Event-Participant and Incremental Planning over Event-Based Social Networks

Yurong Cheng, *Member, IEEE*, Ye Yuan, *Member, IEEE*, Lei Chen, *Member, IEEE*, Christophe Giraud-Carrier, Guoren Wang, and Boyang Li

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. Since these problems 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.

Index Terms—Planning, event-based social networks, incremental planning, approximate algorithm

1 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.

> *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 percent 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.

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

- *Y. Cheng and G. Wang are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100811, China. E-mail: {yrcheng, wanggr}@bit.edu.cn.*
- *Y. Yuan and B. Li are with the School of Computer Science and Engineering, Northeastern University, Qinhuangdao, Hebei 066004, China. E-mail: {yuanye, liboyang}@mail.neu.edu.cn.*
- *L. Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. E-mail: leichen@cse.ust.hk.*
- *C. Giraud-Carrier is with the Department of Computer Science, Brigham Young University, Provo, UT 84602 USA. E-mail: cgc@cs.byu.edu.*

Manuscript received 11 May 2018; revised 27 June 2019; accepted 15 July 2019. Date of publication 30 July 2019; date of current version 11 Jan. 2021. (Corresponding author: Guoren Wang.)
Recommended for acceptance by Y. Xia.
Digital Object Identifier no. 10.1109/TKDE.2019.2931906

- *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 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

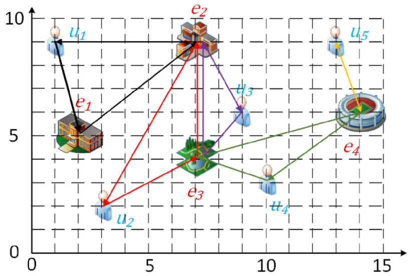


Fig. 1. Locations of users and events.

algorithms and approximation ratio are in fact quite different. Since the GEPC problem is NP-hard [5], subsequently proposing a two-step framework, that not only satisfies the minimum-participant requirement constraint for each event but also provides approximate guarantees. Similarly, the IEP problem is also NP-hard [5], 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.

This paper extends our previous work [5] in several significant ways. First, our earlier work focused on only 3 basic changes in the IEP problem, namely decrease in participation upper bound, increase in participation lower bound, and modification of start and/or end times. Here, we add a fourth basic change to the IEP problem, namely decrease in travel budget (see Section 4.4). We show that all other possible changes are special cases of these 4 basic changes, and provide the corresponding approximation ratios (see Section 4.5). Second, the solutions proposed for the IEP problem in our earlier work were restricted to individual changes to a single constraints (e.g., participation lower bound, start time), and thus do not handle situations where multiple kinds of changes are expected to be made at the same time (e.g., participation upper bound *and* travel budget). We present an algorithm here that can create a new plan for such multiple changes in a single run, and provide corresponding approximation ratio (see Section 4.6). Finally, we extend our earlier experiments with an evaluation of these new algorithms (see Section 5).

2 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 $(\mathbf{l}_{e_j}, \xi_j, \eta_j, t^s_j, t^t_j)$ 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.

Table 1 shows the other information associated with each user and event.

TABLE 1
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.

Users and their travel budgets are shown on row 1. 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 $\mu_1 = \mu(u_1, e_1) + \mu(u_1, e_2) = 0.7 + 0.6 = 1.3$.

2.1 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 2.1 (GEPC problem). *Given an EBSN, the GEPC problem is to find a feasible global plan P^*, such that $\mathcal{U}_{P^*} = \max_P \mathcal{U}_P$, 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 1 correspond to a global plan. The EBSN's global utility score under the given plan is $\mu(u_1, e_1) + \mu(u_1, e_2) + \cdots + \mu(u_5, e_4) = 6.3$.

According to [5], the GEPC problem is NP-hard.

2.2 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.

2.2.1 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 2:00 p.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.

2.2.2 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 the start and end times, or an alternate location may have to be found.

We refer to the above changes as *atomic* operations. 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 2.2 (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'^*, such that $\mathcal{U}_{P'^*} = \max_{P'} \mathcal{U}_{P'}$, subject to* $\mathrm{dif}(P, P'^*) = \min_{P'} \mathrm{dif}(P, P')$.

3 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.

According to [5], the ξ-GEPC problem is NP-hard, so we go on to provide two approximate algorithms with bounded approximation ratio to solve it.

3.1 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 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 [6].

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

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.

Algorithm 1. Conflict Adjusting Algorithm

Input: $E, U, \{\mu(u_i, e_j)\}, P$
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** $U_e \neq \emptyset$ *and e is not assigned* **do**
8: Find $u_k \in U_e$ whose utility to e is the largest
9: **if** e *is not in conflict with events in* P_k *and* $D_k \leq B_k$ *after adding* e *to* P_k **then**
10: $P_k := P_k \cup \{e\}$
11: break
12: **else**
13: $U_e := U_e - \{u_k\}$
14: $P' := \{P_i\}$
15: return P'

According to [5], the approximation ratio of our GAP-based algorithm is $\frac{1}{Uc_{\max}-1} - O(\epsilon)$. The computational complexity $O(n(m^+)^2 \log m^+ + n^2 \times maxCF \times Uc_{\max} \times \log Uc_{\max})$. Here $Uc_{\max} = \max_{i=1}^{n} Uc_i$, where Uc_i is the number of events that fall within a distance $B_i/2$ of l_{u_i}.

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.

3.2 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 3.1 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.

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.

Algorithm 2. Greedy-based Algorithm

Input: $E, U, \{\mu(u_i, e_j)\}$
Output: P^*
$E' := E, U' := U$
while $E' \neq \emptyset$ **do**
  Randomly select a user u_i from U'
  $P_i = \emptyset$
  $D_i := 0$
  while $D_i < B_i$ **do**
    Find the event $e \in E'$ that maximizes $\mu(u_i, e)$
    if *e is not conflicting with the events in* P_i **then**
      Calculate a new D_i' if e is added into P_i
      if $D_i' < B_i$ **then**
        Add e into P_i
        Delete e from E'
        $D_i := D_i'$
  Delete u_i from U'
$P^* = \{P_i\}$
return P^*

According to [5], the approximation ratio of the greedy-based algorithm is $\frac{1}{2Uc_{\max}}$, and the computation complexity is $O((m^+)^2 + Uc_{\max}))$. Here $Uc_{\max} = \max_{i=1}^{n} Uc_i$, where Uc_i is the number of events that fall within a distance $B_i/2$ of l_{u_i}.

4 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. Interestingly, solving for changes caused by the 4 basic atomic operations: (1) "η_j decreased", (2) "ξ_j increased", (3) "t_j^s and/or t_j^t modified", and (4) "B_i is decreased", turns out to be sufficient since, as we will also show, solving for all other atomic operations can be reduced to these ones. We begin by describing solutions to the four basic changes, and then illustrate how all other atomic operations can be handled as either special cases or combination of the basic ones. Finally, we provide solutions for the problem of solving for multiple atomic operations in one run, to account for situations when more than one change is made simultaneously to a plan.

