Utility-Aware Social Event-Participant Planning

Jieying She[†], Yongxin Tong[‡], Lei Chen[†]
[†]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong SAR, PR China
[‡]SKLSDE Lab, School of Computer Science and Engineering, Beihang University, PR China
[†]{jshe, leichen}@cse.ust.hk [‡]yxtong@buaa.edu.cn

ABSTRACT

Online event-based social network (EBSN) platforms are becoming popular these days. An important task of managing EBSNs is to arrange proper social events to interested users. Existing approaches usually assume that each user only attends one event or ignore location information. The overall utility of such strategy is limited in real world: 1) each user may attend multiple events; 2) attending multiple events will incur spatio-temporal conflicts and travel expenses. Thus, a more intelligent EBSN platform that provides personalized event planning for each participant is desired. In this paper, we first formally define the problem of *Utility-aware Social Event-participant Planning* (USEP), which is proven to be NP-hard. To solve the USEP problem, we first devise a greedy-based heuristic algorithm, which performs fast under certain circumstances but has no approximation guarantee. We then present a two-step approximation framework, which not only guarantees a $\frac{1}{2}$-approximation ratio but also includes a series of optimization techniques to improve its space/time efficiency. Finally, we verify the efficiency and effectiveness of the proposed methods through extensive experiments on real and synthetic datasets.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

Keywords

Event-based social network; Event planning

1. INTRODUCTION

A new type of social media, event-based social network (EBSN)[21], is becoming popular recently. In EBSNs, *online* users organize and register for *offline* social activities. For example, Plancast[1] provides a platform for users to organize, attend and share social events, and Meetup[2] allows users to create groups and organize events for other users to join. Such EBSNs facilitate social event organization and ease the recruitment of activity participants.

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

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2749446.

However, existing EBSNs only provide an information sharing platform[21] without strategic event organization and personalized event planning. Recently, the problem of global event organization strategy in EBSNs has been brought to attention. [19] proposes the Social Event Organization (SEO) problem to assign events to users, which maximizes the overall satisfaction of users towards the arranged events and also the social affinity among the users participating the same event. However, this work only considers a simple case of assigning one single event to each participant and thus naturally ignores both spatio-temporal conflicts of different events and multiple events planning for each participant. [26] also studies a global event organization strategy, but does not take travel expenditure into consideration. Imagine the following scenario. Being a sports enthusiast and a music fan, Alice attends various sports activities and concerts published on Meetup on weekends. However, Alice encounters a dilemma on Friday night since Meetup recommends her three interesting but conflicting activities on Saturday: a running club activity from 9:00 a.m. to 11:00 a.m., a tennis match from 10:00 a.m. to 1:30 p.m., and a jazz music party from 2:00 p.m. to 3:00 p.m. In addition to time conflicts, location information is another concern since Alice needs a half hour by taxi or two hours by bus from the tennis gymnasium to the party venue. Although Alice is interested in all the three events, she has to face tradeoffs according to spatio-temporal conflicts of events and her budget of travel expenditure in terms of time, money, etc. Clearly, the approaches in [19][26] cannot handle the aforementioned scenario since the constraints of spatio-temporal conflicts, multiple events allocation for each participant or travel expenditure are ignored.

As discussed above, many users often face the dilemma of choosing events to attend properly from a long and probably conflicting recommendation list on EBSN platforms. On the other hand, existing EBSNs focus on pushing recommendation to all potential participants for each event, in which case the capacities of events are out of consideration, and the satisfaction for both event organizers and participants is not optimized. Therefore, it is appealing to have a new event-participant planning strategy that satisfies spatio-temporal conflict and travel expenditure constraints, and optimizes the planning benefits on a global scale as well.

In this paper, we propose a new global event planning problem considering both spatio-temporal conflict and travel

Table 1: Utility between Events and Users and Time of Events

	u_1 (59)	u_2 (29)	u_3 (51)	u_4 (9)	u_5 (33)	Time
v_1(1)	0.2	0.6	0.7	0.3	0.6	1-4p.m.
v_2(3)	0.5	0.1	0.3	0.9	0.5	3-6p.m.
v_3(4)	0.6	0.2	0.9	0.4	0.5	1-2p.m.
v_4(2)	0.4	0.7	0.2	0.5	0.1	6-7p.m.

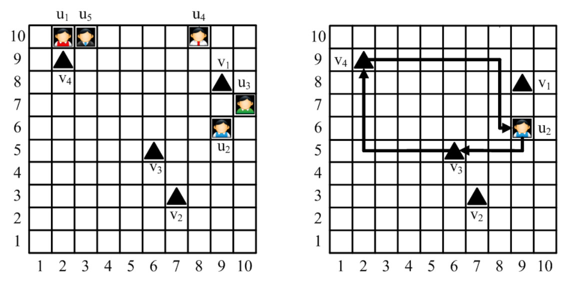


Figure 1: Example 1.

expenditure constraints to maximize the overall satisfaction of event participants. Specifically, given a set of events and a set of users, each event is associated with a location and a capacity, which is the maximum number of attendees allowable, and each user is also associated with a location, which is her/his initial location from which s/he travels to the first attended event and also the final location to which s/he returns from the last attended event. Each event is additionally associated with a time period. Note that users can only attend events that are non-overlapping in time. Each user is additionally associated with a travel budget, which is the maximum travel expenditure the user would like to spend in order to attend the arranged events. There is also a travel cost between each pair of events/users. Users may have different preferences towards the events, each of which is measured as a non-negative "utility value". The satisfaction of a user is defined as a linear combination of the sum of utility values over her/his attended events. Our task is to find an event-participant planning to maximize the overall satisfaction of users, such that the capacity, spatio-temporal conflict and travel budget constraints are satisfied. Similar to [19][26], we also adopt the sum of satisfaction over all users as the overall utility of the planning.

Example 1. *Suppose we have four events($v_1 - v_4$) and five users($u_1 - u_5$) in an EBSN. Table 1 shows the utility values between each event-user pair, and also the time intervals of the events. The capacities of the events and the budgets of the users are shown in brackets, e.g. v_1 has capacity 1, and u_4 has budget 9. Figure 1a shows the locations of users and events, and we use Manhattan distance to calculate the cost among them. Existing methods do not consider conflicts and budgets and thus cannot yield a feasible planning. Figure 1b shows a feasible schedule of u_2.*

Notice that for event planning, many other constraints can be taken into consideration, for example, considering friends who may want to attend the same event, reliability of event organizers, etc. We choose to consider spatio-temporal conflict and travel cost as constraints when optimizing users' interests in this planning problem of EBSNs with the following reasons. First, spatio-temporal conflicts among events are prevalent in real-life EBSNs, which have to be taken into consideration to make a feasible planning for users. Second, travel cost is the fundamental constraint in a planning problem. [6] indicates that venue constraints (e.g. locations), time constraints (e.g. time budget and transit time between venues) and users' preferences are the three main factors that must be taken into consideration in a planning problem. Also, travel cost, in terms of time or money, is the major constraint considered in many existing planning problems [24][8][4].

Table 2: Summary of Symbol Notations

Notation	Description
$V = \{v_1, \cdots, v_{\|V\|}\}$	Set of events
$U = \{u_1, \cdots, u_{\|U\|}\}$	Set of users
c_v	Capacity of v
t_1^v	Start time of v
t_2^v	End time of v
b_u	Travel budget of u
$cost(v_i, v_j)$	Travel cost from v_i to v_j
$cost(u, v)$ $(cost(v, u))$	Travel cost from $u(v)$ to $v(u)$
$\mu(v, u)$	Utility value between v and u
S_u	Schedule of u
$A = \cup_u\{S_u\}$	A planning for all users
$\Omega(A)$ $(\Omega(S_u))$	Total utility score of A (S_u)
cr	Conflict ratio of the set of events
f_b	Budget factor of the set of users

The contributions of our work are summarized as follows.

- To the best of our knowledge, this is the first work to formulate the *Utility-aware Social Event-participant Planning (USEP) problem*, which conducts strategic multiple-event planning for every user and considers the constraints of spatio-temporal conflicts and travel cost. We prove that this problem is NP-hard.
- We develop a greedy-based heuristic algorithm, which is efficient but has no approximation guarantee, to address the USEP problem.
- We present a two-step framework, which not only provides a $\frac{1}{2}$-approximation ratio but also includes a series of optimization techniques to improve its space and time efficiency.
- We verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.

The rest of the paper is organized as follows. In Section 2, we formally formulate the USEP problem and prove its NP-hardness. Section 3 presents a greedy-based heuristic algorithm. Section 4 proposes a two-step framework and its optimization techniques. Section 5 gives an empirical study and Section 6 reviews the related work. Section 7 concludes the paper.

2. PROBLEM STATEMENT

Let V be a set of events. For each $v \in V$, it is associated with a capacity c_v, a location l_v, and a time interval $[t_1^v, t_2^v]$ that describes the starting and ending time of the event. Notice that c_v is any integer in $\mathbb{Z}_+$. For events that may not have capacity constraints, such as firework shows, we can simply set c_v to an extremely large integer. U is a set of users. For each $u \in U$, it is associated with a location l_u that is the initial and also the final location of the user, and a travel budget b_u. We take b_u as an input parameter, following the assumptions of other planning problems [6][24][8][4]. We denote $S_u = \{v_1^u, v_2^u, ..., v_{|S_u|}^u\}$ as the schedule of arranged events in increasing time order for user u. We call two events in a schedule *neighboring* if one will be attended right after the other one finishes. The schedule is feasible if and only if there is no time conflict among the arranged events. And we define the feasibility of a schedule formally as follows.

Definition 1 (Feasible Schedule). *A schedule is feasible if and only if* $t_2^{v_i^u} \le t_1^{v_{i+1}^u}, \forall 1 \le i \le |S_u| - 1$.

Each user u has a utility value for each event v, denoted as $\mu(v,u) \in [0,1]$. Notice that this value can be discovered by existing data mining techniques [36]. However, calculation of the value is not our focus in this work. There is a travel cost (e.g. time, money) between each pair of events v_i, v_j. Without loss of generality, for any two events, suppose their time order is $t_1^{v_i} \le t_1^{v_j}$. If v_i and v_j do not conflict in time and users can attend v_j on time after attending v_i, the travel cost $cost(v_i, v_j)$ is a bounded non-negative integer. Otherwise, $cost(v_i, v_j) = +\infty$. There is also travel cost, $cost(u,v)$ and $cost(v,u)$, between a user and an event, which is the travel cost from l_u to l_v and also that from l_v returning to l_u. The cost between any event-event(event-user) pair satisfies the triangle inequality.

We finally define our Utility-aware Social Event-participant Planning (USEP) problem as follows.

Definition 2 (USEP Problem). *Given a set of events V and a set of users U and their associated attributes, find a planning $A = \cup_u\{S_u\}$ for the users to maximize the total utility score $\Omega(A)$ of the planning A such that the following constraints are satisfied:*

1. *Capacity constraint of events: no event is arranged to users more than its capacity, i.e. $\sum_u \mathbf{1}_{S_u}(v) \le c_v, \forall v$, where $\mathbf{1}_{S_u}(v)$ is the indicator function.*
2. *Budget constraint of users: no user needs to spend more than her/his budget to complete the schedule, i.e. $cost(u, v_1^u) + \sum_{i=2}^{|S_u|} cost(v_{i-1}^u, v_i^u) + cost(v_{|S_u|}^u, u) \le b_u, \forall u$*
3. *Feasibility constraint: each user is arranged a feasible schedule.*
4. *Utility constraint: no user is arranged events that s/he is not interested in at all, i.e. $\mu(v,u) > 0, \forall v \in S_u, \forall u$.*

where the total utility score $\Omega(A)$ is defined as follows:

$$\Omega(A) = \sum_u \sum_{v \in S_u} \mu(v,u) \tag{1}$$

Note that $\Omega(A)$ is the overall utility of the planning towards both event organizers and users.

Remark1. Another setting of the USEP problem is that instead of arranging events from the entire set V for a user, each user u can provide a set $V_u \subseteq V$, only events in which will be arranged to u. That is, $S_u \subseteq V_u$. This variation of the USEP problem can be reduced to the original USEP problem by letting $\mu(v,u) = 0, \forall v \in V \setminus V_u$.

