Hindawi
Complexity
Volume 2019, Article ID 1532013, 12 pages
https://doi.org/10.1155/2019/1532013

Research Article

Incremental Bilateral Preference Stable Planning over Event Based Social Networks

Boyang Li,[1] **Yurong Cheng**,[2] **Guoren Wang**,[2] **and Yongjiao Sun**[1]

[1]*School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China*
[2]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China*

Correspondence should be addressed to Yurong Cheng; yrcheng@bit.edu.cn

Received 24 January 2019; Accepted 2 April 2019; Published 16 April 2019

Guest Editor: Jiajie Xu

In recent years, Event Based Social Networks (EBSNs) appear in people's daily life and are becoming increasing popular. In EBSNs, one typical task is to make personalized plans for users. Existing studies only consider the preference of users. They make plans by selecting interesting events for users. However, for organizers of events, they also would like more high-quality users to participate in their events, which may make the events more exciting. Existing studies are user-centered and ignore the requirement of organizers. What is more, organizers are allowed to modify their events dynamically before they are held. The platforms should be able to dynamically adjust the schedules of users. Therefore, we identify a new Incremental Bilateral Preference Stable Planning (IBPSP) problem over EBSNs and propose several solutions to deal with different situations. We conduct extensive experiments to verify the efficiency and effectiveness of the proposed algorithms.

1. Introduction

In recent years, Event Based Social Networks (EBSNs) [1] have experienced rapid development and attracted much attention from both industry and academia fields. EBSNs, such as Meetup (https://www.meetup.com/) and Plancast (http://plancast.com/), link the online social groups and the offline events. Taken Meetup(https://www.meetup.com/) as an example, it has attracted more than 16 million users with more than 300 thousand events held each month.

In EBSNs, one typical task is to select suitable events and make personalized plans for users to participate in according to the labels that users select as their interested points. Therefore, we can evaluate the interest of users to events based on the similarity of labels of users and events, called utility score [2–6]. The higher the utility score is, the more interested the user is to the event. Besides, the spatial distance is another import factor in personalized planning; the total travel cost of a plan should not be more than the travel budget of the user. Moreover, a user may participate in more than one events; the platform should guarantee that the time periods of events that the user participate in are not overlapped. Therefore, the goal is to make plans for all the users to maximize the utility score where the travel cost of each user cannot be more than his budget and events in the same plan do not conflict with each other.

Example 1. Suppose we have 5 users and 3 events, and details are shown in Table 1. The first row describes users and their travel budget, and the first column gives the events and the upper user bound. The last column is the time period when each event is held, and the time period of e_2 and e_3 are overlapped. It means that users cannot participate in both e_1 and e_3. The other columns are the utility scores of each user. The locations of users and events are shown in Figure 1. The locations are described in a 2D space and the distance between any two locations is Euclidean distance. For a user, he starts from his current location, participates in each event in his plan one by one, and goes back to his start location. For example, the purple lines are the plan of u_2. He starts from $(8, 2)$, participates in e_3 and e_1, and go back to $(8, 2)$ at last. The travel cost is 16.9, which is no more than the travel budget. The total utility of all users whose plans are shown in Figure 1 is 3.7.

Table 1: Users' utility to each event.

	$u_1(40)$	$u_2(40)$	$u_3(10)$	$u_4(52)$	$u_5(18)$	Time
$e_1(3)$	0.6	0.3	0.4	0.5	0.2	12:30-14:00
$e_2(1)$	0.1	0.4	0.8	0.6	0.7	10:00-12:00
$e_3(2)$	0.1	0.7	0.9	0.2	0.9	09:00-12:00

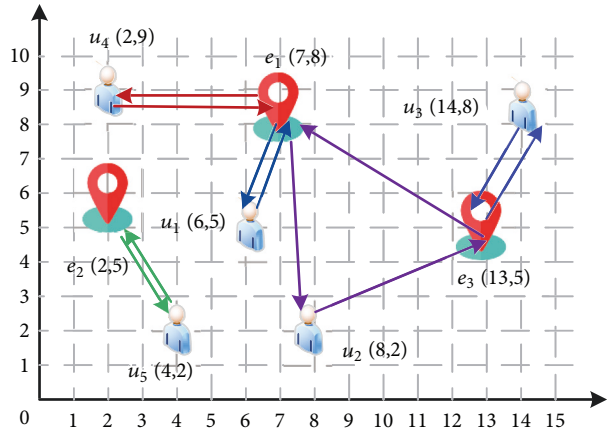


Figure 1: The location of users and events.

Table 2: Events' utility to each user.