4.1 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_j Decreasing Algorithm

Input: $E, U, \{\mu(u_i, e_j)\}, P, \eta_j'$
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 3, 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 3, 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.

Based on [5], the approximation ratio is $\frac{1}{(n_j - \eta_j')(Uc_{\max} - 1)}$, the computational complexity is $O(n_j m(m + \max_{j=1}^{m} \eta_j))$.

4.2 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, 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.

Initially, we have the original plan P obtained from either of our algorithms of Section 3, 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.

Algorithm 4. ξ_j Increasing Algorithm

Input: $E, U, \{\mu(u_i, e_j)\}, P, \xi'_j$
Output: P'
1: **if** $n_j \geq \xi'_j$ **then**
2: return P
3: **else**
4: **for** *each event* $e_{j'}$ **do**
5: **if** $n_{j'} > \xi_{j'}$ **then**
6: **for** *each user* u_i *assigned to* $e_{j'}$ **do**
7: Calculate $\Delta = \mu(u_i, e_{j'}) - \mu(u_i, e_j)$ and insert into H in decreasing order of Δ
8: **for** *int* $k = 0$; $k < \xi'_j - n_j$; $k := k + 1$ **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** $n_{j'} == \xi_{j'}$ **then**
16: Delete all such Δ related to $e_{j'}$ from H
17: Use methods in [4] to check if the $\xi'_j - n_j$ users can attend other events
18: **if** *true* **then**
19: Add these events to P'
20: return P'

According to [5], the approximation ratio is $\frac{1}{(\xi'_j - n_j)(Uc_{\max} - 2)}$, and the computational complexity is $O(mn_{\max}\log mn_{\max} + m(\xi'_j - n_j)(m + \eta_{\max}))$.

4.3 t^s_j or t^t_j Is Changed

It is obvious that updates are needed only when the change to e_j's start or end times, t^s_j or t^t_j, 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.

Algorithm 5. t^s_j/t^t_j Changing Algorithm

Input: $E, U, \{\mu(u_i, e_j)\}, P, t^{s'}_j, t^{t'}_j$
Output: P'
1: **for** *each user* u_i *assigned to* e_j *in* P **do**
2: **if** $t^{s'}_j$ *or* $t^{t'}_j$ *causes conflicts with its plan* **then**
3: Remove e_j from its plan and obtain P'
4: $n_j := n_j - 1$
5: **if** $n_j \geq \xi_j$ **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* && $n_j < \eta_j$ **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** $n_j \geq \xi_j$ **then**
15: return P'
16: **else**
17: Let $\xi'_j := \xi_j$, $\xi_j := n_j$
18: Call Algorithm 4 and update P'
19: return P'

When we get the new holding time t^s_j or t^t_j 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 participation 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.

According to [5], the approximation ratio is $\frac{1}{(uc_j + \xi_j - n'_j)(Uc_{\max} - 1)}$, and the computational complexity is $O((uc_j + \xi_j)Uc_{\max} + mn_{\max}\log mn_{\max} + m(\xi'_j - n_j)(m + \eta_{\max}))$, where $n_{\max} = \max^m_{j'=1} n_{j'}$ and $\eta_{\max} = \max^m_{j'=1} \eta_{j'}$.

4.4 B_i is Decreased

Denote the decreased travel budget of u_i as B'_i. If B'_i is still sufficient to handle the original plan, i.e., $D_i \leq B'_i$, clearly no changes are needed. Otherwise, we order the events assigned to u_i in decreasing order according to their distance to u_i's location. We then proceed to delete iteratively the next farthest event from u_i's original plan until the cost, D'_i, of the new plan is less than or equal to B'_i. This ensures that we delete the smallest number of events so as to satisfy

B'_i, i.e., $dif(P, P')$ is minimized. For each deleted e_j from u_i's original plan, if its currently assigned number of users, n_j, is less than ξ_j, we call Algorithm 4 to satisfy its participation lower bound. The detailed pseudo-code is shown in Algorithm 6.

Algorithm 6. B_i Decreasing Algorithm

Input: $E, U, \{\mu(u_i, e_j)\}, P, B'_i$
Output: P'
1: **if** $D_i \leq B'_i$ **then**
2: return P
3: **else**
4: Order the events assigned to u_i in decreasing order of distance from u_i and store in H
5: $D'_i := D_i$
6: **while** *H is not empty* && $D'_i > B'_i$ **do**
7: Pop the first event, e_j, from H
8: Delete e_j from u_i's plan
9: Update D'_i
10: **for** *each deleted* e_j **do**
11: **if** $n_j < \xi_j$ **then**
12: Let $\xi'_j := \xi_j$, $\xi_j := n_j$
13: Call Algorithm 4 and update P'
14: return P'

Initially, we have the original plan P obtained from either of our algorithms of Section 3, with a new travel budget B'_i for user u_i. The travel cost of u_i in the original plan P is D_i. If $D_i \leq B'_i$, the original plan stands unchanged, with no negative impact (Lines 1-2). Otherwise, we successively delete the farthest event from P, until the new travel cost D'_i is not larger than B'_i (Lines 4-9). For each event assigned to u_i in the original plan P, we calculate the distance, $d(u_i, e_j)$, between u_i and e_j. Here, we use a heap H to store these distances in decreasing order, each with its corresponding event e_j (Line 4). Then at each step, we pop the event, e_j, corresponding to the largest distance, $d(u_i, e_j)$, from H, and remove e_j from P. This can ensure that negative impact is minimized. If any deletion of e_j from u_i's plan causes the number of participants of e_j to become smaller than its participation lower bound, ξ_j, then we are in the situation where e_j's participation lower bound is increased. Denote as n_j the number of users assigned to e_j. Let $\xi'_j = \xi_j$, $\xi_j = n_j$, and call Algorithm 4 to resolve this problem (Lines 11-13). This process does not have any negative impact. Thus, the algorithm guarantees that the negative impact is minimized.

Example 3. As before, we continue with the problem in Example 1, together with the global plan of Table 1. Assume u_1's travel budget, B_1, is decreased from 18 to 8. Then, since $D_1 = d(u_1, e_1) + d(e_1, e_2) + d(e_2, u_1) = \sqrt{17} + \sqrt{41} + 6 = 16.53 > 8$, we must make adjustment to u_1's current plan. We consider the events assigned to u_1, and order them in decreasing order of distance from u_1, yielding e_2 (6) and e_1 ($\sqrt{17}$). After deleting e_2 from u_1's plan, the travel cost, D'_1, is $2\sqrt{17}$, which is still larger than 8. Hence, we proceed to remove e_1 from u_1's plan. Then $D'_1 = 0 \leq 8$ and we stop. u_1's plan is now empty. After these changes, however, no users are assigned to e_1, which breaks the constraint on its participation lower bound. Calling Algorithm 4, we find that u_4 can attend e_1, and consequently add e_1 to u_4's plan.

