On Bottleneck-Aware Arrangement for Event-Based Social Networks

Yongxin Tong #1, Rui Meng *2, Jieying She *2

State Key Laboratory of Software Development Environment, School of Computer Science and Engineering
Beihang University, China
[1] yxtong@buaa.edu.cn

* Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong SAR, China
{[2]rmeng, [3]jshe}@cse.ust.hk

***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, is 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: distances between events and users, attribute similarities between events and users and friend relationships among users. However, none of existing approaches consider all aforementioned influential factors when they recommend users to proper events. Furthermore, existing recommendation strategies neglect the bottleneck cases on the global recommendation. Thus, it is impossible for the existing recommendation solutions to achieve the optimal utility 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, we devise two greedy-based heuristic algorithms, *Greedy* and *Random+Greedy*. In particular, the *Random+Greedy* algorithm is faster and more effective than the *Greedy* algorithm in most cases. Finally, we conduct extensive experiments on real and synthetic datasets which verify the efficiency and accuracy of our proposed algorithms.**

I. Introduction

In recent years, with the widespread usage of mobile computing and pervasive computing, all kinds of new social media techniques are emerging constantly [1][2]. In particular, various event-based social networks (EBSNs)[3] 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 attendances. To sum up, EBSNs provide a novel approach of facilitating *online* users to organize and manage *offline* social activities.

Unfortunately, most existing EBSNs only are 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. [4] introducs the social event organization (SEO) problem, which is to assign users to activities such that maximizes 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 [4], Meetup recommends him the following two related activities: a hiking activity from 8:30 a.m. to 10:30 a.m. and a photographic party from 11:00 a.m. to 1:00 p.m. Although Tony is interested in both activities and has friends in each activity, he needs at least one hour by taxi or two hours by bus from the hiking place to the party venue. Therefore, Tony faces a dilemma since the arrangement ignores the influence of spatial information. Clearly, the solution of [4] cannot handle the aforementioned scenario. In fact, *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.*

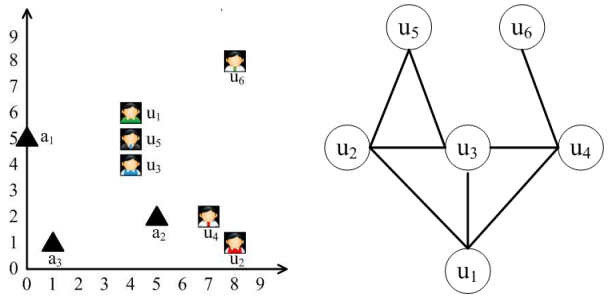
Moreover, organizers of each activity always hope to recruit as many interested users for their activity as possible. However, existing social network recommendation systems usually adopt recommendation strategies for each single activity instead of providing globally optimal arrangements. Since each user only attends one activity at the same time, it easily leads that some activities only attract a few attendances but participants in other activities are excess. Therefore, in order to satisfy all registered activities in EBSN, a reasonable global goal is to find the optimized arrangement that maximizes the minimum average total utility of activities. The utility should should take all of the aforementioned three factors into consideration. In other words, *this optimized goal can keep balance of the satisfactions from different activities in a global view*.

Therefore, a new social event arrangement strategy should not only consider the aforementioned three influential factors,

[1]http://www.meetup.com/
[2]http://plancast.com
[3]http://www.eventbrite.com/

TABLE I: Utility between Activities and Users

	u_1	u_2	u_3	u_4	u_5	u_6
a_1 (2)	**0.73**	0.37	**0.76**	0.60	0.60	0.54
a_2 (4)	0.79	0.72	0.80	**0.76**	0.92	**0.62**
a_3 (3)	0.67	**0.68**	0.51	0.67	**0.80**	0.53