	$e_1(3)$	$e_2(1)$	$e_3(2)$
$u_1(12)$	0.3	0.4	0.8
$u_2(40)$	0.1	0.8	0.7
$u_3(10)$	0.6	0.4	0.6
$u_4(52)$	0.2	0.7	0.5
$u_5(10)$	0.9	0.5	0.9

Existing studies [2–6] only focus on the utility of users. However, the event organizers also have preference towards users for holding the event successfully. For example, organizers prefer users who are more influential in the social network to participate in their events, which may improve the quality of the events. Therefore, the personalized planning problem over EBSNs should be a bilateral planning problem among users and events. The existing techniques cannot give plans that satisfy both users and event organizers. For Example 1, we give the utility scores of events to users in Table 2. u_1, u_2, and u_4 are the worst three users of e_1. e_2 prefers u_4 than u_5. u_2 and u_3 are not the best users of e_3. Through analysis, these methods cannot consider the preference of both users and events at the same time.

Based on the aforementioned example, it is more reasonable to consider bilateral preference of users and events than to only consider the preference of users in real applications. Moreover, the events may be modified for some reasons; the plans for users should be updated quickly once it happens. Therefore, we formally define a new personalized planning problem in EBSNs. Different from existing studies, this problem is a bilateral planning problem which considers the preference of both users and events instead of only considering the preference of user and the attributes of user and events change dynamically. In summary, to solve this problem, we make the following contributions:

(i) We identify a new Incremental Bilateral Preference Stable Planning (IBPSP) problem over EBSNs, which aims to dynamically make plans satisfying both users and event organizers.

(ii) We present solutions to give stable plans in different situations as the attributes of users and events change.

(iii) We verify the effectiveness and efficiency of the proposed methods with extensive experiments on real dataset.

The rest of this paper is organized as follows. Section 2 briefly introduces some basic concepts and formalizes the problem. In Section 3, we give the solutions for different conditions. Extensive experiments are conducted on a series of real-life datasets and the performance is evaluated in Section 4. Section 5 discusses the related works. Finally, we give our conclusion in Section 6.

2. Problem Statement

In this section, we introduce some basic concepts of EBSNs firstly. And then, we formally define the Incremental Bilateral Preference Stable Planning (IBPSP) problem. Symbols used in our paper are shown in Table 3.

Given an event based social network with n users and m events, we denote U as the user set and E as the event set. Each user $u_i \in U$ is described by a 2-tuple $\langle \mathbf{l}_{u_i}, B_i \rangle$, where $\mathbf{l}_{u_i}$ is the location of u_i in a 2D space, and B_i is the travel cost budget. Each event $e_j \in E$ is described by a 4-tuple $\langle \mathbf{l}_{e_j}, \eta_j, t_j^s, t_j^e \rangle$, where $\mathbf{l}_{e_j}$ is the location of e_j in a 2D space, η_j is the largest

Table 3: Summary of symbols used in our paper.

Symbol	Description
E	The event set
e	An event
U	The user set
u	A user
$\mathbf{l}_{u_i}$	Location of u_i
B_i	The travel cost budget of u_i
$\mathbf{l}_{e_j}$	Location of e_j
η_j	The largest number of participants of e_j
t_j^s	The start time of e_j
t_j^e	The end time of e_j
D_i	The travel cost of u_i
$p_u(u_i, e_j)$	The utility score of u_i to e_j
$p_e(e_j, u_i)$	The utility score of e_j to u_i
$PU(u_i)$	The preference rank of u_i to all the events
$PE(e_j)$	The preference rank of e_j to all the users
$\mathscr{P}$	The global plan
$\mathscr{P}(u_i)$	The set of events that u_i will participate in
$\mathscr{P}(e_j)$	The set of users who will participate in e_j

number of participants, and t_j^s and t_j^e are the start time and end time of e_j. For user u_i, the utility score to event e_j is denoted as $p_u(u_i, e_j)$, and the utility score of e_j to u_i is denoted as $p_e(e_j, u_i)$, where $p_u(u_i, e_j), p_e(e_j, u_i) \in [0, 1]$. We denote the preference rank of u_i is $PE(u_i)$, and the preference rank of e_j is $PE(e_j)$. The order of utility scores is strict; that is, no two utility scores are equal.

A global plan $\mathscr{P}$ is a set of plans for all the users in the platform, denoted as $\mathscr{P} = \{(u_i, e_j) \mid 1 \le i \le n,\ 1 \le j \le m\}$. Events in each user's plan cannot conflict with each other. In other words, for two events e_k and e_h in the same user's plan, if e_k starts before e_h, e_k should end earlier than e_h. A user may participate in several events according to his plan; the travel cost is the sum of the distance that he travels. The distance between users and events is Euclidean distance.

Definition 2 (blocking pair). For a user u_i and an event e_j, where $(u_i, e_j) \notin \mathscr{P}$ and the travel budget is enough to participate in e_j, if u_i prefers e_j to at least one event in his plan and e_j prefers u_i to at least one user that will participate in e_j, then (u_i, e_j) is a *blocking pair*.