4.4.1 Approximation Ratio

The approximation ratio of Algorithm 6 is influenced by two main factors. One is the order of events in Lines 11 and 13, and the other is due to Algorithm 4. For each event deletion from the user's plan causing the event's participation lower bound to be violated, we call Algorithm 4. However, the order of events may influence the approximation ratio. For example, assume that events e_1, e_2 and e_3 are deleted and cause underflow on their respective participation lower bounds. Processing e_1, then e_2, and finally e_3 may result in a different overall utility than processing e_1, then e_3, and finally e_2, since our algorithms are greedy. Based on the derivation of the approximate ratio of Algorithms 2 and 4, the approximation ratio of Algorithm 6 is $\frac{1}{2(Uc_{\max}-2)}$.

4.4.2 Complexity Analysis

The computational complexity of Lines 4-9 of Algorithm 6 is $O(m_i)$, where m_i is the number of events that are assigned to user u_i in its original plan. According to the computational complexity of Algorithm 4, the computational complexity of Lines 11-13 is $O(m_u(mn_{\max}\log mn_{\max} + m(m + \eta_{\max})))$, where $n_{\max} = \max_{j'=1}^{m} n_{j'}$ and $\eta_{\max} = \max_{j'=1}^{m} \eta_{j'}$.

4.5 Other Atomic Operations

In this subsection, we show how all other atomic operations in the IEP problem can be handled as either special cases, or combination, of the four basic operations discussed above, and the algorithms relevant to the GEPC problem.

4.5.1 η_j is Increased

Assume that the participation upper bound, η_j, of event e_j is increased. This situation can be handled naturally by Step 2) of the GEPC framework described in Section 3.

4.5.2 ξ_j is Decreased

Assume that the participation lower bound, ξ_j, of event e_j is decreased. The current plan is still adequate. Thus, to minimize the negative impact on users, the overall planning is left unchanged.

4.5.3 Event is Added or Removed

Assume that a new event e_k is added to the platform, with participation lower bound ξ_k and participation upper bound η_k. Then, e_k must now be assigned a sufficient number of users to satisfy its participation lower bound. This can be achieved by simply considering that e_k's participation lower bound increased from 0 to ξ_k, and applying Algorithm 4.

Similarly, if an existing event e_j is withdrawn from the platform, then all users assigned to e_j must be removed, which is equivalent to decreasing e_j's participation upper bound from η_j to 0. This change can be handled by Algorithm 3.

4.5.4 Utility Score is Increased

Assume that the utility score, $\mu(u_i, e_j)$, of user u_i to event e_j is increased. The minimum $dif(P, P')$ should be 0. Since the original plan provided by the GEPC algorithms is (approximately) optimal, it follows that user u_i cannot attend more events within its current travel budget B_i. Thus, no updates are necessary.

4.5.5 Utility Score is Decreased

Assume that the utility score, $\mu(u_i, e_j)$, of user u_i to event e_j is decreased to some non-zero value. Then, the minimum $dif(P, P')$ is 0. As in the situation in which the utility score is increased, no updates are necessary in this case.

If the utility score, $\mu(u_i, e_j)$, of user u_i to event e_j is actually decreased to 0, however, the plan must be altered. Similar to Algorithm 5, we delete e_j from u_i's original plan, and check whether this deletion breaks e_j's participation lower bound. If so, as in Lines 16-19 of Algorithm 5, we call Algorithm 4 to fix the problem and ensure that e_j's participation lower bound is satisfied. As per Algorithm 5, the approximation ratio is $\frac{1}{2(Uc_{\max}-1)}$, and the corresponding complexity is $O(\xi_j Uc_{\max} + mn_{\max}\log mn_{\max} + m(m+\eta_{\max}))$, where $n_{\max} = \max_{j'=1}^{m} n_{j'}$ and $\eta_{\max} = \max_{j'=1}^{m} \eta_{j'}$.

4.5.6 Travel Budget is Increased

Assume that the travel budget, B_i, of user u_i is increased. As with the case of increasing an event's participation upper bound, this situation can be handled naturally by Step 2) of the GEPC framework described in Section 3, to check whether u_i can attend more events.

4.6 Multiple Atomic Operations in One Run

In practice, any number of events can change their associated properties, and any number of users can change their associated properties. In this section, we consider this natural situation in which multiple changes are proposed at the same time, and how to handle all of the corresponding atomic operations in one run. Recall that all allowable atomic operations can be handled by the 4 basic atomic operations detailed in Sections 4.1, 4.2, 4.3, and 4.4. Hence, we simply need to show how to effectively and efficiently apply Algorithms 3, 4, 5, and 6 in one run.

The main idea is as follows. Given a set of users together with their associated proposed property changes, and a set of events together with their associated proposed property changes, all associated events and users are dissociated, and affected plans are re-planned to satisfy the new constraints imposed by the atomic operations, using mechanisms from the corresponding algorithms. We then check whether the participation lower bounds of any event are no longer satisfied, and if so, make the necessary adjustments, again using existing mechanisms. The detailed pseudo-code is shown in Algorithm 7. For simplicity, we denote changed events by $(\eta'_j, \xi'_j, t_j^{s'}, t_j^{t'})$, and changed events by (B'_i), where it is understood that some of the properties may remain unchanged (e.g., $\eta'_j = \eta_j$).

The algorithm is given as input an original plan P, obtained from one of the algorithms of Section 3, together with a set of revised travel budgets, $\{(B'_i)\}$, for some users, and a set of revised participation lower/upper bounds and start/end times, $\{(\eta'_j, \xi'_j, t_j^{s'}, t_j^{t'})\}$, for some events. If other atomic operations are intended, they are first transformed into their equivalent forms involving only B'_i, η'_j, ξ'_j, $t_j^{s'}$, and $t_j^{t'}$, according to the methods introduced in Section 4.5. If u_i's proposed new travel budget B'_i is decreased, such that $D_i > B'_i$, and simultaneously, either event e_j's proposed new participation upper bound is decreased, such that $n_j > \eta'_j$, or its proposed new holding times $t_j^{s'}$, $t_j^{t'}$ cause conflicts, we dissociate u_i from e_j in the original plan P. This is to make sure that the negative impact $dif(P, P')$ is minimized. If the participation upper bound of any event e_j is decreased, we execute Lines 4-5 of Algorithm 3, to remove $\eta_j - \eta'_j$ users' plan whose utility scores are the smallest (Lines 4-5). If the starting time $t_j^{s'}$ and ending time $t_j^{t'}$ of any event e_j are changed, we execute Lines 1-4 of Algorithm 5 to remove conflicts from the original plan P (Lines 6-7). If the travel budget of any user is decreased, we execute Lines 4-9 of Algorithm 6 to remove plans beyond the new budget (Lines 9-10). After removing all such plans that do not satisfy the constraints, as in Algorithms 5 and 6, we check whether any event's participation lower bound is no longer satisfied, and deal with the new increased lower bound ξ'_j if it exists. We execute Lines 4-16 to satisfy all participation lower bounds. Finally, we execute Lines 17-19 of Algorithm 4 to check whether any user can attend more events within current constraints. Note that the execution order of Lines 4-5, Lines 6-7, and Lines 9-10 can be arbitrary, and has no influence on the competitive approximation ratio of Algorithm 7.