Remark2. Another variance of travel cost is that there is a participation fee fee_v for an event v. This variance can be reduced to the original USEP problem by taking $cost$ as cost of money and letting $cost'(u,v) = cost(u,v) + fee_v$ and $cost'(v_i,v_j) = cost(v_i,v_j) + fee_{v_j}$, where $cost'(u,v)$ ($cost'(v_i,v_j)$) is the new travel cost in this variance of USEP, and $cost(u,v)$ ($cost(v_i,v_j)$) is the original travel cost in the original USEP problem.

The notations of symbols are summarized in Table 2. And we have the following theorem that states the hardness of USEP, the proof of which is in the Appendix.

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

3. GREEDY-BASED ALGORITHM

In this section, we present a greedy-based heuristic algorithm, RatioGreedy for the USEP problem. To make a planning that has a high total utility score, we need to consider both the utility value and the travel cost induced by each event-user pair. On one hand, an event-user pair with larger utility value will result in a higher total utility score if the pair is included in the planning. On the other hand, a pair with lower induced travel cost will possibly allow more pairs to be added to the schedule of the user if the pair is included in the planning. Thus, the RatioGreedy algorithm is to consider the utility-cost ratio of each unarranged event-user pair and add the pair with the largest ratio value into the planning if it satisfies all the constraints.

Specifically, the utility-cost ratio for each event-user pair that is not yet in A is defined as follows.

$$ratio(v,u) = \frac{\mu(v,u)}{inc\_cost(v,u)} \tag{2}$$

$$inc\_cost(v,u) = \begin{cases} cost(u,v) + cost(v,u) & S_u = \emptyset \\ cost(u,v) + cost(v,v_1^u) \\ -cost(u,v_1^u) & t_2^v \le t_1^{v_1^u} \\ cost(v_i^u,v) + cost(v,v_{i+1}^u) \\ -cost(v_i^u,v_{i+1}^u) & t_2^{v_i^u} \le t_1^v, t_2^v \le t_1^{v_{i+1}^u}, \\ & 1 \le i < |S_u| \\ cost(v_{|S_u|}^u,v) + cost(v,u) \\ -cost(v_{|S_u|}^u,u) & t_2^{v_{|S_u|}^u} \le t_1^v \\ \infty & otherwise \end{cases} \tag{3}$$

In Equation (2), $inc\_cost(v,u)$ is the additional travel cost that will be induced if the v is added to the current schedule S_u of u. Equation (3) defines $inc\_cost(v,u)$ formally. Specifically, when S_u is empty, the additional cost is that of traveling to v and returning to u if v is arranged for u. When S_u is non-empty, the additional cost depends on the time order of S_u if v is arranged for u, which is determined by the time interval of v and also those of the events that are already in S_u. If v will the first event to be attended when it is added to the current schedule, the additional cost is that of traveling from u to v and then to v_1^u subtracting the cost traveling from u to v_1^u (since u will no longer travel to v_1^u directly if v is added to the schedule), where v_1^u is the first event to be attended in the current schedule. If v will be attended after v_i^u and before v_{i+1}^u, where v_i^u and v_{i+1}^u are some neighboring events in S_u, the additional cost is that of traveling from v_i^u to v and then to v_{i+1}^u subtracting the cost from v_i^u to v_{i+1}^u (since u will no longer travel from v_i^u to v_{i+1}^u directly if v is added to the schedule). Similarly, if v will be the last event to be attended, the additional cost is that of traveling from $v_{|S_u|}^u$ to v and then retuning to u subtracting the cost from $v_{|S_u|}^u$ to u (since u will not long return to her/his initial location after attending $v_{|S_u|}^u$ if v is added to the schedule), where $v_{|S_u|}^u$ is the last event to be attended in the current schedule.

The detailed algorithm is as follows. Initially, the planning A is empty, i.e. $S_u = \emptyset, \forall u \in U$. We maintain a heap H for storage of unarranged event-user pairs. At each iteration, we extract a pair (v,u) with the largest ratio value from H and add it to A if possible. Initially, for each $v_i \in V$, we find a $u_{v_i} \in U$ such that $ratio(v_i, u_{v_i}) \ge ratio(v_i, u')$ for all u'

```
Algorithm 1: RatioGreedy
   input  : $V, U, \{c_v\}, \{\mu(v,u)\}, \{t_1^v, t_2^v\}, \{b_u\}, \{cost\}$
   output: A feasible planning $A$
1  $S_u \leftarrow \emptyset, \forall u$;
2  $H \leftarrow \emptyset$;
3  foreach $v$ do
4      $u_v \leftarrow \arg\max_{u|S_u=\{v\} \text{ is valid}}\ ratio(v,u)$;
5      Push $(v, u_v)$ into $H$ if $u_v\exists$;
6  foreach $u$ do
7      $v_u \leftarrow \arg\max_{v|S_u=\{v\} \text{ is valid}}\ ratio(v,u)$;
8      Push $(v_u, u)$ into $H$ if $v_u\exists$ and $(v_u, u) \notin H$;
9  while $H \neq \emptyset$ do
10     Pop $(v,u)$ with the largest $ratio(v,u)$ from $H$;
11     Add $v$ into $S_u$ if $\{v\} \cup S_u$ is valid;
12     if $v$ is not full of capacity then
13         $u_v \leftarrow$
           $\arg\max_{u'|S_{u'}=\{v\}\cup S_{u'} \text{ is valid and } v \notin S_{u'}}\ ratio(v,u')$;
14         Push $(v, u_v)$ into $H$ if $u_v\exists$ and $(v, u_v) \notin H$;
15     foreach original $(v_k, u) \in H$ do
16         Remove $(v_k, u)$ from $H$ if $inc\_cost(v_k, u)$
           changes, continue otherwise;
17         $u_{v_k} \leftarrow$
           $\arg\max_{u'|S_{u'}=\{v_k\}\cup S_{u'} \text{ is valid and } v_k \notin S_{u'}}\ ratio(v_k,u')$;
18         Push $(v_k, u_{v_k})$ into $H$ if $u_{v_k}\exists$ and $(v_k, u_{v_k}) \notin H$;
19     $v_u \leftarrow$
       $\arg\max_{v'|S_u=\{v'\}\cup S_u \text{ is valid and } v' \notin S_u}\ ratio(v',u)$;
20     Push $(v_u, u)$ into $H$ if $v_u\exists$ and $(v_u, u) \notin H$;
21 $A \leftarrow \cup_u\{S_u\}$;
22 return $A$
```

whose schedule $\{v_i\}$ satisfies all the constraints. If there are multiple u_{v_i}'s whose ratio values are the same, we pick the one with the least *inc_cost*. We also find a v_{u_j} for each $u_j \in U$ in a similar way. Each pair $(v_i, u_{v_i})((v_{u_j}, u_j))$ is pushed into H. If for some i and j, $v_i = v_{u_j}$ and $u_{v_i} = u_j$, only one pair will be in H. Whenever a pair (v,u) is popped from H, we update H as follows. If v is not yet full of capacity, we find the next u_v such that $ratio(v,u_v) \geq ratio(v,u')$ for all u' such that v is not yet in $S_{u'}$ and the schedule $\{v\} \cup S_{u'}$ satisfies all the constraints. Again, if there are multiple u's whose ratio values are the same, we pick the one with the least *inc_cost*. For u, we also find its corresponding v_u in a similar way. The new found pair(s) will be pushed into H. If there are some other pairs (v_k, u) in H that are incident to u, since the schedule of u may have been updated and thus $inc\_cost(v_k, u)$ may have changed, we need to find a possibly new u_{v_k} for each v_k to ensure that for v_k, the pair (v_k, u_{v_k}) that has the largest ratio value and satisfies all the constraints is in H. Therefore, we update the u_{v_k} for each v_k and also update H accordingly. The whole procedure terminates when no more feasible pair can be pushed into H and H is empty.

The procedure of the RatioGreedy algorithm is illustrated in Algorithm 1. In lines 2-8, we initialize H by finding a valid event-user pair with the largest ratio value for each event(user). In lines 9-20, we repetitively pop the pair (v,u) with the largest ratio value from H, add the pair to the planning if possible and update H. In lines 12-14 and lines 19-20, we find a new valid event-user pair with the largest ratio value for v and u respectively. In lines 15-18, we update the elements in H that contain u in case that the *inc_cost*'s and thus the ratio values for such elements have changed. The iteration terminates when H is empty.

Table 3: Example 2

	u_1	u_2	u_3	u_4	u_5
v_1	0.011(18)	0.15(4)	0.175(4)	0.05(6)	0.0375(16)

	States of H
After Initialization	$(v_4,u_1):0.2(2)$, $(v_1,u_3):0.175(4)$, $(v_1,u_2):0.15(4)$, $(v_3,u_3):0.075(12)$, $(v_1,u_4):0.05(6)$, $(v_1,u_5):0.0375(16)$, $(v_2,u_5):0.023(22)$
After 1st Iteration	$(v_1,u_3):0.175(4)$, $(v_1,u_2):0.15(4)$, $(v_3,u_3):0.075(12)$, $(v_1,u_4):0.05(6)$, $(v_1,u_5):0.0375(16)$, $(v_3,u_1):0.0375(16)$, $(v_4,u_2):0.035(20)$, $(v_2,u_5):0.023(22)$

EXAMPLE 2 (RATIOGREEDY). *Back to our running example in Example 1. Initially, H is empty. In the second row of Table 3, we present the ratio values between v_1 and different users and also the inc_cost (in brackets). As pair (v_1, u_3) is valid and has the largest ratio value, we push (v_1, u_3) into H. We continue the initialization step, and present the state of H after initialization in Table 3, where we present the ratio value of each event-user pair in H and also its inc_cost (in bracket). Then during the 1st iteration, the pair v_4, u_1 is popped from H as it has the largest ratio value 0.2. Then we add v_4 to S_{u_1}, and update H, the state of which is presented in Table 3. The procedure continues until H is empty, and we have the following planning $S_{u_1} = \{v_3, v_4\}, S_{u_2} = \{v_3, v_4\}, S_{u_3} = \{v_1\}$ and $S_{u_5} = \{v_3, v_2\}$, which has total utility score* 3.6.

Complexity analysis. In RatioGreedy, initializing H takes $O(|V||U|)$ time as we scan through each event-user pair. Then during each iteration, it takes $O(|S_u|)$ time to check the validity of the popped pair and $O((|V|+|U|)|S_u|)$ time to push new pairs into H. Also, as there are at most $|V|$ event-user pairs in H that are incident to u, it takes at most $O(|V||U||S_u|)$ additional time to update the elements in H. Since there are at most $|V||U|$ event-user pairs, the number of iterations is $O(|V||U|)$. Thus, the overall time complexity of RatioGreedy is $O(|V||U|(|V|+|U|+|V||U|)|S_u|)$ in the worst case. The major space consumption of RatioGreedy comes from H and S_u. Thus, the space complexity of RatioGreedy is $O(|V||U|)$.

4. A TWO-STEP APPROXIMATION FRAMEWORK

In this section, due to the NP-hardness of the USEP problem, we present a two-step approximate solution framework. Based on the framework, we present a *De*composed *D*ynamic *P*rogramming (DeDP) approximation algorithm with approximation ratio $\frac{1}{2}$. To improve the space and time efficiency and also the effectiveness of the DeDP algorithm, we further present techniques that optimize the DeDP algorithm. Finally, we present a DeGreedy algorithm following the two-step approximation solution framework, which is faster than the DeDP algorithm but returns a planning with lower total utility score.

4.1 Preliminary

We first introduce some background of the framework. Specifically, our framework is based on the decomposition idea of Local Ratio Theorem[3][7], which is stated as follows.