(a) Location Information between Activities and Users

(b) Friendship among Users

Fig. 1: Location Information and Friendship

location information, similarity of attributes and social friendship, but also guarantees a balanced satisfactions in 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 three activities($a_1 - a_3$) 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 categorization of this activity and core tags, etc. Similarly, the attributes of users represent preferences of users, users' personalized tags, etc. According to the information of attributes between 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 attended users. In this example, the capacities of $a_1 - a_3$ are 2, 4 and 3, respectively (in brackets). In addition, Fig. 1(a) shows the locations of the three activities($a_1 - a_3$) and the six users ($u_1 - u_6$), and we use Euclidian distance to calculate the spatial utility among them. Fig. 1(b) represents the social friendship among six users ($u_1 - u_6$). If we assume that the utility function between a pair of activity and user is the linear interpolation of normalized factors between their location influence and attribute similarity, the optimal goal is to maximize the minimum average total utility of one activity such that each assigned user in this activity has at least a friend in Fig. 1(b), who is also assigned in the same activity. We present the utility score between each pair of activity and user in TABLE I. A feasible arrangement is shown by bold fonts in TABLE I, where the minimum averaged total utility of activity is 0.69 (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 of the average of total utility 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.

TABLE II: 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	Capacity of a
$U = \{u_1, \cdots, u_{\|U\|}\}$	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
$sim(\boldsymbol{t}_a, \boldsymbol{t}_u)$	the cosine similarity between attribute values of a and u
$\mu(a, u)$	the utility between a and u
M	a social event arrangement
$Ave_M(a)$	average utility of a in the arrangement M

If there is no aforementioned capacity and social friendship constraints, the ESBA problem can be reduced to a classical problem, bottleneck assignment problem[5]. Even if ESBA problem is similar to the traditional bottleneck assignment problem, they have the essential difference from the computational complexity view. On the one hand, the bottleneck assignment problem can be solved by several classical polynomial-time algorithms, e.g. the threshold algorithm[6]. On the other hand, the ESBA problem is NP-hard due to the two additional constraints. The capacity condition makes the bottleneck assignment problem from the one-to-one assignment problem to the many-to-many assignment problem. Moreover, the social friendship constraint also increases the computational complexity. Therefore, the two new conditions are the main challenge 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 a formal definition of bottleneck-aware social event arrangement (BSEA) problem.
- Since the BSEA problem is NP-hard, we devise a baseline algorithm and two heuristic algorithms, Greedy and Random+Greedy. In particular, the Random+Greedy algorithm is faster and more effective than the Greedy algorithm in most cases.
- 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 II, we formally formulate the BSEA problem. In Section III, we devise a baseline algorithm and two heuristic algorithms. Section IV shows extensive experiments on both synthetic and real datasets. We also present related works in Section V. We finally conclude this paper in Section VI.

II. Problem Statement

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

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 attended users 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 mutually friends.

For any feasible arrangement M of activity and users, we denote $m(a, u) = 1$ or $\{a, u\} \in M$ as user u is assigned to activity a, and $m(a, u) = 0$ or $\{a, u\} \notin M$ as 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

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

where $\alpha \in (0, 1)$ is a parameter used to balance spatial distance and 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 MaxD, which is the maximum distance between any activity and any user; the attribute similarity can be used any well-known similarity functions, e.g. Cosine similarity, Jaccard distance, Overlap distance, etc. Any similarity function is applicable to our utility function, in this paper, we choose Cosine similarity as our similarity function, which is shown as follows.

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

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

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

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

where $M(a)$ is a set of users who is assigned to the activity a in the arrangement M, thus $|M(a)|$ is the cardinality of $M(a)$.

We finally define our problem as follows.

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

- $\sum_u m(a, u) \leq \delta_a$ or $\sum_u m(a, u) = 0, \forall a \in A$
- Each user $u \in M(a)$ has at least another user $v \in M(a)$ such that $e(u, v) \in F$ in $G(U, F), \forall a \in A$

The notations of symbols are summarized in Table II. And we have the following theorem that states the hardness of the BESA problem.

Theorem 1: The BESA problem is NP-hard.

Due to limited space, the proof of Theorem 1 is omitted here.

III. Solutions for BSEA

In this section, we present solutions for the BSEA problem. We first present a baseline, which is a random 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.