Algorithm 7. Multiple Atomic Operations Algorithm

Input: E, U, $\{\mu(u_i, e_j)\}$, P, $\{(\eta'_j, \xi'_j, t_j^{s'}, t_j^{t'})\}$, $\{(B'_i)\}$
Output: P'
1: **for** *each* u_i *providing* (B'_i) *and each* e_j *providing* $(\eta'_j, \xi'_j, t_j^{s'}, t_j^{t'})$ **do**
2: Dissociate events associated to u_i and users associated to e_j in P
3: **for** *each* e_j *providing* $(\eta'_j, \xi'_j, t_j^{s'}, t_j^{t'})$ **do**
4: **if** $\eta'_j < \eta_j$ **then**
5: Execute Lines 4-5 of Algorithm 3 to remove $\eta_j - \eta'_j$ users' plan whose utility scores are the smallest and obtain P'
6: **if** $t_j^{s'}$, $t_j^{t'}$ *cause conflicts* **then**
7: Execute Lines 1-4 of Algorithm 5 to remove plans in conflict and update P'
8: **for** *each* u_i *providing* (B'_i) **do**
9: **if** $D_i > B'_i$ **then**
10: Execute Lines 4-9 of Algorithm 6 to remove plans beyond the new budget and update P'
11: **for** *each* e_j *providing* $(\eta'_j, \xi'_j, t_j^{s'}, t_j^{t'})$ **do**
12: **if** $n_j < \xi'_j$ **then**
13: Execute Lines 4-16 of Algorithm 4 to satisfy ξ'_j and update P'
14: Execute Lines 17-19 of Algorithm 4 to check whether any user can attend more events within all constraints and update P'
15: Return P'

Example 4. We consider one last time the problem of Example 1, together with the global plan of Table 1. Assume that e_1 is modified so as to be held from 3:30 p.m. to 5:30 p.m., and simultaneously, u_1's travel budget is decreased from 18 to 8. If these changes are made, e_1 will have conflicts with e_2, but no conflicts with e_3, and u_1's travel budget will no longer be sufficient. According to Lines 1-2, we dissociate u_1 from e_1 in P. In this case, this is the only change necessary as all new constraints are satisfied.

4.6.1 Approximation Ratio

Based on the analysis of approximation ratios in the above 3 subsections, the approximation ratio of Algorithm 7 is: $\frac{\Psi'}{\Psi'_{OPT}} \geq \frac{1}{(n_j-\eta'_j+\xi'_j-n_j+uc_j+\xi_j-n_j+2))(Uc_{\max}-1)} \geq \frac{1}{(uc_j+\xi_j-n_j+2)(Uc_{\max}-1)}$,

TABLE 2
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

where uc_j is the number of individual plans exhibiting conflicts caused by new $t_j^{s'}$ and $t_j^{t'}$ values, n_j is the number of users assigned to event e_j in the original plan, $Uc_{\max} = \max_{i=1}^{n} Uc_i$, and Uc_i is the number of events that fall within a distance $B_i/2$ of $\mathbf{l}_{u_i}$.

4.6.2 Complexity Analysis

The computational complexity of Lines 1-2 is $O(1)$. The computational complexity of Lines 4-5 is $O(n_j \log n_j)$, where n_j is the number of users assigned to event e_j in P. The computational complexity of Lines 6-7 is $O(n_j)$. The computational complexity of Lines 9-10 is $O(m_i)$, where m_i is the number of events assigned to u_i in P. The computational complexity of Lines 11-13 is $O(m(mn_{\max}\log mn_{\max} + (\xi_j' - n_j)(Uc_{\max} + \log mn)))$, and the computational complexity of Lines 14 is $O(m(\xi_j' - n_j)(m + \eta_{max}))$, where $n_{\max} = \max_{j'=1}^{m} n_{j'}$, $\eta_{max} = \max_{j'=1}^{m} \eta_{j'}$. Thus, the total computational complexity of Algorithm 7 is $O(m^2 n_{max} \log mn_{\max} + m(\xi_j' - n_j)(Uc_{\max} + \log mn + m))$.

5 Performance Evaluation

Having introduced our proposed solutions, we now turn to an empirical evaluation of the associated algorithms, in terms of utility, computational cost and memory usage.

5.1 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 @ 2.00 GHz and 64 GB memory. The size of hard disc is 1TB. The memory costs reported here are calculated using system functions that monitor current memory usage. If the memory is not enough for the linear programming of GAP-based algorithm, we use virtual memory to solve this problem.

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 2 summarizes the parameters of the data. The conflict ratio is the proportion of events that have time conflicts.

TABLE 3
"Cut Out" Datasets

Factor	Setting
$\|E\|$	200, **500**, 1000, 2000, 5000
$\|U\|$	100, 500, 1000, 2000, 5000, 10000, 20000, **50000**

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 3 shows the various settings, with default values in bold.

5.2 Results of GEPC Problem

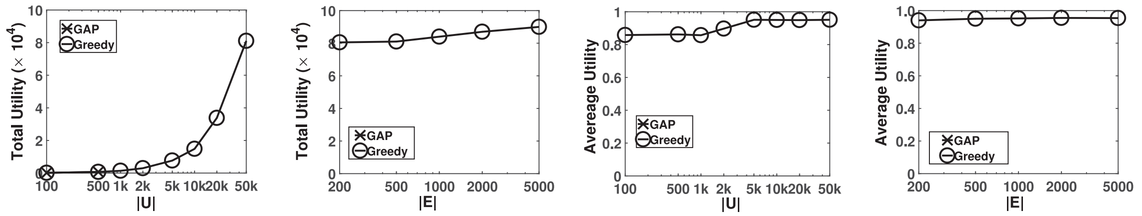
In this section, we test the two algorithms of the GEPC problem. As the GAP-based algorithm (denoted as GAP here) is indeed an extension of the algorithm to solve GAP, we use this algorithm as a baseline to compare the greedy-based algorithm against.

Table 4 shows the results on real datasets. Notice that the memory cost records in [5], we forgot to calculate the part spent by linear programming of the GAP-based algorithm. We refined the memory cost of GAP-based algorithm in this version. 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 extremely 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.

Figs. 2 and 3 report the scalability. To test the total utility, average utility, time cost and memory cost, we first set the number of events $|E| = 500$ and change the number of users $|U|$ from 100 to 50,000; we then set $|U| = 50,000$ and change $|E|$ from 200 to 5,000. Notice that when $|U| > 500$, the GAP-based algorithm cannot run due to the limitation of memory.