Theorem 2 (Local Ratio[3]). *Let $\max f(x) = w \cdot x$ be the objective function, where $w \in \mathbb{R}^n$ is a weight vector, and $x \in \mathbb{R}^n$ is a solution vector satisfying a set of constraints C. Let $w = w_1 + w_2$, and $x \in \mathbb{R}^n$ be an r-approximate solution w.r.t. $w_1 \in \mathbb{R}^n$ and $w_2 \in \mathbb{R}^n$, then x is also an r-approximate solution w.r.t. w.*

Theorem 2 indicates that we can decompose the weight function and thus the USEP problem into a set of simpler problems and aggregate the solutions of the simpler problems to obtain a final solution for the USEP problem. Details of the algorithm are presented in the following.

4.2 DeDP Algorithm

In the DeDP algorithm, the main idea is to first decompose the USEP problem into $|U|$ problems following [3][7], where in each of the $|U|$ problems we design a dynamic programming algorithm to find a feasible schedule for a user u that can maximize the total utility score of the schedule. In the second step, we combine the solutions of each user and resolve the case where certain constraints are violated.

4.2.1 First Step

In the first step, we decompose the original USEP problem into $|U|$ problems. First, we sort the events in V in non-descending order of t_2^v, and denote the i-th element in the sorted list as v_i. That is, $t_2^{v_j} \leq t_2^{v_i}, \forall j < i$. Second, we construct a set of *pseudo-events* with cardinality c_{v_i} for each v_i. Specifically, for each $v_i \in V$ with capacity c_{v_i}, we construct a set of pseudo-events $\{v_{i,k}\}$ where $1 \leq k \leq c_{v_i}$, and each $v_{i,k}$ is identical to v_i except that $c_{v_{i,k}} = 1, \forall 1 \leq k \leq c_{v_i}$. Notice that if $c_{v_i} > |U|$, we simply change c_{v_i} to $|U|$. Thus, $c_{v_i} \leq |U|, \forall v_i$. When arranging events for users, instead of arranging v_i to users directly, we arrange one of the pseudo-events $\{v_{i,k}\}$ to a user. Each $v_{i,k}$ is associated with a utility value $\mu(v_{i,k}, u)$ w.r.t. each user u, where $\mu(v_{i,k}, u) = \mu(v_i, u), \forall 1 \leq k \leq c_{v_i}$ initially. The values of $\mu(v_{i,k}, u)$ will be decomposed as we solve the problem, details of which will be presented shortly. Finally, after constructing the pseudo-event sets, we decompose the USEP problem into $|U|$ problems and iteratively solve each decomposed problem. In the r-th decomposed problem, we find a schedule for user u_r with a dynamic programming algorithm, and update the values of $\mu(v_{i,k}, u)$ accordingly.

More specifically, in the r-th iteration, where $1 \leq r \leq |U|$, we find a schedule for user u_r. For easy illustration, we denote $\mu^r(v_{i,k}, u)$ as the value of $\mu(v_{i,k}, u)$ when we start to process the r-th user, which is updated after the last ($(r-1)$-th) iteration. Particularly, $\mu^1(v_{i,k}, u) = \mu(v_{i,k}, u)$ initially. Then for each event v_i, we first find from its pseudo-event set $\{v_{i,k}\}$ the element that has the largest updated utility value w.r.t. u_r, i.e. $\max_k \mu^r(v_{i,k}, u_r)$, denoted as $\hat{v}_i$ (ties are broken arbitrarily). Let $V_r = \{\hat{v}_i | \mu^r(\hat{v}_i, u_r) > 0\}$. Thus, any $\hat{v}_i$ with non-positive utility value is excluded from V_r. We next find a schedule for user u_r from V_r with a dynamic programming algorithm.

Lemma 1. *For any v_i such that $cost(u, v_i) + cost(v_i, u) > b_u$, it cannot be included in any valid schedule of u.*

Lemma 1, the proof of which is in the Appendix, indicates that we can remove events with $cost(u_r, \hat{v}_i) + cost(\hat{v}_i, u_r) > b_{u_r}$ from V_r as they cannot be in the schedule of u_r. Denote V_r' as the resulted set of events. Then in the dynamic programming algorithm, we scan through each pseudo-event $\hat{v}_i \in V_r'$, and find the best utility score we can obtain with $\hat{v}_i$ being the last event to attend in the schedule. Let $S_{u_r}^{i,T} = \{v^{u_r,i,T}\}$ (simplified as $S^{i,T} = \{v^{i,T}\}$) denote the schedule for u_r with $\hat{v}_i$ being the last event to attend and T being the total travel cost of completing the schedule at $\hat{v}_i$, i.e. $T = cost(u_r, v_1^{i,T}) + \sum_{j=2}^{|S^{i,T}|} cost(v_{j-1}^{i,T}, v_j^{i,T})$ and $v_{|S^{i,T}|}^{i,T}$ is $\hat{v}_i$. Let $\Omega(i,T)$ be the largest utility score of $S^{i,T}$. And we have the following equation to compute $\Omega(i,T)$.

$$\Omega(i,T) = \begin{cases} \mu^r(\hat{v}_i, u_r) & T = cost(u_r, \hat{v}_i) \\ \max_{1 \leq l \leq l_i} & \\ \{\Omega(l, T - cost(\hat{v}_l, \hat{v}_i)) & T \geq cost(\hat{v}_l, \hat{v}_i) \text{ and} \\ +\mu^r(\hat{v}_i, u_r)\} & T + cost(\hat{v}_i, u_r) \leq b_{u_r} \\ 0 & otherwise \end{cases} \quad (4)$$

In Equation (4), l_i is the largest index l such that $t_2^{v_l} \leq t_1^{v_i}$. Recall that $t_2^{v_j} \leq t_2^{v_i}, \forall j < i$. Therefore, $\forall l > l_i$, we have $t_2^{v_l} > t_1^{v_i}$ and thus $\hat{v}_i$ cannot be attended if $\hat{v}_l$ is in the schedule of u_r. By computing $\Omega(i,T)$, we can then find a schedule $\hat{S}_{u_r}$ that corresponds to the largest $\Omega(i,T) (\forall i, 0 \leq T \leq b_{u_r})$ value for u_r.

After finding schedule $\hat{S}_{u_r}$, we finally update the values of μ^r as follows. If the schedule $\hat{S}_{u_r}$ contains pseudo-event $\hat{v}_i$ (recall that $\hat{v}_i$ is one of $\{v_{i,k}\}$), $\mu^{r+1}(\hat{v}_i, u_j)$ is updated as $\mu^r(\hat{v}_i, u_j) - \mu^r(\hat{v}_i, u_r)$ for all $j > r$. In addition, $\mu^{r+1}(v_{i,k}, u_r) = 0, \forall i, k$. The other μ^{r+1} values are the same as their corresponding μ^r. The intuition is to ensure that when $\hat{v}_i$ is arranged to u_r, $\hat{v}_i$ will be rearranged to $u_j (j > r)$ only if $\mu^j(\hat{v}_i, u_j) > 0$ and thus $\mu(v_i, u_j) > \mu(v_i, u_r)$ initially. Then in our second step, $\hat{v}_i$ will be removed from $\hat{S}_{u_r}$ in such case to ensure that $\hat{v}_i$ is only arranged to one single user. Details of this step will be described shortly. After updating the values of $\mu^r(v_{i,k}, u)$, we proceed to the next iteration to find a schedule for user u_{r+1} until $r = |U|$.

The procedure of finding individual schedule for each user is presented in Algorithm 2. In line 1, we construct V_r' according to Lemma 1. In lines 2-9, we compute $\Omega(i,T)$ according to Equation (4). In lines 10-11, we find the schedule with the largest $\Omega(i,T)$ value as $\hat{S}_{u_r}$.

4.2.2 Second Step

In the second step, we resolve the case where $v_{i,k}$ is arranged to multiple users, which violates the capacity constraint of $v_{i,k}$ and thus also that of v_i, since every $v_{i,k}$ must be arranged in this case. Specifically, we scan through u_r in the order of $r = |U|, |U| - 1, ..., 1$. For $r = |U|$, $S_{u_r} = \hat{S}_{u_r}$. And for each $r < |U|$, any $v_{i,k}$ that is already in $\cup_{j>r} S_{u_j}$ is removed from $\hat{S}_{u_r}$, which results in S_{u_r} as the final schedule for u_r. After this step, we obtain a planning where each $v_{i,k}$ is arranged to only one user and thus the capacity constraint of v_i is satisfied. The other three constraints (i.e. budget constraint, feasibility constraint and utility constraint), which are ensured to be satisfied in the first step, are not affected by the removal of events from a schedule. Therefore, the resulted planning of the DeDP algorithm is feasible.

```
Algorithm 2: DPSingle
   input  : $V_r, u_r, \mu^r, \{t_1^v, t_2^v\}, b_{u_r}, \{cost\}$
   output: A feasible schedule $\hat{S}_{u_r}$
1  $V_r' \leftarrow \{v \in V_r | cost(u_r, v) + cost(v, u_r) \le b_{u_r}\}$;
2  for $i \leftarrow 1$ to $|V_r'|$ do
3      $\Omega(i, cost(u_r, \hat{v}_i)) \leftarrow \mu^r(\hat{v}_i, u_r)$;
4      $path(i, cost(u_r, \hat{v}_i) \leftarrow 0$;
5      for $l \leftarrow 1$ to $l_i$ do
6          foreach $T$ s.t. $\Omega(l, T) > 0$ do
7              if $T + cost(\hat{v}_l, \hat{v}_i) + cost(\hat{v}_i, u_r) \le b_{u_r}$ and
               $\Omega(l, T) + \mu^r(\hat{v}_i, u_r) > \Omega(i, T + cost(\hat{v}_l, \hat{v}_i))$
               then
8                  Update $\Omega(i, T + cost(\hat{v}_l, \hat{v}_i))$;
9                  $path(i, T + cost(\hat{v}_l, \hat{v}_i)) \leftarrow l$;
10 Find the largest $\Omega(i, T)$;
11 Construct $\hat{S}_{u_r}$ according to $path(i, T)$;
12 return $\hat{S}_{u_r}$
```

```
Algorithm 3: DeDP
   input  : $V, U, \{c_v\}, \{\mu(v, u)\}, \{t_1^v, t_2^v\}, \{b_u\}, \{cost\}$
   output: A feasible planning $A$
1  $\forall v_i$ s.t. $c_{v_i} > |U| : c_{v_i} \leftarrow |U|$;
2  $\mu^1(v_{i,k}, u_j) \leftarrow \mu(v_i, u_j), \forall 1 \le i \le |V|, 1 \le k \le c_{v_i}, 1 \le j \le |U|$;
3  for $r \leftarrow 1$ to $|U|$ do
4      $V_r \leftarrow \{\hat{v}_i | \mu^r(\hat{v}_i, u_r) > 0\}$;
5      $\hat{S}_{u_r} \leftarrow$ DPSingle ($V_r, u_r, \mu^r, \{t_1^v, t_2^v\}, b_{u_r}, \{cost\}$);
6      Update $\mu^r$;
7  for $r \leftarrow |U|$ downto 1 do
8      $S_{u_r} \leftarrow \{v_i |$the corresponding $\hat{v}_i \in \hat{S}_{u_r} \setminus (\cup_{j>r} \hat{S}_{u_j})\}$;
9  $A \leftarrow \cup_u \{S_u\}$;
10 return $A$
```

4.2.3 Summary

The main procedure of the DeDP algorithm is summarized in Algorithm 3. In line 1, we temporally let $c_{v_i} \le |U|, \forall v_i \in V$. In line 2, we initialize μ^1. In lines 3-6, which is the first step of DeDP, we decompose the USEP problem into $|U|$ problems by finding a schedule for each u_r. In line 4, we construct the set V_r using the selected pseudo-events. We find the schedule for u_r using Algorithm 2 and update μ^r in lines 5-6. In lines 7-9, which is the second step of DeDP, we resolve the case where $v_{i,k}$ is contained in multiple schedules.