A. Baseline Algorithm

The baseline algorithm (Random) 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.

$$\theta = \lfloor |U|/|A| \rfloor \quad (4)$$

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 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 their visited order.

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 are already 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 3-29, 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

Algorithm 1: Random

```
   input : A, U, G
   output: M
1  θ ← ⌊|U|/|A|⌋;
2  A' ← a shuffle-ordered list of A;
3  foreach a_i ∈ A' do
4      C(a_i) ← ∅;
5      for u_j ∈ U do
6          skip u_j if u_j has been arranged;
7          skip u_j with probability 1 − θ/|U|;
8          if u_j has a friend in M(a_i) then
9              add u_j to M(a_i);
10             while |M(a_i)| < min(δ_{a_i}, θ) and u_j has a
               friend in C(a_i) do
11                 u' ← the first v in C(a_i) s.t.
                   e(u_j, v) ∈ F;
12                 add u' to M(a_i);
13                 remove u' from C(a_i);
14         else
15             flag ← false;
16             while |M(a_i)| + 2 ≤ δ_{a_i} and |M(a_i)| < θ
               do
17                 if u_j has a friend in C(a_i) then
18                     flag ← true;
19                     u' ← the first v in C(a_i) s.t.
                       e(u_j, v) ∈ F;
20                     add u' to M(a_i);
21                     remove u' from C(a_i);
22                 else
23                     break;
24             if flag then
25                 add u_j to M(a_i);
26             else
27                 append u_j to the tail of C(a_i);
28         if |M(a_i)| ≥ min{θ, δ_{a_i}} then
29             break;
30 return M
```

probability in lines 8-27. If u_j has a friend in $M(a_i)$ (line 8), we add it to $M(a_i)$ in line 9 and try to add its friends in $C(a_i)$ in lines 10-13. Particularly, we add friends of u_j in order if possible (line 11). If u_j has no friend in $M(a_i)$, we try to find its friends in $C(a_i)$ in lines 15-27. In lines 16-23, we check whether a_i can accommodate at least two more users and try to find a friend of u_j in $C(a_i)$. We add u_j to $M(a_i)$ if at least one of its friends is added to $M(a_i)$ in lines 24-25, and otherwise append u_j to the tail of $C(a_i)$ in line 27.

Complexity anlaysis. There are $|A|$ iterations in the Random 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 $|U|^2$ time to find its friends in $C(a)$. In overall, the worst-case time complexity of the Random algorithm is $O(|A||U|^3)$.

B. Greedy Algorithm

In this section, we present a greedy-based heuristic algorithm (Greedy). Instead of visiting each a in order as the Random algorithm does, we use a globally greedy strategy in this algorithm. More specifically, we maintain a heap H storing pairs of activity and user and try to add the pair at the top of H during each iteration. H uses two keys to maintain the order of each element (a, u): one is the current average utility of a $Ave(a)$, and the other is the updated average utility of a if u is added to $M(a)$, i.e. $Ave'(a) = \frac{Ave(a)|M(a)|+\mu(a,u)}{|M(a)|+1}$. 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 average utility value (or the one that will still have the lowest average utility value even such new arrangement is made if there are multiple activities sharing the same lowest average utility value), so that to increase $\min_a Ave(a)$ as much as possible. H is initialized as follows. For each $a \in A$, we find a $u \in U$ 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 0 and $\mu(a, u)$, which are the values of $Ave(a)$ and $Ave'(a)$ respectively. For each $a \in A$, we maintain a $C(a)$ as we do in the Random 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 Random 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 $C(a)$. Finally, we push v paired with either another friend of u in $C(a)$ or the next user with the largest utility value w.r.t. a if v is not yet full. The whole iteration procedure terminates when H is empty.

We illustrate the procedure in Algorithm 2. In lines 1-4, we initialize H. In lines 6-30, 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 9, and add u to $M(a)$ if u has a friend in $M(a)$ in lines 10-11. 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 lines 13-16. If u has no friend in $C(a)$, it is appended to the tail of $C(a)$ in line 18. If u is added to $M(a)$ in the current iteration, we first try to find its first unarranged friend u' in $C(a)$ and re-push (a, u') into H if a is not yet full in lines 20-25. If no such friend exists, we find the user with the next largest utility value w.r.t a compared with $\mu(a, u)$ and push it paired with a into H if a is not yet full in lines 26-30.

