Recommendations for optimizing the collective user experience

Behzad Golshan* Evimaria Terzi† Panayiotis Tsaparas‡

Abstract

Traditional recommender systems aim to satisfy individual users by providing them with recommendations that match their preferences. Such recommender systems don't take into consideration how the number of users recommended to use a particular item affects the users' experience. For example, a highly-recommended restaurant may match the preferences of many users. However, increasing its popularity via recommendations may make the experience unsatisfactory due to high volume of customers, long lines and inevitably slow service. In this paper, we develop a new recommendation-system paradigm that we call collective recommendations. Collective recommendations take into consideration not only the user preferences, but also the effect of the popularity of a venue to the overall user experience. We formally define the algorithmic problems motivated by collective recommendations and develop an algorithmic framework for solving them effectively. Our experiments with real data demonstrate the effectiveness of our methods in practice.

Nobody goes there any more. It's too crowded

— Yogi Berra

1 Introduction

Traditional recommender systems [1] try to satisfy users by providing recommendations that match the users' individual preferences. Such recommender systems make the implicit assumption that the resulting experience is individual (e.g., watching a movie, or reading a book), and thus the satisfaction derived for the user is fully captured by her preferences. This assumption disregards the fact that, often, even though the preferences are individual, the resulting experience may depend on the *collective* behavior of the users. For example, when recommending physical locations, even though each user may act according to their individual preferences, the result is a crowd, and inevitably the experience becomes a collective one [14]. In such cases it is important for the recommender system to take into account the effect of the recommendations and the subsequent users' actions on the collective satisfaction of the users.

As a concrete example, consider the case of restaurant recommendations in sites such as Yelp. Recommending to users to go to a popular restaurant may match the users' preferences, but at the same time it may result in long lines, poor quality of service, and ultimately user dissatisfaction. On the other hand, when recommending activities to people (e.g., sport activities), it is important for the recommender system to take into account that the experience may be subpar, unless there is a critical mass of participants that have joined the activity. Even for product recommendations, the collective behavior may have an adverse or reinforcing effect on the user experience. For example, for clothing or gadgets, users often look for a sweet spot in the popularity of the products; something that is fashionable (popular), but at the same time not mainstream and ordinary.

In this paper, we introduce a new recommendation paradigm, which we call *collective recommendations*. Collective recommendations take into account both the individual user preferences, and the effect of the collective behavior to the overall user experience. Our recommendations aim to optimize both these aspects that contribute to the user satisfaction.

In our problem formulation, we view recommendations as an assignment (mapping) A of users to items. Our goal is is to maximize the collective user satisfaction, $\mathcal{C}(A)$, that is defined as a weighted function of two terms: (a) a term that measures the alignment of the recommendations to the user preferences, denoted by $\mathcal{P}(A)$, and (b) a term that evaluates the effect of the assignment (collective behavior) on the quality of the item, denoted by $\mathcal{Q}(A)$. Our goal is to find the assignment A that maximizes the collective satisfaction:

$$\mathcal{C}(A) = \lambda\mathcal{P}(A) + (1-\lambda)\mathcal{Q}(A).$$

We call this problem the CollectiveRecs problem.

In principle, $\mathcal{P}(A)$ and $\mathcal{Q}(A)$ can be any function of the preferences and the user assignment. By selecting the appropriate $\mathcal{P}(A)$ and $\mathcal{Q}(A)$ functions, our formulation can capture many different problem instances. In

*Recruit Institute of Technology, USA.

†Computer Science Department, Boston University, USA.

‡Computer Science and Engineering Department, University of Ioannina, Greece

this paper, we investigate the realistic scenario where $\mathcal{P}(A)$ is a linear function of the user preferences, and $\mathcal{Q}(A)$ is a function of the popularity of the items.

Technically, we show that in the general case the CollectiveRecs problem is NP-hard. For the function $\mathcal{P}(A)$ we consider, we show that there is a class of $\mathcal{Q}(A)$ functions for which the CollectiveRecs problem can be solved optimally in polynomial time.

We also show that if there exist polynomial-time algorithms for maximizing $\mathcal{P}(A)$ and $\mathcal{Q}(A)$ independently, (i.e., solving the CollectiveRecs problem for $\lambda = 1$ and $\lambda = 0$), then we can find an assignment that is an $\frac{1}{2}$-approximation to the optimal assignment for CollectiveRecs for any value of λ. This is a result that holds independently of the choice of $\mathcal{P}(A)$ and $\mathcal{Q}(A)$ functions. We apply this result in order to design an $\frac{1}{2}$-approximation algorithm for the functions we consider. We also design efficient and effective heuristics that work reasonably well in practice.

Our experimental evaluation with real data coming from `Yelp`[1], demonstrate the efficacy and the efficiency of our algorithms for our problem. More specifically, we demonstrate that our algorithms consistently make recommendations that lead to high collective satisfaction, which cannot be achieved by algorithms that take into consideration only the user preferences.

