2017 IEEE 33rd International Conference on Data Engineering

Complex Event-Participant Planning and Its Incremental Variant

Yurong Cheng #1, Ye Yuan #1, Lei Chen *2, Christophe Giraud-Carrier ‡3, Guoren Wang #1
#College of Computer Science and Engineering, Northeastern University, China
*Department of Computer Science and Engineering, Hong Kong University of Science and Technology
‡Department of Computer Science, Brigham Young University, USA
1{yrcheng@stumail,yuanye@mail,wanggr@mail}.neu.edu.cn 2leichen@cse.ust.hk 3cgc@cs.byu.edu

***Abstract*—In recent years, online *Event Based Social Network (EBSN)* platforms have become increasingly popular. One typical task of EBSN platforms is to help users make suitable and personalized plans for participating in different interesting social events. Existing techniques either ignore the minimum-participant requirement constraint for each event, which is crucially needed for some events to be held successfully, or assume that events would not change once announced. In this paper, we address the above inadequacies of existing EBSN techniques. We formally define the Global Event Planning with Constraints (GEPC) problem, and its incremental variant. We prove that both are NP-hard, and provide approximate solutions. Finally, we verify the effectiveness and efficiency of our proposed algorithms through extensive experiments over real and synthetic datasets.**

I. Introduction

Event Based Social Network (EBSN) platforms, such as Meetup[1] and Plancast[2], are attracting significant attention from both industry and academia [1]. Also known as *Online to Offline* services, these platforms assist users online with creating, managing, joining, and making suitable and personalized plans for a variety of offline social events of interest. Meetup, for example, counts more than 16 million users, involved in an aggregate 300,000 events held each month.

In practice, EBSN users are generally asked to select labels or categories of events of interest (e.g., sports, music, travelling) at registration time. Based on these preferences and historical records of event participation, a *utility score* capturing each user's interest in each event can be derived [2], [3], [4]. The higher a user's utility score for an event, the higher that user's interest in the corresponding event. In addition to these event-based utility scores, each user has an associated *travel budget* that determines how much can be spent by the user to travel from a place origin to a set of planned events. When designing plans, it is further assumed that a user may participate in multiple non-conflicting events, and that each event has an upper bound on the number of participants it can accommodate. One of the goals of an EBSN is then to create individual event plans for all users that maximize the total utility score of the users to their arranged events. Formally, current EBSN systems solve the following planning problem.

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

Global Event Planning (GEP) [4]: Given sets of users and events, together with utility scores, travel budgets and participation upper bounds, find a plan that assigns users to events such that global utility is maximized.

In practice, however, this formulation of the GEP suffers from at least two significant limitations.

1) The GEP does not account for participation lower bounds on events, that is, it implicitly assumes that all events will effectively take place regardless of the number of users assigned to them.
2) The GEP does not account for possible changes to events and/or by users, that is, it implicitly assumes that once given, all information remains static.

There are a number of situations where the first assumption clearly does not hold. Consider the following examples.

- *Beijing Summer Palace Visit at Discounted Price*. The Summer Palace Office has agreed to offer a 50% discount on all tickets for groups of at least 20 tourists. If the group is smaller, the discount will not be applied.
- *Football (Soccer) Game*. While fewer players may pretend to a friendly game, the rules of the game, that the event organizers may wish to enforce, stipulate that at least 22 players should participate, 11 on each side.
- *Seminar on Healthy Living*. In order for the event organizers to cover their costs (e.g., honorariums for invited speakers, rental fee for the venue), a minimum of participants must register. If the revenue from registrants is less than the anticipated costs, the event may be cancelled.

In all of these cases, and many others, the event cannot be held unless enough participants are assigned to it. Assuming otherwise may result in suboptimal solutions and user dissatisfaction. Thus, it is not only reasonable, but indeed critical, to enable the specification of minimum-participant requirements, or participation lower bounds, on events, and to enforce their satisfaction. This leads to the following extension of the Global Event Planning problem.

Global Event Planning with Constraints (GEPC): Given sets of users and events, together with utility

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.135

scores, travel budgets, participation upper and lower bounds, find a plan that assigns users to events such that global utility is maximized, subject to the constraint that the number of participants assigned to each event exceeds that event's participation lower bound.

Similarly, it is difficult in practice to expect the second assumption to hold. There are too many variables at play to expect things to remain unchanged over time. Consider the following simple examples.

- *Unexpected Work Assignment*. Jessica is looking forward to attending her favorite band's outdoor concert in the local park next Thursday. She receives a phone call from her boss announcing that she must run a 2-day site inspection of one of her company's plants the following Thursday and Friday. Jessica will have to forego the concert.
- *Change of Venue*. Alan is organizing a training seminar. He had planned on a specific venue capable of accommodating 200 participants and advertised accordingly. A week before the seminar, he finds out that the venue has already been booked. He must settle for a smaller venue, and decrease the event's participation upper bound.

In such cases, changes must be made to an existing plan. Recomputing a plan from scratch with the new information is computationally prohibitive. What is needed is an incremental mechanism, where the existing plan can be adapted efficiently. We formulate the problem as follows.

Incremental Event Planning (IEP): Given a solution to the GEPC problem, together with changes to a user's utility scores, a user's travel budget, an event's times, an event's location, or an event's participation upper or lower bound, find a new solution to the GEPC problem.

To the best of our knowledge, this paper presents the first attempt at solving the GEPC, and associated IEP, problems. All previous work has inherent limitations, thus only addressing restricted forms of the GEP problem. For example, it does not account for event participation lower bounds [4]; or it assumes that users can only attend one event and there are no conflicts among events [3]; or it does not consider users' travel budgets [2]. While some of these differences may appear rather small, the resulting algorithms and approximation ratio are in fact quite different. We begin by showing that the GEPC problem is NP-hard, subsequently proposing a two-step framework, that not only satisfies the minimum-participant requirement constraint for each event but also provides approximate guarantees. We similarly show that the IEP problem is also NP-hard, and devise a series of approximate solutions with theoretical guarantees. Finally, we present the results of an extensive empirical study that verifies the effectiveness and efficiency of the proposed methods on real datasets.

II. Problem Statement

In this section, we present a formal mathematical account of the GEPC problem, and its incremental variant, the IEP problem. We assume that the EBSN contains a set $U = \{u_i\}$ of n users, and a set $E = \{e_j\}$ of m events.

Each user $u_i \in U$ is described by a pair $(\mathbf{l}_{u_i}, B_i)$ consisting of a location and a travel budget. Each event $e_j \in E$ is denoted by a 5-tuple $(\mathbf{l}_{e_j}, \xi_j, \eta_j, t_j^s, t_j^t)$ consisting of a location, participation lower bound, participation upper bound, start time, and end time. For each pair, (u_i, e_j), of user and event, there is a corresponding utility score, $\mu(u_i, e_j) \geq 0$, that captures u_i's interest in e_j. A score of 0 signifies that a user will not or cannot participate in the corresponding event.

Example 1: The following is a simple example of an EBSN platform with five users and four events. The locations of users and events are shown graphically on a 2-D grid in Fig. 1.

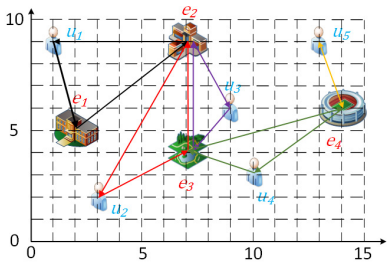


Fig. 1. Locations of Users and Events

Table I shows the other information associated with each user and event. Users and their travel budgets are shown on row 1.

TABLE I
Events and Users Information

$e_j(\xi_j, \eta_j)$	$u_1(18)$	$u_2(20)$	$u_3(20)$	$u_4(30)$	$u_5(10)$	Time
$e_1(1,3)$	**0.7**	0.6	0.4	0.2	0.3	1:00-3:00 p.m.
$e_2(2,4)$	**0.6**	**0.5**	**0.7**	0.3	0.1	4:00-6:00 p.m.
$e_3(3,4)$	0.9	**0.8**	**0.9**	**0.8**	0.6	1:30-3:00 p.m.
$e_4(1,5)$	0.3	0.4	0.5	**0.6**	**0.7**	6:00-8:00 p.m.

Events together with their respective participation lower and upper bounds are shown in column 1, with their start and end times shown in column 7. Finally, the utility scores that users have assigned to events are in columns 2-6. □

When creating plans, the EBSN must operate within the confines of a predefined time horizon, $\mathcal{H}$. For simplicity, and without loss of generality, we assume a time horizon of $\mathcal{H} = 1$ day, or daily planning, so that every day users are provided with their individualized "Plan for Today."