EXAMPLE 3. *Back to our running example in Example 1. Note that in this example, the sorted list of V is v_3, v_1, v_2, v_4, and we still use the original subscript of v's instead of referring to v_3 as v_1, v_1 as v_2 etc. as we do in the algorithm. In the first iteration, the values of μ^1 are identical to those of μ. We present the selected pseudo-events and their corresponding μ^1 values (in bracket) in the second row of Table 4. We then construct V_1', which is presented in Table 5, and find a schedule for u_1 using dynamic programming, events of which are shown in bold font in Table 5. Then the values of μ^2 are updated. We show the selected pseudo-events and their updated μ^2 values (in bracket) in the third row of Table 4. After updating μ^2, we then proceed to find a schedule*

Table 4: Updated μ^r of Example 3

	$\hat{v}_1$	$\hat{v}_2$	$\hat{v}_3$	$\hat{v}_4$
u_1	$v_{1,1}(0.2)$	$v_{2,1}(0.5)$	$v_{3,1}(0.6)$	$v_{4,1}(0.4)$
u_2	$v_{1,1}(0.6)$	$v_{2,2}(0.1)$	$v_{3,2}(0.2)$	$v_{4,1}(0.7)$
u_3	$v_{1,1}(0.1)$	$v_{2,2}(0.3)$	$v_{3,2}(0.9)$	$v_{4,2}(0.2)$
u_4	$v_{1,1}(-0.3)$	$v_{2,3}(0.9)$	$v_{3,3}(0.4)$	$v_{4,2}(0.5)$
u_5	$v_{1,1}(0)$	$v_{2,3}(0.5)$	$v_{3,3}(0.5)$	$v_{4,2}(0.1)$

Table 5: Results of First Step of Example 3

User	V_r'
u_1	$\mathbf{v_{3,1}}$, $v_{1,1}$, $\mathbf{v_{2,1}}$, $v_{4,1}$
u_2	$v_{3,2}$, $\mathbf{v_{1,1}}$, $v_{2,2}$, $\mathbf{v_{4,1}}$
u_3	$\mathbf{v_{3,2}}$, $v_{1,1}$, $\mathbf{v_{2,2}}$, $v_{4,2}$
u_4	$\emptyset$
u_5	$\mathbf{v_{3,3}}$, $\mathbf{v_{2,3}}$, $v_{4,2}$

for u_2. After the first step, the initial planning is shown in bold font in Table 5. Then after the second step, we have the final planning: $S_{u_1} = \{v_3, v_2\}, S_{u_2} = \{v_1, v_4\}, S_{u_3} = \{v_3, v_2\}, S_{u_5} = \{v_3, v_2\}$. The total utility score is 4.6.

Complexity analysis. In the first step of DeDP, the number of iterations is $|U|$. Then in each of the decomposed $|U|$ problems, it takes $O(|V| \max_v\{c_v\})$ time to construct V_r and $O(|V|^2 \max_u\{b_u\})$ time to compute each $\Omega(i, CT)$ value in the worst case. Constructing $\hat{S}_{u_r}$ takes $O(|V|)$ time. Updating μ^r takes $O(|V||U|)$time. Therefore, the first step of DeDP takes $O(|U|(|V|^2 \max_u\{b_u\} + |V||U|))$ time. The second step of DeDP takes $O(|V||U|)$ time to obtain the final schedules. Thus, the overall time complexity of DeDP is $O(|U|(|V|^2 \max_u\{b_u\} + |V||U|))$.

The major space consumption of DeDP comes from storing values of μ^r, which takes $O(|V||U| \max_v\{c_v\})$ space. Computing $\Omega(i, CT)$ takes $O(|V| \max_u\{b_u\})$ space, and storing schedules of users takes $O(|V||U|)$ space. Thus, the overall space complexity of DeDP is $O(|V||U| \max_v\{c_v\} + |V| \max_u\{b_u\})$.

Approximation ratio. We next present the approximation ratio of DeDP with the following theorem, the proof of which is in the Appendix.

THEOREM 3. *DeDP has approximation ratio of $\frac{1}{2}$, i.e. $\Omega(A) \ge \frac{1}{2}\Omega(A^\star)$, where A is the planning returned by DeDP, and $A^\star$ is the optimal planning.*

4.3 Optimizing DeDP Algorithm

Note that though the DeDP algorithm returns a planning with guaranteed approximation ratio of $\frac{1}{2}$, it has the following issues. First, notice that storing values of μ^r requires $O(|V||U| \max_v\{c_v\})$ space, which is infeasible especially when $|U|$ and c_v are large. Also, updating values of μ^r takes time, which reduces the efficiency of DeDP. Second, since some events may be removed from some users' schedules, it is possible that the removed events may be added to some schedules to improve the resulted planning. In this subsection, we address these two issues and propose two enhanced algorithms, DeDPO and DeDPO+RG, which still have guaranteed approximation ratio of $\frac{1}{2}$.

4.3.1 Optimizing Space and Speed

We first optimize the space consumption of DeDP by reducing the space of μ^r, which is the major space consumption of DeDP. Reducing μ^r can also speed up DeDP. We call this improved algorithm DeDPO. We first present the following lemma, the proof of which is in the Appendix.

Algorithm 4: DeDPO

```
   input  : $V, U, \{c_v\}, \{\mu(v,u)\}, \{t_1^v, t_2^v\}, \{b_u\}, \{cost\}$
   output: A feasible planning $A$
1  $\forall v_i$ s.t. $c_{v_i} > |U| : c_{v_i} \leftarrow |U|$;
2  $select(v_i,k) \leftarrow 0, \forall 1 \le i \le |V|, 1 \le k \le c_{v_i}$;
3  for $r \leftarrow 1$ to $|U|$ do
4      for $i \leftarrow 1$ to $|V|$ do
5          $k' \leftarrow \arg\max_k(\{\mu(v_i,u_r)|select(v_i,k)=0\} \cup$
           $\{\mu(v_i,u_r)-\mu(v_i,select(v_i,k))|select(v_i,k)>0\})$;
6          $\hat{v}_i \leftarrow v_{i,k'}$;
7          $\mu'(\hat{v}_i) \leftarrow \mu(v_i,u_r)$ if $select(v_i,k')=0$ or
           $\mu(v_i,u_r)-\mu(v_i,select(v_i,k'))$ otherwise;
8      $V_r \leftarrow \{\hat{v}_i|\mu'(\hat{v}_i)>0\}$;
9      $\hat{S}_{u_r} \leftarrow$ DPSingle ($V_r, u_r, \mu', \{t_1^v,t_2^v\}, b_{u_r}, \{cost\}$);
10     Update $select$;
11 $S_{u_r} \leftarrow \emptyset, \forall 1 \le r \le |U|$;
12 for $i \leftarrow 1$ to $|V|$ do
13     for $k \leftarrow 1$ to $c_{v_i}$ do
14         $S_{select(v_i,k)} \leftarrow S_{select(v_i,k)} \cup \{v_i\}$ if
           $select(v_i,k)>0$;
15 $A \leftarrow \cup_u\{S_u\}$;
16 return $A$
```

Lemma 2. *For a pseudo-event $v_{i,k}$, let $u_{r_1}, u_{r_2}, ..., u_{r_n}$ be the list of users who have $v_{i,k}$ in their schedules $\hat{S}_{u_{r_j}}$, where $1 \le r_1 < ... < r_j < ... < r_n \le |U|$. It holds that $\forall l$ s.t. $1 \le l \le r$,*

$$\mu^l(v_{i,k},u_r) = \begin{cases} \mu(v_i,u_r) & 1 \le l \le r_1 \\ \mu(v_i,u_r) - \mu(v_i,u_{r_{j_l}}) & r_{j_l} < l \le r_{j_l+1}, j_l \ge 1 \end{cases} \tag{5}$$

Notice that the values of $\mu^r(v_{i,k},u_r)$ are used only when DeDP is performing the r-th iteration in the first step. Lemma 2 states that instead of keeping tracking of and updating $\mu^r(v_{i,k},u_r)$ during the whole procedure of the first step of DeDP, we only need to know *who is the last user to have $v_{i,k}$ in her/his schedule $\hat{S}$* when DeDP starts to process the r-th user in the first step, which has similar consideration as [7]. Therefore, we can remove storage of μ^r values by defining a new $|V| \times c_v$ array $select(v_i,k)$, where $select(v_i,k)$ keeps tracking of the last user to have $v_{i,k}$ in her/his $\hat{S}$ before execution of the r-th iteration of the first step. That is, $select(v_i,k) = \max\{j|1 \le j < r, v_{i,k} \in \hat{S}_{u_j}\}$.

The procedure of the improved algorithm DeDPO is illustrated in Algorithm 4. In line 2, we initialize $select(v_i,k)$. In lines 4-7, we find the pseudo-event with the largest utility value for each v_i as we do in DeDP, and calculate the updated μ values, which are stored in a temperate array μ'. Note that $\mu'(\hat{v}_i)$ is equivalent to $\mu^r(\hat{v}_i,u_r)$ in DeDP. We construct V_r in line 8 and find an optimal schedule for u_r based on μ' in line 9. In line 10, we update the values of $select(v_i,k)$ with u_r if $v_{i,k}$ is included in $\hat{S}_{u_r}$. In lines 11-14, we construct the planning by arranging v_i to the last user who has $v_{i,k}$ in $\hat{S}$, which is already recorded in $select(v_i,k)$. Thus, step 2 of DeDPO is equivalent to step 2 of DeDP.

Complexity analysis. DeDPO improves the time efficiency of DeDP by saving $O(|V||U|)$ time to update the values of μ^r in each iteration. Therefore, the first step of DeDPO takes $O(|U|(|V|^2 \max_u\{b_u\} + |V| \max_v\{c_v\}))$ time. The second step of DeDP takes $O(|V| \max_v\{c_v\})$ time to obtain the final schedules. Thus, the overall time complexity of DeDPO is $O(|U|(|V|^2 \max_u\{b_u\} + |V| \max_v\{c_v\}))$, which reduces that of DeDP by $O(|U||V|(|U| - \max_v\{c_v\}))$.

DeDPO improves the space consumption of DeDP by saving $O(|V||U| \max_v\{c_v\})$ space to store values of μ^r while introduces $O(|V| \max_v\{c_v\})$ extra space to store values of *select*. Therefore, the overall space complexity of DeDPO is $O(|V| \max_v\{c_v\} + |V| \max_u\{b_u\} + |V||U|)$, which reduces that of DeDP by one order of magnitude.

4.3.2 *Optimizing Utility*

Notice that in second step of DeDPO (DeDP), some of the events may not be full in terms of capacity, i.e. some of the $select(v_i,k)$ may be 0 for some v_i. Also. some arranged events in $\hat{S}$ of some users may be removed in the second step, and thus such users may still have enough budget to attend some more events. Therefore, our second optimization for DeDPO (DeDP) is to improve the total utility score of the planning by adding events whose capacity are not yet full to schedules of users whose budgets are not yet exceeded.

Specifically, after running the DeDPO(DeDP) algorithm, we obtain an initial planning A. We construct a set of events $V' = \{v|v$ is not full of capacity$\}$, including all the events that are not yet full of capacity w.r.t. A. We next run RatioGreedy algorithm with input V' and U to add more arranged pairs into A if possible. Note that when running RatioGreedy (Algorithm 1), in lines 1-8, the *inc_cost* of each event-user pair should be calculated by taking the existing schedules of A into consideration. We call this algorithm DeDPO+RG. Note that the planning A' returned by DeDPO+RG is still a $\frac{1}{2}$-approximate solution.

4.4 DeGreedy Algorithm

Note that the DeDP algorithm, even the DeDPO algorithm, could be time-consuming since in each decomposed problem of the first step, it takes $O(|V|^2 \max_u\{b_u\})$ time to find an optimal schedule for the user using the dynamic programming algorithm. Thus, in this subsection, we introduce the DeGreedy algorithm, where instead of using the dynamic programming algorithm to solve each decomposed problem, we use a ratio-based greedy algorithm to find a suboptimal schedule for each user in a faster way.