Contributions: Our contributions in this work can be summarized as follows:

- We introduce the novel concept of *collective recommendations*, which take into account both the user preferences, and the quality of an item as a result of the recommendations to the users.
- We formally define the CollectiveRecs problem that aims to maximize the collective satisfaction of the users. We prove that it is in general NP-hard, but under certain conditions it can be approximated within a factor of $\frac{1}{2}$. We identify classes of functions for which the CollectiveRecs problem can be solved optimally.
- Guided by the theoretical analysis, we design approximation algorithms for the general CollectiveRecs problem, as well as efficient heuristics.
- We experiment with real review data from `Yelp`, and we demonstrate the efficiency and the efficacy of our algorithms. We also make our code available at: `cs-people.bu.edu/behzad/collective-recs.tar.gz`.

Roadmap: The rest of the paper is organized as follows. After reviewing the related work in Section 2, we describe our model in Section 3 and our algorithmic problem in Section 4. We provide the algorithmic framework for solving CollectiveRecs in Section 5 and give a thorough experimental evaluation in Section 6. We conclude the paper in Section 7.

[1] http://www.yelp.com/dataset_challenge

2 Related work

Recommender systems have been studied extensively in the past decades, both in research and in practice. In [5, 16, 1] there is a survey of the different techniques and applications. These techniques aim at identifying the best item to recommend to a user, without taking into account the effect of the recommendation to the overall experience in the system. There is also substantial work on location recommendations (e.g., see [23]), an application which we also consider, but it is mainly focused on adding spatial constraints in the recommendations, and implementing them efficiently. Our work builds on top of such recommender systems in order to improve the collective experience of the users.

The work most closely related to ours is the work on activity recommendations, where the social utility of the formed group is also considered [11, 7, 18]. This can be thought of a special case of our formulation where the $\mathcal{Q}()$ function depends on the social network of the participants. Our formulation is more general, and we study a different class of quality functions $\mathcal{Q}()$. Also, in [21, 22], they consider the problem of recommending questions in a discussion forum for Massive Open Online Courses (MOOCs), where they aim to maximize the expertise and minimize the load of the experts. We note that in that setting the goal is to minimize the load of a user, rather than maximize the quality, that is, the load has always a negative effect on the user. Furthermore, that paper considers a specific load function that makes the problem tractable.

The notion of collective experience appears in the case where we want to recommend an item to a group of users, in which case we need to aggregate the different preferences in order to satisfy all users as well as possible [8, 15, 3, 11, 7, 6, 20]. In this case, only one item is recommended to a group of users, and the overall quality of the item for the group is a function of only the preferences of the users. The interactions between users are also taken into account in the case of social recommendations, where we use the social network of the users to determine their preferences (see [17] for a survey). In this case the social network provides additional information for determining the preference of the users for the items, but it does not determine the quality of the item.

Our work is also related to the problem of contextual recommendations (see [2] for a survey in the field), where information additional to the preferences of the

user is used to determine the best recommendation. Typically, this information relates to previous actions of the user, such as a query posted by the user, or to environmental variables, such as time or location. None of these works considers the collective user experience as part of the context.

There are different techniques for finding and assignment of items to users under load constraints. For example, in [9], they consider the problem of finding the best assignment of papers to reviewers under hard load constraints. This is different from our framework, where we assume that the load affects the quality of the recommendation. Our problem is also related to facility-location problems where the cost of opening a facility depends on the final load [12]. However, the problem considered in this case is a minimization problem, that cannot be directly mapped to our problem. Finally, our problem is related to the welfare maximization problem [4], where we want to find an assignment of items to buyers such that the overall utility of the buyers is maximized. In our case the buyers are the items, and the utility function is defined over subsets of items, as the sum of the user preferences plus a function of the size of the subset. The work in this area focuses mostly on submodular functions [19], and online variations of the problem [10].

3 The model

In this section, we define the fundamental concepts we will use throughout the paper and define the necessary notation.

Users, items and preferences: Throughout, we will consider a set I of m items and a population of users U with $|U| = n$. In addition to the regular items in I, we also assume that there is an extra item $\diamond$, which intuitively corresponds to "null" item which is of no interest or value to the users. We always assume that $\diamond \in I$. We use the "null" item to model the case where the system does not produce a recommendation for some user.

We assume that every user i has a preference score for every item j, which we denote by $p(i, j)$; the higher the value of $p(i, j)$ the more user i likes item j. We assume that $p(i, \diamond) = 0$.

Assignment of users to items: An assignment A maps every user to exactly one of the items in I (including the $\diamond$ item which represents no assignment for the user). That is, for every user $i \in U$, $A(i) \in I$.

Collective satisfaction: Given an assignment A, the *collective satisfaction*, denoted by $\mathcal{C}(A)$, is a weighted function of two terms: (a) a term that measures the alignment of the recommendations to the user preferences, denoted by $\mathcal{P}(A)$, and (b) a term that evaluates the effect of the assignment on the quality of the item as experienced by the users assigned to it, denoted by $\mathcal{Q}(A)$. Thus,

$$\mathcal{C}(A) \quad = \quad \lambda \mathcal{P}(A) + (1 - \lambda)\mathcal{Q}(A).$$

where $0 \leq \lambda \leq 1$. Depending on the importance that one wants to place on individual preferences or quality of service one can tune the $\mathcal{C}(A)$ function by changing the value of λ.