Due to the existing of travel budget, the travel cost may be larger than the budget of u_i after the platform arrangements u_i participate in e_j. In this case, (u_i, e_j) is not a blocking pair.

Back to Example 1, according to the plans in Figure 1 and the utility scores in Tables 1 and 2, (u_4, e_2) is a blocking pair. The reason is that u_4 prefers e_2 to e_1, and e_2 prefers u_4 to u_5. (u_5, e_3) is not a blocking pair. The reason is that the travel budget of u_5 is not enough to travel to e_3, though they prefer each other. Figure 2 shows planning results without the blocking pairs.

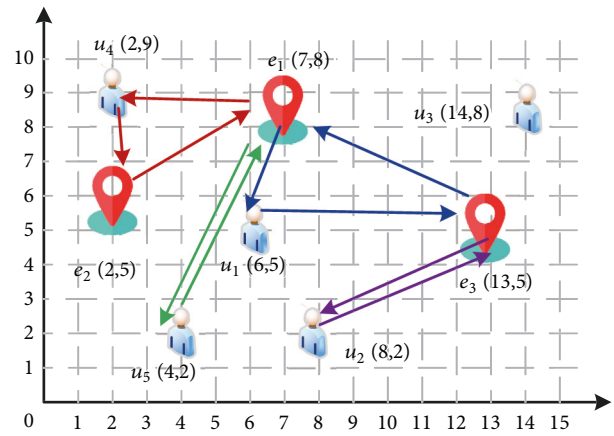


Figure 2: A planning without blocking pairs.

After defining the above concepts, we next introduce the IBPSP problem as follows.

Definition 3 (IBPSP problem). Given a set of users U, a set of events E, the preference rank PE, and PU over an EBSN platform, which allows the attributes of users and events change dynamically, the IBPSP problem is to find a global plan $\mathscr{P}$ for all the users such that the following constraints are satisfied:

(i) Events in the same user's plan do not conflict with each other, where $\forall_i\ \forall e_k \ne e_h \in \mathscr{P}(u_i),\ t_{e_k}^s < t_{e_h}^s \Longrightarrow t_{e_k}^e < t_{e_h}^e$.

(ii) The travel cost of users is not more than their travel budget, where $\forall_i\ D_i \le B_i$.

Input: PE, PU, E, U, $\mathscr{P}_{old}$, u_i
Output: $\mathscr{P}$
1 $\mathscr{P} \longleftarrow \mathscr{P}_{old}$
2 $E_{ac} \longleftarrow \emptyset$
3 **While**($D_i > B_i$)
4 $e_j \longleftarrow$ event with the worst utility score in $\mathscr{P}(u_i)$
5 Put e_j into E_{ac}, remove e_j, u_i from $\mathscr{P}(u_i)$, $\mathscr{P}(e_j)$
6 **EndWhile**
7 **Return** Update_event_plan(PE, PU, E, U, $\mathscr{P}_{old}$, E_{ac})

Algorithm 1: Budget decreasing algorithm.

(iii) The number of participants is not more than the upper bound of events, where $\forall_j \, |\{(u_i, e_j) \in \mathscr{P}, \ 1 \leq i \leq n\}| \leq \eta_j$.

(iv) The global plan does not contain any blocking pairs.

(v) The plan is updated when the attributes change.

In the IBPSP problem, both users and events do not prefer other events and users that are out of the plan, it will reach a stable state. From the perspective of economics, in the existence of competitive relations, the stable state is the most reasonable and the most consistent with the law of development of things [7]. In practice, information of events and profiles of users are subject to change. Thus, a reasonable event planning system should support incremental updates.

3. Solutions to IBPSP Problem

In this section, we first introduce which attributes of users and events may change. And then, we propose several solutions to deal with the changes and update the plans that are made based on the former attributes.

3.1. Changes Caused by Users and Events

Travel Budget of Users. Users can modify their travel budget according to their own intentions. For example, if a user has enough to enjoy the events, he may increase the travel budget to travel further or participate in more events. A user may also travel a shorter distance or even cancel the travel plan due to the bad weather or fall ill.

Participation Upper Bounds of Events. The maximum capacity of the events is limited by a number of factors. For example, if the event is more popular than expected, the organizers may choose a larger venue and increase the upper bound of participations. The opposite, however, is also possible. If a tourism interest group organizes a visit to a scenic spot, the organizer may reduce the number of people due to the restricted traffic conditions of the scenic spot.

Start Times and Times of Events. The organizer may modify the start or end time if the venue is occupied during the time period when the event will be held or some important guests cannot participate in the event on time.

Events Are Added or Cancelled. In EBSNs, new events will be added at any time. When a new event is added, the platform needs to arrange suitable users to participate in this event. The planned events can also be cancelled for all kinds of reasons. When it happens, the platform needs to arrange users to participate in other events that are suitable for them.