More specifically, the framework of DeGreedy is the same as that of DeDPO, as we use the techniques of DeDPO to improve the space and time efficiency of DeGreedy. The only difference is that we find a schedule for user u_r in the r-th iteration of the first step in a greedy way. We then only describe the details of the greedy algorithm to find a schedule for u_r. Initially, $\hat{S}_{u_r}$ is empty. We then iteratively add a valid event v with the largest ratio score, which is defined in Equation (2), into $\hat{S}_{u_r}$. The validity of v is that adding v into $\hat{S}_{u_r}$ does not violate the travel budget and schedule feasibility constraints of u_r. Note that the utility constraint is ensured to be satisfied when we construct V_r. We maintain a heap H to keep tracking of feasible event candidates and pop the one with the largest ratio score from H and add it to $\hat{S}_{u_r}$ during each iteration.

The details of maintaining H are as follows. Initially, H is empty. We then scan through each pseudo-event in V'_r, which is constructed as in Algorithm 2, and push the one with the largest ratio score into H. Each time an pseudo-event $\hat{v}_i$ is popped from H and added to $\hat{S}_{u_r}$, we update H as follows. Let p_i be the index of the precedent of $\hat{v}_i$ in $\hat{S}_{u_r}$, and s_i be index of the successor of $\hat{v}_i$ in $\hat{S}_{u_r}$ if

```
Algorithm 5: GreedySingle
  input  : $V_r, u_r, \mu^r, \{t_1^v, t_2^v\}, b_{u_r}, \{cost\}$
  output: A feasible schedule $\hat{S}_{u_r}$
1  $V_r' \leftarrow \{v \in V_r | cost(u_r, v) + cost(v, u_r) \le b_{u_r}\}$;
2  $v \leftarrow \arg\max_{v \in V_r'} ratio(v, u_r)$;
3  Push $v$ into $H$;
4  $\hat{S}_{u_r} \leftarrow \emptyset$;
5  while $H \neq \emptyset$ do
6      Pop $\hat{v}_i$ with the largest $ratio(\hat{v}_i, u_r)$ from $H$;
7      Add $\hat{v}_i$ into $\hat{S}_{u_r}$;
8      if $\hat{v}_i$ is the first in $\hat{S}_{u_r}$ then
9          $v' \leftarrow \arg\max_{\text{valid } v \in \{\hat{v}_1, \cdots, \hat{v}_{i-1}\} \cap V_r'} ratio(v, u_r)$;
10     else
11         $v' \leftarrow$
           $\arg\max_{\text{valid } v \in \{\hat{v}_{p_i+1}, \cdots, \hat{v}_{i-1}\} \cap V_r'} ratio(v, u_r)$;
12     Add $v'$ into $H$ if $v' \exists$;
13     if $\hat{v}_i$ is the last in $\hat{S}_{u_r}$ then
14         $v' \leftarrow \arg\max_{\text{valid } v \in \{\hat{v}_{i+1}, \cdots, \hat{v}_{|V|}\} \cap V_r'} ratio(v, u_r)$;
15     else
16         $v' \leftarrow$
           $\arg\max_{\text{valid } v \in \{\hat{v}_{i+1}, \cdots, \hat{v}_{s_i-1}\} \cap V_r'} ratio(v, u_r)$;
17     Add $v'$ into $H$ if $v' \exists$;
18 return $\hat{S}_{u_r}$
```

they exist. That is, u_r attends $\hat{v}_{p_i}$, $\hat{v}_i$ and $\hat{v}_{s_i}$ respectively in order w.r.t. $\hat{S}_{u_r}$. We then scan through each pseudo-event in $\{\hat{v}_{p_i+1}, \hat{v}_{p_i+2}, \cdots, \hat{v}_{i-1}\}$ ($\{\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_{i-1}\}$ if $\hat{v}_i$ is the first event in $\hat{S}_{u_r}$), and push the one that is valid and has the largest ratio score into H. Similarly, we scan through each pseudo-event in $\{\hat{v}_{i+1}, \hat{v}_{i+2}, \cdots, \hat{v}_{s_i-1}\}$ ($\{\hat{v}_{i+1}, \hat{v}_{i+2}, \cdots, \hat{v}_{|V|}\}$ if $\hat{v}_i$ is the last event in $\hat{S}_{u_r}$), and push the one that is valid and has the largest ratio score into H. The following lemma states that the way we maintain H ensures that we pop a valid event with the largest ratio score among all the other valid event candidates from H each time, the proof of which is in the Appendix.

Lemma 3. *Let v be the latest event popped from H. It holds that $ratio(v, u_r) = \max_{valid\ v' \in V_r' \setminus \hat{S}_{u_r}} ratio(v', u_r)$.*

The main procedure of DeGreedy is the same as that of Algorithm 4, except that in line 9 of Algorithm 4, we call procedure *GreedySingle* instead to find the schedule $\hat{S}_{u_r}$. The procedure of *GreedySingle* is illustrated in Algorithm 5. In line 1, we construct V_r' as we do in DeDPO. In lines 2-3, we find a valid event with the largest ratio value and push it into H. In lines 5-17, we keep popping events from H and updating $\hat{S}_{u_r}$ and H until H is empty.

Example 4. *Back to our running example in Example 1. DeGreedy returns the following planning after the first step: $\hat{S}_{u_1} = \{v_{3,1}, v_{4,1}\}, \hat{S}_{u_2} = \{v_{1,1}, v_{4,2}\}, \hat{S}_{u_3} = \{v_{3,2}, v_{2,1}\}, \hat{S}_{u_5} = \{v_{3,3}, v_{2,2}\}$. After the second step, the final planning is as follows: $S_{u_1} = \{v_3, v_4\}, S_{u_2} = \{v_1, v_4\}, S_{u_3} = \{v_3, v_2\}, S_{u_5} = \{v_3, v_2\}$. The total utility score is* 4.5.

Notice that similar to DeDPO+RG, we can also run the RatioGreedy algorithm after the second step of the DeGreedy algorithm to improve the total utility score if possible. We call this improved algorithm DeGreedy+RG.

Table 6: Real Datasets

City	$\|V\|$	$\|U\|$	Mean of c_v	cr
Vancouver	225	2012	50	0.25
Auckland	37	569		
Singapore	87	1500		

Table 7: Synthetic Datasets

Factor	Setting
$\|V\|$	20, 50, **100**, 200, 500
$\|U\|$	100, 200, 500, 1000, **5000**
$\mu(v, u)$	**Uniform**, Normal(0.5, 0.25), Power: 0.5, 4
Mean of c_v	10, 20, **50**, 100, 200
Distributions of c_v	**Uniform**, Normal
f_b	0.5, 1, **2**, 5, 10
Distributions of b_u	**Uniform**, Normal
cr	0, **0.25**, 0.5, 0.75, 1
Scalability	$\|U\|$ = 10K, 20K, 30K, 40K, 50K, 100K

Complexity analysis. The difference between DeDPO and DeGreedy is GreedySingle. In GreedySingle, it takes $O(|V|)$ time to update H each time, and the number of iterations is at most $|V|$. Therefore, the overall complexity of DeGreedy is $O(|U|(|V|^2 + |V| \max_v\{c_v\}))$, and the space complexity is $O(|V| \max_v\{c_v\} + |V||U|)$.

5. EXPERIMENTAL STUDY

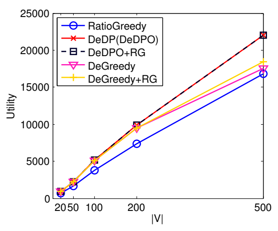
5.1 Experiment Setup

Parameters. We first define a *conflict ratio cr* to reflect the conflicting degree of events. Specifically, conflict ratio is the percentage of pairs of events that are spatio-temporally conflicting with each other. The time and cost values are generated based on the conflict ratio.

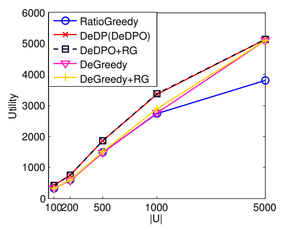
We next use a *budget factor f_b* to control the budget of user b_u. Specifically, given f_b, which is universal for all users, for each user u, the value of b_u is generated uniformly in $[2 \min_v cost(u, v), 2 \min_v cost(u, v) + mid \times f_b \times 2]$, where $mid = \frac{1}{2}(\max_{v,v'} cost(v, v') + \min_{v,v'} cost(v, v'))$.

Datasets. We use the Meetup dataset from [21] as real dataset. In the dataset, each user is associated with a set of tags and a location. Each event in the dataset is also associated with a location, and we use the tags of the group who creates the event as the tags of the event itself since the events themselves do not have tags. We use the similarity of the tags between an event and a user as their utility scores[36]. Since it is unlikely for a user living in a city to attend a meet-up event held in another city, we focus on the event/users pairs located in the same city. We select three popular cities, Vancouver, Auckland and Singapore, and extract events and users located within the area around each city. We use the Manhatan distance between a user and an event or that between two events as their travel cost in our experiments. We generate capacities of events following Uniform distribution, and generate the time and budgets with varying parameters described previously. TABLE 6 presents the statistics and configuration. We also use synthetic dataset for evaluation. We generate the utility values following Uniform, Normal and Power distributions respectively. The statistics and configuration of synthetic data are illustrated in TABLE 7, where we mark our default settings in bold font.

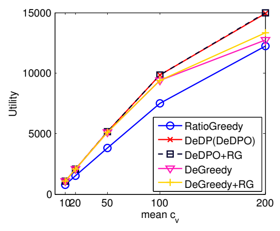
Notice that spatio-temporal conflicts, capacities and travel budgets are also generated syntactically for real datasets. The reason is that though such information is often contained or can be inferred from the description of events on Meetup, it is not contained in the dataset of [21].



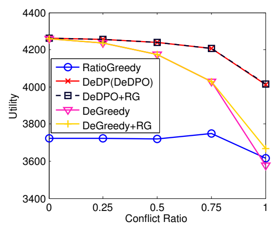
(a) Utility of varied $|V|$



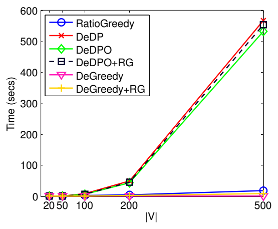
(b) Utility of varied $|U|$



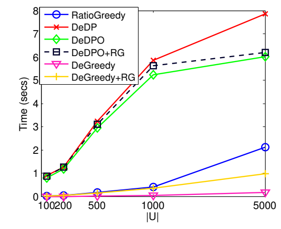
(c) Utility of varied c_v



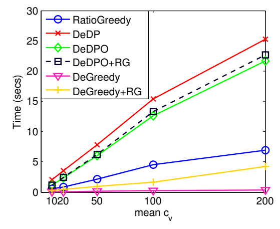
(d) Utility of varied cr



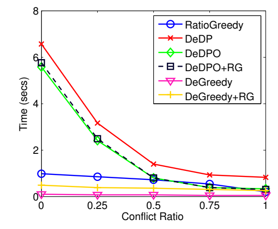
(e) Run time of varied $|V|$



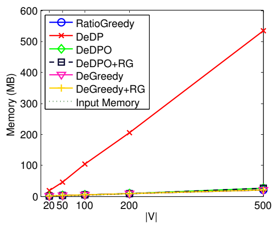
(f) Run time of varied $|U|$



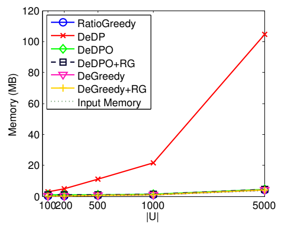
(g) Run time of varied c_v



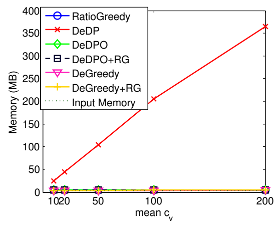
(h) Run time of varied cr



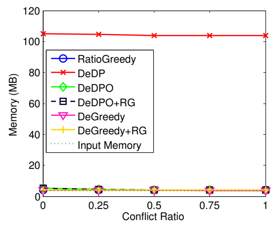
(i) Memory of varied $|V|$



(j) Memory of varied $|U|$



(k) Memory of varied c_v



(l) Memory of varied cr

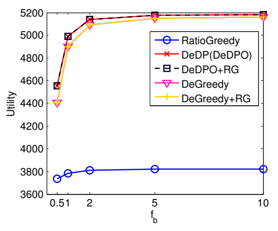
Figure 2: Results on varied $|V|$, $|U|$, mean of c_v, and conflict ratio.