While many of the results we discuss in the paper are general and do not depend on the specific forms of these functions, we work here with the following specific preference and quality functions.

Preference function: Given an assignment A, we use $\mathcal{P}(A)$ to denote the function that evaluates the satisfaction of the users from assignment A based on their preferences. We call this function the preference satisfaction (or simply preference) function. A simple way to define $\mathcal{P}$ is as a linear function of the individual user preferences:

$$\mathcal{P}(A) = \sum_{i \in U} p(i, A(i)).$$

Quality function: Here we assume that the quality of service that users receive depends on the item they are assigned to, but also on how many other people are assigned to the same item. Let $q_j(x)$ be the quality of service that item j can deliver to any of its users when x users are assigned to it. Then, the overall quality of an assignment A is:

$$\mathcal{Q}(A) = \sum_{i \in U} q_{A(i)}\left(\ell_{A(i)}\right) = \sum_{j \in I} \ell_j^A q_j\left(\ell_j^A\right).$$

In the above equation, we use ℓ_j^A to denote the *popularity*, i.e., the number of users assigned to item j under assignment A. When A is clear from the context we simply use ℓ_j instead of ℓ_j^A.

Throughout we assume that $q_\diamond(x)$ is 0 for every integer x. The form of the quality function $q_j(x)$ depends on the type, the nature and the capacity of the different items. In our case, the quality functions q_j can be any function of the popularity of item j. Different functions can be used to model different scenarios. We discuss the choice of the quality function in detail in Section 6.

4 The CollectiveRecs problem

In this section, we define our problem, prove that it is NP-hard and also demonstrate some general approximability results, which we will exploit when designing our algorithms in the next section.

4.1 Problem definition Given the above definitions, we define the CollectiveRecs problem as the problem of finding the assignment A such that the collective satisfaction of the user base is maximized. Formally, this problem is defined as follows:

Problem 1. (CollectiveRecs) *Given n users, m items, and user preferences p, find an assignment A such that*

$$\mathcal{C}(A) = \lambda\mathcal{P}(A) + (1-\lambda)\mathcal{Q}(A)$$

is maximized.

Although the definition of the CollectiveRecs problem assumes that every user is assigned to one item, we can extend it to find assignments where every user is assigned to a multiple items. In this case, we assume that the user picks one of the recommended items uniformly at random and we measure the expected collective user satisfaction.

4.2 Complexity results First, we show here that in the general case, the CollectiveRecs problem is NP-hard.

Theorem 4.1. *The quality-maximization problem is NP-hard.*

The proof of this theorem is provided in the Supplementary material.

Although the CollectiveRecs problem is NP-hard in general, there is a large class of instances that can be solved in polynomial time. This is summarized in the following theorem, the proof of which is given in the Supplementary material.

Theorem 4.2. *The* CollectiveRecs *problem can be solved optimally when the quality function q_j for every item $j \in I$ is concave and decreasing.*

4.3 Approximability results In this section, we discuss a general approximation result for the CollectiveRecs problem. The result is stated in the following theorem, the proof of which is given in the Supplementary material.

Lemma 4.1. *If the instances of the* CollectiveRecs *problem for $\lambda = 0$ and $\lambda = 1$ can be solved optimally in polynomial time, then there exists an $\frac{1}{2}$-approximation algorithm for the* CollectiveRecs *problem for any value of $\lambda \in (0,1)$.*

Theorem 4.3. *There exists a polynomial-time $\frac{1}{2}$-approximation algorithm for the* CollectiveRecs *problem.*

Proof. According to Lemma 4.1, it is enough to show that the CollectiveRecs problem for $\lambda = 0$ and for $\lambda = 1$ can be solved optimally in polynomial time. We prove that these two algorithms exist below.

A polynomial-time algorithm for $\lambda = 1$: In this instance we ignore the quality of service of the assignment and we only focus on maximizing the preferences of users, $\mathcal{P}(A)$. This instance of the CollectiveRecs problem can be solved efficiently in time linear to the size of the input, i.e., $O(nm)$. This optimal solution is obtained by assigning each user to the items she prefers the most.

A polynomial-time algorithm for $\lambda = 0$: In this case, the goal is to maximize the quality of service that users receive. Formally, we seek to maximize $\mathcal{Q}(A) = \sum_{j\in I} \ell_j q_j(\ell_j)$.

This instance of the CollectiveRecs problem (i.e., when $\lambda = 0$) can also be solved in polynomial time using dynamic programming. For this, let us define $\mathbf{Q}(i,j)$ to denote the maximum quality one can achieve by assigning i individuals only to the first j items. Now $\mathbf{Q}(i,j)$ can be expressed recursively as follows:

(4.1)
$$\mathbf{Q}(i,j) = \max_{x=0,\ldots,\min\{i,\tau\}} \{\mathbf{Q}(i-x, j-1) + xq_j(x)\},$$

where τ is an upper bound any item can have (e.g., an upper bound on the number of customers a restaurant can facilitate). In the worst case, $\tau = O(n)$, but in practice $\tau \ll n$. The corner cases for the dynamic programming are: $\mathbf{Q}(i,0) = 0$ for every $i \in U$ and $\mathbf{Q}(0,j) = 0$ for every $j \in I$. The above dynamic-programming algorithm solves the CollectiveRecs problem optimally when $\lambda = 0$ in time $O(nm\tau)$.