A global plan, P, is a set of individual plans that assign events to each user within $\mathcal{H}$, i.e., $P = \{P_i : P_i \subseteq E, 1 \leq i \leq n\}$. User plans are designed to be free of time conflicts. That is, if an event e_k starts before an event e_h in some plan P_i, e_k should also end before e_h starts. In Example 1, events e_1 and e_3 have a time conflict since e_3 starts before e_1 ends. Similarly, events e_2 and e_4 also have a conflict since e_4 starts when e_2 ends leaving no time to go from e_2 to e_4

Assuming that more than one event may be scheduled within $\mathcal{H}$, a user's travel cost, D_i, is the sum of the costs of

traveling from event to event within his/her plan. While such costs may consist of one, or a combination, of distance (e.g., Euclidean, Manhattan), cost of attendance (e.g., admission fee), and other considerations, here we simply use Euclidean distance. Similarly, a user's utility, μ_i, is the sum of its utility scores over the events in his/her plan. In Example 1, if u_1's plan were made up of e_1 and e_2, its travel cost would be $D_1 = d(u_1, e_1) + d(e_1, e_2) + d(e_2, u_1) = \sqrt{17} + \sqrt{41} + 6 = 16.53$, and its utility would be $\mu_1 = \mu(u_1, e_1) + \mu(u_1, e_2) = 0.7 + 0.6 = 1.3$.

A. Complex Event Planning: GEPC Problem

The EBSN's global utility score for a plan P, denoted $\mathcal{U}_P$, is the sum of the users' utility scores in P.

Definition 1 (GEPC problem): Given an EBSN, the GEPC problem is to find a feasible global plan $P^\star$, such that $\mathcal{U}_{P^\star} = \max_P \mathcal{U}_P$, subject to the following constraints:

1) Users' plans have no time conflicts, i.e., $\forall i\ \forall e_k \neq e_h \in P_i \quad t^s_{e_k} < t^s_{e_h} \Rightarrow t^t_{e_k} < t^s_{e_h}$.
2) Users' travel costs are within budget, i.e., $\forall i\ D_i \leq B_i$.
3) Events' participation upper bounds are satisfied, i.e., $\forall j\ |\{P_i : e_j \in P_i\}| \leq \eta_j$
4) Events' participation lower bounds are satisfied, i.e., $\forall j\ |\{P_i : e_j \in P_i\}| \geq \xi_j$ □

Example 2: The cells with colored entries in Table I correspond to a global plan. It is easy to verify that all constraints are satisfied. There are no time conflicts among assigned events. All travel costs are within budget (e.g., $D_1 = 16.53 \leq 18 = B_1$, $D_2 = 17.53 \leq 20 = B_2$). All events's participation upper bounds are met (e.g., e_2 is in 3 individual plans and $3 \leq 4 = \eta_2$). Finally, all events's participation lower bounds are also met (e.g., e_3 is in 3 individual plans and $3 \geq 3 = \xi_3$). The EBSN's global utility score under the given plan is $\mu(u_1, e_1) + \mu(u_1, e_2) + \cdots + \mu(u_5, e_4) = 6.3$. □

Theorem 1: The GEPC problem is NP-hard.

Proof 1: The proof is straightforward. The version of the GEP problem presented in [4] was shown to be NP-hard. Letting $\xi_j = 0$ for all $(1 \leq j \leq m)$ (i.e., removing event participation lower bounds) reduces that version to the GEPC problem. Hence, the GEPC problem is NP-hard. □

B. Incremental Variant: IEP Problem

In practice, event information and user preferences are subject to change. Thus, a reasonable event planning system should support incremental updates. In other words, even though an optimal plan arranges users to suitable events according to the users' requirement when the events are posted, the planning may have to be altered before some events start. While changes to the global plan are accommodating to users who change their individual requirements, they often have a negative impact on users who require no such changes but whose plans must nevertheless be modified. As a result, it is important, when incrementally finding a new maximum for the utility score, to also minimize the negative impact on users.

We begin with a description of which system parameters are affected by changes made by users and to events, respectively.

1) Changes Caused by Users' Actions:

- **Utility scores**. As users' interests and availability change, utility scores are affected, either explicitly or implicitly. In Example 1, if u_1's availability changes, say, from the whole day to 2:00p.m.-8:00 p.m., then, u_1 can no longer attend e_1, and $\mu(u_1, e_1)$ would become 0.
- **Travel budgets.** As users' circumstances change, travel budgets may be adapted. For example, if the weather turns bad, making road conditions hazardous, a user may decide not to travel at all, or to travel a shorter distance than what had previously been planned.

2) Changes Caused by Events' Actions:

- **New events.** Given the nature of EBSNs, it is inevitable that new events will be added at any time.
- **Participation lower and upper bounds.** As event organizers work through the logistics and constraints of their events, participation lower bounds may change. In the case of the Beijing Summer Palace Visit, for example, if the tourist season is proving less busy than expected, the Palace Office may choose to increase the minimum number of participants required for the discounted price to apply. Conversely, an event organizer may have to decrease the maximum number of participants if the venue is smaller than anticipated.
- **Start times, end times, and locations.** As an event organizer is notified that the planned place for the event is not available during the planned period of time, changes may have to be made to start and end times, or an alternate location may have to be found.

We refer to the above changes as *atomic* operations. Our incremental version is designed for single atomic operations. It is not entirely clear how multiple atomic changes could be handled in a single run. This consideration is left to future work, and the case where multiple atomic operations take place is treated here as running the incremental version multiple times. As per the above discussion, when making changes to global plans, we must minimize the negative impact on users. When a plan P is transformed into a new plan P' following some atomic operation, the negative impact, denoted as $dif(P, P')$, is defined as the sum of the number of events that each user can no longer attend in P', i.e., $dif(P, P') = \sum_{i=1}^{n} |P_i \setminus P'_i|$.

Definition 2 (IEP problem): Given an EBSN, an original planning P, and an atomic operation on P, the IEP problem is to find a new planning $P'^\star$, such that $\mathcal{U}_{P'^\star} = \max_{P'} \mathcal{U}_{P'}$, subject to $dif(P, P'^\star) = \min_{P'} dif(P, P')$. □

Example 3: Consider the plan in Table I, and assume that η_4 is decreased from 5 to 1. The solution of the IEP problem is the one where e_4 is removed from u_4's plan (rather than from u_5's plan, since $\mu(u_5, e_4) > \mu(u_4, e_4)$), to satisfy e_4's updated participation upper bound, but where e_2 is then added to u_4's plan, to maximize utility, while maintaining all other constraints. Hence, $P_4 - P'_4 = \{e_3, e_4\} - \{e_2, e_3\} = \{e_4\}$, and there is one event that u_4 can no longer attend. None of the plans for the other users are altered. The negative impact is

thus: $|P_1 \setminus P_1'| + |P_2 \setminus P_2'| + |P_3 \setminus P_3'| + |P_4 \setminus P_4'| + |P_5 \setminus P_5'| = 0 + 0 + 0 + 1 + 0 = 1$. □

III. Solutions to the GEPC Problem

We take a two-step approach to solve the GEPC problem, as follows.

1) We solve a restricted version of the GEPC problem, denoted ξ-GEPC, where the values of all events' participation upper bounds are temporarily set to the values of those events's participation lower bounds (i.e., $\forall j\ \eta_j = \xi_j$). In other words, the global plan found by ξ-GEPC assigns exactly ξ_j users to each event, thus meeting the constraint of the GEPC problem on the participation lower bounds, but not assigning any more users than are strictly necessary to each event.
2) We then check whether users can possibly participate in more events than those assigned by the ξ-GEPC plan. That is, we now update the ξ-GEPC plan by solving for event participation upper bounds set to $\eta_j - \xi_j$.

Since the second step can be solved using existing methods with provable approximation ratio (e.g., see [4]), the challenge is to provide an adequate solution for the first step.

Theorem 2: The ξ-GEPC problem is NP-hard.

Proof 2: The proof consists of reducing the Generalized Assignment Problem (GAP), a well-known NP-hard problem, to the ξ-GEPC problem. An instance of the GAP is defined as follows. Let J be a set of m jobs and M be a set of n parallel machines. Each job is to be processed by exactly one machine. Processing job j on machine i requires time $p_{i,j}$, and incurs cost $c_{i,j}$. Each machine can work for no longer than T_i time units. The GAP is to find a schedule $S = \cup_{i=1}^{n} S_i$ of the jobs, such that the total cost $C = \sum_{i\in[1,n], j\in S_i} c_{i,j}$ is minimum, subject to $\sum_{j\in S_i} p_{i,j} \le T_i$. We now construct an instance of ξ-GEPC from an instance of GAP. Let $E = J$, $U = M$, and $\xi_j = 1$ for all events. Let the start and end times of each event be any values that guarantee no conflicts. Let $d(u_i, e_j) = p_{i,j}/2$, and let the distance between any pair of events e_j and e_j' be any value satisfying $d(e_j, e_j') < \max_{i=1}^{n}(p_{i,j} + p_{i,j'})$. It can be shown that $D_i \le \sum_{j\in S_i} p_{i,j} \le (2+\epsilon)D_i$, so let $B_i = \frac{1}{2+\epsilon}T_i$. Finally, let $\mu(u_i, e_j) = 1 - c_{i,j}$. Our objective is to find $P = \cup_{i=1}^{n} P_i$ such that $\sum_{i\in[1,n], e_j\in P_i} \mu(u_i, e_j) = m - C$. If S_i exists, we have $\sum_{j\in S_i} p_{i,j} \le T_i$. Because $D_i \le \sum_{j\in S_i} p_{i,j} \le (2+\epsilon)D_i$, we have $D_i \le \sum_{j\in S_i} p_{i,j} \le T_i = (2+\epsilon)B_i$, i.e., $D_i \le B_i$ holds. Meanwhile, we know that $\sum_{i\in[1,n], e_j\in P_i} \mu(u_i, e_j) = m - C$. On the other hand, if P_i exists, we have $D_i \le B_i = \frac{1}{2+\epsilon}T_i$. Because $D_i \le \sum_{j\in S_i} p_{i,j} \le (2+\epsilon)D_i$, we have $\frac{1}{2+\epsilon}\sum_{j\in S_i} p_{i,j} \le D_i \le B_i = \frac{1}{2+\epsilon}T_i$, i.e., $\sum_{j\in S_i} p_{i,j} \le T_i$ holds. Meanwhile, we know that $\sum_{i\in[1,n], j\in S_i} c_{i,j} = C$. □

We go on to provide two approximate algorithms with bounded approximation ratio to solve the ξ-GEPC problem.