We evaluate RatioGreedy, DeDP, DeDPO, DeDPO+RG, DeGreedy, and DeGreedy+RG in terms of total utility score, running time and memory cost, and study the effect of different parameters on the performance of the algorithms. The algorithms are implemented in C++, and the experiments were performed on a Windows 7 machine with Intel i7-2600 3.40GHZ 8-core CPU and 8GB memory.

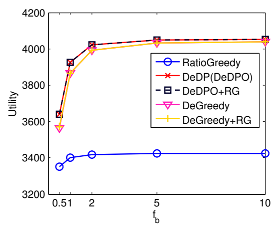
5.2 Experiment Results

Effect of cardinality of V. We first study the effect of $|V|$. The first column of Figure 2 presents the results when $|V|$ varies from 20 to 500. In terms of total utility scores, we can first observe that DeDP-based algorithms, i.e. DeDP(DeDPO) and DeDPO+RG, perform the best, while the heuristic-based RatioGreedy algorithm achieves the lowest utility scores. Note that the difference between two total utility scores can be considered as at least how many more interesting events are suggested to users. For example, the difference of 4500 and 5000 when $|V| = 100$ indicates that in the planning with utility score of 5000, at least 500 more interesting events are suggested to users compared to the one with total utility score of 4500. Second, DeGreedy+RG enhances DeGreedy to a certain degree, while DeDPO+RG does not enhance DeDPO significantly. The reason is that DeGreedy returns a worse planning than DeDPO does, and thus there could be more available event-user pairs in the planning returned by DeGreedy, which increases the possibility that the enhanced algorithm can add some more event-user pairs to the original planning. Third, we can observe that the utility scores increase as $|V|$ increases, which is natural as there are more events available for users when $|V|$ increases. As for running time, we can observe that DeDP-based algorithms run slower than other algorithms do, especially when $|V|$ is as large as 500. We can also see that DeDPO is slightly faster than DeDP as expected. Finally, in terms of memory consumption, we can observe that all the algorithms perform well except DeDP, which is highly memory-consuming. The results reflect that our techniques that optimize DeDP are very effective. Overall, DeDPO-based algorithms are the best in terms of utility score, while DeGreedy-based algorithms have a better trade-off between total utility score and running time.

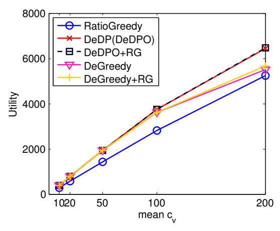
Effect of cardinality of U. We next study the effect of varying $|U|$, the results of which are presented in the second column of Figure 2. $|U|$ is varied from 100 to 5K. For utility score, we can again observe that DeDP-based algorithms perform the best, while RatioGreedy performs the worst. However, DeGreedy-based algorithms are almost as good as DeDP-based algorithms when $|U|$ is as large as 5000. The reason is that when $|U|$ is 5000, the number of events w.r.t. users is relatively small, and thus the optimality of DeDP-based algorithms in finding a schedule for each user becomes less significant. As for running time, we can observe that DeGreedy is the fastest, while DeDP is again the slowest among the six. Also, we can see that DeDPO is again more time-efficient than DeDP. Finally, for memory consumption, we can observe once again that DeDP, without optimization, performs much worse than the other algorithms. Overall, DeDPO-based algorithms still return the best planning, while DeGreedy-based algorithms again show better trade-off between optimality and running time.



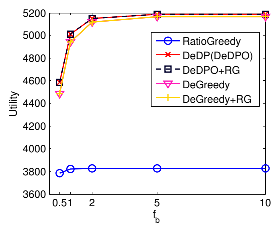
(a) Utility of varied f_b



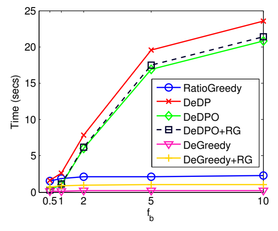
(b) Utility of Power(0.5) μ



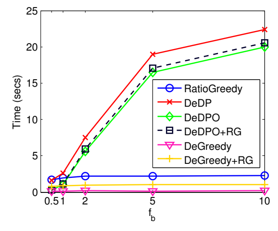
(c) Utility of Normal c_v



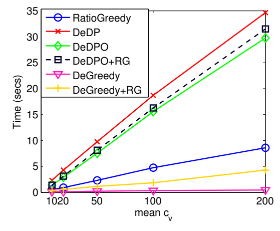
(d) Utility of Normal b_u



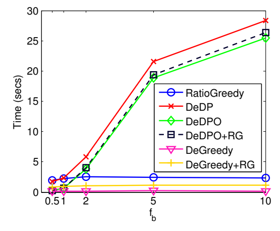
(e) Run time of varied f_b



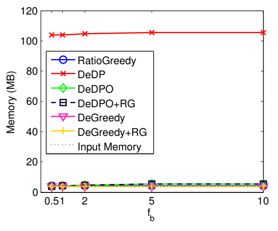
(f) Run time of Power(0.5) μ



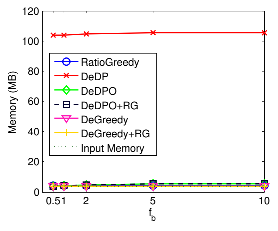
(g) Run time of Normal c_v



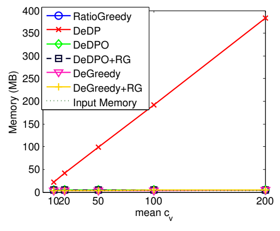
(h) Run time of Normal b_u



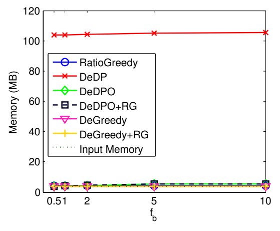
(i) Memory of varied f_b



(j) Memory of Power(0.5) μ



(k) Memory of Normal c_v



(l) Memory of Normal b_u

Figure 3: Results on varied f_b and distributions.

Effect of capacity. We next study the effect of varying the mean of c_v, the results of which are presented in the third column of Figure 2. We vary the mean of c_v from 10 to 200. For utility score, we can first observe that the utility scores increase as the mean of c_v increases, which is reasonable as more event-pairs can be added to the planning when the capacity of events increases. Second, DeDP-based algorithms again achieve the best utility results, while RatioGreedy is still the worst. We can finally observe that DeDPO+RG improves the utility results of DeDP(DeDPO) slightly, while DeGreedy+RG improves the utility results of DeGreedy more significantly. For running time, the running time of all algorithms increases as the mean of c_v increases, which makes sense as each event can be arranged for more users. Second, we can again obesve that DeGreedy is the fastest, while DeDP is the least time-efficient. Finally, as for memory consumption, DeDP without optimization is still the most memory-consuming algorithm.

Effect of conflict ratio. We then study the effect of varying conflict ratio cr, the results of which are presented in the last column of Figure 2. We vary the conflict ratio from 0.0 to 1.0. For utility score, our first observation is that the utility scores decrease as the conflict ratio increases. The reason is that when more pairs of events are conflicting with each other, the availability of events is decreased when the algorithms find schedules for users. As we can see from Figure 2d, the utility scores are quite low when $cr = 1$, in which case each user can attend at most one event as all events are conflicting with each other. Finally, we can observe that DeDP-based algorithms perform significantly better than the other algorithms when the conflict ratio increases. The reason is that DeDP-based algorithms always return optimal schedules for each user in each decomposed problem, while DeGreedy-based algorithms may return much worst schedules in each decomposed problem when the conflict ratio increases since the greedy strategy of DeGreedy does not consider conflicting degree of events. As for running time, the first interesting observation is that the running time of all the algorithms, especially that of DeDPO-based algorithms and DeGreedy-based algorithms, decreases as cr increases. The major reason is that when more events are conflicting with each other, the number of iterations for both DeDPO-based algorithms and DeGreedy-based algorithms will decrease. As we can see from Figure 2h, DeDPO-based algorithms are nearly as fast as DeGreedy-based algorithms when cr is 0.5 or above, in which cases the time efficiency advantage of the greedy strategy adopted by DeGreedy-based algorithms becomes less significant. Finally, for memory consumption, DeDP is once again the worst among all the algorithms. Overall, DeDPO-based algorithms demonstrate their advantages in utility results when conflict ratio increases, and they can be as fast as DeGreedy-based algorithms when conflict ratio increases.

Effect of budget factor. We next study the effect of varying budget factor f_b, the results of which are presented in the first column of Figure 3. We vary f_b from 0.5 to 10. For utility, we can first observe that the utility scores increase as f_b increases, which makes sense as users can attend more events as f_b increases and thus b_u increases. However, when $f_b \geq 2$, the utility scores increase much slower as f_b increases. The reason is that though budgets of users increase, the number of events available to users is limited.

Thus, when f_b is large, all events may have been full of capacity and thus no more events can be added to users' schedules though users' budgets are sufficient. As for running time and memory consumption results, we can observe again that DeGreedy-based algorithms are the most time-efficient and DeDP is the most memory-consuming.

Effect of distribution. We then study the effect of varying distributions of μ, c_v and b_u respectively. In the second column of Figure 3, we present the results when values of μ are generated following a power distribution with parameter 0.5. We plot against different f_b values, and the other parameters are set to default. We can observe that the trending patterns of utility scores, running time and memory consumption are similar to those in the first column of Figure 3, where values of μ are generated uniformly. We also generate values of μ following normal distribution with mean 0.5 and std 0.25 and a power distribution with parameter 4, whose results are similar and we omit them for brevity.

In the third column of Figure 3, we present the results when values of c_v are generated following a normal distribution, whose mean is varied from 10 to 200 and the std is the mean value multiplied by 0.25. The trending patterns of utility scores, running time and memory consumption are also similar to those in the third column of Figure 2, where the values of c_v are generated uniformly.

Finally, in the last column of Figure 3, we present the results when values of b_u are generated following a normal distribution, whose mean is $2\min_v cost(u,v) + mid * f_b$ and standard deviation is the mean value multiplied by 0.25, where $mid = \frac{1}{2}(\max_{v,v'} cost(v,v') + \min_{v,v'} cost(v,v'))$. We vary f_b from 0.5 to 10. Again, we can observe that the trending patterns of utility scores, running time and memory consumption are similar to those in the first column of Figure 3, where the values of b_u are generated uniformly.

Scalability. We next test the scalability of the algorithms. Since DeDP is memory-consuming and thus not scalable as our previous results show, we only test the scalability of RatioGreedy, DeDPO, DeDPO+RG, DeGreedy and DeGreedy+RG. The results are presented in the first three columns of Figure 4, where we set $|V|$ to 100, 200 and 500 respectively and vary $|U|$. The mean of c_v is set to 200, and the other parameters are set to default. For utility, we can see that DeDPO-based algorithms still perform the best, while RatioGreedy is the worst. For running time, we can observe that RatioGreedy is not very scalable as its running time increases quickly as the scale of data increases. We can also see that DeGreedy-based algorithms are highly-efficient even when the scale of data is large. Particularly, DeGreedy is much more efficient than the other algorithms when $|U|$ is as large as 100K. In our special test case where $|V| = 500$, $|U| = 200K$ and the mean of c_v is 500, DeGreedy returns a planning with total utility score of 229,234 in around 13 minutes while DeDPO returns one with totally utility score of 230,585 in more than 1.4 hours. It indicates that DeGreedy has better a trade-off between optimality and running time. As for DeDPO-based algorithms, though they are slower than the DeGreedy-based algorithms, their running time increases slowly as the scale of data increases. Finally, for memory consumption, all the algorithms consume only very little memory in addition to the memory taken up by input data. Thus, all algorithms are scalable in space. Overall, DeDPO-based algorithms return the best planning among all the algorithms and are scalable. DeGreedy-based algorithms return worse planning but are more scalable, and are preferable on extremely large datasets, e.g. cities with millions of citizens, given their high efficiency.