An upper bound: An upper-bound for the CollectiveRecs problem with paramter λ can be computed by solving the problem once for $\lambda = 0$ and once for $\lambda = 1$, and combining the obtained objective values. Clearly, this does not yield a feasible solution since the optimal assignments for $\lambda = 0$ and $\lambda = 1$ may not agree.

As a consequence we have the following:

Corollary 4.1. *If there is an α_0-approximation (resp. α_1-approximation) algorithm for the* CollectiveRecs *problem with $\lambda = 0$ (resp. $\lambda = 1$), with $\alpha_1, \alpha_0 < 1$, then there is an $\frac{1}{2} \times \min\{\alpha_1, \alpha_2\}$-approximation algorithm for the* CollectiveRecs *problem for any $\lambda \in (0,1)$.*

5 Algorithms

In this section, we discuss algorithms for the general CollectiveRecs problem.

The `Pref-Only` algorithm: This algorithm simply assigns every user i to his most-preferred item. That is, $A(i) = \arg\max_{j\in I} p(i,j)$.

This algorithm is optimal when $\lambda = 1$ and when there is no capacity constraint τ, which corresponds to the ultimate upper bound on the number of users that can be assigned to a single item. In this case, the running time of this algorithm is simply $O(nm)$.

In practice, it is reasonable to impose an upper bound to every item (e.g., say that no more than τ people can go to a restaurant). In the presence of such a bound the `Pref-Only` algorithm needs to solve a maximum-weight matching problem, on the bipartite graph that has as nodes users and items and there are weighted edges with weight $p(i, j)$ between every user i and item j. This problem can be solved in polynomial time using the Hungarian algorithm in time $O\left(n^3\right)$, assuming that $n = \max\{n, m\}$.

We can solve the same problem more efficiently by greedily picking the user-item pair (i, j) with the highest $p(i, j)$ value such that i has not yet been matched and the number of users assigned to j are less than τ. This algorithm is an $\frac{1}{2}$-approximation algorithm to the maximum-weight matching problem [13] and runs in time $O(nm \log(nm))$.

The `DP+Matching` algorithm: The `DP+Matching` algorithm is a two-step algorithm. In the first step, it uses a dynamic-programming algorithm for finding the optimal number of users that need to be assigned to each item. This step optimizes only the quality of service that the items can offer to the people they are assigned to them. The dynamic-programming routine we use is the one described by recursion (4.1) in the proof of Theorem 4.3. The output of this step is a number x_j associated with each item j. This number corresponds to the number of users that the algorithm should assign to this item in the end of its execution. As we discussed in the proof of Theorem 4.3, the running time of this step of the algorithm is $O(nm\tau)$, where in theory $\tau = O(n)$, but in practice is a small constant.

Given the number of users that should be assigned to each item, the second step of the `DP+Matching` algorithm actually solves a matching problem, where the actual user preferences are taken into consideration. In this step every user is assigned to one item and every item has a hard constraint on the number of people it can cater – item j can cater x_j users, where x_j is the number reported by the dynamic-programming routine of the first step. The goal of this step is to respect these constraints and find an assignment that maximizes $\mathcal{P}(A)$, subject to these constraints. As before, this step can be done exactly using the Hungarian algorithm in time $O(n^3)$, or approximately in time $O(nm \log(nm))$.

We also consider a variant of `DP+Matching`, which we call `DP+Random`. The `DP+Random` algorithm computes the optimal number of users that need to be assigned to each item similar to `DP+Matching`. However, in the second step, it does not invoke a matching routine, but rather does a random assignment of people to items respecting the constraints on the number of people to be assigned to each facility in time $O(n)$. The `DP+Random` algorithm acts as the opposite extreme of the `Pref-Only` algorithm: While `Pref-Only` optimizes solely for the preference function, disregarding the quality of the assignment, `DP+Random` optimizes solely for the quality function, disregarding the individual preferences.

Discussion: Given Lemma 4.1 and Theorem 4.3, we have the following result.

COROLLARY 5.1. *The algorithm that runs both `Pref-Only` and `DP+Matching` (or `DP+Random`) and takes the best of the two is an $\frac{1}{2}$-approximation algorithm for the* COLLECTIVERECS *problem when $\lambda \in (0, 1)$.*

The `Greedy` algorithm: `Greedy` constructs an assignment A by assigning one user at a time. That is, at every iteration t the algorithm makes the assignment of a user to an item such that the collective satisfaction $\mathcal{C}(A^t)$ from the partial assignment up to this iteration is maximized. The greedy assignment can be implemented efficiently by using a maxheap data structure. More precisely, for every possible assignment of user i to item j, we create an entry in the maxheap with a value representing how much this assignment contributes to the objective function. In each iteration, the algorithm removes the top entry from the heap, and makes the corresponding assignment if (a) the user is not assigned before and (b) if number of assigned users to the item is smaller than τ. After making the assignment, the algorithm needs to update the value of those entries which are associated with the item j – there are at most n such entries. Given that there are $m \times n$ entries in the maxheap, the cost of creating the heap is $O(mn \log(mn))$. `Greedy` may do up to $m \times n$ iterations, and in each iteration it may need to update the weight of n entries which takes $O(n \log(n))$. Thus the overall running time of the `Greedy` is $O(mn^2 \log(n))$.