Complexity analysis. 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 $O(|U|)$ to time to find a friend in $C(a)$.

Algorithm 2: Greedy

input : A, U, G
output: M

foreach $a \in A$ **do**
  $u \leftarrow \arg\max_{u \in U} \mu(a, u)$;
  push (a, u) into H with keys 0 and $\mu(a, u)$;
heapify H;
$C(a) \leftarrow \emptyset, \forall a \in A$;
while $H \neq \emptyset$ **do**
  pop (a, u) from H;
  $push\_flag \leftarrow false$;
  if $|M(a)| < \delta_a$ *and* u *is unarranged* **then**
    if u_j *has a friend in* $M(a)$ **then**
      add u to $M(a)$;
    else if $|M(a)| + 2 \leq \delta_a$ **then**
      if u *has an unarranged friend in* $C(a)$ **then**
        $u' \leftarrow$ the first unarranged v in $C(a)$ s.t. $e(u, v) \in F$;
        add u, u' to $M(a)$;
        remove u' from $C(a)$;
      else
        append a to the tail of $C(a)$;
    if u *is added to* $M(a)$ *and* $|M(a)| < \delta_a$ **then**
      if u *has an unarranged friend in* $C(a)$ **then**
        $u' \leftarrow$ the first unarranged v in $C(a)$ s.t. $e(u, v) \in F$;
        $Ave'(a) \leftarrow (Ave(a)|M(a)| + \mu(a, u'))/(|M(a)| + 1)$;
        push (a, u') into H with keys $Ave(a)$ and $Ave'(a)$;
        remove u' from $C(a)$;
        $push\_flag \leftarrow true$;
  if a *is not full and not* $push\_flag$ **then**
    $u' \leftarrow$ the user with the next largest μ w.r.t. a;
    if $u'\exists$ **then**
      $Ave'(a) \leftarrow (Ave(a)|M(a)| + \mu(a, u'))/(|M(a)| + 1)$;
      push (a, u') into H with keys $Ave(a)$ and $Ave'(a)$;
return M

In overall, the time complexity of the Greedy algorithm is $O(|A||U|(|U| + log|U| + \delta))$.

C. Random+Greedy Algorithm

In this section, we present another heuristic that combines the Random algorithm and the Greedy Algorithm, which we call it Random+Greedy (RG). We again 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 Random 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 do in the Random algorithm.

