World Wide Web (2016) 19:1151–1177
DOI 10.1007/s11280-015-0377-6

Bottleneck-aware arrangement over event-based social networks: the max-min approach

Yongxin Tong[1] · Jieying She[2] · Rui Meng[2]

Received: 22 May 2015 / Revised: 11 September 2015 /
Accepted: 5 November 2015 / Published online: 8 December 2015

Abstract With the popularity of mobile computing and social media, various kinds of *online event-based social network* (EBSN) platforms, such as Meetup, Plancast and Whova, are gaining in prominence. A fundamental task of managing EBSN platforms is to recommend suitable social events to potential users according to the following three factors: spatial locations of events and users, attribute similarities between events and users, and friend relationships among users. However, none of the existing approaches considers all the aforementioned influential factors when they recommend users to proper events. Furthermore, the existing recommendation strategies neglect the bottleneck cases of the global recommendation. Thus, it is impossible for the existing recommendation solutions to be fair in real-world scenarios. In this paper, we first formally define the problem of *bottleneck-aware social event arrangement* (BSEA), which is proven to be NP-hard. To solve the BSEA problem approximately,

Electronic supplementary material The online version of this article (doi:10.1007/s11280-015-0377-6) contains supplementary material, which is available to authorized users.

✉ Yongxin Tong
yxtong@buaa.edu.cn

Jieying She
jshe@cse.ust.hk

Rui Meng
rmeng@cse.ust.hk

[1] State Key Laboratory of Software Development Environment and National Engineering Research Center for S & T Resources Sharing Service, School of Computer Science and Engineering and International Research Institute for Multidisciplinary Science, Beihang University, Beijing, China

[2] Department of Computer Science and Engineering, Hong Kong University of Science, Technology, Clear Water Bay, Kowloon, Hong Kong SAR, China

we devise two greedy heuristic algorithms, *Greedy* and *Random+Greedy*, and a local-search-based optimization technique. In particular, the *Greedy* algorithm is more effective but less efficient than the *Random+Greedy* algorithm in most cases. Moreover, a variant of the BSEA problem, called the Extended BSEA problem, is studied, and the above solutions can be extended to address this variant easily. Finally, we conduct extensive experiments on real and synthetic datasets which verify the efficiency and effectiveness of our proposed algorithms.

Keywords Event-based social networks · Social networks · Assignment problem

1 Introduction

In recent years, with the widespread usage of mobile computing and pervasive computing, all kinds of new social media techniques are emerging [12, 26]. In particular, various event-based social networks (EBSNs) [13, 23, 24, 27] are getting popular. Recent success stories include Meetup,[1] Plancast,[2] and Eventbrite.[3] For example, on Meetup, organizers can create different activities, such as career day, social parties, etc., and users can register and attend these activities according to the recommendations from Meetup. Furthermore, Eventbrite is another social event platform for organizers to share information of activities to potential participants. To sum up, EBSNs provide a novel approach for facilitating *online* users to organize and manage *offline* social activities.

Unfortunately, most existing EBSNs are only simple information sharing platforms about social activities and do not provide an intelligent and global arrangement for social activities and potential users. Thus, how to design a global arrangement strategy in EBSNs has become a fundamental issue and attracted much attention in the database and data mining communities recently. Li et al. [14] introduces the social event organization (SEO) problem, which is to assign users to activities to maximize the overall innate and social affinities. However, this work only considers two factors, the similarity of attributes and social friendship among users, for the assignment and neglects the spatial influence between activities and users. Imagine the following scenario. Being a shutterbug and hiking enthusiast, Tony intends to attend photographic and hiking activities organized by Meetup on the coming Saturday. According to the solution of [14], Meetup recommends him the following two related activities: a hiking activity and a photographic party. Although Tony is interested in both activities and has friends in each activity, he has to spend at least 3 hours by taxi to the hiking place from his home and only 0.5 hour to the photographic party venue. Tony faces a dilemma if the arrangement ignores the influence of spatial information. Clearly, the solution of [14] cannot handle the aforementioned scenario and may recommend the hiking activity to Tony. In fact, Tony prefers to attend the photographic party due to its short spatial distance. Therefore, *a reasonable arrangement strategy in EBSNs should seamlessly integrate the following three factors: the location influence between activities and users, the similarity of attributes between activities and users, and the social friendship among users.*

Moreover, organizers of each activity always hope to recruit as many interested users for their activity as possible. However, existing social network recommendation systems

[1] http://www.meetup.com/

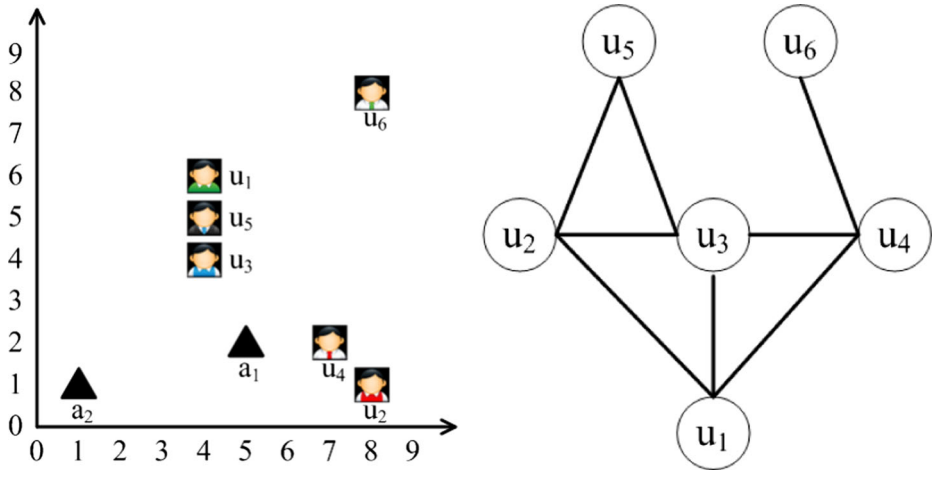
[2] http://plancast.com

[3] http://www.eventbrite.com/

usually adopt recommendation strategies for each single activity instead of providing globally optimal arrangements. Since each user only attends one activity at the same time, it easily results in that some activities only attract few attendees but participants in other activities are excess. A possible global goal may be to maximize the average utility value of all the activities in the platform. However, in this case, the activities whose utility values are below-average may not be satisfied since the utility values of these activities may be arbitrarily low without any guarantee. In other words, many activities may not be organized well and their organizers will be unhappy. Therefore, in order to satisfy all the registered activities in EBSN, a reasonable global goal is to find the optimized arrangement that maximizes the minimum normalized utility of activities. The utility should take all the aforementioned three factors into consideration. In other words, *this optimized goal can keep balance of the satisfactions of different activities in a global view*.

Therefore, a new social event arrangement strategy should not only consider the aforementioned three influential factors, location information, similarity of attributes and social friendship, but should also guarantee a balanced satisfaction of different activities in a globally optimal perspective. In the following, we illustrate a toy example to explain our motivation in details.

Example 1 Suppose we have two activities ($a_1 - a_2$) and six users ($u_1 - u_6$) in an EBSN. We also assume that each activity/user includes a profile, which consists of a list of attributes. For an activity, the corresponding attributes show the category and core tags, etc., of this activity. Similarly, the attributes of users represent preferences of users. According to the attributes of activities and users, we can obtain the similarity of interests between each pair of them. Furthermore, each activity includes a capacity, which is the maximum number of participants. In this example, the capacities of a_1 and a_2 are 4 and 3, respectively (in brackets). In addition, Figure 1a shows the locations of the two activities ($a_1 - a_2$) and the six users ($u_1 - u_6$), and we use Euclidian distance to calculate the spatial utility among them. Figure 1b represents the social friendship among six users ($u_1 - u_6$). If we assume



(a) Locations of Activities and Users (b) Friendship among Users

Figure 1 Location Information and Friendship

that the utility function between a pair of activity and user is the linear interpolation of the normalized factors between their location influence and attribute similarity, the optimal goal is to maximize the minimum normalized utility of one activity such that each assigned user in this activity has at least a friend in Figure 1b, who is also assigned to the same activity. We present the utility score between each pair of activity and user in Table 1. A feasible arrangement is shown in bold fonts in Table 1, where the minimum normalized utility score of activity is 0.493 (a_2).

As discussed in the motivation example, a novel social event arrangement strategy, called *bottleneck-aware social event arrangement* (BSEA), is introduced. Specifically, given a set of activities and a set of users, each activity includes the location information, the attributes and the capacity, and each user has the location information, the attributes and the corresponding social friendship, the BESA problem is to find an arrangement, such that the minimum normalized utility score of all assigned users to one activity is maximized, while the capacity and social friendship constraints are satisfied. In this paper, the social friendship constraint is specified that each user in one activity has at least one friend in the social graph, who is also assigned to the same activity.

If the aforementioned capacity and social friendship constraints are omitted, the ESBA problem can be reduced to a classical problem, bottleneck assignment problem [2]. Even if the ESBA problem is similar to the traditional bottleneck assignment problem, they are essentially different from the computational complexity view. The bottleneck assignment problem can be solved by several classical polynomial-time algorithms, e.g. the threshold algorithm [7]. However, the ESBA problem is NP-hard due to the two additional constraints. The capacity condition makes the bottleneck assignment problem a many-to-many assignment problem. Moreover, the social friendship constraint also increases the computational complexity. Therefore, the two new conditions are the main challenges to solve the ESBA problem. To the best of our knowledge, our work is the first study about the ESBA problem. In particular, we make the following contributions.

- We introduce a new social event arrangement problem and propose the formal definitions of the bottleneck-aware social event arrangement (BSEA) problem and a variant of the BSEA problem.
- We prove that the BSEA problem and its variant are NP-hard, respectively.
- For the BSEA problem, we devise a baseline algorithm, two heuristic algorithms, *Greedy* and *Random+Greedy*, and a local-search-based optimization technique. In particular, the *Greedy* algorithm is more effective but less efficient than the *Random+Greedy* algorithm in most cases. Furthermore, the aforementioned solutions can be extended to address the variant of the BSEA problem easily.
- We conduct extensive experiments on real and synthetic datasets which verify the efficiency and effectiveness of our proposed algorithms.