6 Experiments

In this section, we experimentally explore the efficacy of our algorithms for the COLLECTIVERECSproblem using a number real-world datasets which we describe next. Our code is available at: `cs-people.bu.edu/behzad/collective-recs.tar.gz`.

Datasets: For our experiments, we use the `Yelp` Academic Challenge data[2] which contains the ratings and

[2] `https://www.yelp.com/dataset_challenge`

Datasets	n	m	Missing entries original	Missing entries completed	# of ratings per user	# of ratings per business	Avg. ratings
`Charlotte`	3,679	1,577	1.0%	42.3%	651.0	1,518.8	3.85
`Scottsdale`	5,116	1,447	0.9%	41.9%	606.8	2,145.4	3.87
`Phoenix`	9,778	2,901	0.4%	41.6%	1,206.2	4,065.8	3.93

Table 1: Statistics on the datasets.

reviews of customers on different businesses. More precisely, we focus on the three cities for which the dataset contains a large number of businesses and reviews, namely `Charlotte`, `Scottsdale` and `Phoenix`, and we use these subsets in our experiments. Throughout, we view each customer as a user and each business as an item. Also, the reported ratings correspond to the preference of users for different items.

These datasets are extremely sparse as each customer only rates and reviews a limited number of businesses. To improve the quality of our dataset, we use an item-based *nearest-neighbor* collaborative filtering technique for matrix completion [1] which can be described as follows. If a user i has no rating on item j, we first find the top $k = 3$ similar items (using cosine similarity) that are rated by user i. We compute the missing entry as the weighted average of user i's ratings for the neighboring items, where weights are proportional to their similarity with item j. For the entries that we cannot fill, we replace the missing entry with value 0 to reflect that the user has no interest in the item[3].

Table 1 summarizes some insightful statistics about our datasets. The first two columns show the number of users and items respectively. The next two columns report the percentage of missing entries before and after running the matrix completion. Finally, the last three columns report the number of available ratings per user, the number of available ratings per item, and the average rating respectively.

Quality functions: While user preferences can be extracted from their ratings, we still need to specify the quality functions $q_j(x)$ associated with each item. For our experiments, we try a wide range of quality functions with different distinct behaviors which we discuss below.

- `Exp-Decay` can be written as: $q_j(x) = e^{-\alpha_j x}$, where α_j is sampled uniformly from the interval $[0, 0.1]$.
- `Inc-Logistic` is written as: $q_j(x) = 1/(1 + e^{\alpha_j * (\mu_j - x)})$, where α_j and μ_j are sampled uniformly from intervals $[0, 0.2]$ and $[5, 30]$ respectively.
- `Dec-Logistic` can be expressed as: $q_j(x) = 1 - 1/(1+e^{\alpha_j * (\mu_j - x)})$, where α_j and μ_j are sampled uniformly from intervals $[0, 0.2]$ and $[5, 30]$ respectively.
- `Gaussian` is essentially a scaled Gaussian curve and can be expressed as: $q_j(x) = (e^{-(x-\mu_j)^2})/(2\sigma_j^2)$, where μ_j and σ_j are sampled uniformly from the interval $[5, 30]$ and $[3, 13]$ respectively.
- `Concave-Squared` can be expressed as: $q_j(x) = \max\{0, -(x/\alpha_j - 1)^2 + 1\}$, where α_j is sampled uniformly from the interval $[5, 20]$.
- `Convex-Squared` is written as: $q_j(x) = \min\{1, (x/\alpha_j - 1)^2\}$, where α_j is sampled uniformly from interval $[5, 20]$.

Although the choice for the parameters of each function may seem arbitrary, they are selected such that we observe reasonable behaviors when the population is between 0 and 40 as we set $\tau = 40$ in our experiments. Functions `Exp-Decay` and `Dec-Logistic` are strictly decreasing. These types of functions describe scenarios where the fewer the people the higher the quality of service (e.g., a car repair shop). The `Exp-Decay` assumes an exponential drop in quality, while `Dec-Logistic` models a steady decrease that becomes sharp when a critical mass is reached. The `Inc-Logistic` models the opposite effect. The function captures scenarios where more participation improves the experience of users (e.g., a marathon race). However, when a critical mass is reached, the additional crowd has a diminishing effect on quality. The `Concave-Squared` and `Gaussian` are examples of unimodal functions. Such functions occur in settings where users generally enjoy sharing an experience with a group but when the group size grows over a threshold the quality of service drops due to over-population (e.g., a bar, or a concert venue). The `Convex-Squared` captures the exact opposite trend, where mid-size groups are not as desirable as small or large groups. For instance, students in a small study group with a tutor work nicely and have the tutor's attention. As the group grows the attention of the tutor gets divided and the student satisfaction drops. However in a large group the students benefit from the ability to form smaller subgroups with students with similar interests, and the experience improves. Finally, since

