1 = 0.2$.

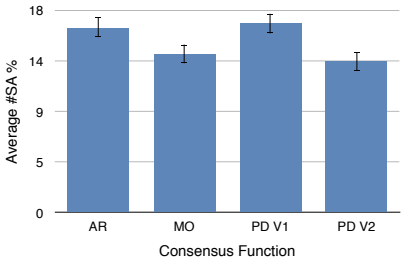

Figure 8: Average Percentage of SAs for Different Consensus Functions

All results clearly demonstrate that GRECA achieves significant saveups for those consensus functions. They also show that **PD V2** outperforms **PD V1**. During our post-analysis, we observed that a higher weight on disagreement allows faster stopping, because the items have smaller scores. **MO** is the next best performer achieving as high as 83% in accesses' saveups.

5. RELATED WORK

Group recommendation has been designed for various domains such as news pages [21], tourism [11] and music [7]. A group may

be formed at any time by a random set of users with different interests, a number of persons who explicitly choose to be part of a group, or by computing similarities between users with respect to some similarity functions and then clustering similar users together [19, 3].

There are two dominant strategies for group recommendations [4, 3]. The first approach creates a pseudo-user representing the group and then makes recommendations to that pseudo-user, while the second strategy computes a recommendation list for each group member and then combines them to produce a group's list. For the latter, a widely adopted approach is to apply an aggregation function to obtain a *consensus group preference* for a candidate item. However, to the best of our knowledge, none of the existing functions account for the influence between group members [24].

Affinities may be strong emotional bonds, like links between family members or a clique of close friends. Those links may also be relatively weak thereby breaking with the passage of time or the occurrence of relationship-damaging events. In [16], an affinity model is proposed for group recommendation based on NEO-FFI [4] personality test. Another model is proposed in [12] where 3 different components (social relationship, expertise and disagreement) are aggregated to form affinity. A possible extension of our work could make use of that affinity definition.

On e-commerce platforms, recent studies have proved the importance of time in recommender systems. In [8], Yi Ding et al. assign different weights to different rating records based on their creation time, and reveal the existence of a dynamic change in user interests. Liang Xiong et al. [25] improved the accuracy of recommendations by incorporating the global evolution pattern of user preferences. Potentially the most similar contributions to ours on time models are [17, 15] where Yehuda Koren et al. take temporal dynamics of user and item biases into consideration for individual recommendations. To the best of our knowledge, no previous work has studied a time model for group recommendations.

Threshold algorithms [9] have been used extensively for recommendation. Their attractiveness lies in monotonic score aggregation functions, which operate on sorted input and enable the early pruning of low-ranked answers. In this work, we adapt `NRA` to aggregate individual preferences, disagreement and temporal affinity lists to compute temporal recommendations and propose a novel stopping condition and prove correctness and instance optimality.

6. CONCLUSION

We examined affinity-aware group recommendation over time and developed `GRECA`, an efficient algorithm with unique features that distinguish it from state-of-the-art recommendation algorithms. Our proposed semantics is compatible with popular group consensus functions. Our extensive experiments with real Facebook users and Movielens datasets assess the high quality of temporal affinity-aware recommendations for groups with different characteristics (small/large groups, similar/dissimilar groups, high and low affinity groups). In the future we would like to study the maintenance of our index structures over time in relationship with how often affinity between users changes. In particular, we are examining how to combine incremental clustering with our indices in order to determine the minimum amount of information to store that guarantees instance optimality. Moreover, we plan to extend our performance studies to larger groups with thousands of users.

[4] *Neuroticism Extroversion Openness Five Factor Inventory*

References

[1] 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.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, June 2005.

[3] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.

[4] S. Berkovsky and J. Freyne. Group-based recipe recommendations: analysis of data aggregation strategies. In *RecSys*, pages 111–118, 2010.

[5] J. R. Bettman, E. J. Johnson, and J. W. Payne. Consumer decision making. *Handbook of consumer behavior*, 44(2):50–84, 1991.

[6] 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*, pages 547–550, 2009.

[7] A. Crossen, J. Budzik, and K. J. Hammond. Flytrap: intelligent group music recommendation. In *IUI*, pages 184–185, 2002.

[8] Y. Ding and X. Li. Time weight collaborative filtering. In *CIKM*, pages 485–492. ACM, 2005.

[9] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2):109–118, 2002.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In P. Buneman, editor, *PODS*. ACM, 2001.

[11] I. Garcia, L. Sebastia, and E. Onaindia. On the design of individual and group recommender systems for tourism. *Expert Syst. Appl.*, 38(6):7683–7692, 2011.

[12] M. Gartrell, X. Xing, Q. Lv, A. Beach, R. Han, S. Mishra, and K. Seada. Enhancing group recommendation by incorporating social relationship interactions. In *GROUP*, pages 97–106. ACM, 2010.

[13] A. Jameson and B. Smyth. Recommendation to groups. In *The adaptive web*, pages 596–627. Springer, 2007.

[14] N. M. Klein and M. Yadav. Context effects on effort and accuracy in choice: An inquiry into adaptive decision making. *Journal of Consumer Research*, 16:410–420, 1989.

[15] N. Koenigstein, G. Dror, and Y. Koren. Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In *RecSys*, pages 165–172, 2011.

[16] M. Kompan and M. Bieliková. Social structure and personality enhanced group recommendation. In *EMPIRE 2014, Aalborg, Denmark*, pages 1613–0073, 2014.

[17] Y. Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, 2010.

[18] D. A. Lussier and R. W. Olshavsky. Task complexity and contingent processing in brand choice. *Journal of Consumer Research*, 6:154–165, 1979.

[19] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H.-P. Kriegel. Fast group recommendations by applying user clustering. In *ER*, pages 126–140, 2012.

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

[21] S. Pizzutilo, B. De Carolis, G. Cozzolongo, and F. Ambruoso. Group modeling in a public space: Methods, techniques, experiences. AIC'05. WSEAS, 2005.

[22] S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Space efficiency in group recommendation. *VLDB J.*, 19(6):877–900, 2010.

[23] S. B. Roy, S. Thirumuruganathan, S. Amer-Yahia, G. Das, and C. Yu. Exploiting group recommendation functions for flexible preferences. In *ICDE Conference*, 2014.

[24] N. Shabib, J. A. Gulla, and J. Krogstie. On the intrinsic challenges of group recommendation. In *RSWeb@RecSys*, 2013.

[25] L. Xiong, X. Chen, T.-K. Huang, J. G. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM*, pages 211–222. SIAM, 2010.