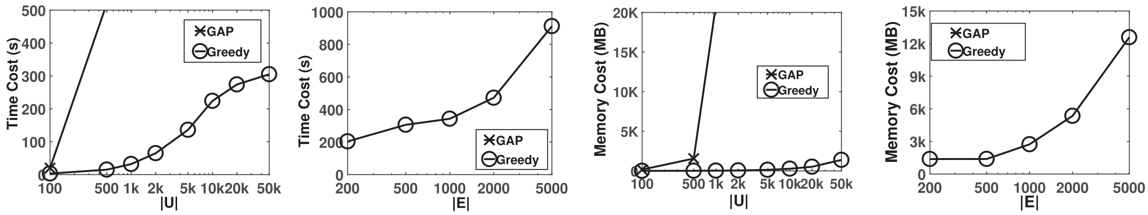
TABLE 4
Algorithms for GEPC on Real Datasets

	GAP			*Greedy*		
Datasets	Total Utility	Time Cost (s)	Memory Cost (MB)	Total Utility	Time Cost (s)	Memory Cost (MB)
Beijing	34306	0.49	4955.22	32095	0.01	0.70
Vancouver	5.903×10^7	768.83	48256.53	5.903×10^7	1.10	23.70
Auckland	1.62×10^6	14.04	7181.21	1.61×10^6	0.12	1.98
Singapore	6.93×10^6	202.71	21343.91	6.93×10^6	0.66	7.76



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

Fig. 2. Utilities of algorithms for GEPC.



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

Fig. 3. Time and memory cost of algorithms for GEPC.

From Figs. 2a and 2b, 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 3. From Figs. 3a and 3b, we can see that the time cost of the greedy-based algorithm is much smaller than that of the GAP-based algorithm. This confirms the complexity analysis of Section 3. From Fig. 3, we can see that the memory cost of the GAP-based algorithm is much larger than that of the greedy-based algorithm, since the linear programming part of the GAP-based algorithm spends large amount of memory. Fig. 3 does not show the memory cost of GAP-based algorithm, since all its memory cost is larger than 64 GB. These results further suggest that the greedy-based algorithm is more effective and efficient than the GAP-based algorithm. Moreover, we test the average utility of the planning, which is calculated as $\mathcal{U}_P/|P|$, where $\mathcal{U}$ is the total utility, $|P|$ is the number of matched pairs of events and users. From Figs. 2c and 2d, the average utility keeps larger than 0.84, which means that the algorithm can assign user to the events that has a large utility score. Moreover, we can see that the average utility almost keeps the same with the increase of $|U|$ and $|E|$. This means that the average utility has a strong scalability. In Fig. 3, we can see that the influence of $|E|$ is larger than $|U|$ for both time cost and memory cost. The GAP-based algorithm cannot run, but the greedy algorithm's scalability is better. When $|U|$ increases from 100 to 50k (500 times), the time cost increases from about 20s to 300s (about 15 times). When $|E|$ increases from 200 to 5000 (25 times), the time cost increases from about 200s to 900s (about 4.5 times). Same phenomenon can be found in the experiments of memory cost. However, for event based social network apps (such as Meetup etc.), users usually never attend to events held in other places that are far from where they work and live (such as other cities). According to our problem definition, we consider the planning of one day. Usually, just hundreds of events are held in one city in each day. Like Meetup, the number of events held all over the world in each day is about 10,000. Thus, the planning algorithms have few chance to compute too large numbers of events. According to Figs. 3a and 3c our algorithms can still run efficiently when $|U|$ becomes millions or even tens of millions, which can sufficiently match the number of users of each city. So our algorithms can work well in real applications.

5.3 Results of IEP Problem

In this section, we test the performance of the algorithms associated with the IEP problem, the incremental version of the GEPC problem. We consider the 4 basic atomic operations of Sections 4.1, 4.2, 4.3, and 4.4, denoted here as η-De, ξ-In, t^s-t^t, and B-De, respectively, as well as the situation when multiple changes are made simultaneously. As above, we test the algorithms on both real data and "cut out" data.

We test each atomic operation as follows. For each dataset, we randomly select either one event or one user, and change one of their constraints, i.e., decrease the event's η value, increase the event's ξ value, modify the event's t^s and/or t^t values, or decrease the user's budget B. To test multiple atomic operations in one run, we first randomly select one user, and randomly change his B_i and $\mu(u_i.e_j)$, and calculate a new plan using Algorithm 7 and methods introduced in Section 4.5. This experiment is denoted as *Mul-1U*. Then, we select one event, and randomly change its η_j, ξ_j, t_j^s, or t_j^t, or randomly add or delete one event from the event set. We calculate the corresponding new plan using Algorithm 7 and methods introduced in Section 4.5. This experiment is denoted as *Mul-1E*. Notice that Mul-1U and Mul-1E are conducted only on one event or user. Finally, we randomly select events or users from the event set and user set, and randomly select some changes and obtain new parameters of these new events and users. We calculate the new plan using Algorithm 7 and methods introduced in Section 4.5. This experiment is denoted as *Mul-All*. Notice that in Mul-All, all changes are randomly conducted on all selected events and users. As a comparison, we run the greedy algorithms again, denoted as *Re-Greedy*, and run each basic operations one by one, denoted as *1By1*.

We conduct the experiment 50 times and calculate the average total utility, time cost, and memory cost. Recall that

TABLE 5
Results of η-De on Real Datasets

Datasets	Beijing	Vancouver	Auckland	Singapore
Utility (η-De)	218115	5.11×10^7	1.58×10^6	7.31×10^6
Utility (Re-Greedy)	307850	4.69×10^7	1.74×10^6	7.02×10^6
Utility (Re-GAP)	330721	5.15×10^7	1.78×10^6	7.45×10^6
Time (s)	0.005	0.001	0.003	0.001
Memory (MB)	2.42	486.2	25.09	98.28

TABLE 6
Results of ξ-in on Real Datasets

Datasets	Beijing	Vancouver	Auckland	Singapore
Utility (ξ-In)	320295	5.24×10^7	1.84×10^6	7.88×10^6
Utility (Re-Greedy)	360905	4.79×10^7	1.23×10^6	1.03×10^7
Utility (Re-GAP)	364758	5.35×10^7	1.93×10^6	1.35×10^7
Time (s)	0.003	0.007	0.005	0.004
Memory (MB)	3.91	784.76	50.18	127.44

TABLE 7
Results of t^s-t^t on Real Datasets

Datasets	Beijing	Vancouver	Auckland	Singapore
Utility (t^s-t^t)	78793	5.44×10^7	2.07×10^6	6.72×10^6
Utility (Re-Greedy)	78039	4.71×10^7	1.63×10^6	7.11×10^6
Utility (Re-GAP)	80412	5.72×10^7	2.83×10^6	7.92×10^6
Time (s)	0.001	0.004	0.003	0.001
Memory (MB)	3.06	545.92	40.53	90.27

TABLE 8
Results of B-De on Real Datasets

Datasets	Beijing	Vancouver	Auckland	Singapore
Utility (B-De)	32155	6.01×10^7	1.59×10^6	6.86×10^6
Utility (Re-Greedy)	31999	5.83×10^7	1.59×10^6	6.85×10^6
Utility (Re-GAP)	32598	6.02×10^7	1.59×10^6	6.86×10^6
Time (s)	0.002	0.009	0.002	0.005
Memory (MB)	3.41	448.72	36.85	88.47

TABLE 9
Results of Mul-1U on Real Datasets

Datasets	Beijing	Vancouver	Auckland	Singapore
Utility (Mul-1U)	38986.5	4.14×10^7	1.38×10^6	7.33×10^6
Utility (Re-Greedy)	37724.5	4.03×10^7	1.37×10^6	7.21×10^6
Utility (Re-GAP)	37099.5	3.95×10^7	1.33×10^6	7.55×10^6
Time(s)	0.013	0.03	0.12	0.009