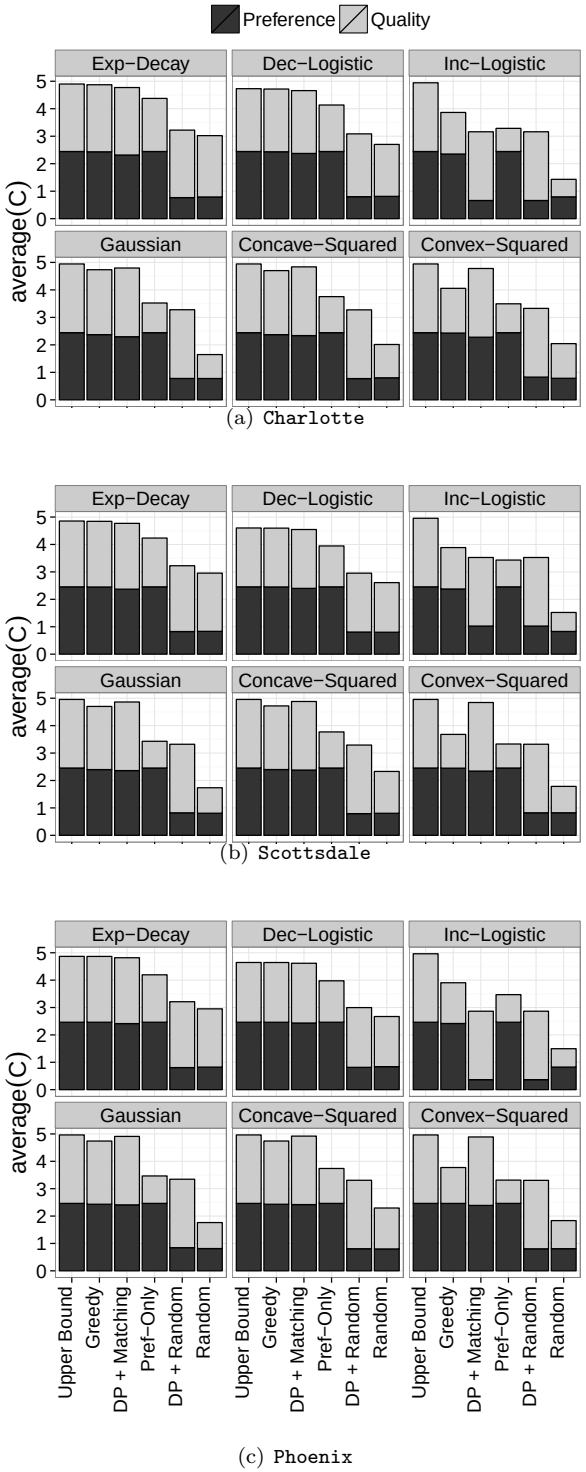
[3] An implementation of this matrix completion technique can be found in the python package `fancyimpute`.

the user ratings take values between 0 and 5, we scale each quality function by a factor of 5 to make preference values and service qualities comparable.

Experimental results: Here, we evaluate all algorithms for all previously mentioned datasets considering the quality functions discussed above. To better evaluate our algorithms, we compare our results against a theoretical upper bound which we compute as follows. Let A_p and A_q be the optimal assignments for maximizing preferences and quality, computed using `Pref-Only` and `DP+Matching` respectively. Then a natural upper bound is given by $\mathcal{P}(A_p) + \mathcal{Q}(A_q)$. We also compare our results, against a random baselines which we refer to as `Random`. The `Random` algorithm assigns each user to an item selected uniformly at random.

Figure 1 illustrates the relative performance of the algorithms for different quality functions and all three datasets. More specifically, it highlights how much of the obtained user satisfaction is due to preferences (i.e., the dark shade) and how much is due to the experienced quality (i.e., the light shade). The y-axis shows the average collective satisfaction of users. For this experiment, we set the value of λ to 0.5 so that we put a balanced emphasis on the importance of preferences and experienced qualities.

We can observe a number of interesting trends in Figure 1. First, we can observe that `Greedy` and `DP+Matching` algorithms outperform the other algorithms with a performance very close to the theoretical upper-bound. Also, we can observe that both `DP+Matching` and `DP+Random` algorithms (as expected) are very effective in maximizing the experienced service quality. Nevertheless, the significant difference between the two algorithms demonstrates that the second step in the `DP+Matching` algorithm is quite effective. Finally, we can observe that the `DP+Matching` algorithm outperforms the `Greedy` algorithm when the quality function is of type `Gaussian`, `Concave-Squared` or `Convex-Squared`. A common characteristic of these functions is that they are not monotone. As a result, the greedy algorithm is completely oblivious to the changes in the service quality that arise with assigning users to items. In case of monotone functions, we can observe that both `Greedy` and `DP+Matching` algorithm are performing well except for the `Inc-Logistic` function which is a monotone increasing function. When dealing with such functions, the first step of the `DP+Matching` algorithm decides to assign all users to a limited number of items since the more users assigned to an item, the higher the quality. By limiting the number of items, the `DP+Matching` algorithm fails to maximize the user preferences as users may not be interested in this limited set. In fact, we can see that both `DP+Random` and



(a) `Charlotte`

(b) `Scottsdale`

(c) `Phoenix`

Figure 1: Average collective satisfaction per user for different algorithms ($\lambda = 0.5$).