A. GAP-based Approximation Algorithm

Because the ξ-GEPC problem can be reduced to the GAP when time constraints are ignored, our first algorithm finds a candidate solution to the GAP, and subsequently adjusts it to remove time conflicts.

In the GAP, each event is assigned to exactly one user. Thus, we first transform the ξ-GEPC problem by creating ξ_j copies, $\{e_j^1, \ldots, e_j^{\xi_j}\}$, of each event e_j, with the same location, start time, end time, and utility to all users. Now, we have $m^+ = \sum_{j=1}^{m} \xi_j$ events, with the copies of the same event having conflicts. Then, the ξ-GEPC problem consists of assigning each of the m^+ events to exactly one user, considering both the original conflicts among different events and the conflicts among copies of the same event, with the other constrains remaining unchanged.

We can then construct an instance of GAP from an instance of ξ-GEPC ignoring time conflicts, such that (1) $J = E$ with size $m^+ = \sum_{k=1}^{m} \xi_k$, and $M = U$ with size n; (2) $p_{i,j} = 2d(u_i, e_j)$, and $T_i = (2+\epsilon)B_i$; and (3) $c_{i,j} = 1 - \mu(u_i, e_j)$. If a plan, P, exists in ξ-GEPC, the maximum total utility is $\sum_{i=1}^{n}\sum_{j=1}^{m^+} \mu(u_i, e_j) = \Psi$. Then, the minimum total cost is $C = m^+ - \Psi$. It is easy to see that the instance of ξ-GEPC ignoring time conflicts is YES if and only if the instance of GAP is YES, and can be solved using linear programming with the relaxation method of [5].

Now, let us consider how to adjust the events that have conflicts. The approach is summarized in Algorithm 1.

Algorithm 1: Conflict Adjusting Algorithm

```
   Input: E, U, {μ(u_i, e_j)}, P
   Output: P'
1  for each user u_i do
2      Find all the conflict events in P_i
3      while P_i has conflict events do
4          Find the conflicting event e whose utility is the smallest
5          P_i := P_i − {e}
6          U_e := U − {u_i}
7          while U_e ≠ ∅ and e is not assigned do
8              Find u_k ∈ U_e whose utility to e is the largest
9              if e is not in conflict with events in P_k and D_k ≤ B_k after
               adding e to P_k then
10                 P_k := P_k ∪ {e}
11                 break
12             else
13                 U_e := U_e − {u_k}
14 P' := {P_i}
15 return P'
```

Given the plan P obtained from the linear programming algorithm, for each user u_i, we find all the conflicting events in u_i's plan P_i (Line 2). We then find the event e from u_i's conflicting events whose utility is the smallest and delete it from P_i (Lines 4-5). For all users except u_i, we find the user u_k whose utility to e is the largest. If in u_k's plan, there are no events that have conflicts with e, and u_k's travel cost is still within budget after adding e to P_k, then we add e to P_k. Otherwise, we continue to find another user whose utility to e is the second largest, and so on, until event e is assigned (Lines 7-12). We go on to find another event from P_i whose utility is the second smallest, and apply the above procedure until no conflicting events can be found in P_i. We repeat the process until the plans of all users are checked, and return the

updated global plan.

Example 4: Consider the problem in Example 1. Assume that the preliminary plan obtained from the linear programming algorithm is shown in red in Table II.

TABLE II
PLAN OBTAINED FROM GAP-BASED ALGORITHM

$e_j(\xi_j, \eta_j)$	$u_1(18)$	$u_2(20)$	$u_3(20)$	$u_4(30)$	$u_5(10)$	Time
$e_1(1,3)$	**0.7**	0.6	0.4	**0.2**	0.3	1:00-3:00 p.m.
$e_2(2,4)$	0.6	**0.5**	**0.7**	0.3	0.1	4:00-6:00 p.m.
$e_3(3,4)$	**0.9**	**0.8**	**0.9**	0.8	0.6	1:30-3:00 p.m.
$e_4(1,5)$	0.3	0.4	0.5	0.6	**0.7**	6:00-8:00 p.m.

We find that in u_1's planning P_1, e_1 and e_3 have conflicts. As the utility of e_1 is smaller than that of e_3, we delete e_1 from P_1. For the other users, u_2's utility to e_1 is the largest. But in u_2's plan, e_3 has conflict with e_1. So e_1 cannot be added to u_2. Neither can e_1 be added to u_3 for the same reason. Then, u_5 has the third largest utility to e_1, and no conflicting events are in u_5's plan, but the travel budget of u_5 cannot afford him/her to participate in e_1. Finally we find that u_4 is available to attend e_1, and add e_1 to u_4's plan (shown in blue in Table II). □

1) Approximation Ratio: Let $m^+ = \sum_{k=1}^{m} \xi_k$. The utility provided by plan P_i is then $\Psi = \sum_{i=1}^{n} \sum_{j=1}^{m^+} \mu(u_i, e_j)$. It was shown in [6] that for any given cost C and time load T_i in GAP, the linear programming algorithm with the relaxation method of [5] can find an assignment of cost at most $(1+\epsilon)C$ and time load at most $(2+\epsilon)T_i$. It follows that $m^+ - \Psi \leq (1+\epsilon)(m^+ - \Psi)_{OPT} = (1+\epsilon)(m^+ - \Psi_{OPT})$. Thus, $\Psi \geq (1+\epsilon)\Psi_{OPT} - m^+\epsilon$, or $\frac{\Psi}{\Psi_{OPT}} \geq 1+\epsilon-\frac{m^+\epsilon}{\Psi_{OPT}} = 1+(1-\frac{m^+}{\Psi_{OPT}})\epsilon$. Because each $\mu(u_i, e_j) \in [0,1]$, Ψ_{OPT} is the sum of m^+ numbers whose values are between 0 and 1. Thus, $m^+ > \Psi_{OPT}$, i.e., $(1-\frac{m^+}{\Psi_{OPT}}) < 0$. Hence, linear programming provides a $(1-O(\epsilon))$-approximation ($\epsilon > 0$) to the ξ-GEPC problem with no time conflicts.

When checking the conflicts of user u_i in the Conflict Adjusting algorithm, we iteratively delete the conflicting event whose utility is the smallest. Thus, the remaining events have larger utilities. Let P_i^{OPT} be u_i's optimal plan, and P_i be u_i's plan after running the Conflict Adjusting algorithm. It is clear that the largest number of events u_i can attend is bounded by Uc_i, the number of events that fall within a distance $B_i/2$ of $\mathbf{l}_{u_i}$, i.e., $|P_i^{OPT}| \leq Uc_i$. In the worst case, the event $e^{\max}$ whose utility is the highest has conflicts with all of the other events, while the other events have no conflicts among themselves. The linear programming algorithm originally assigned Uc_i events to u_i. But when adjusting the conflicts, all events except $e^{\max}$ are removed from P_i. Thus, P_i only contains $e^{\max}$, while P_i^{OPT} contains all the other $Uc_i - 1$ events. Thus, $\frac{\sum_{e_j \in P_i} \mu(u_i,e_j)}{\sum_{e_j \in P_i^{OPT}} \mu(u_i,e_j)} \geq \frac{1}{Uc_i - 1}$, and the global utility satisfies $\sum_{i,j} \mu(u_i, e_j) = \sum_{i=1}^{n} (\sum_{e_j \in P_i} \mu(u_i, e_j)) \geq \sum_{i=1}^{n} (\frac{1}{Uc_i - 1} OPT_i) \geq \frac{1}{Uc_{\max} - 1} \sum_i OPT_i = \frac{1}{Uc_{\max} - 1} OPT$, where $Uc_{\max} = \max_{i=1}^{n} Uc_i$. It follows that the approximation ratio of the Conflict Adjusting algorithm is $\frac{1}{Uc_{\max} - 1}$.

Finally, since the input of the Conflict Adjusting algorithm is the output of the linear programming algorithm, the conflict adjusting algorithm makes a further approximation on the approximate basis of linear programming, so that the approximation ratio of our GAP-based algorithm is $\frac{1}{Uc_{\max} - 1} - O(\epsilon)$.

2) Complexity Analysis: According to [6], the complexity of using the linear programming algorithm is $O(n(m^+)^2 \log m^+)$. Let $maxCF$ be the maximum number of events that have conflicts with each other. Then, the complexity of the conflict adjusting algorithm is $O(n^2 \times maxCF \times Uc_{\max} \times \log Uc_{\max} + n \times maxCF^2 \times \log maxCF)$. Thus, the total complexity of our GAP-based algorithm is $O(n(m^+)^2 \log m^+ + n^2 \times maxCF \times Uc_{\max} \times \log Uc_{\max})$.