Algorithm 3: Random+Greedy

input : A, U, G
output: M

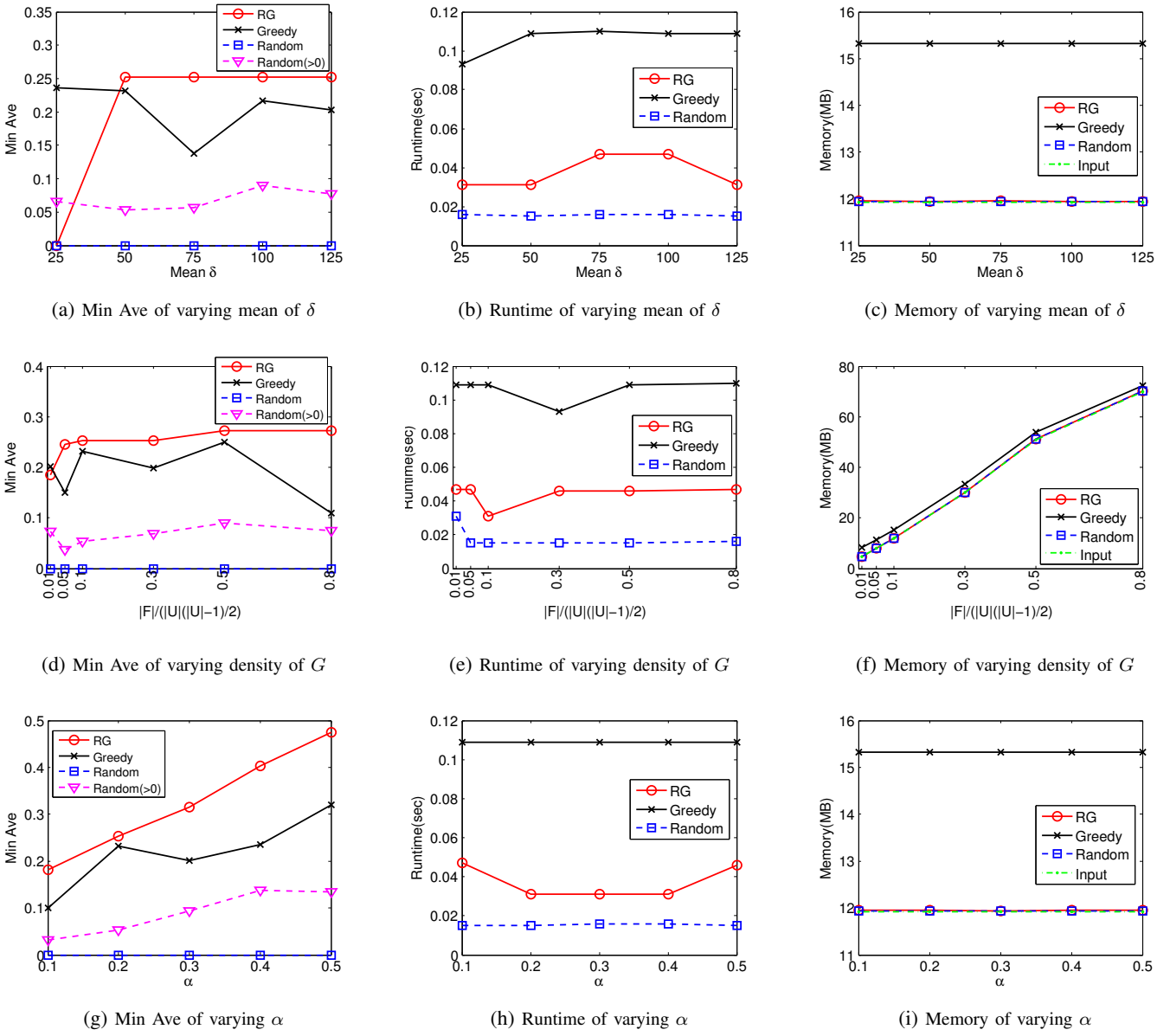
$\theta \leftarrow \lfloor |U|/|A| \rfloor$;
$A' \leftarrow$ a shuffle-ordered list of A;
foreach $a_i \in A'$ **do**
  $C(a_i) \leftarrow \emptyset$;
  $L \leftarrow$ sorted list of unarranged $u \in U$ in non-ascending order of $\mu(a_i, u)$;
  foreach $u_j \in L$ **do**
    if u_j *has a friend in* $M(a_i)$ **then**
      add u_j to $M(a)$;
      while $|M(a_i)| < \min(\delta_{a_i}, \theta)$ *and* u_j *has a friend in* $C(a_i)$ **do**
        $u' \leftarrow$ the first v in $C(a_i)$ s.t. $e(u_j, v) \in F$;
        add u' to $M(a_i)$;
        remove u' from $C(a_i)$;
    else
      $flag \leftarrow false$;
      while $|M(a_i)| + 2 \leq \delta_{a_i}$ *and* $|M(a_i)| < \theta$ **do**
        if u_j *has a friend in* $C(a_i)$ **then**
          $flag \leftarrow true$;
          $u' \leftarrow$ the first v in $C(a_i)$ s.t. $e(u_j, v) \in F$;
          add u' to $M(a_i)$;
          remove u' from $C(a_i)$;
        else
          break;
      if $flag$ **then**
        add u_j to $M(a_i)$;
      else
        append u_j to the tail $C(a_i)$;
    if $|M(a_i)| \geq \min\{\theta, \delta_{a_i}\}$ **then**
      break;
return M

We illustrate the procedure in Algorithm 3, which is similar to Algorithm 1, except that we visit each unarranged u in a certain order instead of visiting u with a probability.