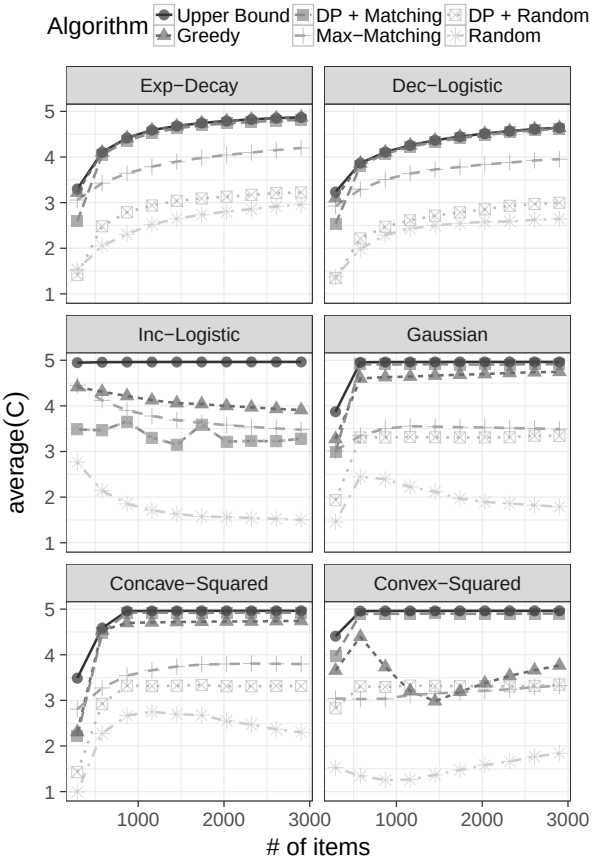


Figure 2: Average collective satisfaction per user for different values of λ (`Phoenix` dataset).

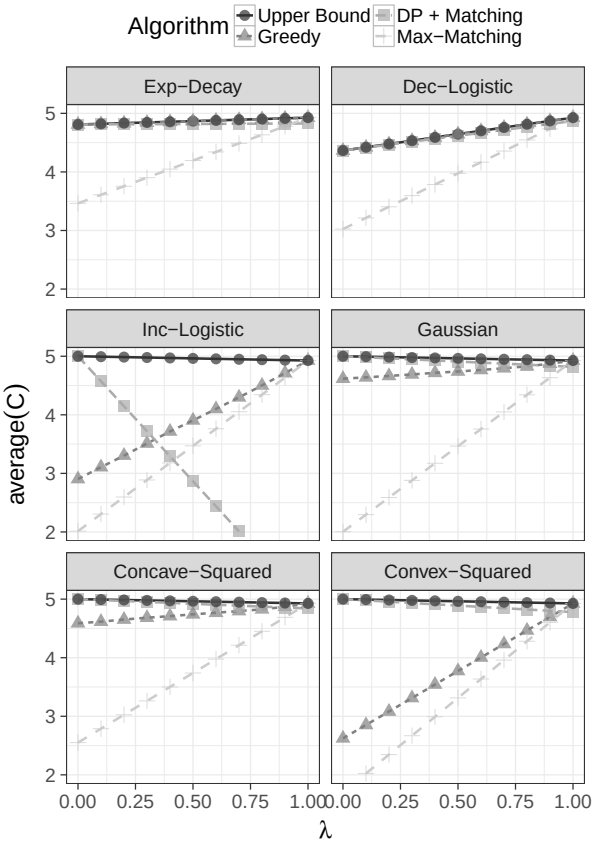


Figure 3: Average collective satisfaction per user for different number of items (`Phoenix` dataset, $\lambda = 0.5$).

`DP+Matching` achieve the same level of user preference.

Figure 2 shows how well algorithms perform for different values of λ. We present our results only for the `Phoenix` dataset as the results for the other datasets shows a similar trend. Also, we drop the `Random` and `DP+Random` algorithms from the plot as their performance was significantly worse than the rest of the algorithms. The x-axis shows the different values of λ and the y-axis shows the average collective satisfaction of users. Note that $\lambda = 1$ implies that we only care about maximizing user preferences which can be done optimally using the `Pref-Only` algorithm which explains why the performance of this algorithm increases with λ. Nevertheless, it is always outperformed by the other two algorithms (excluding the case of `Inc-Logistic` function). As before, we can observe that the performance of `Greedy` and `DP+Matching` algorithms is close to the theoretical upper bound with `DP+Matching` outperforming the `Greedy` for unimodal functions. Finally, as explained earlier, the poor performance of `DP+Matching` in the `Inc-Logistic` case is due to the fact that maximizing the service quality limits the flexibility of the algorithm in maximizing the user preference.