TABLE 10
Results of Mul-1E on Real Datasets

Datasets	Beijing	Vancouver	Auckland	Singapore
Utility (Mul-1E)	37577.4	4.14×10^7	1.50×10^6	7.36×10^6
Utility (Re-Greedy)	35306.9	4.05×10^7	1.18×10^6	7.21×10^6
Utility (Re-GAP)	38576.4	4.2×10^7	1.56×10^6	7.38×10^6
Time(s)	0.026	0.037	0.039	0.036

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 greedy algorithm after an atomic operation is performed on the EBSN platform (denoted as Re-GAP and Re-Greedy, respectively). When testing the scalability of IEP algorithms, we do not re-run the GAP-based algorithm. The reason is that the GAP-based algorithm cannot provide results over large sizes of datasets. According to the analysis in the above subsection, the total utility of Greedy-based algorithm is only a little smaller than the GAP-based algorithm. Thus, Re-Greedy can almost present the total utility of re-running the GEPC algorithms. Experimental results are shown in Tables 5, 6, 7, 8, 9, 10, and 11 and Figs. 4 and 7.

Tables 5, 6, 7, and 8 show the total utility, time cost and memory cost of 4 basic operations of IEP problem over real datasets. We compare the total utility obtained by 4 basic operations with that obtained from Re-GAP and Re-Greedy respectively. Overall, the total utilities obtained with IEP are almost the same as those obtained Re-Greedy. 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. This also indicate that our problem definition of IEP problem is more reasonable than just re-arrange all the users and events. The reason is that the IEP problem definition can keep "unhappy" users as few as possible, in the condition that the total utility of the platform has little loss. However, re-arrange the users and events can only obtain a little larger total utility, but cause a much larger "unhappiness". 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 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 total utility obtained from Re-GAP is almost the largest. Moreover, the time cost of IEP problem is mostly much smaller than Re-Greedy and Re-GAP, which means the 4 basic operation algorithms of the IEP problem more efficient than re-running the GEPC algorithm to calculate the changes.

Now, we analyze the performance of each basic operation respectively. For η-De, comparing Tables 4 and 5, most utility scores in Table 5 are smaller than those in Table 4. It is reasonable since when η decreases, fewer events and users are planned. While there also exists the case in which the utility scores in Table 5 are larger, such as in the dataset of Singapore. This is because when some users are "discarded" by event e_j (due to the decreased η_j), they are arranged to other events whose utility scores are larger than $\mu(u, e_j)$, according to their travel budgets. For ξ-In, comparing Tables 4 and 6, we can find most utility scores in Table 6 are larger than those in Table 4. It is reasonable since when ξ is decreased, more events and users may be planned. The case in which the utility scores in Table 6 are small is caused when the users forcibly re-arranged to event e_j (due to the increased ξ_j) prefer those events in the original plan. For t^s-t^t, comparing Table 7 and Table 5, if the utility scores in Table 7 are larger, it means

TABLE 11
Results of Mul-All on Real Datasets

Datasets	Beijing	Vancouver	Auckland	Singapore
Utility (Mul-All)	32032	6.01×10^7	1.58×10^6	6.72×10^6
Utility (Re-Greedy)	32723	5.93×10^7	1.72×10^6	7.03×10^6
Utility (Re-GAP)	31881	6.01×10^7	1.59×10^6	6.85×10^6
Time(s)	0.014	0.024	0.012	0.004
Memory(MB)	5.78	742.36	79.42	112.35

that the changed new time of events makes the number of conflict events become smaller. On the other hand, if the utility scores in Table 7 are smaller, it means that the changed new time of events makes the number of conflict events become larger. For B-De, comparing Table 8 and Table 5, we can find most utility scores in Table 5 are smaller than those in Table 4. It is reasonable since when B decreased, the number of events that a user can attend becomes smaller. A few scores are larger, which means that when user u_i gives up some event e_js, due to the decreased B_i, more users can attend these e_j, and the utility scores of these new attending users to e_js are larger than $\mu(u_i, e_j)$.

Tables 9, 10, and 11 show the performance of multiple operations in one run of IEP problem. The memory cost of Mul-1U, Mul-1E, and Mul-All is almost the same, so we just report the memory cost of Mul-All in Tabel 11. The memory cost of multiple algorithms in one run is almost the same with that of basic operations of IEP problem and the greedy algorithm of GEPC problem. Comparting the time cost of Mul-1U, Mul-1E, and Mul-All, we find that the time cost of Mul-1E is a little smaller. This is because the calculation of basic operation B-De is a little less efficient than the other three basic operations. Comparing the utility scores of Mul-1U, Mul-1E, and Mul-All with Re-Greedy and Re-GAP, we can find that the utility scores obtained from Mul-1U, Mul-1E, and Mul-All are not much smaller than, or event almost equals to those obtained from Re-Greedy and Re-GAP. This means that when multiple changes happen, the algorithms of IEP problem are still strong. Because the IEP problem does not lose much more total utility scores, and at the same time, they can keep the "unhappy" users as few as possible.

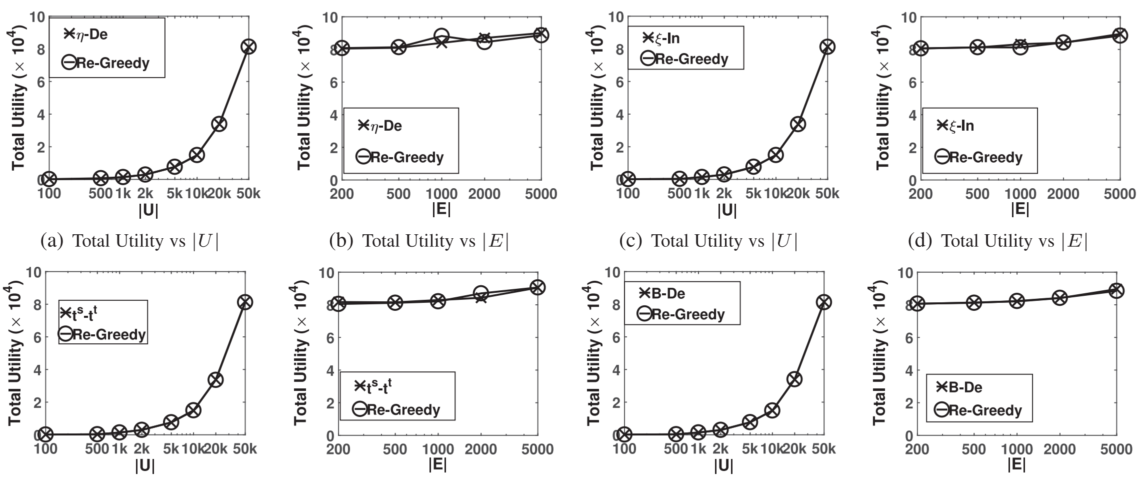
Figs. 4, 5, and 6 report the scalability of total utility of algorithms in IEP problem w.r.t $|U|$ and $|E|$. We can see that no matter basic operations and multiple operations in one run of IEP problem, the utility scores increase with the increase of $|U|$ and $|E|$. It is reasonable since more events and users means more planning, which makes the total utility score larger. The utility scores obtained by Re-Greedy are sometimes smaller, sometimes almost equal to, or sometimes larger than the IEP algorithms. This means that with the change of $|U|$ and $|E|$, IEP algorithms can always obtain a good planning result, while the utility scores obtained from Re-Greedy depends on the selection order of users of greedy-based algorithm.