The rest of the paper is organized as follows. In Section 2, we formally formulate the BSEA problem and its variant, called the Extended ESEA problem. In Section 3,

Table 1 Utility between activities and users

	u_1	u_2	u_3	u_4	u_5	u_6
a_1 (4)	**0.79**	0.72	**0.80**	**0.76**	0.92	**0.62**
a_2 (3)	0.67	**0.68**	0.51	0.67	**0.80**	0.53

for the BSEA problem, we devise a baseline algorithm, two greedy heuristic algorithms, and a local-search-based optimization technique. Moreover, the corresponding algorithms extended by the aforementioned solutions are proposed to address the Extended ESEA problem in Section 4. Section 5 shows extensive experiments on both synthetic and real datasets. The related works are reviewed in Section 6. We finally conclude this paper in Section 7.

2 Problem statement

We first introduce several basic concepts and then formally define the bottleneck-aware social event arrangement (BSEA) problem.

Definition 1 (Activity) An activity is defined as $a =< \boldsymbol{l}_a, \boldsymbol{t}_a, \delta_a >$ where $\boldsymbol{l}_a =< l_a^1, l_a^2 >$ is a 2-dimensional vector used to represent the longitude and the latitude of the activity, $\boldsymbol{t}_a =< t_a^1, t_a^2, \cdots, t_a^d >$ with $t_a^i \in [0, T], \forall 1 \leq i \leq d$ is a d-dimensional vector used to record attribute values of the activity, and δ_a is the capacity of the activity, namely the maximum number of attendees (participants) of the activity.

Definition 2 (User) A user is defined as $u =< \boldsymbol{l}_u, \boldsymbol{t}_u >$ where $\boldsymbol{l}_u =< l_u^1, l_u^2 >$ is a 2-dimensional vector used to represent the location of the user, and $\boldsymbol{t}_u =< t_u^1, t_u^2, \cdots, t_u^d >$ with $t_u^i \in [0, T], \forall 1 \leq i \leq d$ is a d-dimensional vector to describe attribute values of the user.

Definition 3 (Social Network) Given a social network $G = (U, F)$, where each vertex $u \in U$ is a user, and any two users (vertices) u and v are connected by an edge $e_{u,v} \in F$ if and only if they are friends mutually.

For any feasible arrangement M of the activities and users, we denote $m(a, u) = 1$ or $\{a, u\} \in M$ as that user u is assigned to activity a, and $m(a, u) = 0$ or $\{a, u\} \notin M$ as that u is not assigned to a. We then define the utility of an assigned pair of activity and user.

Definition 4 (Utility Function) Given a set of activities A and a set of users U, the utility that the user $u \in U$ is assigned to the activity $a \in A$ is measured by the following function

$$\mu(a, u) = \alpha \left(1 - \frac{D(\boldsymbol{l}_a, \boldsymbol{l}_u)}{\widehat{MaxD}}\right) + (1 - \alpha) sim(\boldsymbol{t}_a, \boldsymbol{t}_u) \tag{1}$$

where $\alpha \in [0, 1]$ is a parameter used to balance the spatial distance and the attribute similarity; the spatial distance between a and u is measured by Euclidian distance $D(\boldsymbol{l}_a, \boldsymbol{l}_u)$ and is normalized by $\widehat{MaxD} = MaxD + \epsilon$, where $MaxD$ is the maximum distance between any activity and any user, and ϵ is an arbitrary small positive real number. Furthermore, the attribute similarity can be calculated by any well-known similarity function, e.g. Cosine similarity, Jaccard distance, Overlap distance, etc. In this paper, we choose Cosine similarity as our similarity function, which is shown as follows.

$$sim(\boldsymbol{t}_a, \boldsymbol{t}_u) = \frac{\boldsymbol{t}_a \cdot \boldsymbol{t}_u}{\sqrt{|\boldsymbol{t}_a| \cdot |\boldsymbol{t}_u|}} \tag{2}$$

According to the aforementioned utility function, we define the concept of normalized utility of an activity.

Definition 5 (Normalized Utility of An Activity) Given an activity a, a set of users U and an arrangement M, the normalized utility of a is represented by the following equation

$$Ave_M(a) = \frac{\sum_{u \in U,(a,u) \in M} \mu(a,u)}{\delta_a} \tag{3}$$

where δ_a is the capacity of the activity a.

We define our BSEA problem as follows.

Definition 6 (BSEA Problem) Given a set of activities A, each a of which is associated with location $\boldsymbol{l}_a$, attributes $\boldsymbol{t}_a$, and capacity δ_a, a set of users U, each u of which is associated with location $\boldsymbol{l}_u$ and attributes $\boldsymbol{t}_u$, and a social network $G = (U, F)$, the BSEA problem is to find an arrangement M among the activities and users to maximize $\min_{a_i \in A}\{Ave(a_i)\}$ such that

- $\sum_u m(a,u) \leq \delta_a, \forall a \in A$
- Each user $u \in M(a)$ has at least one another user $v \in M(a)$ such that $e(u,v) \in F$, $\forall a \in A$

In particular, we call the activity with the smallest normalized utility score, i.e. $\arg\min_{a_i \in A}\{Ave(a_i)\}$, the bottleneck activity. The notations of symbols are summarized in Table 2. And we have the following theorem that states the hardness of the BSEA problem.

Theorem 1 *The BSEA problem is NP-hard.*

Proof In order to complete the proof, we reduce the PARTITION problem, which is a well-known NP-complete problem [8], to the BSEA problem. The following is an instance of the

Table 2 Summary of symbol notations

Notation	Description
$A = \{a_1, \cdots, a_{\|A\|}\}$	the set of activities
$\boldsymbol{l}_a =< l_a^1, l_a^2 >$	the longitude and the latitude of a
$\boldsymbol{t}_a =< t_a^1, \cdots, t_a^d >$	the attribute values of a
δ_a	the capacity of a
δ_u	the capacity of u
$U = \{u_1, \cdots, u_{\|U\|}\}$	the set of users
$\boldsymbol{l}_u =< l_u^1, l_u^2 >$	the longitude and the latitude of u
$\boldsymbol{t}_u =< t_u^1, \cdots, t_u^d >$	the attribute values of u
$G = (U, F)$	the social network based on the set of users U
$MaxD$	the maximum distance between any activity and any user
$\widehat{MaxD}$	$MaxD + \epsilon$
$sim(\boldsymbol{t}_a, \boldsymbol{t}_u)$	the cosine similarity between attribute values of a and u
$\mu(a,u)$	the utility between a and u
M	a social event arrangement
$Ave_M(a)$	the normalized utility of a in the arrangement M

PARTITION problem. Given a set of n positive integers $S = \{s_1, s_2, \cdots, s_n\}$ and a function $\sigma(X) = \sum_{x \in X} x$, the PARTITION problem is to decide if S can be partitioned into two subsets S_1 and S_2 such that $\sigma(S_1) = \sigma(S_2) = \frac{1}{2}\sigma(S)$.

We then construct an instance with two activities of the BSEA problem from the instance of the PARTITION problem accordingly:

(1) The two partitioned sets, S_1 and S_2, correspond to the two activities, a_1 and a_2, respectively. Furthermore, let $\delta_{a_1} = \delta_{a_2} = n$, where n is the number of positive integers in S.

(2) Each positive integers s_i in the set S corresponds to a user u_i in U such that $\mu(a_1, u_i) = \mu(a_2, u_i) = \frac{s_i}{\max_j\{s_j\}}$ where $\mu(\cdot, \cdot)$ is the utility score between the activity and the user.

(3) All users are mutually friends.

Let a parameter $K = \frac{\sigma(S)}{2n \times \max_j\{s_j\}}$, where $\max_j\{s_j\}$ is the maximum positive integer in the set S consisting of n positive integers, which is used to normalize the utility values to [0, 1]. The corresponding decision problem of the BSEA problem, in the given instance, is to decide whether $min\{Ave(a_1), Ave(a_2)\}$ is equal to K. Thus, it is easy to see that the instance of the PARTITION problem is YES if and only if the instance of the BSEA problem is YES. In general, if a decision problem is an NP-Complete problem, then the corresponding optimization problem is NP-Hard. Therefore, the BSEA problem is NP-Hard, which completes the proof. □

In the aforementioned BSEA problem, each user is required to attend one activity only. In some real-world scenarios, however, some users hope to attend multiple activities. Therefore, we present an extended BSEA problem where each users can attend multiple activities.

Definition 7 (Extended BSEA Problem) Given a set of activities A, each a of which is associated with location $\boldsymbol{l}_a$, attributes $\boldsymbol{t}_a$ and capacity δ_a, a set of users U, each u of which is associated with location $\boldsymbol{l}_u$, attributes $\boldsymbol{t}_u$ and capacity δ_u, and a social network $G = (U, F)$, the extended BSEA problem is to find an arrangement M among activities and users to maximize $min_{a_i \in A}\{Ave(a_i)\}$ such that

- $\sum_u m(a, u) \leq \delta_a, \forall a \in A$
- $\sum_a m(a, u) \leq \delta_u, \forall u \in U$
- Each user $u \in M(a)$ has at least one another user $v \in M(a)$ such that $e(u, v) \in F$ in $G(U, F), \forall a \in A$

Similar to the BSEA problem, the Extended BSEA problem is proven as an NP-hard problem as follows.

Theorem 2 *The Extended BSEA problem is NP-hard.*

Proof The BSEA problem is a special case of the Extended BSEA problem when each user's capacity is set as one. Based on Theorem 1, we know the BSEA problem is NP-hard, so the Extended BSEA problem is also NP-hard. □

3 Solutions for BSEA

In this section, we present solutions for the BSEA problem. We first present a baseline algorithm. We then propose two non-trivial heuristic algorithms. The first heuristic algorithm is based on a globally greedy strategy, while the second one is based on a locally greedy strategy. More details are presented as follows.