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

TABLE III: Dataset

City	$\|V\|$	$\|U\|$	Mean δ	Var δ	$\frac{\|F\|}{\|U\|(\|U\|-1)/2}$	α
VA	225	2012	25, **50**, 75, 100, 125	10	0.01 0.05 **0.1** 0.3 0.5 0.8	0.1 **0.2** 0.3 0.4 0.5



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

(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

Fig. 2: Results on varying mean of δ, density of G and α.

IV. Experimental Evaluation

In this section, we mainly evaluate proposed three algorithms, Random (baseline algorithm), Greedy and Random+Greedy(RG), in terms of Min Ave, running time and memory cost, and test the performance of proposed algorithms via varying following three parameters: capacity δ, the density of graph and balance parameter α. All algorithms were implemented in C++, and the experiments were performed on a machine with i7-3520M 2.90GHz 4-core CPU and 4GB memory.

Datasets. We use both real and synthetic datasets for experiments. The real dataset is the Meetup dataset from [3]. In this dataset, each user is associated with a set of tags. Note that each activity in the Meetup dataset does not include explicit tags but has the information of corresponding created group, which is associated with a set of tags. Thus, 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 III, where we mark our default settings in bold font. Note that all generated capacity values are converted into 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.

Effect of δ. We then study the effect of varying δ. Particularly, we vary the mean of δ, which is generated according to the Normal distribution. We present the result in the first row of Fig. 2. We can first observe that the Min Ave values do not change too much with varying δ. Also, Random+Greedy algorithm outperforms Greedy algorithm and Random algorithm in terms of Min Ave and also beats Greedy algorithm in running time. It indicates that Random+Greedy algorithm is more effective and more efficient than Greedy algorithm in practice.

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 are presented in the second row of Fig. 2. We can first observe that the Min Ave values increase when $|F|/(|U|(|U|-1)/2)$ increases from 0.01 to 0.05 in overall. However, when the graph becomes denser, the Min Ave values do not change too much in overall. We also observe that Random+Greedy algorithm outperforms Greedy algorithm in overall.

Effect of α. We next study the effect of varying α. The results are presented in the third row of Fig. 2. We can first observe that the Min Ave values increase with increasing α. The possible reason is that the sets of tags among users can be very different and thus the cosine similarity between two users can be low. Thus, increasing α can reduce the contribution of the low cosine similarity. Random+Greedy algorithm again outperforms Greedy algorithm in almost all aspects and all settings.

Scalability. We finally study the scalability of the algorithms, the results of which are presented in Fig. 3. $|A|$ is 200, the mean of δ is 50, and the density of graph is set to 0.05. We vary the size of U. We can observe that all the algorithms run very fast even when the dataset is larger. We can also observe that Random and Random+Greedy algorithms are still the most efficient algorithms. Finally, the algorithms consume little memory in addition to the memory consumed by input data.

Conclusion. All the algorithms are quite efficient and scalable. In particular, Random+Greedy algorithm performs the best in terms of Min Ave, and Random+Greedy algorithm consumes both less time and less space than Greedy algorithm in practice. The results indicate that Random+Greedy algorithm is the best algorithm among the all.

V. Related Work

In this section, we will review the related works from 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. [3] is the first work on studying unique features of EBSNs. Recently, some other issues on EBSN have been studied. Some researches, such as [7][8][9][10], train datasets of EBSNs to derive learning models to recommend events to potential users. However, goals of these works are just recommendation rather than optimizing a global arrangement, which is our goal. Furthermore, the other type of work [11] is to find the most influential event organizers in EBSNs. This study integrates the classical maximization influence model [12] and the team formation model [13] to discover the most influential set of organizers. However, the optimization goal of [11] is still different with that of our problem. In particular, a closely related work, *Social Event Organization* problem [4], has been proposed recently. This problem is to maximize the overall innate affinities of users toward the arranged events and the social affinity among the users attending the same event. However, the aforementioned problem substantially differs from ours. On the one hand, [4] does not consider all the following three factors, the location influence, the similarity of attributes, and the social friendship, simultaneously. On the other hand, the goals of [4] and ours are also different.