3.2. Solutions. In this section, suppose the platform has made plans for users. And then, the changes which are introduced above come. We provide solutions on how to solve the changes when the changes happen.

3.2.1. B_i Is Decreased. When the travel budget of a user decreases, it may lead to that he cannot participate in some events that are planned by the platform for him. The platform needs to remove some events from his plan until the total travel cost is no more than the budget. What is more, the plans also need to be adjusted to make sure that there are no blocking pairs in the new plan.

The pseudocode is illustrated in Algorithm 1. When the travel budget of u_i changes, we first initialize a set E_{ac} as an empty set (Line 2). E_{ac} stores the events that are affected by the change. While the travel cost is larger than the travel budget, we find an event e_j in $\mathscr{P}(u_i)$ whose utility score is the lowest in $\mathrm{PU}(u_i)$, remove it from $\mathscr{P}(u_i)$, and put it into E_{ac} (Lines 3-6). After this step, we make sure that the travel cost is not greater than the new travel budget. Since u_i cannot participate in some events, there may be blocking pairs in the plans of these events. Finally, Algorithm 2 is called to update the plans of events in E_{ac}.

Algorithm 2 terminates until E_{ac} is empty and returns a new global plan. In each loop, we pop an event e_j from E_{ac} (Line 3) and start to enumerate each user u_i whose utility is not greater than the lowest utility in current plan $\mathscr{P}(e_j)$ until the number of participants equals η_j (Lines 4-5). If e_j conflicts with e_k that has a higher utility in (u_j), then we start again from the next user (Lines 7-9). Otherwise, we remove the events whose utility is less than e_j from $\mathscr{P}(u_i)$ and put them in a conflict set and they are also put into E_{ac} (Lines 10-12). Then we can add e_j to $\mathscr{P}(u_j)$ (Line 13). If the travel cost

Input: PE, PU, E, U, $\mathscr{P}_{old}$, E_{ac}
Output: $\mathscr{P}$
1 $\mathscr{P} \longleftarrow \mathscr{P}_{old}$
2 **While**($E_{ac} \neq \emptyset$)
3 Pop e_j from E_{ac}
4 $u_h \longleftarrow$ user with the worst utility in $\mathscr{P}(e_j)$
5 **Foreach** (u_i after u_h in $PE(e_j) \bigwedge$ the number of participants does not exceed η_j)
6 $conflict\_events = \phi$
7 **If**($\exists e_k \in \mathscr{P}(u_j)$ that conflicts with e_j)
8 **continue**
9 **EndIf**
10 **Foreach**($e_k \in \mathscr{P}(u_j)$)
11 Remove e_k from $\mathscr{P}(u_i)$, put e_k into $conflict\_events$, put e_k into E_{ac}
12 **EndFor**
13 Put e_j into $\mathscr{P}(u_j)$
14 **While**($D_i > B_i$)
15 Remove the worst event e_k from $\mathscr{P}(u_i)$
16 **If**(e_j is removed)
17 **break**
18 **EndIf**
19 Put e_k into $conflict\_events$, put e_k into E_{ac}
20 **EndWhile**
21 **If**(e_j has been removed from $\mathscr{P}(u_i)$)
22 **Foreach**($e_k \in conflict\_events$)
23 Put e_k back to $\mathscr{P}(u_i)$, remove e_k from E_{ac}
24 **EndFor**
25 **EndIf**
26 **EndFor**
27 **Endwhile**
28 **Return** $\mathscr{P}$

Algorithm 2: Update_event_plan.

is greater than the travel budget, we remove some events and put them into the conflict set and E_{ac} (Lines 14-20). If e_j is also removed in the former step, it means e_j is too fat for u_j to participate in, and $\mathscr{P}(u_i)$ is rehabilitated (Lines 21-25).

Example 4. Based on the planning of Figure 2, suppose that the travel budget of u_1 decreases to 5. Then, e_3 is removed from $\mathscr{P}(u_1)$ and put into E_{ac}. We pop e_3 from E_{ac}, and u_3 can participate in it; a new stable planning is constructed.

3.2.2. η_j Is Decreased. When η decreases, the number of participants may exceeds the new upper bound in current plans. The platform needs to remove some participants and adjust the plans of these participants.

The pseudocode is illustrated in Algorithm 3. When η_j changes, we first initialize a set U_{ac} as an empty set (Line 2). U_{ac} stores the users that are affected by the change. While the number of participants is larger than the new upper bound, we find a user u_i in $\mathscr{P}(e_j)$ whose utility score is the lowest in $PE(e_j)$, remove him from $\mathscr{P}(e_j)$, and put it into U_{ac} (Lines 3-6). After this step, we make sure that the new η is satisfied. Finally, Algorithm 4 is called to update the plans of users in U_{ac}.