While relatively simple, the GAP-based algorithm will not scale well. When the size of the dataset becomes large, the computational cost is very large. Hence, in the next section, we provide a much faster approximate algorithm with a bounded approximation ratio just a little looser than the one offered by the GAP-based algorithm.

B. Greedy-Based Algorithm

The main idea of the greedy-based algorithm is as follows. First, the equivalent transformation of the ξ-GEPC problem introduced in Section III-A is applied. Then, at each step, we randomly select a user and let him/her greedily choose his/her favorite events. Of course, the user cannot choose new events having conflicts with previously chosen ones. The algorithm terminates when all the m^+ events have been chosen. The pseudo-code is shown in Algorithm 2.

Algorithm 2: Greedy-based Algorithm

```
   Input: E, U, {μ(u_i, e_j)}
   Output: P*
1  E' := E, U' := U
2  while E' ≠ ∅ do
3      Randomly select a user u_i from U'
4      P_i = ∅
5      D_i := 0
6      while D_i < B_i do
7          Find the event e ∈ E' that maximizes μ(u_i, e)
8          if e is not conflicting with the events in P_i then
9              Calculate a new D'_i if e is added into P_i
10             if D'_i < B_i then
11                 Add e into P_i
12                 Delete e from E'
13                 D_i := D'_i
14     Delete u_i from U'
15 P* = {P_i}
16 return P*
```

Initially, we create working copies, U' and E', of U and E, respectively (Line 1). At each step, we randomly select a user u_i from U', and initialize u_i's plan $P_i = \emptyset$ and travel cost $D_i = 0$ (Lines 3-5). As long as D_i is smaller than u_i's travel budget B_i, we pick u_i's current favorite event e in E' (Line 7). If e has no conflicts with the other events in P_i, and if when inserting e into P_i, the new travel cost D_i' is still within budget, we insert e into P_i, delete e from E', and update D_i to D_i' (Lines 8-13). That process continues until u_i's travel budget cannot afford any more of its favorite events (Line 6). We then delete u_i from U' (Line 14), and return to randomly

choosing a user from U', repeating the above process until E' is empty (Line 2). Recall that all events have been copied ξ_j times so that an event may be selected by several users. Finally, the global plan is returned.

Example 5: Consider again the problem in Example 1. Assume that the users are selected in the order u_1 ,u_2, u_3, u_4, u_5. The final plan is shown in Table III in red.

TABLE III
PLAN OBTAINED FROM GREEDY-BASED ALGORITHM

$e_j(\xi_j, \eta_j)$	$u_1(18)$	$u_2(20)$	$u_3(20)$	$u_4(30)$	$u_5(10)$	Time
$e_1(1,3)$	0.7	0.6	0.4	**0.2**	0.3	1:00-3:00 p.m.
$e_2(2,4)$	0.6	**0.5**	**0.7**	0.3	0.1	4:00-6:00 p.m.
$e_3(3,4)$	**0.9**	**0.8**	**0.9**	0.8	0.6	1:30-3:00 p.m.
$e_4(1,5)$	0.3	0.4	0.5	**0.6**	**0.7**	6:00-8:00 p.m.

First, u_1 chooses e_3 as $\mu(u_1, e_3)$ is the largest. Because e_1 has conflicts with e_3, and u_1's travel budget is not sufficient for him/her to attend e_2 or e_4, u_1's plan is complete. Then, u_2 chooses his/her events, namely e_3 and then e_2; and so on. After u_4 chooses e_4 and e_1, each event has been selected by ξ_j users, and the algorithm terminates. □

1) Approximation Ratio: We observe that except for the greedy method of each user, the order of users to select events also has an influence on the final total utility. In Example 5, if the order of users is changed to $\{u_1, u_2, u_3, u_5, u_4\}$, u_5 will choose e_4 before u_4 can choose it. Thus, the arrangement of e_4 will be u_5 instead of u_4 (shown in blue), and the total utility will be larger.

Let P_i be u_i's plan obtained from the greedy-based algorithm, and P_i^{OPT} be u_i's plan in the optimal global plan. If an event e_j is in P_i^{OPT} but not in P_i, it is either (1) selected by another user before u_i, or (2) still available for u_i and there exists at least one event in P_i whose utility is larger than $\mu(u_i, e_j)$. Let X_i be the set of events that are in $P_i^{OPT} \setminus P_i$ and are still available when u_i is selecting events. In other words, all the events in X_i are in situation (2). As stated above, the largest number of events that u_i can attend is bounded by Uc_i. Thus $|X_i| \leq Uc_i$. Similarly, in the worst case, after u_i greedily selects the event e^{max} whose utility is the largest, he/she can select no more events either due to time conflicts or excess in travel budget. On the other hand, the optimal solution is to give up e^{max} and select Uc_i events each of whose utility is just a little smaller than that of e^{max}. It follows, therefore, that $\sum_{e_j \in P_i} \mu(u_i, e_j) \geq \frac{1}{Uc_i} \sum_{e_j \in X_i} \mu(u_i, e_j)$, and thus $\sum_{i=1}^{n} \sum_{e_j \in P_i} \mu(u_i, e_j) \geq \frac{1}{Uc_{\max}} \sum_i (\sum_{e_j \in X_i} \mu(u_i, e_j))$.

If we know the relationship between $\sum_i (\sum_{e_j \in X_i} \mu(u_i, e_j))$ and OPT, we can get the approximation ratio for the greedy-based algorithm. Let $Y_i = P_i^{OPT} \cap P_i$. It is not hard to see that the utilities of events in Y_i are larger than those of the events in X_i. If $\sum_i (\sum_{e_j \in X_i} \mu(u_i, e_j)) \geq OPT/2$, according to the above inequality, $\sum_{i=1}^{n} \sum_{e_j \in P_i} \mu(u_i, e_j) \geq \frac{OPT}{2Uc_{\max}}$. If $\sum_i (\sum_{e_j \in X_i} \mu(u_i, e_j)) < OPT/2$, i.e., the sum of utilities in X is less than half of OPT, then, according to the definition of X_i and Y_i, the other half of OPT (more than $OPT/2$) must be in Y_i. Thus, we have $\sum_{i=1}^{n} \sum_{e_j \in P_i} \mu(u_i, e_j) \geq OPT/2$. Obviously, $OPT/2 \geq \frac{OPT}{2Uc_{\max}}$. Hence, $\sum_{i=1}^{n} \sum_{e_j \in P_i} \mu(u_i, e_j) \geq \frac{OPT}{2Uc_{\max}}$ holds, and it follows that the approximation ratio of the greedy-based algorithm is $\frac{1}{2Uc_{\max}}$.

2) Complexity Analysis: The while loop starting on Line 2 is executed at most $O(m^+)$ times. Each time, the complexity of finding the event with the greatest utility is $O(m^+)$. The complexity of checking whether there exist conflicts when adding an event and calculating the corresponding travel cost is $O(Uc_{\max})$. Thus, the total computational complexity is $O((m^+)^2 + Uc_{\max}))$.

IV. SOLUTION TO THE IEP PROBLEM

In this section, we describe our IEP framework. Recall our list of atomic operations, i.e., participation upper bound (η_j) increased/decreased, participation lower bound (ξ_j) increased/decreased, start time (t_j^s) and/or end time (t_j^t) changed, new event (e_j) added, utility score (μ_{u_i,e_j}) increased/decreased, and travel budget (B_i) increased/decreased. It should be clear that solving for changes caused by the 3 atomic operations: (1) "η_j decreased", (2) "ξ_j increased," and (3) "t_j^s and/or t_j^t modified", is sufficient since solving for all other atomic operations can be reduced to one of these. For example, the atomic operation "e_j added" may be reduced to increasing e_j's participation lower bound η_j from 0 to some positive value, so that the algorithm for atomic operation (2) can be used to solve this problem. Thus, we provide algorithms for only these three atomic operations.

A. η_j is decreased

Assume that n_j users have been assigned to event e_j in the original plan P, and that e_j's participation upper bound is decreased from η_j to η'_j. Recall that our objective in updating P to P' is to minimize the negative impact $dif(P, P')$. Clearly, if $\eta'_j \geq n_j$, there is no need for updating, i.e., $P' = P$ and $dif(P, P') = 0$. If not, the minimum negative impact is obtained by removing e_j from exactly $n_j - \eta'_j$ users' plans, so that $dif(P, P') = n_j - \eta'_j$. To maintain maximum utility in P', the users whose plans are altered are those who have the smallest utility scores for e_j. It is then possible to check whether these $n_j - \eta'_j$ users can attend other events within their current travel budget, using, for example, algorithms in [4]. The pseudo-code is shown in Algorithm 3.