Real datasets. We finally study results on real datasets. In the last column of Figure 4, we present the results on the Singapore dataset. We vary values of f_b, and set the other parameters to default. We can observe that the trending patterns are similar to those in the first column of Figure 3, where synthetic data is used. The results on the other two real datasets are similar, and we omit them for brevity.

Summary. We finally summarize our findings.

- DeDPO improves the space and time efficiency of DeDP significantly, and is much more scalable than DeDP.
- DeDP(DeDPO)-based algorithms always achieve the largest total utility score among all the algorithms. DeGreedy-based algorithms are the most time-efficient, though achieve slightly worse total utility score than DeDP(DeDPO)-based algorithms do.
- Both DeDPO-based algorithms and DeGreedy-based algorithms are scalable in space and time.

6. RELATED WORK

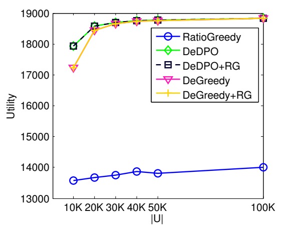
In this section, we review the related work from three categories, mining and managing event-based social networks, location and event recommendation in social networks and bipartite graph matching.

Mining and managing event-based social networks. With the prevalence of various kinds of EBSN platforms, [21] is the first work on formulating and analysing the properties of EBSNs. Recently, some other problems in EBSN are studied. A set of studies, such as [36][10][13], utilize learning-based models to train data of EBSNs and then recommend events to related users. However, they focus on recommendation rather than optimizing a global planning.

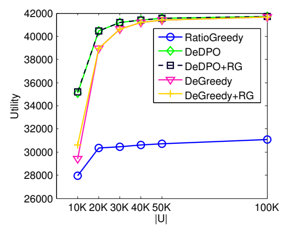
[11] studies the problem of discovering influential event organizers based on the influence maximization problem[14] and team formation problem[17][2], which only considers interests of event organizers. Our work differs from them in that we study the maximum utility event-participant planning problem.

In particular, a closely related research, *Social Event Organization (SEO)* problem [19], has been proposed recently. This problem aims to maximize the overall satisfaction of users towards the arranged events and the social affinity among the users participating the same event. In SEO, each event is associated with a lower bound and an upper bound on the number of attendees, and every pair of users attending the same event contributes to the social affinity score. Due to the pair-wise social affinity score, the SEO problem is NP-hard to approximate and thus only heuristics are proposed. More importantly, the SEO problem substantially differs from ours since this work only considers a simple case of assigning one single event to each participant and thus naturally ignores both the spatio-temporal conflicts of events and also multiple-event planning for each user.

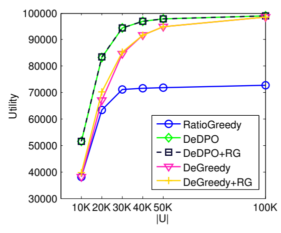
Another related work [26] also studies a global event assignment strategy, but does not consider travel expenditure. [29] studies another objective function, max-min, regarding the event arrangement problem, but still does not consider travel cost between events.



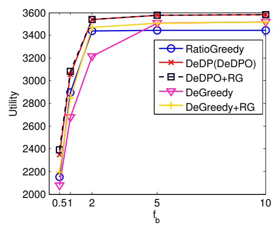
(a) Utility ($|V| = 100$)



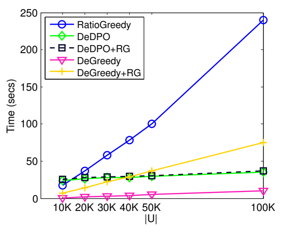
(b) Utility ($|V| = 200$)



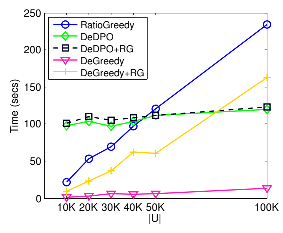
(c) Utility ($|V| = 500$)



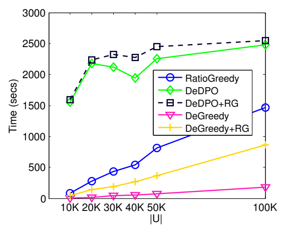
(d) Utility of Singapore



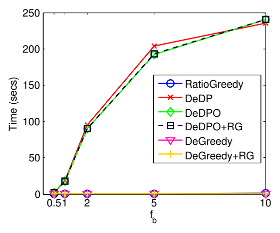
(e) Run time ($|V| = 100$)



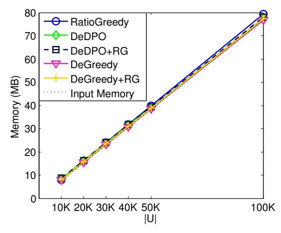
(f) Run time ($|V| = 200$)



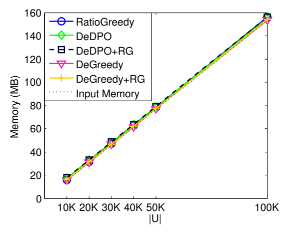
(g) Run time ($|V| = 500$)



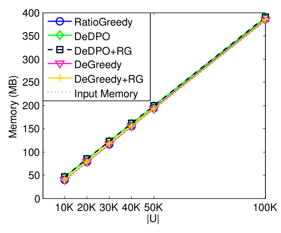
(h) Run time of Singapore



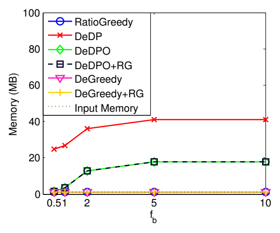
(i) Memory ($|V| = 100$)



(j) Memory ($|V| = 200$)



(k) Memory ($|V| = 500$)



(l) Memory of Singapore

Figure 4: Results on scalability test and real dataset.

Location recommendation in social networks. Due to the recent popularity of location-based social networks (LBSNs), this topic is quite hot [25][18][21][34][20][9][28][35][22][15][15][1]. However, they focus on mining single user's interests in certain venues and make user-oriented recommendation. Also, they do not consider spatio-temporal conflicts, capacity of events or travel budget of users. Our problem is different from theirs since we satisfy the interests of most users with a systematic planning while consider spatio-temporal conflicts and travel cost.

In addition, location-based travel route recommendation[16][6][24][8][4] tries to recommends some landmarks to tourists, where travel expenditure is considered. However, these works focus on making recommendation to one single user and do not consider conflicts, and we differ from them in that we make a global planning for users. Thus, their solutions cannot be applied to our problem.

Bipartite graph matching. The maximum weighted bipartite matching [32][5] is quite related to our problem. However, the bipartite matching problem does not take spatio-temporal conflicts or travel cost between nodes or capacity of nodes into consideration. Recent works [33][31][30][27][23] study the problem of spatial matching, which integrates spatial information and capacities of nodes into the weighted bipartite matching problem. However, they still do not consider spatio-temporal conflicts or travel cost of nodes. Notice the original maximum weighted bipartite matching can be solved in polynomial time. However, our problem is different since our problem is NP-hard due to the conflicts and travel cost between nodes.

7. CONCLUSION

In this paper, we propose a novel social event-participant planning problem called *u̲tility-aware s̲ocial e̲vent-participant p̲lanning* (USEP). We first analyze our differences with other problems in EBSNs and prove the NP hardness of our problem. In order to solve this problem, we first devise a greedy-based heuristic solution, called RatioGreedy, which performs fast in certain circumstances but has no approximation guarantee. To obtain better approximation quality, a two-step approximation framework is presented. Based on this framework, we first propose the DeDP algorithm which has a $\frac{1}{2}$-approximation ratio but consumes more space for large datasets. In order to improve the time/space efficiency of the DeDP algorithm, we further devise the DeDPO algorithm using a series of optimized techniques. Furthermore, to address the challenge of massive data, we also design the DeGreedy algorithm following the two-step approximation framework, which runs faster than the DeDP algorithm but returns a planning with a lower total utility score. We conduct extensive experiments that verify the efficiency, effectiveness and scalability of the proposed approaches.

Acknowledgements: We are grateful to anonymous reviewers for their constructive comments on this work. 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, NSFC Grant No. 61328202/61300031, Microsoft Research Asia Gift Grant, Google Faculty Award 2013 and Microsoft Research Asia Fellowship 2012.

8. REFERENCES

[1] S. Amer-Yahia, S. B. Roy, A. Chawlat, G. Das, and C. Yu. Group recommendation: Semantics and efficiency. In *VLDB'09*.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Online team formation in social networks. In *WWW'12*.

[3] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon even 1935-2004. *ACM Computing Surveys (CSUR)*, 2004.

[4] I. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso. Where shall we go today?: Planning touristic tours with tripbuilder. In *CIKM'13*.

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

[6] C. Chen, D. Zhang, B. Guo, X. Ma, G. Pan, and Z. Wu. Tripplanner: Personalized trip planning leveraging heterogeneous crowdsourced digital footprints. *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 2014.

[7] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 2006.

[8] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic construction of travel itineraries using social breadcrumbs. In *HT'10*.

[9] T. De Pessemier, J. Minnaert, K. Vanhecke, S. Dooms, and L. Martens. Social recommendations for events. In *CEUR Workshop'13*.

[10] 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*.

[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] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. 1979.

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

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

[15] H. Khrouf and R. Troncy. Hybrid event recommendation using linked data and user diversity. In *RecSys'13*.

[16] T. Kurashima, T. Iwata, G. Irie, and K. Fujimura. Travel route recommendation using geotags in photo sharing sites. In *CIKM'10*.

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

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

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

[20] 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*.

[21] 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*.

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

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

[24] E. H.-C. Lu, C.-Y. Chen, and V. S. Tseng. Personalized trip recommendation with multiple constraints by mining user check-in behaviors. In *GIS'12*.

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

[26] J. She, Y. Tong, L. Chen, and C. C. Cao. Conflict-aware event-participant arrangement. In *ICDE'15*.

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

[28] Y.-C. Sun and C. C. Chen. A novel social event recommendation method based on social and collaborative friendships. In *SocInfo'13*.

[29] Y. Tong, R. Meng, and J. She. On bottleneck-aware arrangement for event-based social networks. In *ICDE Workshop SSEPM'15*.

[30] L. H. U, K. Mouratidis, M. L. Yiu, and N. Mamoulis. Optimal matching between spatial datasets under capacity constraints. *ACM Transactions on Database Systems (TODS)*, 2010.

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

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

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

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

[35] Q. Yuan, G. Cong, and C.-Y. Lin. Com: a generative model for group recommendation. In *KDD'14*.

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

APPENDIX

Proof of Theorem 1:

PROOF. In order to complete the proof, we reduce the knapsack problem, which is a well-known NP-complete problem [12], to the USEP problem. The following is an instance of the knapsack problem. Given n items $\{x_1, x_2, ..., x_n\}$, each with value $a_i > 0$ and weight $w_i > 0$, the maximum weight W that a bag can carry, decide if there is a collection of items $C = \{x_{s_1}, x_{s_2}, ..., x_{s_m}\}$ (without loss of generality $s_i < s_j, \forall 1 \le i < j \le m$) such that $\sum_{i=1}^{m} a_{s_i} = K$ and $\sum_{i=1}^{m} w_{s_i} \le W$.

We then construct an instance of the USEP problem from the instance of the knapsack problem accordingly:

(1) Let $|U| = 1$, $U = \{u\}$, and $b_u = W$.

(2) Each item of the knapsack instance corresponds to an event. Let $\mu(v_i, u) = \frac{a_i}{\max a_i}, \forall 1 \le i \le n$, and $c_{v_i} = 1, \forall 1 \le i \le n$.

(3) Assign any value to $\{t_1^{v_i}, t_2^{v_i}\}$ subject to $t_1^{v_i} < t_2^{v_i}, \forall 1 \le i \le n$ and $t_2^{v_i} < t_1^{v_{i+1}}, \forall 1 \le i < n$.

(4) The travel cost between u and an event v_i is constructed as: $cost(u, v_i) = cost(v_i, u) = \frac{w_i}{2}, \forall 1 \le i \le n$.