Algorithm 4 terminates until U_{ac} is empty and return a new global plan. In each loop, we pop a user u_i from U_{ac} (Line 3) and start to enumerate each event e_j whose utility are not greater than the lowest utility in current plan $\mathscr{P}(u_i)$ (Lines 4-5). If e_j conflicts with e_k that has a higher utility in $\mathscr{P}(u_i)$, then we start again from the next event (Lines 6-8). Otherwise, add e_j, u_i into $\mathscr{P}(u_i)$, $\mathscr{P}(e_j)$ (Line 9). If the number of participants exceeds η_j, we remove the user with the lowest utility in $\mathscr{P}(e_j)$ and put the user into U_{ac} if he is not u_i (Lines 9-18).Finally the new global plan $\mathscr{P}$ is returned (Line 21).

Example 5. Based on the planning of Figure 2, suppose that the participant bound of e_3 decreases to 1. Then, e_3 is removed from the planning of u_2, and u_2 is put into U_{ac}. We pop u_2 from U_{ac}, then e_2 is removed from the planning of u_4 and u_4 is put into U_{ac}. u_4 cannot participate in other events; a new stable planning is constructed.

3.2.3. t_j^s or t_j^e Is Changed. If the start or end time changes, it may cause many conflicts in the original plan $\mathscr{P}$. There are four kinds of cases that the global plan needs to be updated: (1) the new event conflicts with events that with higher utility

Input: $PE, PU, E, U, \mathcal{P}_{old}, e_j$
Output: $\mathcal{P}$
1 $\mathcal{P} \longleftarrow \mathcal{P}_{old}$
2 $U_{ac} \longleftarrow \emptyset$
3 **While**$(|\mathcal{P}(e_j)| > \eta_j)$
4 $u_i \longleftarrow$ user with the worst utility score in $\mathcal{P}(e_j)$
5 Put u_i into U_{ac}, remove e_j, u_i from $\mathcal{P}(u_i)$, $\mathcal{P}(e_j)$
6 **Endwhile**
7 **Return** Update_user_plan($PE, PU, E, U, \mathcal{P}_{old}, U_{ac}$)

Algorithm 3: η decreasing algorithm.

Input: $PE, PU, E, U, \mathcal{P}_{old}, U_{ac}$
Output: $\mathcal{P}$
1 $\mathcal{P} \longleftarrow \mathcal{P}_{old}$
2 **While**$(U_{ac} \neq \emptyset)$
3 Pop u_i from U_{ac}
4 $e_k \longleftarrow$ event with the worst utility in $\mathcal{P}(u_i)$
5 **Foreach** (e_j after e_k in $PU(u_i)$)
6 **If**($\exists e_h \in \mathcal{P}(u_i)$ conflicts with e_j)
7 **break**
8 **EndIf**
9 Add e_j, u_i into $\mathcal{P}(u_i)$, $\mathcal{P}(e_j)$
10 **If**(the number of participants exceeds η_j)
11 **If**(u_i is the worst user in $\mathcal{P}(e_j)$)
12 Remove e_j, u_i from $\mathcal{P}(u_i)$, $\mathcal{P}(e_j)$
13 **break**;
14 **ElseIf**
15 Remove the worst user u_k from $\mathcal{P}(e_j)$
16 Put u_k into U_{ac}
17 **EndIf**
18 **EndIf**
19 **EndFor**
20 **Endwhile**
21 **Return** $\mathcal{P}$

Algorithm 4: Update_user_plan.

scores in users' plans, (2) the event does not conflict with events with higher utility scores in the same plan, however, some events with lower utility scores that conflict with it, (3) the original event conflicts with some events in the users' plan but the conflicts do not exist after the change, and (4) some events out of users' conflict with the original event but the conflicts do not exist after the change.

The pseudocode is illustrated in Algorithm 5. We initialize U_{ac} and E_{ac} at first (Line 1). And then, we find all the users whose plans are affected by the change, remove e_j from their plans, and put them into U_{ac} (Lines 3-5). The plan of e_j is cleared and needs to be replanned (Line 6). For users in U_{ac} and events in E_{ac}, we call Algorithms 2 and 3 to update their plans (Lines 7-12).

3.2.4. The Other Situations. We discuss how to solve the other situations, based on the above algorithms.

B_i Is Increased. When the travel budget of a user increases, the user can participate in more events. We can update his plan by putting him into U_{ac} and calling Algorithm 4. However, we need to make a small change to Algorithm 4. We cannot enumerate events starting from whose utility is not greater than the lowest utility in current plan; it will result in blocking pairs. The user may participate in events with higher utility than events in the current plan, because his travel budget may be large enough to travel to the locations of these events. Therefore, we enumerate events starting from whose utility is the largest and add them into the plan if possible.