Algorithm 3: η Decreasing Algorithm

Input: E, U, $\{\mu(u_i, e_j)\}$, P, η'_j
Output: P'
1 **if** $n_j \leq \eta'_j$ **then**
2 return (P)
3 **else**
4 Sort the users assigned to e_j in decreasing order of utility scores
5 Remove e_j from the plans of the last $n_j - \eta'_j$ users to get P'
6 Use methods in [4] to check if the $n_j - \eta'_j$ users can attend other events
7 **if** *true* **then**
8 Add these events to the corresponding plans in P'
9 return P'

Initially, we have the original plan P obtained from either of our algorithms of Section III, with n_j users assigned to

e_j, and a new lower participation upper bound, η'_j, for some event e_j. If $\eta'_j \geq n_j$, the original plan stands unchanged, with no negative impact (Lines 1-2). Otherwise, we arrange the n_j users assigned to e_j according to decreasing utility scores (Line 4), and remove e_j from the individual plans of the last $n_j - \eta'_j$ users (Line 5). This ensures that the remaining η'_j users have the largest utility scores to e_j, and $dif(P, P') = n_j - \eta'_j$, which is minimized. As each event is assigned at least ξ_j users after this step, according to the analysis in Section III, we can now use algorithms in [4] to check whether the $n_j - \eta'_j$ users can attend other events that have no conflicts with their current plans and are within their travel budget. If so, we add these events to their respective plans (Lines 6-9). Since it only adds events to users' plans, this step does not have any negative impact. Thus, the algorithm guarantees that negative impact is minimized, and greedily obtains a new global utility score.

Example 6: Consider the problem in Example 1, and the original plan as shown in Table I in color (red and blue). If η_4 is decreased from 5 to 4, no update is needed, $P = P'$, and $dif(P, P') = 0$. If η_4 is decreased from 5 to 1, we consider the two users, u_4 and u_5, assigned to e_4, and remove e_4 from u_4's plan since $\mu(u_4, e_4) < \mu(u_5, e_4)$. This results in a new global plan P' with $dif(P, P') = 1$. We then check whether u_4 may attend other events, and find that u_4 can now indeed participate in e_2 so we add e_2 to u_4' plan in P', with no negative impact. Overall, $dif(P, P') = 1$, which is clearly minimized. □

1) Approximation Ratio: Let $\{\mu_1, \ldots \mu_{\eta'_j}, \ldots, \mu_{n_j}\}$ be the utilities for e_j corresponding to $\{u_1, \ldots, u_{\eta'_j}, \ldots, u_{n_j}\}$ in decreasing order. In the worst case, after e_j is removed from $\{u_{\eta'_j}, \ldots, u_{n_j}\}$'s plans, no events can be added to these users' plans. In the optimal solution, if we remove e_j from $\{u_{2\eta'_j - n_j - 1}, \ldots, u_{\eta'_j - 1}\}$'s plans whose total utility is a little larger, then $Uc_i - 1$ events can be added to u_i's plan. Denote the original total utility as Ψ; the updated total utility is Ψ', and the optimal updated total utility is Ψ'_{OPT}. Similar to the Conflict Adjusting algorithm, the approximation ratio here is $\frac{\Psi'}{\Psi'_{OPT}} = \frac{\Psi - \sum_{i=\eta'_j}^{n_j} \mu_i}{\Psi - \sum_{i=2\eta'_j - n_j - 1}^{\eta'_j - 1} \mu_i + \sum_{i=2\eta'_j - n_j - 1}^{\eta'_j - 1} \sum_{j=1}^{Uc_i - 1} \mu(u_i, e_j)} \geq \frac{1}{(n_j - \eta'_j)(Uc_{\max} - 1)\mu_{2\eta'_j - n_j - 1}} \geq \frac{1}{(n_j - \eta'_j)(Uc_{\max} - 1)}$.

2) Complexity Analysis: If $\eta'_j \geq n_j$, the computational complexity is $O(1)$. Otherwise, the complexity of sorting the n_j users' utility scores is $O(n_j \log n_j)$, and the complexity of the fastest algorithm in [4] is $O(n_j m(m + \max_{j=1}^{m} \eta_j))$. Thus, the computational complexity of our η Decreasing algorithm is $O(n_j m(m + \max_{j=1}^{m} \eta_j))$.

B. ξ_j is increased

Assume that n_j users have been assigned to event e_j in the original plan P, and that e_j's participation lower bound is increased from ξ_j to ξ'_j. Clearly, if $\xi'_j \leq n_j$, there is no need for updating, i.e., $P' = P$ and $dif(P, P') = 0$. If not, we find other events $e_{j'}$ that have extra users (i.e., $n_{j'} > \xi_{j'}$) and greedily spare $\xi'_j - n_j$ users to e_j, so that $dif(P, P') = \xi'_j - n_j$. Then, similar to Algorithm 3, we check whether these $\xi'_j - n_j$ users can attend other events within their current travel budget. The pseudo-code is shown in Algorithm 4.

Algorithm 4: ξ Increasing Algorithm

```
Input: E, U, {μ(u_i, e_j)}, P, ξ'_j
Output: P'
1  if n_j ≥ ξ'_j then
2      return P
3  else
4      for each event e_j' do
5          if n_j' > ξ_j' then
6              for each user u_i assigned to e_j' do
7                  Calculate Δ = μ_j − μ_j' and insert into H in decreasing
                   order of Δ
8      for int k = 0; k < ξ'_j − n_j; k := k + 1 do
9          Pop the largest Δ from H, and get the corresponding u_i and e_j'
           providing such Δ
10         Check whether changing e_j' to e_j in u_i's plan causes conflicts and is
           still within travel budget B_i
11         if true then
12             Delete e_j' from u_i's plan and add e_j
13             Delete all such Δ related to u_i from H
14             n_j' := n_j' − 1
15             if n_j' == ξ_j' then
16                 Delete all such Δ related to e_j' from H
17     Use methods in [4] to check if the ξ'_j − n_j users can attend other events
18     if true then
19         Add these events to P'
20     return P'
```

Initially, we have the original plan P obtained from either of our algorithms of Section III, with n_j users assigned to e_j, and a new higher participation lower bound ξ'_j for some event e_j. If $\xi'_j \leq n_j$, the original plan stands unchanged, with no negative impact (Lines 1-2). Otherwise, we scan all the other events and find which have extra users that may be "transferred" to e_j (Lines 4-16). For each event $e_{j'}$, such that $n_{j'} > \xi_{j'}$, we calculate the utility difference $\Delta_i = \mu(u_i, e_{j'}) - \mu(u_i, e_j)$ for each of its assigned users. Here, we use a heap H to store the Δ's, each with its corresponding event $e_{j'}$ and user u_i that provides such Δ. The Δ's in H are in decreasing order (Lines 4-7). Then, at each step, we pop the largest Δ (and its corresponding u_i and $e_{j'}$) from H (Line 9), and check whether e_j can replace $e_{j'}$ in u_i's plan, i.e., it causes no conflicts in u_i's plan and u_i's travel cost is still within budget (Line 10). If so, we proceed with the replacement, and remove all the Δ's provided by u_i from H (Lines 12-13). At that point, $n_{j'}$ is changed into $n_{j'} - 1$. If, while going through this process, $n_{j'}$ reaches $\xi_{j'}$, then we would no longer be able to transfer any of its users, so, we delete all such Δ's provided by $e_{j'}$ in H (Lines 14-16). The process terminates when $\xi'_j - n_j$ users are assigned to e_j (Line 8), leading to a minimized negative impact $dif(P, P') = \xi'_j - n_j$. Finally, similar to Algorithm 3, we use algorithms in [4] to check whether the $\xi'_j - n_j$ can attend other events that have no conflicts with their current plans and are within their travel budget. If so, we add these events to their respective plans (Lines 17-19). This step again has no negative impact. Thus, the algorithm guarantees that negative impact is minimized, and greedily obtains a new global utility score.

Example 7: We still consider the problem in Example 1,

together with the global plan of Table I. If ξ_4 is increased from 1 to 2, no update is needed, $P = P'$, and $dif(P, P') = 0$. If ξ_4 in increased from 1 to 3, we need to find 1 user from e_2, e_3 and e_4 to assign to e_1. As $n_2 > \xi_2$, e_2 has extra users to spare to e_1. Thus, the utility values in H are $\{\Delta_1 = \mu(u_2, e_4) - \mu(u_2, e_2) = -0.1, \mu(u_3, e_4)\mu(u_3, e_2) = -0.2, \mu(u_1, e_4) - \mu(u_1, e_2) = -0.3\}$. We pop Δ_1 from H and find that changing e_2 into e_4 in u_2's plan causes no conflicts and B_2 is still sufficient. Thus, we change u_2's plan accordingly, and no further updates are needed. Overall, $dif(P, P') = 1$, which is clearly minimized. □

1) Approximation Ratio: Let $\{\Delta_1, \dots \Delta_{n_j}, \dots, \Delta_{\xi'_j}\}$ be the Δ_i's corresponding to $\{u_1, \dots, u_{n_j}, \dots, u_{\xi'_j}\}$ in decreasing order. In the worst case, after changing e_j to $e_{j'}$ in $\{u_{n_j}, \dots, u_{\xi'_j}\}$'s plans, no events can be added to these users' plans. In the best case, if we change $\{u_{2n_j - \xi'_j - 1}, \dots, u_{n_j - 1}\}$'s plans whose total utility is a little larger, then $Uc_i - 2$ events can be added to u_i's plan. Denote the original total utility as Ψ; the updated total utility is Ψ', the optimal updated total utility is Ψ'_{OPT}, and the approximation ratio is $\frac{\Psi'}{\Psi'_{OPT}} = \frac{\Psi + \sum_{i=n_j}^{\xi'_j} \Delta_i}{\Psi + \sum_{i=2n_j - \xi'_j - 1}^{n_j - 1} \Delta_i + \sum_{i=2n_j - \xi'_j - 1}^{n_j - 1} \sum_{j=1}^{Uc_i - 2} \mu(u_i, e_j)} \geq \frac{1}{(n_j - \eta'_j)(Uc_{\max} - 2)}$.