Location-Based Social Networks. There are a lot of related studies of this topic in recent years due to the wide popularity of location-based social networks (LBSN). For example, [14][15][16][17][18][19][20][21][22] aim the issue of user-oriented recommendation. These researches mainly focus on discovering potential preferences of users and then recommend related events/tasks to a single user. In other words, the goals of these works are not an optimized arrangement problem, which is the main difference with our work. Moreover, [23][24] study the problems of query processing over LBSNs. The two studies only consider the two factors, location information and friendship, and ignore the similarity of attribute between activities and users.

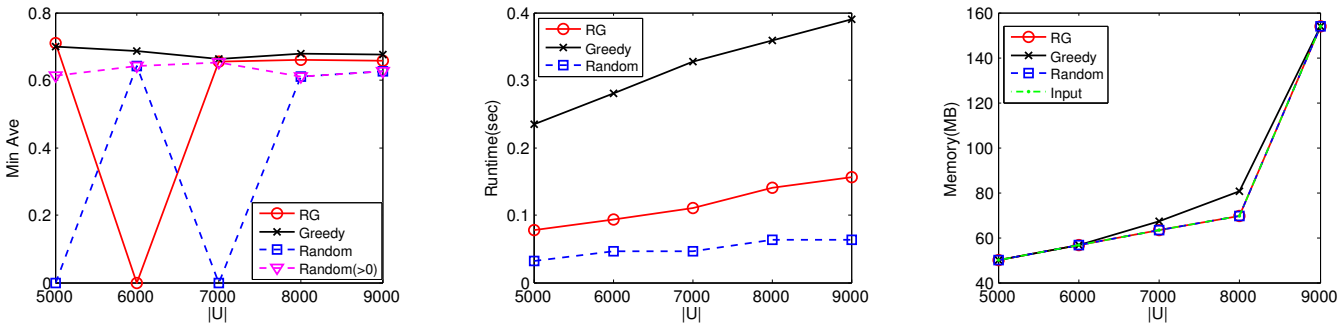
Spatial Matching. In recent years, there have been a series of works about spatial matching, such as [25][26][27][28][29]. These researches aim to integrate spatial information and capacities of spatial objects into the weighted bipartite matching scenario. For example, [25] uses the problem of stable marriage as its optimization goal, and [26] 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 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[30][5]. Besides classical bipartite graph matching, another close related work is the bottleneck assignment problem [5]. However, the original bottleneck assignment problem does not consider the capacity and social friendship constraints proposed in our problem. Thus, existing solutions of bottleneck assignment problem cannot address our issue.

VI. Conclusion

In this paper, we identify a novel social event arrangement problem, called *bottleneck-aware social event arrangement* (*BSEA*) problem. We first discuss the main differences between our new problem and traditional bottleneck assignment problem and prove that the proposed problem is NP-hard. Then,

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



(a) Min Ave of scalability test (b) Runtime of scalability test (c) Memory of scalability test

Fig. 3: Results on scalability test.

we design a baseline algorithm and two greedy-based heuristic algorithms to solve the *BSEA* problem. We first a global greedy algorithm to improve the effectiveness of the baseline algorithm. Then, we further propose the Random+Greedy heuristic algorithm which performs significantly faster than greedy algorithm and is more effective than the Greedy algorithm in most cases. 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 2014CB340303, National Science Foundation of China (NSFC) under Grant No. 61232018 and 61300031, Microsoft Research Asia Gift Grant, Microsoft Research Asia Fellowship 2012, and Google Faculty Award 2013.