Input: PE, PU, E, U, $\mathcal{P}_{old}$, e_j
Output: $\mathcal{P}$
1 $U_{ac} = \phi, E_{ac} = \emptyset$
2 $\mathcal{P} \longleftarrow \mathcal{P}_{old}$
3 **Foreach**($u_i \in \mathcal{P}(e_j)$)
4 Remove e_j from $\mathcal{P}(u_i)$, put u_i into U_{ac}
5 **EndFor**
6 Empty $\mathcal{P}(e_j)$, put e_j into E_{ac}
7 **If**($U_{ac} \neq \emptyset$)
8 $\mathcal{P} \longleftarrow$Update start users($PE$, PU, E, U, $\mathcal{P}$, U_{ac})
9 **EndIf**
10 **If**($E_{ac} \neq \emptyset$)
11 $\mathcal{P} \longleftarrow$Update start events($PE$, PU, E, U, $\mathcal{P}$, E_{ac})
12 **EndIf**
13 **Return** $\mathcal{P}$

Algorithm 5: t_j^s or t_j^e changing algorithm.

Input: PE, PU, E, U, $\mathcal{P}_{old}$, e_j, u_i
Output: $\mathcal{P}$
1 $U_{ac} = \phi, E_{ac} = \emptyset$
2 $\mathcal{P} \longleftarrow \mathcal{P}_{old}$
3 **While**($D_i > B_i$)
4 $e_j \longleftarrow$ event with the worst utility score in $\mathcal{P}(u_i)$
5 Put e_j into E_{ac}, remove e_j, u_i from $\mathcal{P}(u_i)$, $\mathcal{P}(e_j)$
6 **Endwhile**
7 **Foreach**($u_i \in \mathcal{P}(e_j)$)
8 Remove e_j from $\mathcal{P}(u_i)$, put u_i into U_{ac}
9 **EndFor**
10 Empty $\mathcal{P}(e_j)$, put e_j into E_{ac}
11 **If**($U_{ac} \neq \emptyset$)
12 $\mathcal{P} \longleftarrow$Update start users($PE$, PU, E, U, $\mathcal{P}$, U_{ac})
13 **EndIf**
14 **If**($E_{ac} \neq \emptyset$)
15 $\mathcal{P} \longleftarrow$Update start events($PE$, PU, E, U, $\mathcal{P}$, E_{ac})
16 **EndIf**
17 **Return** $\mathcal{P}$

Algorithm 6: Multiple changes in one run.

η_j Is Increased. When η increases, the event can accommodate more participants; we then put it into E_{ac} and call Algorithm 2 to update the plans. Note that if the number of participants is less than the original upper bound, the plans need not to be update.

3.2.5. Multiple Changes in One Run. We consider a more complex case in which multiple changes come at the same time. As introduced above, all the changes can be solved by two basic algorithms. The main idea is that we find the events and users whose plans need to be updated and conduct Algorithms 2 and 4 to update their plans.

The pseudocode is illustrated in Algorithm 6. We initialize U_{ac} and E_{ac} at first (Line 1). While the travel cost is larger than the travel budget, we find an event e_j in $\mathcal{P}(u_i)$ whose utility score is the lowest in PU(u_i), remove it from $\mathcal{P}(u_i)$, and put it into E_{ac} (Lines 3-6). And then, we find all the users whose plans are affected by the change, remove e_j from their plans, and put them into U_{ac} (Lines 7-9). The plan of e_j is cleared and needs to be replanned (Line 10). For users in U_{ac} and events in E_{ac}, we call Algorithms 2 and 3 to update their plans (Lines 11-17).

4. Performance Evaluation

In this section, we provide an empirical evaluation of our proposed algorithms, including the experimental environment, algorithm running time, memory cost, and the total sum of utility scores of users.

4.1. Experiment Environment and Dataset. We conduct our algorithms over a real-life dataset, the Meetup dataset [1],

Table 4: Real datasets.

City	$\|U\|$	$\|E\|$	Mean of η	Conflict ratio
Beijing	113	16	50	0.25
Auckland	569	37	50	0.25
Hawaii	2967	817	50	0.25
Hong Kong	3528	1324	50	0.25
Singapore	9893	4257	50	0.25
Vancouver	16095	11536	50	0.25

Table 5: Synthetic datasets.

	Value
$\|U\|$	200, 500, 1000, **5000**
$\|E\|$	200, 500, **1000**, 5000

which is a popular event based social network. The dataset records the locations of users and the tags that indicate the interesting points of the users. It also records the locations of events where they are held. The events are held by interesting groups; therefore, the tags of events are the tags of whom hold them. The utility scores are calculated as the method in [1]. We extract six datasets of different cities according to the longitude and latitude and generate other parameters, summarized in Table 4. We also use a synthetic dataset to test the scalability of our algorithms, shown in Table 5.

The algorithms are implemented in Visual C++ 2017, and the experiments are conducted in a Windows 10 machine with Intel(R) Core(TM) i5-6500 3.20GHz CPU and 16GB main memory. When the changes come, we treat all the events as new added and get a global plan as the contrast experiment, named Re-Plan. In each experiment, we repeat 10 times and report the average results.