3.1 Baseline algorithm

The baseline algorithm is based on a random strategy, in which we visit each activity $a \in A$ in a random order and find an arrangement for each such a from a set of users that are not yet arranged again in a random way. More specifically, let A' be a shuffle-ordered list of A, and we visit each element a_i in A' in order. We obtain a threshold θ to indicate when we should stop processing a_i as follows.

$$\theta = \left\lfloor \frac{|U|\delta_{a_i}}{\sum_a \delta_a} \right\rfloor \tag{4}$$

Algorithm 1 Baseline

```
   input  : A, U, G
   output: M
 1 A' ← a shuffle-ordered list of A;
 2 foreach a_i ∈ A' do
 3     calculate θ;
 4     for u_j ∈ U do
 5         skip u_j with probability 1 − θ/|U| or if u_j has been arranged;
 6         if u_j has friends in M(a_i) then
 7             add u_j to M(a_i);
 8             add friends of u_j from C(a_i) to M(a_i) if not exceeding θ and δ_{a_i};
 9         else
10             if u_j has friends in C(a_i) then
11                 add u_j and its friends from C(a_i) to M(a_i) if not exceeding θ and
                   δ_{a_i};
12             else
13                 append u_j to the tail of C(a_i);
14         if |M(a_i)| ≥ min{θ, δ_{a_i}} then
15             break;
16 return M
```

That is, we aim to balance the numbers of users arranged to each activity according to θ, so that to balance the $Ave(a)$ value of each activity as much as possible. For each a_i, we process it as follows. We visit each user $u_j \in U$ that has not yet been arranged to any activity with probability $\frac{\theta}{|U|}$. During this process, we maintain an ordered list $C(a_i)$, which stores a list of isolated candidate users that have been visited before. That is, $\forall u_k \in C(a_i), k < j, u_k$ was visited before we visit u_j, and u_k has no friend in the set of users that we have visited, i.e. $\nexists u' \in M(a_i) \cup C(a_i)$ s.t. $e(u_k, u') \in F$. Elements in $C(a_i)$ are stored in the order that they were visited.

We then process each visited u_j as follows. We first check whether u_j has a friend in $M(a)$. If u_j can find its friend(s) in the set of users that have already been arranged to a, i.e. $\exists v \in M(a)$ s.t. $e(u_j, v) \in F$, we safely update $M(a)$ by adding u to $M(a)$. Otherwise, we check whether u_j has any friend in $C(a_i)$. If u_j can find its friend(s) in $C(a_i)$, i.e. $\exists v \in C(a_i)$ s.t. $e(u_j, v) \in F$, and a_i can accommodate at least two more users, i.e. $|M(a_i)| + 2 \leq \delta_{a_i}$, we add both u_j and v into $M(a_i)$ and also remove v from $C(a_i)$. If u_j is added to $M(a_i)$ successfully, we keep checking whether there exist other friends of u_j in $C(a_i)$ and add them into $M(a_i)$ until the size of $M(a_i)$ reaches θ or δ_{a_i}. The iteration for each a_i terminates when the size of $M(a_i)$ reaches θ or δ_{a_i}.

We illustrate the procedure in Algorithm 1. In lines 2-15, we visit each $a_i \in A$ in a random order. Then during each iteration, we visit each unarranged $u_j \in U$ with a certain probability in lines 6-15. If u_j has a friend in $M(a_i)$, we add it to $M(a_i)$ and try to add its friends in $C(a_i)$. Otherwise, we try to find its friends in $C(a_i)$ in line 10. We try to add u_j with her/his friends in $C(a_i)$ to $M(a_i)$ if possible in line 11, and append u_j to the tail of $C(a_i)$ in line 13 otherwise.

Example 2 Back to our running example in Example 1. We first obtain a random shuffle order of A, which is a_2, a_1. We first visit a_2, and calculate $\theta = 2$. Each u is skipped with probability $\frac{2}{3}$, and u_1 and u_2 are skipped. We next visit u_3, which has no friend in $M(a_2)$ as $M(a_2)$ is currently empty. Thus, we push u_3 to $C(a_2)$ and $C(a_2) =< u_3 >$. We then visit u_4, which has friend u_3 in $C(a_2)$. Thus, we add u_3, u_4 to $M(a_2)$. Note that $|M(a_2)| = 2$ reaches the threshold θ, so we continue to visit the next activity a_1. The procedure continues and we have the following final arrangement $M(a_1) = \{u_2, u_5\}, M(a_2) = \{u_3, u_4\}$ with minimum normalized utility score of 0.39.

Algorithm 2 Initialization

```
   input : A, U, G
   output: M
1  P ← sorted list of {u ∈ U | u has friends in G} in non-descending order of degrees;
2  foreach a ∈ A in a random order do
3      u_top ← top user in P;
4      u' ← friend of u_top with least remaining degree;
5      assign u_top, u' to M(a);
6      remove u_top, u' from P;
7      update the degrees of users and P;
8  return M
```

Complexity analysis There are $|A|$ iterations in the baseline algorithm. Then during each iteration, we visit at most $|U|$ users. When visiting each user, we spend $O(\delta)$ time to check its friends in $M(a)$ and $O(|U|^2)$ time to find its friends in $C(a)$. In overall, the worst-case time complexity of the baseline algorithm is $O(|A||U|^3)$.

3.2 Greedy algorithm

In this section, we present a greedy-based heuristic algorithm (Greedy). Instead of visiting each a in order as the baseline algorithm does, we use a globally greedy strategy in this algorithm. More specifically, we first make an initial arrangement in a greedy

way to avoid unnecessary cases where some activities are not arranged any user and that the minimum normalized utility score of an activity is zero. We then continue to make arrangement based on the initial arrangement by maintaining a heap H storing pairs of activity and user and trying to add the pair at the top of H during each iteration. We make an initial arrangement M as follows. We maintain a list P of users in U with at least one friend in non-descending order of their degrees, i.e. numbers of friends, in G. That is, each user in P has degree at least one. We then visit each activity a in A in an arbitrary order, and assign the top user u_{top} in P, i.e. the one with the least friends in P, and one of her/his friend to a. The friend of u_{top} with the least degree is chosen to be assigned to a. After assigning u_{top} and her/his friend to a, we update elements of P in non-descending order of their remaining degrees, i.e. numbers of unassigned friends, in G, and continue to arrange for the next activity in A.

After the initial step, we continue to make arrangement based on M and maintain H in the following way. H uses two keys to maintain the order of each element (a, u): one is the current normalized utility of a $Ave(a)$, and the other is the updated normalized utility of a if u is added to $M(a)$, i.e. $Ave'(a) = \frac{Ave(a)|\delta_a| + \mu(a,u)}{\delta_a}$. In other words, elements in H are arranged first in ascending order of $Ave(a)$, and then in non-descending order of $Ave'(a)$ if the elements have the same value of $Ave(a)$. The intuition is that we try to arrange a new user to an activity that currently has the lowest normalized utility value (or the one that will still have the lowest normalized utility value even such new arrangement is made if there are multiple activities sharing the same lowest normalized utility value), so that to increase $\min_a Ave(a)$ as much as possible. We also maintain a list Q_a of users for each a, elements in which are sorted in non-ascending order of $\mu(a, u)$. H is then initialized as follows. For each $a \in A$, we pop the $u \in Q_a$ with the largest $\mu(a, u)$ w.r.t. a, i.e. $u = \arg\max_{u \in U} \mu(a, u)$. We then push the pair (a, u) into H with keys of $Ave(a)$ and $Ave'(a)$ respectively. For each $a \in A$, we maintain a $C(a)$ as we do in the baseline algorithm.

Details of each iteration are as follows. Let (a, u) be the pair popped from H in the current iteration. If a is not full, i.e. $|M(a)| < \delta_a$, and u is not yet arranged to any activity, i.e. $u \notin M(a'), \forall a' \in A$, we then try to add u to $M(a)$ as follows. If u can find its friends in $M(a)$, we safely add u to $M(a)$. Otherwise, we check whether u has any friend in $C(a)$ as we do in the baseline algorithm. More specifically, if $M(a)$ can accommodate at least two more users, i.e. $|M(a)| + 2 \leq \delta_a$, and u can find its an unarranged in $C(a)$, i.e. $\exists v \in C(a)$ s.t. $e(u, v) \in F$ and $v \notin M(a'), \forall a' \in A$, we add both u and the first such v (i.e. the foremost in $C(a)$) into $M(a)$ and remove v from $C(a)$. If u is successfully added to $M(a)$, we find her/his friends in $C(a)$ and re-push them in to Q_a. Finally, we push v paired with the next user with the largest utility value w.r.t. a in Q_a if v is not yet full and Q_a is non-empty. The whole iteration procedure terminates when H is empty.

We illustrate the procedure in Algorithm 2 and Algorithm 3. Algorithm 2 illustrates the initialization procedure of M. In line 1, we construct the sorted list P of users. We then make initial arrangement for each activity in lines 2-7. Particularly, we add the top user in P paired with her/his friend to M in lines 3-5 and then update P accordingly in lines 6-7. Algorithm 3 illustrates the main procedure of the Greedy algorithm. In lines 1-3, we initialize M, Q_a, $\forall a$ and H. In lines 4-17, we iteratively pop a pair (a, u) from H and try to update M and H. Specifically, we check whether a is full and whether u has been arranged in line 6, and add u to $M(a)$ if u has a friend in $M(a)$ in line 8. Otherwise, we check whether u has an unarranged friend in $C(a)$ and add u and its first friend in $C(a)$ into $M(a)$ if the friend exists in line 11. If u has no friend in $C(a)$, it is appended to the tail of $C(a)$ in line

13. If u is added to $M(a)$ in the current iteration, we try to find its unarranged friends in $C(a)$ and re-push them to Q_a if a is not yet full in line 15. We then push a paired with the next user in Q_a into H if a is not yet full in line 17.