(5) The travel cost between two events is constructed as:

$$cost(v_i, v_j) = \begin{cases} \frac{w_i + w_j}{2} & 1 \le i < j \le n \\ +\infty & \text{otherwise} \end{cases}$$

We want to decide if there is a feasible schedule S_u for u such that $\sum_{v_i \in S_u} \mu(v_i, u) = \frac{K}{\max a_i}$ and it satisfies all the constraints.

We can see that if the collection C exists, then the schedule $S_u = \{v_{s_1}, v_{s_2}, ..., v_{s_m}\}$ is feasible, and it satisfies that $\sum_{v_i \in S_u} \mu(v_i, u) = \frac{K}{\max a_i}$ and that $cost(u, v_{s_1}) + cost(v_{s_m}, u) + \sum_{i=2}^{m} cost(v_{s_{i-1}}, v_{s_i}) = \sum_{i=1}^{m} w_{s_i} \le W$. On the other hand, if the schedule $S_u = \{v_{s_1}, v_{s_2}, ..., v_{s_m}\}$ exists, then there exists a collection $C = \{x_{s_1}, x_{s_2}, ..., x_{s_m}\}$ satisfying that $\sum_{i=1}^{m} a_{s_i} = K$ and that $\sum_{i=1}^{m} w_{s_i} \le W$. This completes the proof. □

Proof of Lemma 1:

PROOF. If $|S_u| = 1$ and $S_u = \{v_i\}$, it is clear that S_u violates the budget constraint of u. If $|S_u| > 1$ and $v_i \in S_u$, due to the triangle inequality property of *cost*, it is clear that $cost(u, v_1^u) + \sum_{j=2}^{|S_u|} cost(v_{j-1}^u, v_j^u) + cost(v_{|S_u|}^u, u) \ge cost(u, v_i) + cost(v_i, u) > b_u$. Thus, S_u is infeasible. □

Proof of Theorem 3:

PROOF. Let $A_r = \cup_{j \ge r}\{S_{u_j}\} \subseteq A$ be the solution w.r.t. μ^r, i.e. $\Omega(A_r) = \sum_{j \ge r} \sum_{v_{i,k} \in S_{u_j}} \mu^r(v_{i,k}, u_j)$. Notice that according to the way we update μ^r, $\mu^{r+1}(v_{i,k}, u_r) = 0, \forall 1 \le i \le |V|, 1 \le k \le c_{v_i}$. Since $\mu^{r+1}(v_{i,k}, u_j) = \mu^r(v_{i,k}, u_j)$, $\forall 1 \le i \le |V|, 1 \le k \le c_{v_i}, j < r$, it is easy to see that $\mu^{r+1}(v_{i,k}, u_j) = 0, \forall 1 \le i \le |V|, 1 \le k \le c_{v_i}, j \le r$. Thus, when calculating the total utility score of a solution w.r.t. μ^r, we do not need to sum over users in $\{u_j | j < r\}$.

We then prove by induction and start from the last user, i.e. $r = |U|$. When $r = |U|$, since the dynamic programming algorithm returns an optimal schedule for $u_{|U|}$, $A_{|U|} = \{S_{u_{|U|}}\}$ is an optimal solution w.r.t. $\mu^{|U|}(\hat{v}., u_{|U|})$. Since $\mu^{|U|}(\hat{v}_i, u_{|U|}) \ge \mu^{|U|}(v_{i,k}, u_{|U|}), \forall 1 \le i \le |V|, 1 \le k \le c_{v_i}$, $\Omega(A_{|U|}) = \Omega(A^\star_{|U|}) \ge \frac{1}{2}\Omega(A^\star_{|U|})$, where $A^\star_{|U|}$ is the optimal solution w.r.t. $\mu^{|U|}$.

Suppose A_{r+1} is a $\frac{1}{2}$-approximate solution w.r.t. μ^{r+1}. We next prove that A_r is also a $\frac{1}{2}$-approximate solution w.r.t. μ^r. Recall that $\mu^{r+1}(v_{i,k}, u_r) = 0, \forall 1 \le i \le |V|, 1 \le k \le c_{v_i}$. Thus, the schedule of u_r does not affect the total utility score of the solution w.r.t. μ^{r+1}. Since $A_{r+1} \subseteq A_r$ and A_{r+1} is a $\frac{1}{2}$-approximate solution w.r.t. μ^{r+1}, clearly A_r is also a $\frac{1}{2}$-approximate solution w.r.t. μ^{r+1}.

We then define μ_o^r as follows.

$$\mu_o^r(v_{i,k}, u_j) = \begin{cases} \mu^r(v_{i,k}, u_r) & (j = r) \text{ or } (j > r \text{ and } v_{i,k} \in \hat{S}_{u_r}) \\ 0 & \textit{otherwise} \end{cases} \tag{6}$$

Consider an arbitrary feasible planning $A' = \cup_j\{S'_{u_j}\}$ w.r.t. μ_o^r, and we have

$$\begin{aligned} \Omega_o(A') &= \sum_j \sum_{v_{i,k} \in S'_{u_j}} \mu_o^r(v_{i,k}, u_j) \\ &= \sum_{j \ge r} \sum_{v_{i,k} \in S'_{u_j}} \mu_o^r(v_{i,k}, u_j) \\ &= \sum_{v_{i,k} \in S'_{u_r}} \mu^r(v_{i,k}, u_r) \\ &\quad + \sum_{j > r} \sum_{v_{i,k} \in \hat{S}_{u_r} \cap S'_{u_j}} \mu^r(v_{i,k}, u_r) \end{aligned} \tag{7}$$

Notice that in the planing A', each $v_{i,k}$ can be included in at most one of $S'_{u_j}, \forall j > r$. Therefore, we have

$$\sum_{j > r} \sum_{v_{i,k} \in \hat{S}_{u_r} \cap S'_{u_j}} \mu^r(v_{i,k}, u_r) \le \sum_{v_{i,k} \in \hat{S}_{u_r}} \mu^r(v_{i,k}, u_r) \tag{8}$$

Note that the schedule $\hat{S}_{u_r}$ returned by the dynamic programming algorithm for u_r is an optimal solution w.r.t. $\mu^r(\hat{v}., u_r)$ and that $\mu^r(\hat{v}_i, u_r) \ge \mu^r(v_{i,k}, u_r), \forall 1 \le i \le |V|, 1 \le k \le c_{v_i}$. Therefore, we have

$$\sum_{v_{i,k} \in S'_{u_r}} \mu^r(v_{i,k}, u_r) \le \sum_{v_{i,k} \in \hat{S}_{u_r}} \mu^r(v_{i,k}, u_r) \tag{9}$$

Combining Equations (8) and (9), we have

$$\Omega_o(A') \le 2 \sum_{v_{i,k} \in \hat{S}_{u_r}} \mu^r(v_{i,k}, u_r) = 2\Omega_o(\hat{S}_{u_r}) \tag{10}$$

Notice that regarding A_r, for each $v_{i,k} \in \hat{S}_{u_r}$, it is either in S_{u_r} or in $\cup_{j > r} S_{u_j}$, and appears only once in A_r. For the other $v_{i,k}$'s that are not in $\hat{S}_{u_r}$, they cannot be in S_{u_r} and do not affect $\Omega_o(A_r)$ according Equation (6). Therefore, we have $\Omega_o(A_r) = \Omega_o(\hat{S}_{u_r}) \ge \frac{1}{2}\Omega_o(A')$. Thus, A_r is a $\frac{1}{2}$-approximate solution w.r.t. μ_o^r. Notice that $\mu^r(\cdot, \cdot) = \mu^{r+1}(\cdot, \cdot) + \mu_o^r(\cdot, \cdot)$. Based on Theorem 2, A_r is thus a $\frac{1}{2}$-approximate solution w.r.t. μ^r.

By induction, $A = \cup_j\{S_{u_j}\}$ is a $\frac{1}{2}$-approximate solution w.r.t. $\mu^1 = \mu$, which completes the proof. □

Proof of Lemma 2:

PROOF. Clearly, for $1 \le l \le r_1, l \le r$, it holds that $\mu^l(v_{i,k}, u_r) = \mu(v_i, u_r)$ since $v_{i,k}$ is not included in any of

$\hat{S}_{u_j}, 1 \leq j < r_1$ and thus $\mu^l(v_{i,k}, u_r)$ does not change in the first $r_1 - 1$ iterations according to the way we update μ^r.

We prove the remaining of the claim by induction. For $r_1 < l \leq r_2, l \leq r$, we have

$$\begin{aligned}\mu^l(v_{i,k}, u_r) &= \mu^{l-1}(v_{i,k}, u_r) \\ &= \cdots \\ &= \mu^{r_1+1}(v_{i,k}, u_r) \\ &= \mu^{r_1}(v_{i,k}, u_r) - \mu^{r_1}(v_{i,k}, u_{r_1}) \\ &= \mu(v_i, u_r) - \mu(v_i, u_{r_1}) \end{aligned} \tag{11}$$

Suppose $\mu^l(v_{i,k}, u_r) = \mu(v_i, u_r) - \mu(v_i, u_{r_{j_l}})$ holds for $j_l = j - 1$, i.e. $r_{j-1} < l \leq r_j, l \leq r$, we next prove that the equation also holds for $r_j < l \leq r_{j+1}, l \leq r$. For $r_j < l \leq r_{j+1}, l \leq r$, we have

$$\begin{aligned}\mu^l(v_{i,k}, u_r) &= \mu^{l-1}(v_{i,k}, u_r) \\ &= \cdots \\ &= \mu^{r_j+1}(v_{i,k}, u_r) \\ &= \mu^{r_j}(v_{i,k}, u_r) - \mu^{r_j}(v_{i,k}, u_{r_j}) \\ &= (\mu(v_i, u_r) - \mu(v_i, u_{r_{j-1}})) \\ &- (\mu(v_i, u_{r_j}) - \mu(v_i, u_{r_{j-1}})) \\ &= \mu(v_i, u_r) - \mu(v_i, u_{r_j}) \end{aligned} \tag{12}$$

which completes the proof. □

Proof of Lemma 3:

Proof. We prove by induction. Before the 1st iteration, it is clear that we have pushed a valid event with the largest ratio value into H during in the initialization step. Suppose before the i-th iteration, the valid event with the largest ratio value must be in H. We next prove that after updating H in the i-th iteration, all valid event candidates with the possibly largest ratio value must be in H.

Denote $\hat{v}_i$ as the event popped from H in the i-th iteration. First, if $\hat{v}_i$ is the only event in $\hat{S}_{u_r}$, the way we update H ensures that we scan through the remaining $V'_r \setminus \{\hat{v}_i\}$ events to push the next valid event candidates with the largest ratio values into H. Second, if $\hat{v}_i$ is the first but not the only event in $\hat{S}_{u_r}$, recall that $\hat{v}_{s_i}$ is the successor of $\hat{v}_i$ in $\hat{S}_{u_r}$. Notice that only the *inc_cost* values of the events in $\{\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_{i-1}\}$ and $\{\hat{v}_{i+1}, \hat{v}_{i+2}, \cdots, \hat{v}_{s_i-1}\}$ will be changed when $\hat{v}_i$ is added into $\hat{S}_{u_r}$, and these events are not yet in H. Also, the way we update H ensures that the valid event candidates in $\{\hat{v}_1, \cdots, \hat{v}_{i-1}\}$ and $\{\hat{v}_{i+1}, \cdots, \hat{v}_{s_i-1}\}$ with the largest ratio values will be scanned. For the other events whose *inc_cost* values do not change, they have been scanned through in previous iterations. Therefore by induction, the next valid event candidate with the largest ratio value must be in H. Similarly, if $\hat{v}_i$ is the last but not the only event in $\hat{S}_{u_r}$, the way we update H ensures that we scan through all events whose *inc_cost* are changed and the next valid event with the largest ratio value must be in H. Finally, if $\hat{v}_i$ has a precedent $\hat{v}_{p_i}$ and a successor $\hat{v}_{s_i}$ in $\hat{S}_{u_r}$, we also scan through all events whose *inc_cost* are changed. Therefore, in all cases, the next valid event with the largest ratio value must be in H after we update H in the i-th iteration if it exists, which completes the proof. □