4.2. Results on Real Dataset. In this section, we test the performance of our algorithms, denoted as B-De, η-De, $t^s - t^e$, and *Mul*, on Meetup datasets. For each algorithm, we randomly select several users to decrease their travel budget and several events to decrease their upper bound of participations one by one. For the multiple case, we randomly select some users and events of the three kinds of changes.

The experiment results are shown in Tables 6–9. The utility scores of B-De are the same as Re-Plan. The reason is that B-De conducts the same algorithm to update the plans of events as Re-Plan. In other cases, the algorithms are similar with Re-Plan. η-De is better than Re-Plan on Hawaii and Vancouver, $t^s - t^e$ is better on Hawaii and Hong Kong, and *Mul* is better on Singapore and Vancouver. In IBPSP problem, the stable plan is not unique; we can find different stable plans through different algorithms. Though the utility scores are not the same, the plans are still stable. In all the algorithms, the running time is much smaller than the Re-Plan; the reason is that the incremental algorithms only adjust the plans of users and events that are affected by the changes, but the Re-Plan makes plans for all the users and events over again. The running time increases as the data size increases. The incremental algorithms cost the similar memory as Re-Plan due to the fact that they all need memory to store the preference and the final plans. Mul costs a bit more memory than others due to size of U_{ac} and E_{ac} is larger than others.

4.3. Results on Synthetic Datasets. In this section, we test the scalability of the proposed algorithms on synthetic datasets. We first set $|U|$ as 5000 and change $|E|$ from 200 to 5000. Then, we set $|E|$ as 1000 and change $|U|$ from 200 to 5000.

The results of efficiency are shown in Figure 3. The total utility increases as the data size increases. The total utility of B-De is still as same as Re-Plan as shown in Figures 3(a) and 3(b). η-De is better than Re-Plan as $|E|$ increases. $t^s - t^e$ and Mul are better than Re-Plan as $|U|$ and $|E|$ increase.

The results of effectiveness are show in Figure 4. Both the running time and memory cost increase as $|E|$ and $|U|$ increase. $t^s - t^e$ and *Mul* cost more time than the others, the reason is that the plans adjusted in the two algorithms are more than the others. B-De runs faster than η-De when $|E|$ increases but runs the slowest when $|U|$ increases. The reason is that B-De needs to enumerate users while adjusting plans of events; the increasing of $|U|$ affects the running time. To the contrary, η-De needs to enumerate events while adjusting plans of users; the running time increases faster as $|E|$ increases.

5. Related Work

In this section, we review related works from two categories, event based social networks (EBSNs) and stable matching problem.

Studies on EBSNs. By analyzing Meetup and Plancust, X. Liu et al. [1] firstly proposed the concept of EBSN and received more and more attention [8–17]. On the event recommendation problem, Zhang et al. [9] predicted whether a user will participate in some events by learning the historical events that the user participated in. Cao et al. [18] scored users based on location information, attributes, and relations and built a Bayesian model to recommend events. Wang et al. [19] proposed a context-enhanced method to recommend events to users. However, the method relies too much on social information and geographical information; the lack and inaccuracy of data will seriously affect the accuracy. On the event organization problem, Shen et al. [3, 4] considered

Table 6: Results of B-De on real datasets.

City	Re-Plan			B-De		
	Utility	Time(s)	Memory(MB)	Utility	Time(s)	Memory(MB)
Beijing	218.94	0.81	0.78	218.94	0.01	0.78
Auckland	2443.73	4.62	2.71	2443.73	0.07	2.69
Hawaii	12469.67	56.19	166.58	12469.67	1.81	160.73
Hong Kong	18898.82	293.87	366.07	18898.82	6.35	370.36
Singapore	52753.51	11442.20	2902.35	52753.51	29.79	2900.78
Vancouver	70732.40	86492.48	12908.62	70732.40	103.67	13001.34

Table 7: Results of η-De on real datasets.

City	Re-Plan			η-De		
	Utility	Time(s)	Memory(MB)	Utility	Time(s)	Memory(MB)
Beijing	215.77	0.77	0.78	208.78	0.01	0.78
Auckland	2394.26	3.65	2.89	2244.57	0.08	2.75
Hawaii	12489.57	60.78	170.86	12604.53	2.35	177.75
Hong Kong	18796.87	301.78	359.17	18702.48	7.56	388.47
Singapore	52766.12	11304.18	2801.87	52689.48	26.88	2857.44
Vancouver	70655.31	86677.14	12944.63	70689.14	99.57	12903.78

Table 8: Results of $t^s - t^e$ on real datasets.