Example 3 Back to our running example in Example 1. We first make an initial arrangement and first visit a_1. Since u_6 has the least number of friends, we add u_6 with its friend u_4 to $M(a_1)$. We then update the degrees of the remaining users and P. We then visit a_2. Since now u_5 is one of the users with the least number of available friends, which are u_2 and u_3, we next add u_5 with its one of its friend u_3 to $M(a_2)$. After the initialization step, we start to make arrangement greedily. We first construct Q for each activity, which results in $Q(a_1) =< u_1, u_2 >, Q(a_2) =< u_2, u_1 >$. Then after initializing H, we obtain $H =< (a_1, u_1) : (0.345, 0.54), (a_2, u_2) : (0.44, 0.66) >$. That is, a_1 has the minimum current normalized score 0.345, and will have normalized score 0.54 if u_1 is added to $M(a_1)$. We then keep trying to add a pair to M during each iteration. In the 1st iteration, the pair (a_1, u_1) is popped from H. Since u_1 has a friend u_4 in $M(a_1)$, u_1 is added to $M(a_1)$ and $M(a_1) = \{u_1, u_4, u_6\}$. We then update $Q(a_1) =< u_2 >$ and $H =< (a_2, u_2) : (0.44, 0.66), (a_1, u_2) : (0.54, 0.72) >$. Then in the 2nd iteration, (a_2, u_2) is popped from H. Since u_2 has friends u_3 and u_5 in $M(a_2)$, we add u_2 to $M(a_2)$. The final arrangement is $M(a_1) = \{u_1, u_4, u_6\}$, $M(a_2) = \{u_2, u_3, u_5\}$ and has minimum normalized utility score of 0.54.

Algorithm 3 Greedy

```
   input : A, U, G, M
   output: M
 1 M ← Initialization(A, U, G);
 2 construct Q_a for each a ∈ A;
 3 initialize H;
 4 while H ≠ ∅ do
 5     pop (a, u) from H;
 6     if |M(a)| < δ_a and u is unarranged then
 7         if u_j has a friend in M(a) then
 8             add u to M(a);
 9         else if |M(a)| + 2 ≤ δ_a then
10             if u has an unarranged friend in C(a) then
11                 add u and its first friend from C(a) to M(a);
12             else
13                 append a to the tail of C(a);
14         if u is added to M(a) and |M(a)| < δ_a then
15             re-push friends of u in C(a) to Q_a;
16     if a is not full and Q_a ≠ ∅ then
17         pop the next u' from Q_a and push (a, u') into H;
18 return M
```

Complexity analysis For the initialization step, there are $|A|$ iterations and it takes at most $O(|U|)$ time to update P during each iteration. Thus, the time complexity of the initialization step is $O(|A||U|)$. Since each activity-user pair is pushed into H for at most twice, the number of iterations is at most $O(|A||U|)$. As there are at most $|A|$ elements in H, during each iteration, it takes $\Theta(log|U|)$ time to pop a pair from or push one into H. It also takes

$O(\delta)$ time to find a friend in $M(a)$ and $O(|U|^2)$ to time to find a friend in $C(a)$. In overall, the time complexity of the Greedy algorithm is $O(|A||U|(|U|^2 + log|U| + \delta))$.

Algorithm 4 Random+Greedy

```
   input : A, U, G
   output: M
 1 M ← Initialization(A, U, G);
 2 A' ← a shuffle-ordered list of A;
 3 foreach a_i ∈ A' do
 4     calculate θ;
 5     L ← sorted list of unassigned u ∈ U in non-ascending order of μ(a_i, u);
 6     for u_j ∈ L do
 7         if u_j has friends in M(a_i) then
 8             add u_j to M(a_i);
 9             add friends of u_j from C(a_i) to M(a_i) if not exceeding θ and δ_{a_i};
10         else
11             if u_j has friends in C(a_i) then
12                 add u_j and its friends from C(a_i) to M(a_i) if not exceeding θ and
                   δ_{a_i};
13             else
14                 append u_j to the tail of C(a_i);
15         if |M(a_i)| ≥ min{θ, δ_{a_i}} then
16             break;
17 return M
```

3.3 Random+Greedy algorithm

In this subsection, we present another heuristic algorithm that integrates the aforementioned random idea into the Greedy Algorithm, which we call Random+Greedy (RG). We again make an initial arrangement based on Algorithm 2 and then visit each activity $a \in A$ in a random order and use the threshold θ to indicate when we should stop processing each a. Unlike the baseline algorithm in which we randomly pick up some unarranged users, we greedily add users to $M(a)$. More specifically, given a shuffle-ordered list A' of A, we visit each a_i in A' in order. For each a_i, we visit each unarranged users $u \in U$ in non-ascending order of $\mu(a_i, u)$. Then for each u_j we visit, we process it as we make the random arrangement in the baseline algorithm.

We illustrate the procedure in Algorithm 4, which is similar to Algorithm 1, except that we make an initial arrangement using Algorithm 2 and visit each unarranged u in a certain order instead of visiting u with a probability.

Example 4 Back to our running example in Example 1. We first make an initial arrangement, which results in $M(a_1) = \{u_3, u_5\}$, $M(a_2) = \{u_4, u_6\}$. After initialization, we visit each a in the order of a_2, a_1. For a_2, the threshold $\theta = 2$, and thus we do not make additional arrangement for a_2. Then for a_1, unassigned u_1 has the largest utility score and has the friend u_2 in $M(a_1)$. Thus, u_1 is added to $M(a_1)$. The final arrangement is $M(a_1) = \{u_1, u_3, u_5\}$, $M(a_2) = \{u_4, u_6\}$ and has minimum normalized utility score of 0.4.

Complexity analysis Similar to the Baseline algorithm, the worst-case time complexity of the RG algorithm is $O\left(|A||U|^3\right)$.

Algorithm 5 LocalSearch

```
   input : A, U, G, M
   output: M
 1 while M can be updated do
 2     a ← arg min_a Ave_M(a);
 3     if a is full of capacity then
 4         break;
 5     if an unassigned user u has friends in M(a) then
 6         add u to M(a);
 7     else if there exist two unassigned users who are friends and |M(a)| + 2 ≤ δ_a
       then
 8         add the two unassigned users to M(a);
 9     else
10         foreach a' in A in non-ascending order of Ave_M(a') do
11             flag ← false;
12             foreach u' ∈ M(a') do
13                 if (Ave_M(a')δ_a' − μ(a', u')) / δ_a' ≥ Ave_M(a) then
14                     if u' has friends in M(a) or (u' has an unassigned friend and
                       |M(a)| + 2 ≤ δ_a) then
15                         add u' or u' paired with its unassigned friend to M(a);
16                         flag ← true;
17                         break;
18             if flag then
19                 break
20 return M
```

3.4 Local-search-based optimization

Notice that after making the arrangement using the three algorithms we present above, some users may not be assigned to any activity, and the activity with the minimum normalized utility score, i.e. the bottleneck activity, may still have space to accommodate more users. Therefore, we next propose a local search algorithm that tries to increase the normalized utility score of the bottleneck activity.

The basic idea of the local search algorithm is that if the bottleneck activity is not yet full of capacity, we try to add unassigned users or move assigned users from other activities to the bottleneck activity to increase its normalized utility score without decreasing the overall minimum normalized utility score. More specifically, we iteratively try to improve the arrangement of the bottleneck activity a. We first try to add unassigned users to a if they have friends in $M(a)$. If no unassigned user can be added to $M(a)$, we then search other activities and try to re-arrange their users to a. Particularly, we visit each activity a' in non-ascending order of their normalized utility scores. If there exists a user $u' \in M(a')$ s.t. removing u' from $M(a')$ will not lead to a smaller minimum normalized utility score and u' can be added to $M(a)$, we re-arrange u' to a to increase the normalized utility score of the bottleneck activity. We then proceed to the next iteration to improve the arrangement of the new bottleneck activity until no improvement can be made.

We illustrate the procedure in Algorithm 5. We iteratively try to improve the arrangement of the bottleneck activity. In line 2, we find the bottleneck activity a. We then check whether

some unassigned users can be arranged to a in lines 5-8. If not, we then try to re-arrange an assigned user to a in lines 10-17.

Example 5 We use the arrangement returned by Example 2 as example. Recall that after running the baseline algorithm, we have arrangement $M(a_1) = \{u_2, u_5\}$, $M(a_2) = \{u_3, u_4\}$. Since the current bottleneck activity is a_2, we try to improve $M(a_2)$ in the 1st iteration. Since unassigned user u_1 has friends u_3 and u_4 in $M(a_2)$, we assign u_1 to $M(a_2)$ and $M(a_2) = \{u_1, u_3, u_4\}$. Since $Ave_M(a_1) = 0.41$ and $Ave_M(a_2) = 0.62$, a_1 becomes the new bottleneck activity and we next try to improve $M(a_1)$. Since unassigned user u_6 has no friend in $M(a_1)$, we try to find a user in $M(a_2)$. We find that $u_3 \in M(a_2)$ has friends u_2 and u_5 in $M(a_1)$ and removing u_3 from $M(a_2)$ will not result in a lower minimum normalized utility score than 0.41. Thus, we move u_3 from $M(a_2)$ to $M(a_1)$, and $M(a_1) = \{u_2, u_3, u_5\}$, $M(a_2) = \{u_1, u_4\}$. Now $Ave_M(a_1) = 0.61$ and $Ave_M(a_2) = 0.45$, we next try to improve $M(a_2)$. We then add unassigned user u_6 to $M(a_2)$. The final arrangement is $M(a_1) = \{u_2, u_3, u_5\}$, $M(a_2) = \{u_1, u_4, u_6\}$ and has minimum normalized utility score of 0.61.

4 Solutions for extension of BSEA

In this section, we present extended solutions for the extension of BSEA problem where users can attend multiple activities.

Algorithm 6 BaselineExt

```
   input : A, U, G
   output: M
 1 A' ← a shuffle-ordered list of A;
 2 foreach a_i ∈ A' do
 3     calculate θ;
 4     for u_j ∈ U do
 5         skip u_j with probability 1 − θ/|U| or if δ_{u_j} = 0;
 6         if u_j has friends in M(a_i) then
 7             add u_j to M(a_i);
 8             add friends of u_j from C(a_i) to M(a_i) if not exceeding θ and δ_{a_i};
 9             decrease capacities of u_j and its added friends by one;
10         else
11             if u_j has friends in C(a_i) then
12                 add u_j and its friends from C(a_i) to M(a_i) if not exceeding θ and
                   δ_{a_i};
13                 decrease capacities of u_j and its added friends by one;
14             else
15                 append u_j to the tail of C(a_i);
16         if |M(a_i)| ≥ min{θ, δ_{a_i}} then
17             break;
18 return M
```

4.1 Extended baseline algorithm

To extend the baseline algorithm to the cases where users can attend multiple activities, we maintain the remaining capacity of each user and only assign users with non-zero remaining

capacities to an activity. We call this algorithm BaselineExt and illustrate the procedure in Algorithm 6. And the threshold θ is modified as follows.