From Fig. 7, we can see that the time cost increases with increasing values of $|U|$ and $|E|$. The computation time of the η-De algorithm is a little smaller than that of ξ-In and t^s-t^t, most likely due to the fact that its heap size is much smaller. The time cost of B-De is larger than that of the other three atomic operations, since deleting events from a user's plan may cause the participation lower bounds of several events to no longer be satisfied, which, in turn, leads to the need of executing Algorithm 4 several times. For the same reason, the time cost of the three multiple operations in one run is also larger than that of η-De, ξ-In and t^s-t^t. Notice that for the multiple operations in one run algorithms, we find that the total utility of Mul-All algorithm is almost the same with that of Re-Greedy and 1By1, but the time cost is much smaller than Re-Greedy and 1By1. This means that the Algorithm 7 can obtain almost the same total utility with Re-Greedy and 1By1, but it is more efficient, which verifies the effectiveness of Algorithm 7.

6 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.

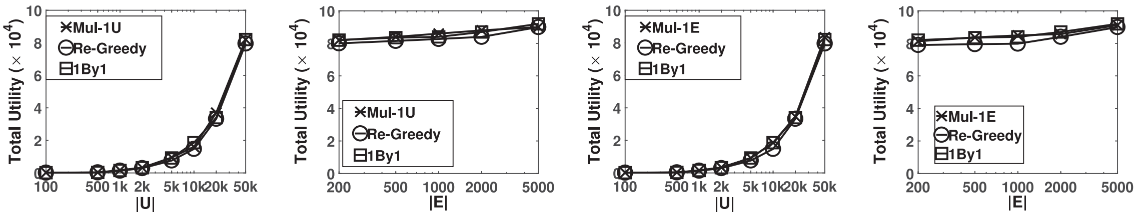
Studies on LBSNs. Recent years have seen an increase in popularity of Online To Offline (O2O) services. One of the hottest topics in O2O services is Location-Based Social Networks (LBSNs) [7], [8], [9], [10], [11], [12]. Although work based on LBSNs recommends or arranges users to events (or places, such as restaurants and shopping malls), it focuses on how to maximize users' individual utilities, i.e., on



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

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

Fig. 4. Performance of four basic operations of IEP.



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

Fig. 5. Performance of multiple operations on single event or user in one run.

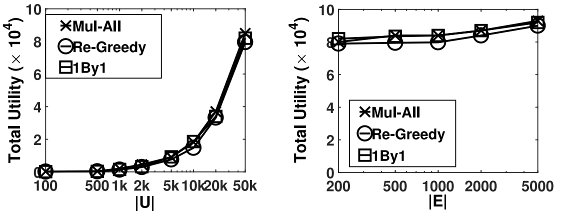
providing user-oriented recommendations. On the other hand, our work, like other works on EBSNs, focuses on maximizing the total utility of the whole system. In other words, EBSNs schedule all users and create global satisfiable plans.

Studies on EBSNs. The 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 global satisfiable planning. After that, different kinds of recommendation problems are studied over EBSN platforms. Purushotham et al studied the personalized group recommendation problem over LBSN/EBSN platforms [16]. Liao et al studied the problem of event recommendation considering the participant influence [17]. Macedo et al studied the context-aware event recommendation problem [18]. Zhang et al proposed a collective Bayesian Poisson factorization model to solve the cold-start local event recommendation problem [19]. These studies are all about the recommendation problem, instead of planning problem. In other words, they only recommend events/groups but do not care whether users would actually participate in these events/groups. 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 [20]. Essentially, the aim is to solve the influence maximization problem [21], together with the team formation problem [22]. We focus on a planning providing maximized total utility.

The work on the Social Event Organization problem, which assigns 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.

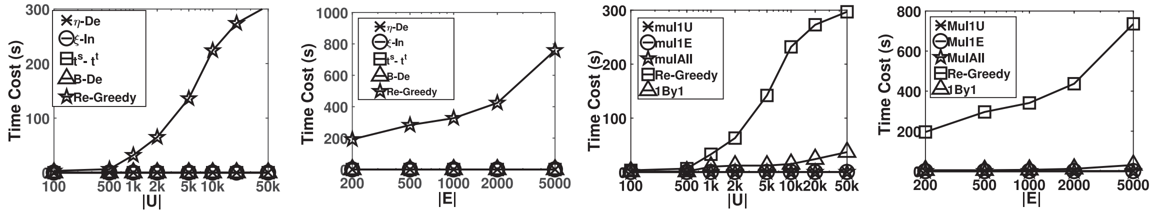
Studies on GAP Variants. Although our problem can be written into integer programming like an operations research problem, we focus on designing algorithms with bounded approximation ratios from the perspective of complexity and algorithmics, since no general operations research methods can provide solutions 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 [23] for a survey). The Bottleneck GAP [24] 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 [25]. The Non-linear Capacity Constraint GAP treats the capacity constraint of machines as a function [26]. Finally, in the Stochastic GAP, the jobs/machines are not consistently given, but follow a random function or are in a sequence [27], [28]. 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 [29]. 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 [30], in that they both study



(a) Utility vs $|U|$ (b) Utility vs $|E|$

Fig. 6. Multiple operations on random events/users in one run.



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

Fig. 7. Time cost of IEP.

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.

7 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, 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. To further accelerate the algorithms to solve the planning problems, in the future studies, parallel algorithms could also be considered. Moreover, the social relationships among users also have an influence on the planning of EBSN platforms. In future work, considering the social relationships among users is also a good direction of the EBSN research.

Acknowledgments

Yurong Cheng is supported by the National Key Research and Development Program of China (Grant No. 2018YFB1004402), the NSFC (Grant No. U1811262) and the China Postdoctoral Science General Program Foundation (No. 2018M631358). Ye Yuan is supported by the NSFC (Grant No. 61572119 and 61622202) and the Fundamental Research Funds for the Central Universities (Grant No. N181605012). Guoren Wang is supported by the NSFC (Grant No. 61672145, 61732003 and 61729201). Lei Chen is supported by the Hong Kong RGC GRF Project 16207617, CRF Project C1031-18G, the National Science Foundation of China (NSFC) under Grant No. 61729201, the Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Hong Kong ITC ITF grants ITS/391/15FX and ITS/212/16FP, Didi-HKUST joint research lab project, the Microsoft Research Asia Collaborative Research Grant, the Wechat Research Grant, and the Webank Research 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 *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 1032–1040.