2) Complexity Analysis: If $n_j > \xi'_j$, the computational complexity is $O(1)$. Otherwise, let $n_{\max} = \max_{j'=1}^{m} n_{j'}$ and $\eta_{\max} = \max_{j'=1}^{m} \eta_{j'}$. The complexity of calculating the Δ's and ordering them into heap H is $O(mn_{\max} \log mn_{\max})$. Then, the complexity of finding $(\xi'_j - n_j)$ users where any $e_{j'} \neq e_j$ can be changed into e_j is $O((\xi'_j - n_j) \times (Uc_{\max} + \log mn))$. Finally, the complexity of the methods in [4] to check whether these $(\xi'_j - n_j)$ users can attend any more events is $O(m(\xi'_j - n_j)(m + \max_{j'=1}^{m} \eta_{j'}))$ time. Thus, the computational complexity of our ξ Increasing algorithm is $O(mn_{\max} \log mn_{\max} + m(\xi'_j - n_j)(m + \eta_{\max}))$.

C. t_j^s or t_j^t is changed

It is obvious that updates are needed only when the change to e_j's start or end times, t_j^s or t_j^t, causes conflicts in the original plan P. Hence, we first ind all users whose plans are conflicted and remove e_j from their plans. If the number of remaining users assigned to e_j is still larger than its participation lower bound ξ_j, the algorithm terminates. Otherwise, we check whether other users can also attend e_j. If, after this step, n_j users are assigned to e_j, and $n_j \geq \xi_j$, the algorithm terminates. Otherwise, we apply Algorithm 4 with e_j's participation lower bound increased from n_j to ξ_j. The detailed pseudo-code is shown in Algorithm 5

When we get the new holding time t_j^s or t_j^t of event e_j, we first find all n_j users assigned to e_j. For each such user u_i, we check whether the change in e_j causes time conflicts, and, if so, delete e_j from u_i's plan (Lines 1-4). Assume that, uc_j users are deleted from e_j, so that $dif(P, P') = uc_j$. If $n_j - uc_j > \xi_j$, the algorithm terminates (Lines 5-6). Otherwise, we use a heap H to store the utility scores of other users assigned to e_j in decreasing order. At each step, we pop the largest utility from the heap, corresponding to user $u_{i'}$. If adding e_j to $u_{i'}$'s plan would not cause conflicts nor exceed $u_{i'}$'s travel budget, we add e_j to $u_{i'}$'s plan (Lines 9-13). We repeat this process until H is empty (i.e., all users are checked) or the number of users assigned to e_j reaches η_j. During this process $dif(P, P') = 0$ since only event additions are performed. If n'_j users are assigned to e_j after this process, and $n'_j \geq \xi_j$, the algorithm terminates (Lines 14-15). Otherwise, we call Algorithm 4 with new participation lower bound $\xi'_j := \xi_j$ and previous participant lower bound $\xi_j := n'_j$ to get the final result (Lines 16-19). The corresponding negative impact is $\xi_j - n'_j$, and thus our algorithm produces $dif(P, P') = uc_j + \xi_j - n'_j$, which is clearly minimized.

Algorithm 5: t_j^s/t_j^t Changing Algorithm

```
   Input: E, U, {μ(u_i, e_j)}, P, t_j^{s'}, t_j^{t'}
   Output: P'
1  for each user u_i assigned to e_j in P do
2      if t_j^{s'} or t_j^{t'} causes conflicts with its plan then
3          Remove e_j from its plan and obtain P'
4          n_j := n_j − 1
5  if n_j ≥ ξ_j then
6      return P'
7  else
8      Order the other users' utility scores to e_j decreasingly and store in H
9      while H is not empty && n_j < η_j do
10         Pop the largest utility score from H with corresponding user u_{i'}
11         if adding e_j to its plan does not cause conflicts and travel cost is still
           within budget then
12             Add e_j to u_{i'}'s plan and get P'
13             n_j := n_j + 1
14     if n_j ≥ ξ_j then
15         return P'
16     else
17         Let ξ'_j := ξ_j, ξ_j := n_j
18         Call Algorithm 4 and get P'
19         return P'
```

Example 8: We again consider the problem in Example 1, together with the global plan of Table I. If e_1 is changed to be held from 3:30 p.m.-5:30 p.m., e_1 will have conflicts with e_2, but no conflicts with e_3. So, u_1 could not attend both e_1 and e_2 as originally planned. Thus, we remove e_1 from u_1's plan. This makes the assigned number of users to e_1 smaller than ξ_1. So we check whether other users can attend e_1. We find that u_4 can attend e_1. If no such user were found, we would Algorithm 4, increasing e_1's participation lower bound from 0 to ξ_1. □

1) Approximation Ratio: Similar to Algorithm 4, the approximation ratio of Algorithm 5 is $\frac{1}{(uc_j + \xi_j - n'_j)(Uc_{\max} - 1)}$.

2) Complexity Analysis: Before running Algorithm 4, the computational complexity is $O((uc_j + \xi_j)Uc_{\max})$. Thus, the computational complexity of t_j^s/t_j^t Changing algorithm is $O((uc_j + \xi_j)Uc_{\max} + mn_{\max} \log mn_{\max} + m(\xi'_j - n_j)(m + \eta_{\max}))$, where $n_{\max} = \max_{j'=1}^{m} n_{j'}$ and $\eta_{\max} = \max_{j'=1}^{m} \eta_{j'}$.

V. Performance Evaluation

This section provides an empirical evaluation of our proposed algorithms, in terms of utility, computational cost and

memory usage.

A. Experiment Environment and Dataset

The algorithms were implemented in C++ with STL, and the experiments were performed on a Linux Fedora 16 (Linux 3.6.11-4.fc16*86_62 GNOME 3.2.1) machine with Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz and 16GB memory. The memory costs reported here are calculated using system functions that monitor current memory usage.

We used the Meetup dataset [1], where there are a tag document and a location document for each user. The tag document records the labels that users selected when they registered on the platform. The location document records the longitude and latitude of each user's location. There are also a location document and a group document for each event, and a tag document for each group. In Meetup, events are created by groups. The group document records the events contained in each group, and the tag document records the interest tags of each group. The location document records the longitude and latitude of the place where each event is held. Using the tag document of users, the tag document of events, and the group document of events, we can calculate the utility of each user to each event according to the method introduced in [1][2]. The generation method for the parameters, B_i, t_j^s, t_j^t and η_j, is the same as in [4]. The ξ_j's are randomly generated from 0 to η_j. Table IV summarizes the parameters of the data. The conflict ratio is the proportion of events that have time conflicts.

TABLE IV
REAL DATASETS

City	$\|U\|$	$\|E\|$	Mean of ξ	Mean of η	Conflict ratio
Beijing	113	16	10	50	0.25
Vancouver	2012	225	10	50	0.25
Auckland	569	37	10	50	0.25
Singapore	1500	87	10	50	0.25

To test the scalability of our algorithms, we also use some "cut out" datasets, where some number of users and events are removed from the original data. Table V shows the various settings, with default values in bold.

TABLE V
"CUT OUT" DATASETS

Factor	Setting
$\|E\|$	20, **50**, 100, 200, 500
$\|U\|$	200, 500, 1000, **5000**

B. Results of GEPC problem

In this section, we test our two approximate algorithms for the GEPC problem. As the GAP-based algorithm (denoted as GAP here) is indeed an extension of the algorithm to solve GAP, we use this algorithm as a baseline to compare the greedy-based algorithm against.

Table VI shows the results on real datasets. We can see that the total utility obtained from the GAP-based approximation is a little larger than that obtained from the greedy-based algorithm. However, the time cost of the GAP-based algorithm is much larger than that of the greedy-based algorithm, and the memory cost of the GAP-based algorithm is a little larger than that of the greedy-based algorithm. This suggests that in practical applications, the greedy-based algorithm may be as effective and more efficient than the GAP-based algorithm.

Figures 2 and 3 report on scalability. To test the total utility, time cost and memory cost, we first set the number of events $|E| = 50$ and change the number of users $|U|$ from 100 to 5000; we then set $|U| = 5000$ and change $|E|$ from 20 to 500. From Figures 2(a) and 2(b), we can see that the total utility of both algorithms increases with increasing values of $|U|$ and $|E|$. The total utility of the GAP-based algorithm is a little larger than that of the greedy-based algorithm. This confirms the approximation ratio analysis of Section III. From Figures 2(c) and 2(d), we can see that the time cost of the greedy-based algorithm is much smaller than that of the GAP-based algorithm. The time cost of the GAP-based algorithm is almost 100 times larger than that of the greedy-based algorithm. This confirms the complexity analysis of Section III, and is as expected, given the computational complexity of the linear programming and conflict adjusting steps. From Figure 3, we can see that the memory cost of the GAP-based algorithm is a little larger than that of the greedy-based algorithm. These results further suggest that the greedy-based algorithm is more effective and efficient than the GAP-based algorithm.