Finally, we study how well our algorithms perform when the number of items in the dataset varies. To do so, we use the `Phoenix` dataset and we randomly sample a fixed number of items and solve the CollectiveRecs problem on this subset. Figure 3 shows the average satisfaction of users (y-axis) for different number of available items (x-axis). Note that we generally expect the algorithms to perform better when more items are available, and we see this general trend in our informed (i.e., non-random) algorithms. As before, we can observe that `Greedy` and `DP+Matching` algorithm's performance is close the theoretical upper bound. Note that in the case of `Convex-Squared` function, the `Greedy` algorithm has an unusual trend, which can be explained as follows. Initially, when the number of items is small, `Greedy` is forced to assign many users to items, and as a result it passes the low point in the `Convex-Squared` curve, achieving high satisfaction. As the number of available items increases, the `Greedy` acts myopically, and it avoids assigning many users to an item to avoid the drop in the service quality in the `Convex-Squared` func-

tion. As a result for medium-sized number of items the satisfaction drops. When the number of items becomes large (larger than 1000) the myopic approach pays off, since small number of users per item results in higher quality, causing an increasing trend.

7 Conclusions

In this paper we introduced collective recommendations, where we recommend items to users taking into account both the user preferences and the effect of the collective behavior on the overall user experience. More specifically, we formalized the CollectiveRecs problem, which aims to find an assignment of users to items so as maximize the collective satisfaction. Our problem definition is general and views collective satisfaction as the weighted sum of two terms: (a) a preference term that measures the alignment of the assignment with the user preferences and (b) a quality term that measures the quality of the item as a function of the collective behavior of the user-base. We proved that this problem is in general NP-hard, but we identified a class of quality functions for which it can be solved in polynomial time. We also demonstrated a technique for designing bounded-factor approximation algorithms for the general problem. In addition to that, we explored other efficient heuristics that appear to work well in practice. Our experiments with real data demonstrated the efficiency and the efficacy of our algorithms in practice.

Acknowledgements: This work was funded by NSF awards: # 1421759 and # 1253393.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[3] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawla, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.

[4] Francis M Bator. The simple analytics of welfare maximization. *The American Economic Review*, 47(1):22–59, 1957.

[5] J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez. Recommender systems survey. *Know.-Based Syst.*, 46:109–132, 2013.

[6] Ludovico Boratto and Salvatore Carta. State-of-the-art in group recommendation and new approaches for automatic identification of groups. *Information Retrieval and Mining in Distributed Environments*, 324:1–20, 2011.

[7] Mike Gartrell, Xinyu Xing, Qin Lv, Aaron Beach, Richard Han, Shivakant Mishra, and Karim Seada. Enhancing group recommendation by incorporating social relationship interactions. In *ACM SIGGROUP*, pages 97–106, 2010.

[8] Anthony Jameson and Barry Smyth. Recommendation to groups. In *The Adaptive Web, Methods and Strategies of Web Personalization*, pages 596–627, 2007.

[9] Maryam Karimzadehgan and ChengXiang Zhai. Constrained multi-aspect expertise matching for committee review assignment. In *CIKM*, pages 1697–1700, 2009.

[10] Nitish Korula, Vahab Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. In *STOC*, pages 889–898, 2015.

[11] Keqian Li, Wei Lu, Smriti Bhagat, Laks V. S. Lakshmanan, and Cong Yu. On social event organization. In *ACM SIGKDD*, pages 1206–1215, 2014.

[12] Mohammad Mahdian and Martin Pál. *Universal Facility Location*, pages 409–421. 2003.

[13] Gianmarco De Francisci Morales, Aristides Gionis, and Mauro Sozio. Social content matching in mapreduce. *PVLDB*, 4(7):460–469, 2011.

[14] Breffni M Noone and Anna S Mattila. Restaurant crowding and perceptions of service quality: The role of consumption goals and attributions. *Journal of Foodservice Business Research*, 12(4):331–343, 2009.

[15] Mark O'Connor, Dan Cosley, Joseph A. Konstan, and John Riedl. Polylens: A recommender system for groups of user. In *ECSCW*, pages 199–218, 2001.

[16] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, 2009.

[17] Jiliang Tang, Xia Hu, and Huan Liu. Social recommendation: a review. *Social Network Analysis and Mining*, 3(4):1113–1133, 2013.

[18] Wenting Tu, David Wai-Lok Cheung, Nikos Mamoulis, Min Yang, and Ziyu Lu. Activity-partner recommendation. In *PAKDD*, pages 591–604, 2015.

[19] Jan Vondrak. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, pages 67–74, 2008.

[20] Min Xie, Laks V. S. Lakshmanan, and Peter T. Wood. Breaking out of the box of recommendations: from items to packages. In *RecSys*, pages 151–158, 2010.

[21] Diyi Yang, David Adamson, and Carolyn Penstein Rosé. Question recommendation with constraints for massive open online courses. In *RecSys*, pages 49–56, 2014.

[22] Diyi Yang, Jingbo Shang, and Carolyn Penstein Rosé. Constrained question recommendation in moocs via submodularity. In *CIKM*, pages 1987–1990, 2014.

[23] Josh Jia-Ching Ying, Eric Hsueh-Chan Lu, Wen-Ning Kuo, and Vincent S. Tseng. Urban point-of-interest recommendation by mining user check-in behaviors. In *UrbComp*, 2012.