REFERENCES

[1] S. Liu, S. Wang, F. Zhu, J. Zhang, and R. Krishnan, "Hydra: Large-scale social identity linkage via heterogeneous behavior modeling," in *SIGMOD'14*.

[2] Y. Tong, C. C. Cao, and L. Chen, "Tcs: efficient topic discovery over crowd-oriented service data," in *KDD'14*.

[3] X. Liu, Q. He, Y. Tian, W.-C. Lee, J. McPherson, and J. Han, "Event-based social networks: linking the online and offline social worlds," in *KDD'12*.

[4] K. Li, W. Lu, S. Bhagat, L. V. Lakshmanan, and C. Yu, "On social event organization," in *KDD'14*.

[5] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems, Revised Reprint*, 2009.

[6] R. S. Garfinkel, "An improved algorithm for the bottleneck assignment problem," *Operations Research*, vol. 19, no. 7, pp. 1747–1751, 1971.

[7] W. Zhang, J. Wang, and W. Feng, "Combining latent factor model with location features for event-based group recommendation," in *KDD'13*.

[8] C. Z. Y. C. L. G. Y. Z. Zhi Qiao, Peng Zhang, "Event recommendation in event-based social networks," in *AAAI'14 Student Abstracts*.

[9] R. Du, Z. Yu, T. Mei, Z. Wang, Z. Wang, and B. Guo, "Predicting activity attendance in event-based social networks: Content, context and social influence," in *UbiComp '14*.

[10] S. Karanikolaou, I. Boutsis, and V. Kalogeraki, "Understanding event attendance through analysis of human crowd behavior in social networks," in *DEBS'14*.

[11] K. Feng, G. Cong, S. S. Bhowmick, and S. Ma, "In search of influential event organizers in online social networks," in *SIGMOD'14*.

[12] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *KDD'03*.

[13] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *KDD'09*.

[14] E. Minkov, B. Charrow, J. Ledlie, S. Teller, and T. Jaakkola, "Collaborative future event recommendation," in *CIKM'10*.

[15] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel, "Lars: A location-aware recommender system," in *ICDE'12*.

[16] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen, "Lcars: A location-content-aware recommender system," in *KDD'13*.

[17] G. Liao, Y. Zhao, S. Xie, and P. S. Yu, "An effective latent networks fusion based model for event recommendation in offline ephemeral social networks," in *CIKM'13*.

[18] Y.-C. Sun and C. C. Chen, "A novel social event recommendation method based on social and collaborative friendships," in *SocInfo'13*.

[19] X. Liu, Y. Tian, M. Ye, and W.-C. Lee, "Exploring personal impact for group recommendation," in *CIKM'12*.

[20] H. Khrouf and R. Troncy, "Hybrid event recommendation using linked data and user diversity," in *RecSys'13*.

[21] C. C. Cao, J. She, Y. Tong, and L. Chen, "Whom to ask?: jury selection for decision making tasks on micro-blog services."

[22] C. C. Cao, Y. Tong, L. Chen, and H. Jagadish, "Wisemarket: a new paradigm for managing wisdom of online social users," in *KDD'13*.

[23] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing."

[24] D.-N. Yang, C.-Y. Shen, W.-C. Lee, and M.-S. Chen, "On socio-spatial group query for location-based social networks," in *KDD'12*.

[25] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao, "On efficient spatial matching," in *VLDB'07*.

[26] L. H. U, M. L. Yiu, K. Mouratidis, and N. Mamoulis, "Capacity constrained assignment in spatial databases," in *SIGMOD'08*.

[27] K. Mouratidis, M. L. Yiu, N. Mamoulis *et al.*, "Optimal matching between spatial datasets under capacity constraints," *ACM Transactions on Database Systems (TODS)*, 2010.

[28] Y. Sun, J. Huang, Y. Chen, R. Zhang, and X. Du, "Location selection for utility maximization with capacity constraints," in *CIKM'12*.

[29] C. Long, R. C.-W. Wong, P. S. Yu, and M. Jiang, "On optimal worst-case matching," in *SIGMOD'13*.

[30] D. B. West, *Introduction to graph theory*, 2001.