[2] J. She, Y. Tong, L. Chen, and C. C. Cao, "Conflict-aware event-participant arrangement," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 735–746.

[3] K. Li, W. Lu, S. Bhagat, L. V. Lakshmanan, and C. Yu, "On social event organization," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1206–1215.

[4] J. She, Y. Tong, and L. Chen, "Utility-aware social event-participant planning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1629–1643.

[5] Y. Cheng, Y. Yuan, L. Chen, C. Giraud-Carrier, and G. Wang, "Complex event-participant planning and its incremental variant," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 859–870.

[6] S. A. Plotkin, D. B. Shmoys, and É. Tardos, "Fast approximation algorithms for fractional packing and covering problems," *Math. Operations Res.*, vol. 20, no. 2, pp. 257–301, 1995.

[7] E. Minkov, B. Charrow, J. Ledlie, S. Teller, and T. Jaakkola, "Collaborative future event recommendation," in *Proc. 19th ACM Int. Conf. Inf. Knowl. Manage.*, 2010, pp. 819–828.

[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 *Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 1655–1660.

[9] H. Khrouf and R. Troncy, "Hybrid event recommendation using linked data and user diversity," in *Proc. 7th ACM Conf. Recommender Syst.*, 2013, pp. 185–192.

[10] Y.-C. Sun and C. C. Chen, "A novel social event recommendation method based on social and collaborative friendships," in *Social Informatics*. Berlin, Germany: 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," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 3, pp. 1259–1273, Jun. 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 *Proc. 20th Int. Conf. Advances Geographic Inf. Syst.*, 2012, pp. 209–218.

[13] W. Zhang, J. Wang, and W. Feng, “Combining latent factor model with location features for event-based group recommendation,” in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 910–918.

[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 *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2014, pp. 425–434.

[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 *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 567–578.

[16] S. Purushotham and C.-C. J. Kuo, “Personalized group recommender systems for location-and event-based social networks,” *ACM Trans. Spatial Algorithms Syst. Regular Papers SIGSPATIAL Paper*, vol. 2, no. 4, 2016, Art. no. 16.

[17] Y. Liao, W. Lam, L. Bing, and X. Shen, “Joint modeling of participant influence and latent topics for recommendation in event-based social networks,” *ACM Trans. Inf. Syst.*, vol. 36, no. 3, pp. 29:1–29:31, 2018.

[18] A. Q. de Macedo, L. B. Marinho, and R. L. T. Santos, “Context-aware event recommendation in event-based social networks,” in *Proc. 9th ACM Conf. Recommender Syst.*, 2015, pp. 123–130.

[19] W. Zhang and J. Wang, “A collective Bayesian poisson factorization model for cold-start local event recommendation,” in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1455–1464.

[20] K. Feng, G. Cong, S. S. Bhowmick, and S. Ma, “In search of influential event organizers in online social networks,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 63–74.

[21] D. Kempe, J. Kleinberg, and É. Tardos, “Maximizing the spread of influence through a social network,” in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 137–146.

[22] T. Lappas, K. Liu, and E. Terzi, “Finding a team of experts in social networks,” in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 467–476.

[23] T. Öncan, “A survey of the generalized assignment problem and its applications,” *INFOR: Inf. Syst. Oper. Res.*, vol. 45, no. 3, pp. 123–141, 2007.

[24] J. Mazzola and A. Neebe, “Bottleneck generalized assignment problems,” *Eng. Costs Prod. Econ.*, vol. 14, no. 1, pp. 61–65. 1988.

[25] M. Laguna, J. P. Kelly, J. Gonz ález-Velarde, and F. Glover, “Tabu search for the multilevel generalized assignment problem,” *Eur. J. Oper. Res.*, vol. 82, no. 1, pp. 176–189, 1995.

[26] J. B. Mazzola, A. W. Neebe, and C. V. Dunn, “Production planning of a flexible manufacturing system in a material requirements planning environment,” *Int. J. Flexible Manuf. Syst.*, vol. 1, no. 2, pp. 115–142, 1989.

[27] D. R. Spoerl and R. K. Wood, “A stochastic generalized assignment problem,” Naval Postgraduate School, Working Paper, 2003.

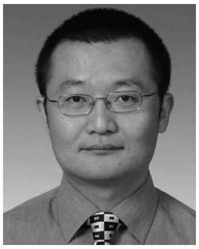
[28] C. Derman, G. J. Lieberman, and S. M. Ross, “A sequential stochastic assignment problem,” *Manag. Sci.*, vol. 18, no. 7, pp. 349–463, 1972.

[29] S. O. Krumke and C. Thielen, “The generalized assignment problem with minimum quantities,” *Eur. J. Oper. Res.*, vol. 228, no. 1, pp. 46–55, 2013.

[30] N. Gupta, L. Kot, S. Roy, G. Bender, J. Gehrke, and C. Koch, “Entangled queries: Enabling declarative data-driven coordination,” *ACM Trans. Database Syst.*, vol. 37, no. 3, 2012, Art. no. 21.

Yurong Cheng received the BS and PhD degrees from the College of Computer Science and Engineering of Northeastern University, China, in 2012 and 2017, respectively. Currently, she is a postdoc in computer science at the Beijing Institute of Technology. Her main research interests include queries and analysis over uncertain graphs, knowledge bases, social networks, and spatio-temporal databases. She is a member of the IEEE.

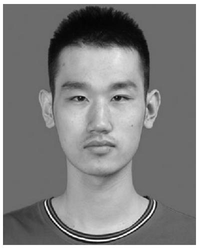
Ye Yuan received the BS, MS, and PhD degrees in computer science from Northeastern University, China, in 2004, 2007, and 2011, respectively. He is now a professor with the College of Information Science and Engineering, Northeastern University. His research interests include graph databases, probabilistic databases, data privacy-preserving, and cloud computing. He is a member o the IEEE.



Lei Chen received the BS degree in computer science and engineering from Tianjin University, China, in 1994, the MA degree from the Asian Institute of Technology, Bangkok, Thailand, in 1997, and the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is currently an associate professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include crowdsourcing over social media, social media analysis, probabilistic and uncertain databases, and privacy-preserved data publishing. He is a member of the IEEE.

Christophe Giraud-Carrier received the BS, MS and PhD degrees in computer science from Brigham Young University, in 1991, 1993, and 1994, respectively. He is currently an associate professor with the Department of Computer Science, Brigham Young University. He was a senior manager at ELCA, a Swiss IT services company, from 2001 to 2004, and a senior lecturer with the Department of Computer Science, University of Bristol, from 1994 to 2001. His research interests include machine learning, data mining, computational health science, and metalearning.

Guoren Wang received the BSc, MSc, and PhD degrees in computer science from Northeastern University, China, in 1988, 1991, and 1996, respectively. Currently, he is a professor with the Department of Computer Science, Beijing Institute of Technology, China. His research interests include XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management.



Boyang Li received the BS degree from the College of Computer Science and Engineering of Northeastern University, China, in 2015. Currently, he is working toward the PhD degree at Northeastern University. His main research interests include social networks, spatio-temporal databases, and machine learning.