Algorithm 7 InitializationExt

input : A, U, G
output: M
1 $P \leftarrow$ sorted list of $\{u \in U | u$ has friends in $G\}$ in non-descending order of degrees and then their capacities;
2 **foreach** $a \in A$ **do**
3 $u_{top} \leftarrow$ top user in P;
4 $u' \leftarrow$ friend of u_{top} with least remaining degree;
5 assign u_{top}, u' to $M(a)$;
6 update $\delta_{u_top}, \delta_{u'}$ and remove from P if full of capacity;
7 update the degrees of users and P;
8 **return** M

$$\theta = \left\lfloor \frac{\delta_{a_i} \sum_u \delta_u}{\sum_a \delta_a} \right\rfloor \tag{5}$$

The main procedure is similar to Algorithm 1. The difference is that we check the availability of u_j in line 5 and update the capacities of the newly assigned users in lines 9 and 13, respectively.

Complexity analysis The time complexity of the BaselineExt algorithm is the same as that of the baseline algorithm, which is $O\left(|A||U|^3\right)$.

Algorithm 8 GreedyExt

input : A, U, G, M
output: M
1 $M \leftarrow InitializationExt(A, U, G)$;
2 construct Q_a for each $a \in A$;
3 initialize H;
4 **while** $H \neq \emptyset$ **do**
5 pop (a, u) from H;
6 **if** $|M(a)| < \delta_a$ *and* u *is not yet full of capacity* **then**
7 **if** u_j *has a friend in* $M(a)$ **then**
8 add u to $M(a)$;
9 update δ_u;
10 **else if** $|M(a)| + 2 \leq \delta_a$ **then**
11 **if** u *has an unarranged friend in* $C(a)$ **then**
12 add u and its first friend from $C(a)$ to $M(a)$;
13 update capacity of u and its added friend;
14 **else**
15 append a to the tail of $C(a)$;
16 **if** u *is added to* $M(a)$ *and* $|M(a)| < \delta_a$ **then**
17 re-push friends of u in $C(a)$ to Q_a;
18 **if** a *is not full and* $Q_a \neq \emptyset$ **then**
19 pop the next u' from Q_a and push (a, u') into H;
20 **return** M

4.2 Extended greedy algorithm

Recall that we make an initial arrangement in the original Greedy algorithm. To extend the Greedy algorithm, we also make extension to the initialization step. More specifically, we keep updating the remaining capacities of users and remove users who are full of capacity or have no available friend from the sorted list L. For the extended Greedy algorithm, we also maintain the capacities of users and only assign users with non-zero remaining capacities to an activity.

We illustrate the procedure in Algorithm 7 and Algorithm 8. Algorithm 7 illustrates the extended initialization procedure of M. Particularly, the difference from Algorithm 2 is that we update the remaining capacities of users and remove users from P when they are full of capacity in line 6. Algorithm 8 illustrates the extended Greedy algorithm GreedyExt. Different from Algorithm 3, we check the validity of a user by its remaining capacity in line 6 and update the capacities of the users who are added to the arrangement in line 9 and 13, respectively.

Complexity analysis Similar to Algorithm 2 and Algorithm 3, the extended initialization step has time complexity $O(|A||U|)$ and the extended Greedy algorithm has time complexity $O(|A||U|(|U|^2 + log|U| + \delta))$.

Algorithm 9 RandomGreedyExt

```
   input : A, U, G
   output: M
 1 M ← InitializationExt(A, U, G);
 2 A' ← a shuffle-ordered list of A;
 3 foreach a_i ∈ A' do
 4     calculate θ;
 5     L ← sorted list of {u ∈ U|δ_u > 0} in non-ascending order of μ(a_i, u);
 6     for u_j ∈ L do
 7         if u_j has friends in M(a_i) then
 8             add u_j to M(a_i);
 9             add friends of u_j from C(a_i) to M(a_i) if not exceeding θ and δ_{a_i};
10             update capacities of u_j and its friends;
11         else
12             if u_j has friends in C(a_i) then
13                 add u_j and its friends from C(a_i) to M(a_i) if not exceeding θ and
                   δ_{a_i};
14                 update capacities of u_j and its friends;
15             else
16                 append u_j to the tail of C(a_i);
17         if |M(a_i)| ≥ min{θ, δ_{a_i}} then
18             break;
19 return M
```

4.3 Extended random+greedy algorithm

We then extend the Random+Greedy algorithm, named as the Random+GreedyExt (RGExt) algorithm. We again make an initial arrangement based on Algorithm 7. Then in the extended Random+Greedy algorithm, we again keep updating the remaining capacities of users and assign a user to an activity only when s/he has non-zero remaining capacity. We illustrate the procedure in Algorithm 9. Different from Algorithm 4, in line 5, L only stores

users with non-zero remaining capacities. Also, we update capacities of users who are added to the arrangement in lines 10 and 14, respectively.

Algorithm 10 LocalSearchExt

```
   input : A, U, G, M
   output: M
 1 while M can be updated do
 2     a ← arg min_a Ave_M(a);
 3     if a is full of capacity then
 4         break;
 5     if a non-full user u has friends in M(a) then
 6         add u to M(a);
 7         update δ_u;
 8     else if there exist two non-full users who are friends and |M(a)| + 2 ≤ δ_a then
 9         add the two non-full users to M(a);
10         update capacities of the two users;
11     else
12         foreach a' in A in non-ascending order of Ave_M(a') do
13             flag ← false;
14             foreach u' ∈ M(a') do
15                 if (Ave_M(a')δ_a' − μ(a', u'))/δ_a' ≥ Ave_M(a) then
16                     if u' has friends in M(a) or (u' has a non-full friend and
                       |M(a)| + 2 ≤ δ_a) then
17                         add u' or u' paired with its non-full friend to M(a);
18                         update capacities of u' and its friend;
19                         flag ← true;
20                         break;
21             if flag then
22                 break
23 return M
```

Complexity analysis Similar to the Random+Greedy algorithm, the worst-case time complexity of the RGExt algorithm is $O\left(|A||U|^3\right)$.

4.4 Extended local-search-based optimization

We finally present the extended Local Search algorithm. Similar to the previous extended algorithms, we check the validity of users by their remaining capacities and update their capacities whenever they are assigned to an activity. We illustrate the procedure in Algorithm 10. Different from Algorithm 4, we only add users with non-zero capacities to the arrangement and update the capacities of the users who are added to the arrangement in lines 7, 10 and 17, respectively.

5 Experimental evaluation

5.1 Experiment settings

In this subsection, we evaluate our proposed algorithms. We first present the settings of the experimental study.

Table 3 Synthetic dataset

Factor	Settings
$\|V\|$	**100**, 200, 300, 400, 500
$\|U\|$	1000, **2000**, 3000, 4000, 5000
δ_a	$\boldsymbol{\mathcal{N}(25, 10)}$, $\mathcal{N}(50, 10)$, $\mathcal{N}(75, 10)$, $\mathcal{N}(100, 10)$, $\mathcal{N}(125, 10)$
δ_u	Normal, mean=1, 2, **3**, 4, 5, variance=2
$\frac{\|F\|}{\|U\|(\|U\|-1)/2}$	0.01, **0.05**, 0.1, 0.15, 0.2
α	0.1, **0.2**, 0.3, 0.4, 0.5
Scalability	$\|V\| = 1000$, $\|U\| = 4K, 6K, 8K, 10K, 12K$
	mean of $\delta_a = 50$, density of graph = 0.0005

Datasets We use both real and synthetic datasets for the experiments. The real dataset is the Meetup dataset from [13]. In this dataset, each user is associated with a set of tags and a location. Each activity in the dataset is also associated with a location, and we use the tags of created group as the tags of the corresponding activity. The Meetup dataset consists of 225 activities and 2012 users.

For synthetic data, we generate attribute values and locations following Uniform distribution, and generate capacities of activities following Normal distribution. Statistics and configuration of synthetic data are illustrated in Table 3, where we mark our default settings in bold font. Note that the generated capacity values are converted to integers. Since the Meetup dataset does not contain social information, we generate social graphs synthetically for both real and synthetic datasets. Specifically, the social graph is generated using a function from the python-graph library[4] with varying density of the graph. The statistics of the real dataset are presented in Table 4.

Furthermore, all algorithms are implemented in C++, and the experiments were performed on a machine with Intel i7-2600 3.40GHZ 8-core CPU and 8GB memory.

5.2 Experiment results

In this section, we mainly evaluate the proposed algorithms in terms of Min Ave, running time and memory cost, and test the performance of proposed algorithms via varying the following parameters: the size of A, the size of U, δ_a and δ_u, the density of graph and the balance parameter α. We explain the notations of the algorithms we use in the results in Table 5.

Effect of $|A|$ We first study the effect of varying $|A|$. We present the results for BSEA in Figure 2a–c and those for Extended BSEA in Figure 2d–f. We can first observe that the Min Ave values decrease with increasing size of A. The reason is that the number of users is limited and thus when the number of activities increases, less users are available to each activity on average. Also, we can observe that the Greedy-based algorithms, i.e. Greedy and GreedyO, perform the best in term of Min Ave values but worst in terms of running time and memory, while the baseline algorithm performs the worst in term of Min Ave values. Another observation is that the Local Search optimization greatly improves the results of the

[4] https://code.google.com/p/python-graph/

Table 4 Real dataset

City	$\|A\|$	$\|U\|$	δ_a	δ_u	$\frac{\|F\|}{\|U\|(\|U\|-1)/2}$	α
Vancouver	225	2012	$\mathcal{N}(25, 10)$	$\mathcal{N}(3, 2)$	0.01 0.05 0.1 0.15 0.2	0.2

baseline and the Random+Greedy algorithms without consuming much additional running time or memory.