C. Results of IEP problem

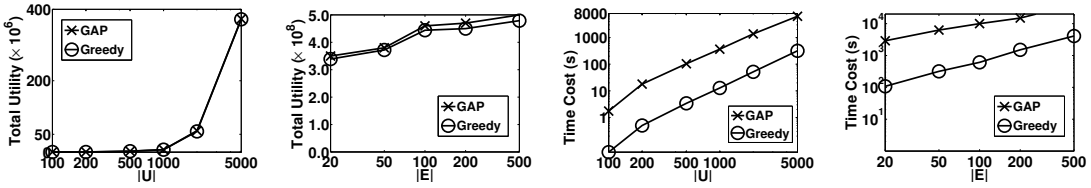
In this section, we test the performance of IEP, the incremental version of GEPC, for the three atomic operations, denoted here as η-De, ξ-In, and t^s-t^t', respectively. As above, we test the algorithms on real data and on "cut out" data.

For each algorithm, we randomly select 1 event, and decrease its η, increase its ξ, and change its t^s and t^t, respectively. We conduct the experiment 50 times and calculate the average total utility, time cost, and memory cost. Recall that with IEP, we must minimize the negative impact (i.e., minimize the number of canceled events for each user). As such, we compare the total utility obtained with our incremental algorithms with the one obtained by re-running the GAP-based algorithm and the greedy algorithm after an atomic operation is performed on the EBSN platform (denoted as Re-GAP and Re-Greedy, respectively). Results are shown in Tables VII-IX and Figs. 4 and 5.

The total utilities obtained with IEP are almost the same as those obtained by re-running the greedy-based algorithm (Re-Greedy) and the GAP-based algorithm (Re-GAP). This means that the results of refining the changes by keeping the minimized"unhappy" changes (i.e., negative impact) are almost the same as re-arranging all of the users to all of the events. Sometimes, the total utility of IEP is larger, while at other times it is smaller. This is reasonable because the greedy-based algorithm is also approximate and the selection order of users influences the total utility. It is quite possible that when performing some changes, the refining made by

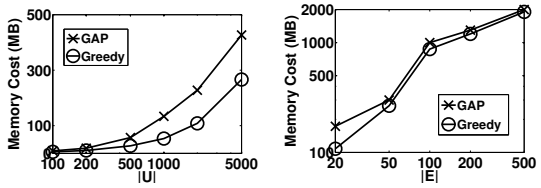
TABLE VI
ALGORITHMS FOR GEPC ON REAL DATASETS

Datasets	*GAP*			*Greedy*		
	Total Utility	Time Cost (s)	Memory Cost (MB)	Total Utility	Time Cost (s)	Memory Cost (MB)
Beijing	34306	1.32	3.9	32095	0.044	1.7
Vancouver	5.903×10^7	12383	545.6	5.903×10^7	231.89	341.2
Auckland	1.62×10^6	7.43	44.3	1.61×10^6	1.48	19.3
Singapore	6.93×10^6	138.02	56.7	6.93×10^6	12.80	53.1



(a) Total Utility vs $|U|$ (b) Total Utility vs $|E|$ (c) Time Cost vs $|U|$ (d) Time Cost vs $|E|$

Fig. 2. Performance of Algorithms for GEPC



(a) Memory Cost vs $|U|$ (b) Memory Cost vs $|E|$

Fig. 3. Memory Cost of Algorithms for GEPC

TABLE VII
RESULTS OF η-DE ON REAL DATASETS

Datasets	Utility (η-De)	Utility (Re-Greedy)	Utility (Re-GAP)	Time (s)	Memory (MB)
Beijing	218115	307850	330721	0.025	2.42
Vancouver	5.11×10^7	4.69×10^7	5.15×10^7	17.39	486.2
Auckland	1.58×10^6	1.74×10^6	1.78×10^6	2.49	25.09
Singapore	7.31×10^6	7.02×10^6	7.45×10^6	11.95	98.28

incremental algorithms makes the total utility larger than that of the original plan, while when re-running the greedy-based algorithm, the total utility becomes smaller due to a poor user selection order. The utilities obtained from Re-GAP algorithm are always a little larger than those of the other two algorithms. This is also reasonable since the GAP-based algorithm can provide a more accurate answer based on our analysis. From Figs. 4(d) and 4(h), we can see that the time cost increases

TABLE VIII
RESULTS OF ξ-IN ON REAL DATASETS

Datasets	Utility (ξ-In)	Utility (Re-Greedy)	Utility (Re-GAP)	Time (s)	Memory (MB)
Beijing	320295	360905	364758	0.085	3.91
Vancouver	5.24×10^7	4.79×10^7	5.35×10^7	22.16	784.76
Auckland	1.84×10^6	1.23×10^6	1.93×10^6	5.53	50.18
Singapore	7.88×10^6	1.03×10^7	1.35×10^7	35.23	127.44

TABLE IX
RESULTS OF t^s-t^t ON REAL DATASETS

Datasets	Utility (t^s-t^t)	Utility (Re-Greedy)	Utility (Re-GAP)	Time (s)	Memory (MB)
Beijing	78793	78039	80412	0.062	3.06
Vancouver	5.44×10^7	4.71×10^7	5.72×10^7	18.66	545.92
Auckland	2.07×10^6	1.63×10^6	2.83×10^6	6.23	40.53
Singapore	6.72×10^6	7.11×10^6	7.92×10^6	24.88	90.27

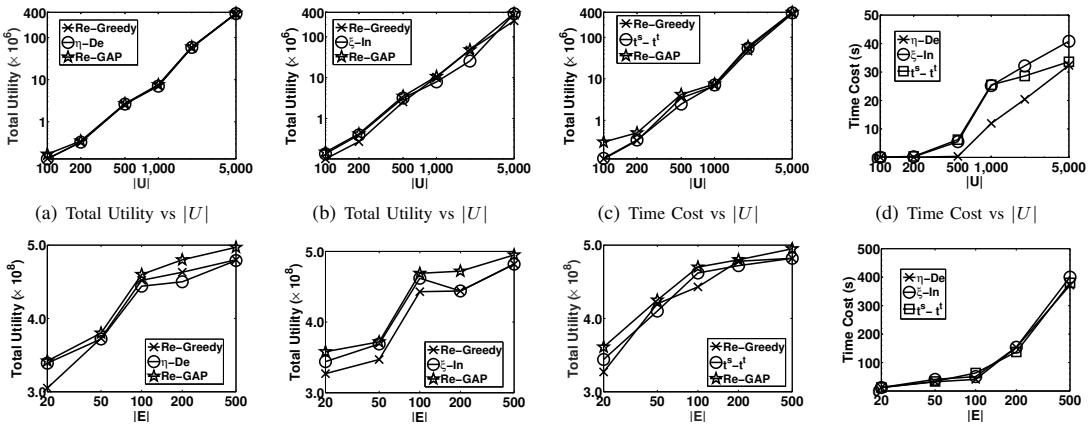
with increasing values of $|U|$ and $|E|$. The computation time of the η-De algorithm is a little smaller than that of the the other two, most likely due to the fact that its heap size is much smaller. Similar results are observed for memory cost as shown in Figs. 5(a) and 5(b). The memory costs of the 3 algorithms are nearly the same, but that of the η-De algorithm is a little smaller.

VI. RELATED WORK

We summarize the related work from three different perspectives: studies on Location-Based Social Networks (LBSNs), studies on EBSNs, and the difference between our work and variants of GAP.

Studies on LBSNs: Recent years have seen an increase in popularity of Online To Offline (O2O) services. One of the hottest topics in O2O services is Location-Based Social Networks (LBSNs) [7], [8], [9], [10], [11], [12]. Although work based on LBSNs recommends or arranges users to events (or places, such as restaurants and shopping malls), it focuses on how to maximize users' individual utilities, i.e., on providing user-oriented recommendations. On the other hand, our work, like other works on EBSNs, focuses on maximizing the total utility of the whole system. In other words, EBSNs schedule all users and create global satisfiable plans.

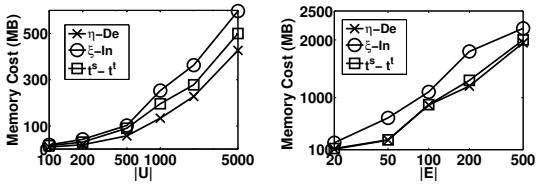
Studies on EBSNs: The concept of EBSN was first proposed in [1]. This work analyzed the characteristics of data from Meetup and Plancust, which are two of the most popular EBSN platforms, and proposed the formulation of EBSN and its corresponding properties. Further research considered recommending events to related users by using machine-learning methods on EBSN historical data [13][14]. Other researchers proposed a general heterogeneous graph model to abstract the EBSN data and provided a general training method to solve 3 kinds of recommendation problems over EBSN: recommending groups to users, recommending tags to groups and recommending events to users [15]. Again, the focus was on individual user recommendations rather than a



(a) Total Utility vs $|U|$ (b) Total Utility vs $|U|$ (c) Time Cost vs $|U|$ (d) Time Cost vs $|U|$

(e) Total Utility vs $|E|$ (f) Total Utility vs $|E|$ (g) Time Cost vs $|E|$ (h) Time Cost vs $|E|$

Fig. 4. Performance of Algorithms for IEP



(a) Memory Cost vs $|U|$ (b) Memory Cost vs $|E|$

Fig. 5. Memory Cost of Algorithms for IEP

global satisfiable planning. A related, although quite different problem, is that of mining an influential cover set (ICS), that is, to find k users who together have the required skills and expertise to organize an event such that they can influence the largest number of users in the social network [16]. Essentially, the aim is to solve the influence maximization problem [17], together with the team formation problem [18]. We focus on a planning providing maximized total utility.