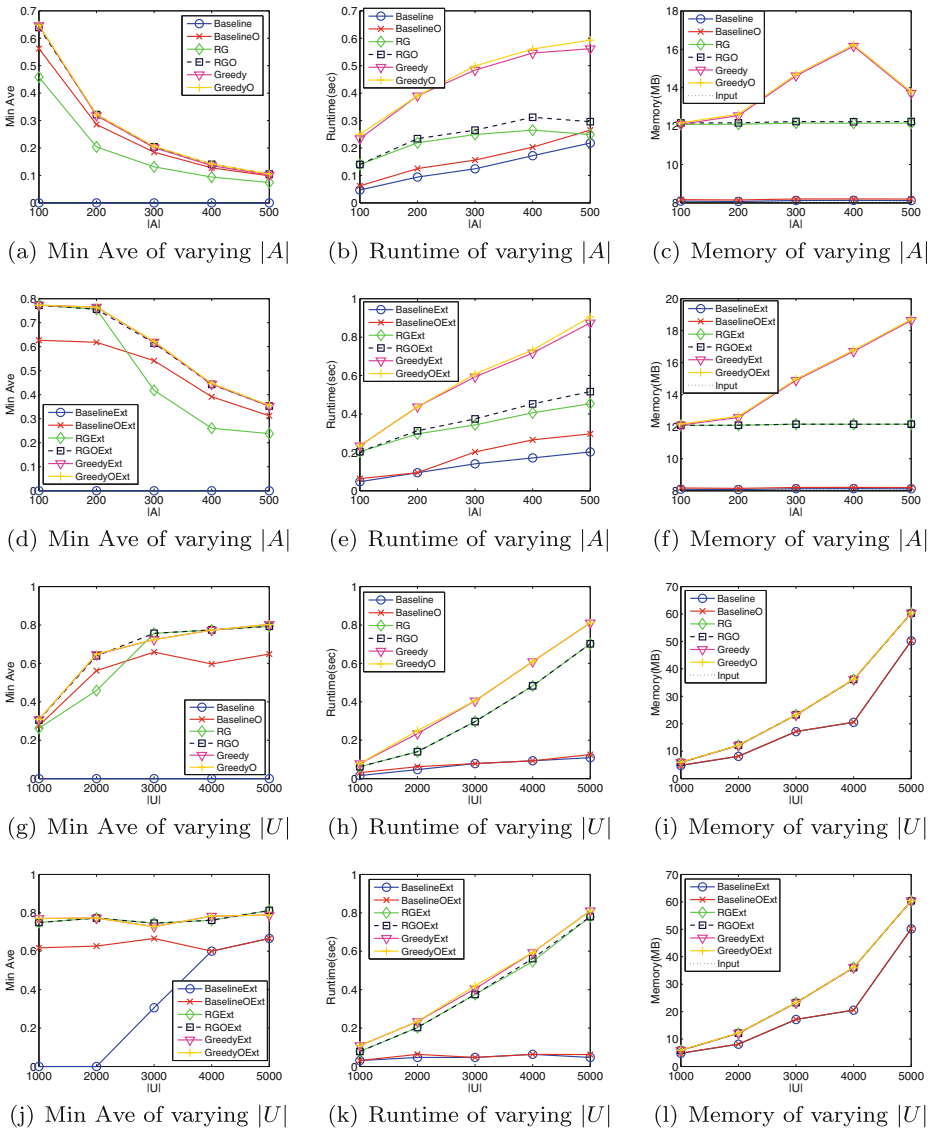
Effect of $|U|$ We then study the effect of varying $|U|$. We present the results for BSEA in Figure 2g–i and those for Extended BSEA in Figure 2j–l. We can first observe that the Min Ave values generally increase with increasing size of A, which is reasonable as more users are available to the activities. However, we can observe that the Min Ave values do not increase much for Extended BSEA. One possible reason is that for Extended BSEA, users can attend multiple activities and thus the activities may have been saturated even when the number of users is small. We can again observe that the Greedy-based algorithms perform the best in overall in term of Min Ave values and the Local Search optimization technique can improve the results significantly especially when the initial results are bad.

Effect of δ_a We then study the effect of varying δ_a. Particularly, we vary the mean of δ_a, which is generated according to the Normal distribution. We present the results for BSEA in Figure 3a–c and those for Extended BSEA in Figure 3d–f. We can first observe that the Min Ave values decrease when the mean of δ_a increases. This is reasonable since with increasing δ_a, the Ave scores of activities will decrease. Also, the Greedy-based algorithms again achieves the best Min Ave results and the Local Search optimization technique are both effective and efficient.

Effect of δ_u We next study the effect of varying δ_u for Extended BSEA. Again, we vary the mean of δ_u, which is generated following the Normal distribution. We present the results in Figure 3g–i. We can observe that the Min Ave values do not increase much with increasing mean of δ_u. One possible reason is that the activities may have been saturated even when δ_u is small as users can attend multiple activities. We can again observe that the Greedy-based algorithms are best in term of Min Ave values but worst in running time and memory. Also, the Local Search optimization technique is effective especially when the initial arrangement is far from optimal.

Table 5 Notations of algorithms

Notation	Algorithm
BaselineO	Baseline + local search optimization
BaselineExt	Extended baseline
BaselineOExt	Extended baseline + Extended local search optimization
GreedyO	Greedy + Local search optimization
GreedyExt	Extended greedy
GreedyOExt	Extended greedy + Extended local search optimization
RGO	Random+greedy + Local Search Optimization
RGExt	Extended random+greedy
RGOExt	Extended random+greedy + Extended local search optimization



(a) Min Ave of varying $|A|$ (b) Runtime of varying $|A|$ (c) Memory of varying $|A|$

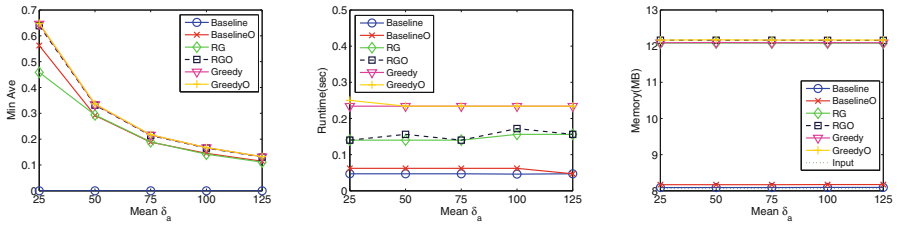
(d) Min Ave of varying $|A|$ (e) Runtime of varying $|A|$ (f) Memory of varying $|A|$

(g) Min Ave of varying $|U|$ (h) Runtime of varying $|U|$ (i) Memory of varying $|U|$

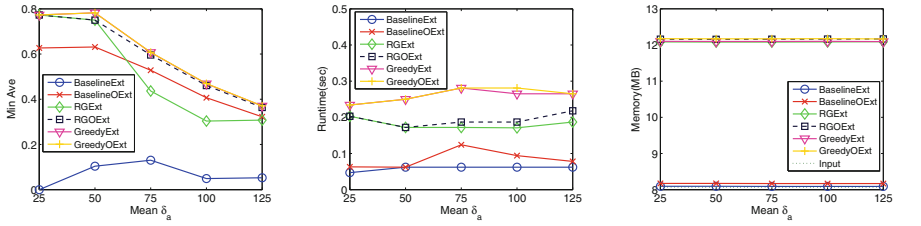
(j) Min Ave of varying $|U|$ (k) Runtime of varying $|U|$ (l) Memory of varying $|U|$

Figure 2 Results on varying $|A|$ and $|U|$

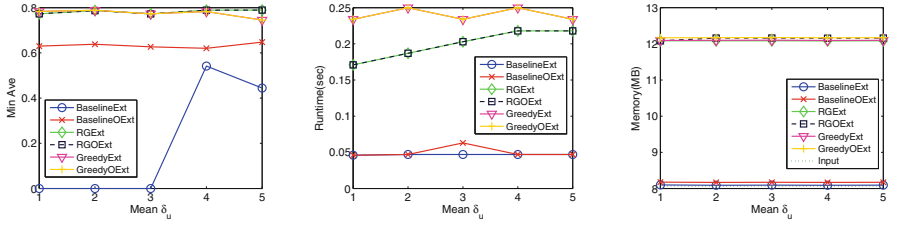
Effect of density of G We then study the effect of the density of G. Particularly, we vary $|F|/(|U|(|U|-1)/2)$, which reflects the density of G. The results for BSEA are presented in Figure 4a–c and those for Extended BSEA are presented in Figure 4d–f. We can first observe that the Min Ave values increase when $|F|/(|U|(|U|-1)/2)$ increases from 0.01 to 0.05 for BSEA. However, when the graph becomes denser, the Min Ave values do not change too much in overall. One possible reason is that the number of available users is limited and thus increasing the density of the social graph cannot improve the results too much. We can again observe that the Greedy-based algorithms achieve better Min Ave results but worse



(a) Min Ave of varying mean of δ_a (b) Runtime of varying mean of δ_a (c) Memory of varying mean of δ_a



(d) Min Ave of varying mean of δ_a (e) Runtime of varying mean of δ_a (f) Memory of varying mean of δ_a



(g) Min Ave of varying mean of δ_u (h) Runtime of varying mean of δ_u (i) Memory of varying mean of δ_u

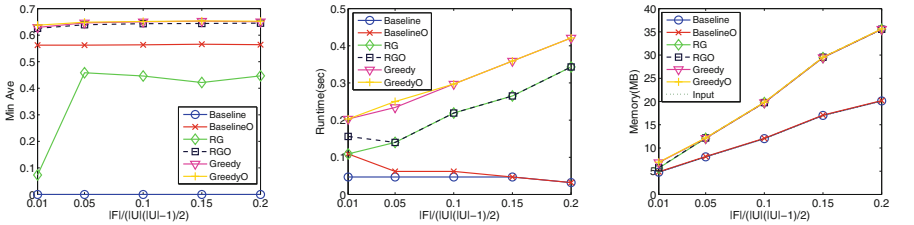
Figure 3 Results on varying mean of δ_a and δ_u

running time and memory. Also, the Local Search optimization technique is again effective and efficient.

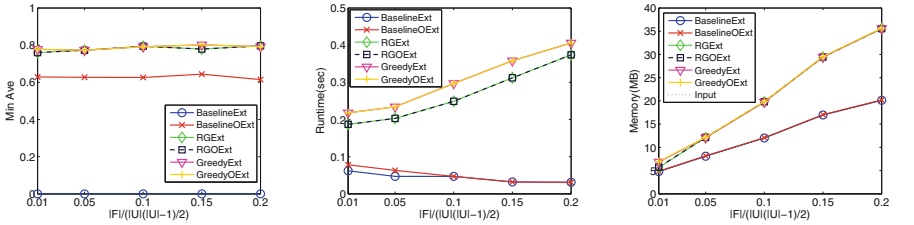
Effect of α We next study the effect of varying α. The results for BSEA are presented in Figure 4g–i and those for Extended BSEA are presented in Figure 4j–l. We can first observe that the Min Ave values do not change much with varied α. Again, the Greedy-based algorithms are best in term of Min Ave values and the Local Search optimization technique is effective when the original arrangement is far from optimal.

Real dataset We then study the results on the real dataset. We present the results for BSEA in Figure 5a–c and those for Extended BSEA in Figure 5d–f. We can observe that the results have similar trending patterns as those for the synthetic dataset in Figure 5a–f.

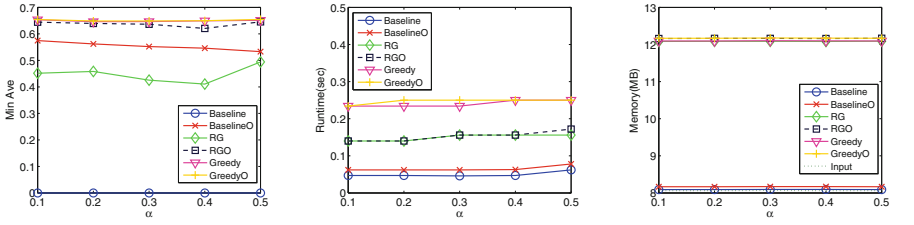
Scalability We finally study the scalability of the algorithms. The settings of some parameters are presented in Table 3, and the other parameters are set to default values. We present the results for BSEA in Figure 5g–i and those for Extended BSEA in Figure 5j–l. We can observe that all the algorithms run very fast even when the dataset is larger. We can



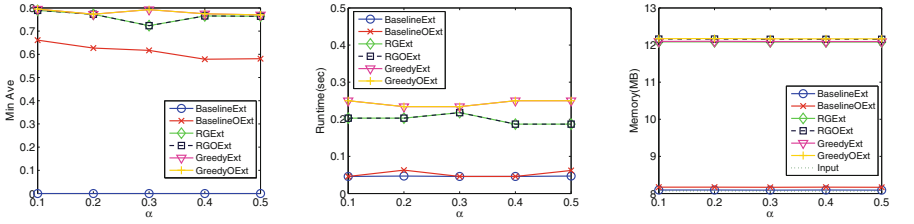
(a) Min Ave of varying density of G (b) Runtime of varying density of G (c) Memory of varying density of G



(d) Min Ave of varying density of G (e) Runtime of varying density of G (f) Memory of varying density of G



(g) Min Ave of varying α (h) Runtime of varying α (i) Memory of varying α

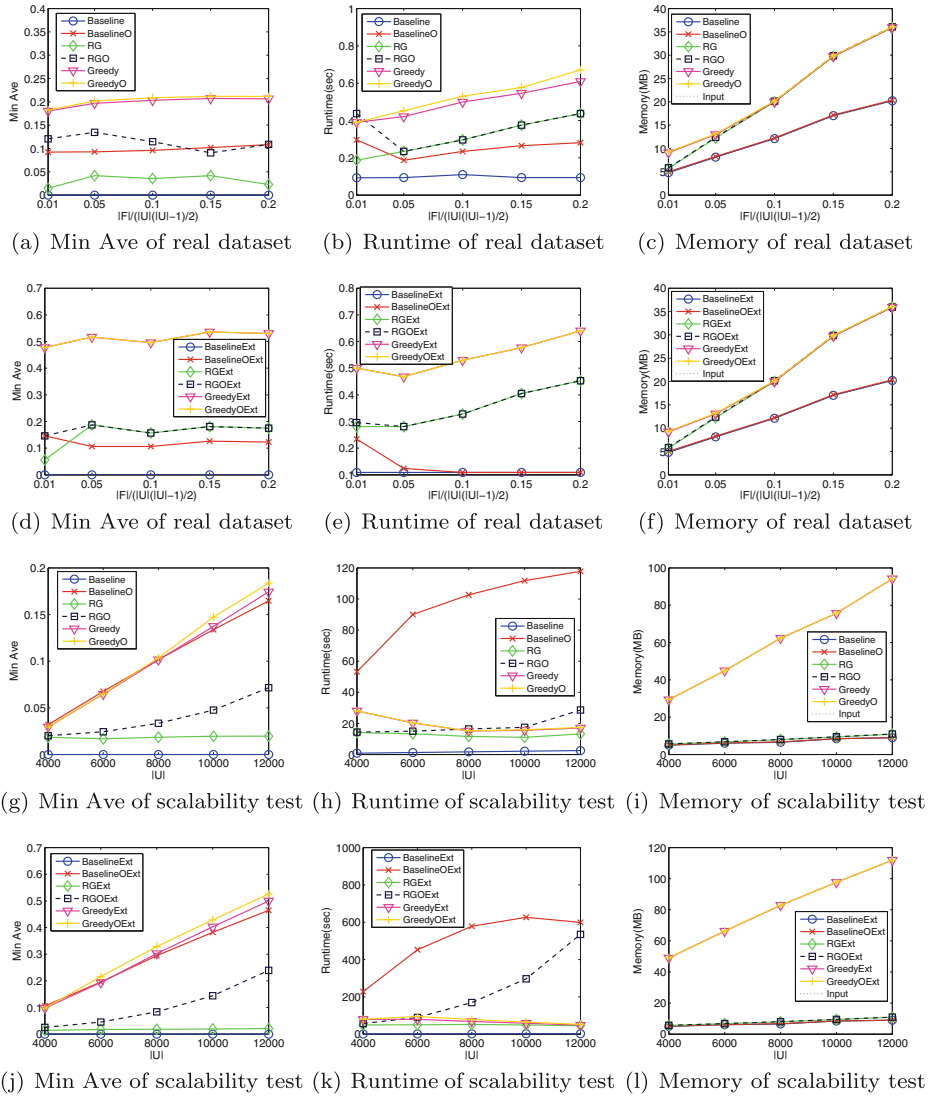


(j) Min Ave of varying α (k) Runtime of varying α (l) Memory of varying α

Figure 4 Results on varying density of G and α

also observe that the baseline and the Random+Greedy-based algorithms are still the most efficient algorithms. Finally, the algorithms except Greedy and GreedyO consume little memory in addition to the memory consumed by input data.

Comparison between BSEA and SEO [14] We compare with the best algorithm of the SEO problem, i.e. the PCADG algorithm. Notice that the objective function and the constraints of our problem are both different from those of the SEO problem. Particularly, our objective function is to maximize the minimum of the weighted linear combination of the



(a) Min Ave of real dataset (b) Runtime of real dataset (c) Memory of real dataset

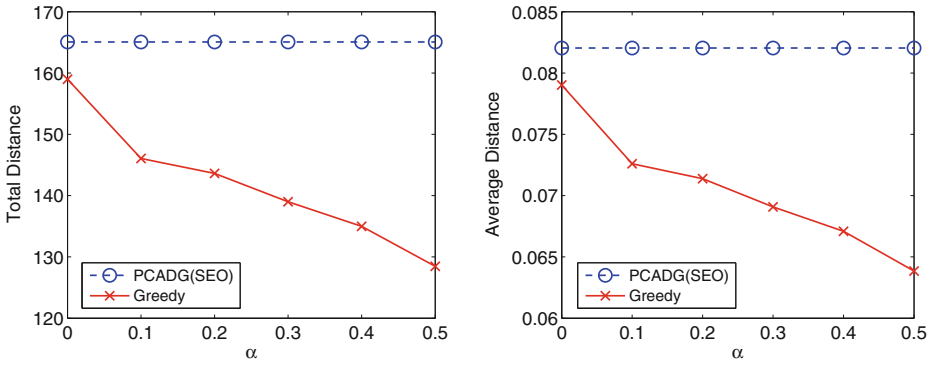
(d) Min Ave of real dataset (e) Runtime of real dataset (f) Memory of real dataset

(g) Min Ave of scalability test (h) Runtime of scalability test (i) Memory of scalability test

(j) Min Ave of scalability test (k) Runtime of scalability test (l) Memory of scalability test

Figure 5 Results on real dataset and scalability test

similarity score and the spatial distance score for an activity, but the objective function of the SEO problem is to maximize the sum of the weighted linear combination of the similarity score and the social affinity score between each activity and user in the arrangement. Also, one constraint of our problem is that no user is isolated in an activity, i.e. each user has at least one friend attending the same activity. However, the SEO has no social constraint but instead a lower bound constraint, i.e. at least γ_a users are allocated to activity a for the arrangement to be feasible. Therefore, to compare with PCADG fairly, we adjust the weight factor α in both objective functions, so that both objective functions of our problem and the SEO problem only consider the similarity score, i.e. $\alpha = 0$ in both problems. In addition, we make the social graph a complete graph, and all the activities in the SEO problem have



(a) Min Ave of real dataset (b) Runtime of real dataset

Figure 6 Comparison between BSEA and SEO [14]

lower bounds of 2. Therefore, the social constraint of our problem is equivalent to the lower bound constraint of the SEO problem.