The work on the Social Event Organization problem, which consists of assigning a set of events for a group of users to attend that provides maximized overall satisfaction, and its variants, are most similar to ours [2][3][4]. The problems they study are typical and representative problems in EBSN. However, none of them contain all of the considerations in our paper, especially for the aspects of the events' participation lower bound and the incremental changes to the constraints on users and events. The main contribution of our paper is that we overcome the shortcomings of these studies. Accordingly, we propose two problems, Global Event Planning with Constraints (GEPC) and Incremental Event Planning (IEP), and propose approximate algorithms, each with a bounded approximation ratio, to solve these problems, since the problems are NP-hard. Prior approaches are special cases of our GEPC problem.

Studies on GAP variants: Although our problem can be written into integer programming like an operations research problem, we focus on designing algorithms with bounded approximation ratios from the perspective of complexity and algorithmics, since no general operations research methods can provide solutions to all questions with proven bounds. Considering our specific objectives and constraints, we showed that the Generalized Assignment Problem (GAP) can be reduced to a special case of the GEPC problem, ξ-GEPC. In other words, even the ξ-GEPC problem is harder than GAP. We further find that if ignoring the conflicts of events in the ξ-GEPC, this simplified problem can be solved by GAP. Then, how to further process the solutions of GAP to get the final answer with a bounded approximation ratio is what we have done in Algorithm 1. Additionally, considering the low efficiency of this GAP-based algorithm, we proposed a greedy one with a much higher efficiency and an approximation ratio not much worse than the GAP-based algorithm. We note also that there are variants of GAP studied in the literature, but we find that these variants are not the same as our GEPC problem (see [19] for a survey). The Bottleneck GAP [20] changes the objective function to a min-max version to minimize the maximum cost of machines. The Multi-level GAP allows the machines in GAP to have several levels [21]. The Non-linear Capacity Constraint GAP treats the capacity constraint of machines as a function [22]. Finally, in the Stochastic GAP, the jobs/machines are not consistently given, but follow a random function or are in a sequence [23][24]. Since the basic GAP, as well as the aforementioned variants, cannot handle events' participation lower bounds and conflicts among events, they are of a different kind than our GEPC problem. A variant of GAP with minimum quantities is proposed in [25]. However, these minimum quantities are used to constrain users rather than events. Besides, neither GAP nor its variants studied the situation in which constraints may incrementally change.

Finally, we note that the work on entangled queries also has some relevance to our own [26], in that they both study a matching problem with several constraints. There are significant differences, however. First, the former aims to find a feasible result satisfying the constraints to answer a specific query, while the latter focuses on a result that maximizes all

users' total satisfaction. Second, no attempt is made in the former to address the situation when constraints are changed incrementally changed, as per the IEP problem. Finally, the solutions are of different kinds. The former approach focuses on how to reduce the hardness of the problem, and efficiently evaluate the SQL queries in relational databases, using strategies like "safe" and "unique" to make the evaluation tractable, and reduce the search space. By contrast, our approach is designed to provide approximate algorithms to directly solve the NP-hard problems, and analyze the approximate-ratio and complexity for each algorithm.

VII. Conclusion

In this paper, we define the Global Event Planning with Constraints (GEPC) problem, which creates a global plan of multiple events for each user with maximized total utility. We consider the following constraints: event participation lower and upper bounds, time conflicts among events, travel costs among events, and user travel budget. To the best of our knowledge, our work is the first to consider all of the above constraints at once. We first prove that this problem is NP-hard and propose two approximate algorithms with provable approximation ratio. The first one is based on linear programming, which has a good approximation ratio but poor scalability. In order to improve the efficiency, we also provide a greedy-based algorithm, with guaranteed approximation ratio. Finally, we also provide an incremental variant of GEPC, called Incremental Event Planning (IEP), that minimally updates the planning when the attribute of a user or of an event is changed. We analyze all possible changes and provide approximate algorithms with bounded approximation ratio for each possibility. Experiments over real and synthetic datasets demonstrate the effectiveness and efficiency of our algorithms.

While our problem formulation refers to users' costs as travel costs, it is clear that such costs could take into account not only travel, but also potential costs associated with attending events (e.g., admission fees). Whether these could be naturally rolled into travel costs and thus be treated uniformly is an interesting question, which future work may address. Similarly, our incremental version assumed changes to a single attribute, running multiple times if multiple changes were to take place. It would be interesting to consider how the algorithm would need to be extended or modified for multiple atomic changes to be handled in a single run.

Acknowledgment

Yurong Cheng and Guoren Wang are supported by the NSFC (Grants No. 61332006, 61332014, 61328202 and U1401256). Ye Yuan is supported by the NSFC (Grants No. 61572119 and 61622202) and the Fundamental Research Funds for the Central Universities (Grant No. N150402005). Lei Chen is supported by Hong Kong RGC GRF Project 16202215, NSFC Guang Dong Grant (No. U1301253, NSFC 61232018), National Grand Fundamental Research 973 Program of China under Grant 2014CB340303 , and Microsoft Research Asia Gift Grant.

References

[1] 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 *SIGKDD*, 2012, pp. 1032–1040.

[2] J. She, Y. Tong, L. Chen, and C. C. Cao, "Conflict-aware event-participant arrangement," in *ICDE*, 2015.

[3] K. Li, W. Lu, S. Bhagat, L. V. Lakshmanan, and C. Yu, "On social event organization," in *SIGKDD*, 2014.

[4] J. She, Y. Tong, and L. Chen, "Utility-aware social event-participant planning," in *SIGMOD*, 2015.

[5] S. A. Plotkin, D. B. Shmoys, and É. Tardos, "Fast approximation algorithms for fractional packing and covering problems," *Mathematics of Operations Research*, vol. 20, no. 2, 1995.

[6] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, no. 1-3, 1993.

[7] E. Minkov, B. Charrow, J. Ledlie, S. Teller, and T. Jaakkola, "Collaborative future event recommendation," in *CIKM*, 2010.

[8] 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*, 2013.

[9] H. Khrouf and R. Troncy, "Hybrid event recommendation using linked data and user diversity," in *RecSys*, 2013.

[10] Y.-C. Sun and C. C. Chen, "A novel social event recommendation method based on social and collaborative friendships," in *Social Informatics*. Springer, 2013.

[11] C. Chen, D. Zhang, B. Guo, X. Ma, G. Pan, and Z. Wu, "Tripplanner: personalized trip planning leveraging heterogeneous crowdsourced digital footprints," *T-ITS*, 2014.

[12] 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*, 2012.

[13] W. Zhang, J. Wang, and W. Feng, "Combining latent factor model with location features for event-based group recommendation," in *SIGKDD*, 2013.

[14] 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*, 2014.

[15] T.-A. N. Pham, X. Li, G. Cong, and Z. Zhang, "A general graph-based model for recommendation in event-based social networks," in *ICDE*, 2015.

[16] K. Feng, G. Cong, S. S. Bhowmick, and S. Ma, "In search of influential event organizers in online social networks," in *SIGMOD*, 2014.

[17] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *SIGKDD*, 2003.

[18] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *SIGKDD*, 2009.

[19] T. Öncan, "A survey of the generalized assignment problem and its applications," *INFOR: Information Systems and Operational Research*, vol. 45, no. 3, 2007.

[20] J. Mazzola and A. Neebe, "Bottleneck generalized assignment problems," *Engineering Costs and Production Economics*, vol. 14, no. 1, 1988.

[21] M. Laguna, J. P. Kelly, J. González-Velarde, and F. Glover, "Tabu search for the multilevel generalized assignment problem," *European Journal of Operational Research*, vol. 82, no. 1, 1995.

[22] J. B. Mazzola, A. W. Neebe, and C. V. Dunn, "Production planning of a flexible manufacturing system in a material requirements planning environment," *International Journal of Flexible Manufacturing Systems*, vol. 1, no. 2, 1989.

[23] D. R. Spoerl and R. K. Wood, "A stochastic generalized assignment problem," 2004.

[24] C. Derman, G. J. Lieberman, and S. M. Ross, "A sequential stochastic assignment problem," *Management Science*, vol. 18, no. 7, 1972.

[25] S. O. Krumke and C. Thielen, "The generalized assignment problem with minimum quantities," *European Journal of Operational Research*, vol. 228, no. 1, 2013.

[26] N. Gupta, L. Kot, S. Roy, G. Bender, J. Gehrke, and C. Koch, "Entangled queries: Enabling declarative data-driven coordination," *ACM Transactions on Database Systems*, vol. 37, no. 3, 2012.