To compare with SEO, we calculate the spatial distance between each pair of user and activity in the arrangement result. Particularly, we show the sum of distance and also the average distance of each user-activity pair. Since both the PCADG algorithm of SEO and our Greedy algorithm allocate an activity for each user but the Random+Greedy and the Random algorithm fail to allocate an activity for every user, we only compare PCADG with Greedy. The results are presented in Figure 6a–b. Notice that we also plot the results when α is varied from 0.1 to 0.5 in our problem to show how the distance between users and activities vary when the weight of the distance score increases. Particularly, since SEO does not consider spatial information, only similarity score is considered in PCADG when α varies, and thus the distance results of SEO do not vary. We can observe that the arrangement generated by our solution results in both smaller total distance and average distance of each user-activity pair compared with the PCADG algorithm of SEO particularly when α increases. Therefore, it indicates that users spend less on traveling when attending activities according to the arrangement generated by our solutions.

Conclusion All the algorithms are quite efficient and scalable. In particular, the Greedy-based algorithms perform the best in terms of Min Ave but worst in running time and memory. Also, the Local Search optimization technique is quite effective and efficient.

6 Related work

In this section, we will review the related works in four categories, event-based social networks, location-based social networks, spatial matching, and bipartite graph matching/bottleneck assignment problem.

Event-Based Social Networks With the widespread usage of mobile computing and pervasive computing techniques, various *online event-based social network* (EBSN) platforms, such as Meetup, Plancast, and Eventbrite, are getting popular. Liu et al. [13] is the first work on studying unique features of EBSNs. Recently, some other issues of EBSNs have been studied. Some research, such as [5, 9, 32], train datasets of EBSNs to derive learning

models to recommend events to potential users. However, these works just make recommendation rather than optimizing a global arrangement, which is our goal. Furthermore, [6] tries to find the most influential event organizers in EBSNs. This study integrates the classical maximization influence model [10] and the team formation model [15] to discover the most influential set of organizers. However, the optimization goal of [6] is still different with that of our problem.

In particular, a closely related work, *Social Event Organization* (SEO) problem [14], has been proposed recently. This problem is to maximize the overall innate affinities of users towards the arranged activities and the social affinity among the users attending the same activity. However, actually our BSEA problem and the SEO problem are quite different as the objective function and the constraints of our problem are both different from those of the SEO problem. Particularly, our objective function is to maximize the minimum of the weighted linear combination of the similarity score and the spatial distance score for an activity, but the objective function of the SEO problem is to maximize the sum of the weighted linear combination of the similarity score and the social affinity score between each activity and user pair in the arrangement. Also, one constraint of our problem is that no user is isolated in an activity, i.e. each user has at least one friend attending the same activity. However, the SEO problem has no such social constraint but instead a lower bound constraint on the number of attendees. Furthermore, the SEO problem does not consider any spatial information, but our ESEA problem considers three factors, the location influence, the similarity of attributes, and the social friendship, simultaneously.

Location-Based Social Networks There are a lot of related studies of this topic in recent years due to the popularity of location-based social networks (LBSN). For example, [3, 4, 11, 16–18, 21, 30] target at the issue of user-oriented recommendation. These researches mainly focus on discovering potential preferences of users and then recommend related events/venues to a single user. In other words, these works do not target at the arrangement optimization problem, which is the main difference with our work. Moreover, [1, 31] study the problems of query processing over LBSNs. The two studies only consider two factors, the location information and friendship, and ignore the similarity of attributes between activities and users.

Spatial Matching In recent years, there have been a series of works about spatial matching, such as [19, 20, 22, 25, 28]. These works aim to integrate spatial information and capacities of spatial objects into the weighted bipartite matching scenario. For example, [28] uses the problem of stable marriage as its optimization goal, and [19] chooses the sum of total scores in the weighted bipartite matching as its optimization goal. Since these works only consider the location information and capacities of objects and neglect the friendship information of social network, they are significantly different with our work.

Bipartite Graph Matching and Bottleneck Assignment Problem Bipartite graph matching and assignment problems have been widely studied for decades. The related researches have been surveyed by the following books [2, 29]. Besides classical bipartite graph matching, another close related work is the bottleneck assignment problem [2]. However, the original bottleneck assignment problem does not consider the capacity and social friendship constraints considered in our problem. Thus, existing solutions of the bottleneck assignment problem cannot address our issue.

7 Conclusion

In this paper, we identify a novel social event arrangement problem, called *bottleneck-aware social event arrangement* (*BSEA*) problem. We first present the formal definitions of the bottleneck-aware social event arrangement (BSEA) problem and a variant of the BSEA problem, called the Extended BSEA problem. Then, we prove that the two proposed problems are NP-hard. In order to solve the two problems, we devise a baseline algorithm, two heuristic algorithms, *Greedy* and *Random+Greedy*, and a local-search-based optimization technique. In particular, the *Greedy* algorithm is more effective but less efficient than the *Random+Greedy* algorithm in most cases. Furthermore, the aforementioned solutions can be modified to address the Extended BSEA problem easily. Finally, we verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.

Acknowledgment This work is supported in part by the Hong Kong RGC Project N_HKUST637/13, National Grand Fundamental Research 973 Program of China under Grant 2014CB340304, National Science Foundation of China (NSFC) under Grant No. 61502021, 61532004, 91118008 and 61232018, and Microsoft Research Asia Fellowship 2012.

References

1. Armenatzoglou, N., Papadopoulos, S., Papadias, D.: A general framework for geo-social query processing. PVLDB **6**(10), 913–924 (2008)
2. Burkard, R.E., Dell'Amico, M., Martello, S.: Assignment Problems, Revised Reprint (2009)
3. Cao, C.C., She, J., Tong, Y., Chen, L.: Whom to ask?: jury selection for decision making tasks on micro-blog services. PVLDB **5**(11), 1495–1506 (2012)
4. Cao, C.C., Tong, Y., Chen, L., Jagadish, H.: Wisemarket: a new paradigm for managing wisdom of online social users. In: KDD, pp. 455–463 (2013)
5. Du, R., Yu, Z., Mei, T., Wang, Z., Wang, Z., Guo, B.: Predicting activity attendance in event-based social networks: Content, context and social influence. In: UbiComp, pp. 425–434 (2014)
6. Feng, K., Cong, G., Bhowmick, S.S., Ma, S.: In search of influential event organizers in online social networks. In: SIGMOD, pp. 63–74 (2014)
7. Garfinkel, R.S.: An improved algorithm for the bottleneck assignment problem. Oper. Res. **19**(7), 1747–1751 (1971)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness (1979)
9. Karanikolaou, S., Boutsis, I., Kalogeraki, V.: Understanding event attendance through analysis of human crowd behavior in social networks. In: DEBS, pp. 322–325 (2014)
10. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: KDD, pp. 137–146 (2003)
11. Khrouf, H., Troncy, R.: Hybrid event recommendation using linked data and user diversity. In: RecSys, pp. 185–192 (2013)
12. Liu, S., Wang, S., Zhu, F., Zhang, J., Krishnan, R.: Hydra: Large-scale social identity linkage via heterogeneous behavior modeling. In: SIGMOD, pp. 51–62 (2014)
13. Liu, X., He, Q., Tian, Y., Lee, W.-C., McPherson, J., Han, J.: Event-based social networks: linking the online and offline social worlds. In: KDD, pp. 1032–1040 (2012)
14. Li, K., Lu, W., Bhagat, S., Lakshmanan, L.V., Yu, C.: On social event organization. In: KDD, pp. 1206–1215 (2014)
15. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: KDD, pp. 467–476 (2009)
16. Levandoski, J.J., Sarwat, M., Eldawy, A., Mokbel, M.F.: Lars: A location-aware recommender system. In: ICDE, pp. 450–461 (2012)
17. Liao, G., Zhao, Y., Xie, S., Yu, P.S.: An effective latent networks fusion based model for event recommendation in offline ephemeral social networks. In: CIKM, pp. 1655–1660 (2013)

18. Liu, X., Tian, Y., Ye, M., Lee, W.-C.: Exploring personal impact for group recommendation. In: CIKM, pp. 674–683 (2012)
19. Leong Hou, U., Yiu, M.L., Mouratidis, K., Mamoulis, N.: Capacity constrained assignment in spatial databases. In: SIGMOD, pp. 15–28 (2008)
20. Long, C., Wong, R.C.-W., Yu, P.S., Jiang, M.: On optimal worst-case matching. In: SIGMOD, pp. 845–856 (2013)
21. Minkov, E., Charrow, B., Ledlie, J., Teller, S., Jaakkola, T.: Collaborative future event recommendation. In: CIKM, pp. 819–828 (2010)
22. Mouratidis, K., Yiu, M.L., Mamoulis, N., et al.: Optimal matching between spatial datasets under capacity constraints. ACM Trans. Database Syst. (TODS) **35**(2), 9 (2010)
23. She, J., Tong, Y., Chen, L.: Utility-aware social event-participant planning. In: SIGMOD, pp. 1629–1643 (2015)
24. She, J., Tong, Y., Lei Chen, L., Cao, C.: Conflict-aware event-participant arrangement. In: ICDE, pp. 735–746 (2015)
25. Sun, Y., Huang, J., Chen, Y., Zhang, R., Du, X.: Location selection for utility maximization with capacity constraints. In: CIKM, pp. 2154–2158 (2012)
26. Tong, Y., Cao, C.C., Chen, L.: Tcs: efficient topic discovery over crowd-oriented service data. In: KDD, pp. 861–870 (2014)
27. Tong, Y., Meng, R., She, J.: On bottleneck-aware arrangement for event-based social networks. In: ICDEW, pp. 216–223 (2015)
28. Wong, R.C.-W., Tao, Y., Fu, A.W.-C., Xiao, X.: On efficient spatial matching. In: VLDB, pp. 579–590 (2007)
29. West, D.B.: Introduction to graph theory (2001)
30. Yin, H., Sun, Y., Cui, B., Hu, Z., Chen, L.: Lcars: A location-content-aware recommender system. In: KDD, pp. 221–229 (2013)
31. Yang, D.-N., Shen, C.-Y., Lee, W.-C., Chen, M.-S.: On socio-spatial group query for location-based social networks. In: KDD, pp. 949–957 (2012)
32. Zhang, W., Wang, J., Feng, W.: Combining latent factor model with location features for event-based group recommendation. In: KDD, pp. 910–918 (2013)