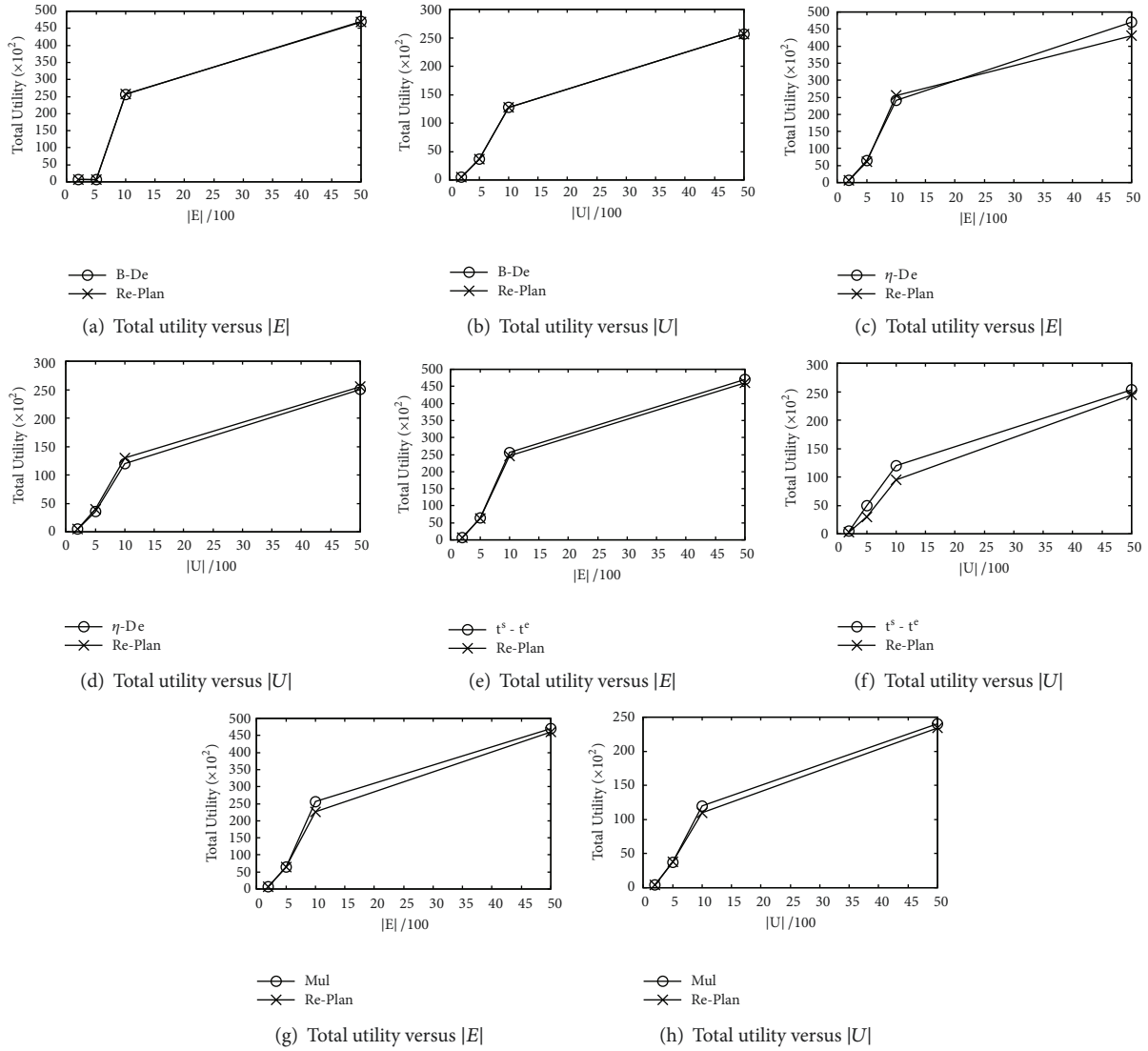
City	Re-Plan			$t^s - t^e$		
	Utility	Time(s)	Memory(MB)	Utility	Time(s)	Memory(MB)
Beijing	221.07	0.97	0.78	211.77	0.01	0.79
Auckland	2507.47	5.21	2.67	2487.45	0.08	3.04
Hawaii	11847.70	55.78	150.43	12078.8	2.02	184.48
Hong Kong	19752.41	321.47	397.15	19987.56	7.59	394.51
Singapore	52078.86	11648.67	2784.45	51978.07	33.48	2875.43
Vancouver	71097.98	87124.35	13478.74	70988.46	112.78	13998.47

Table 9: Results of *Mul* on real datasets.

City	Re-Plan			*Mul*		
	Utility	Time(s)	Memory(MB)	Utility	Time(s)	Memory(MB)
Beijing	202.17	1.21	0.81	198.45	0.01	0.78
Auckland	2378.47	5.14	2.07	2245.87	0.07	2.88
Hawaii	11987.78	60.74	152.78	11624.47	1.75	187.57
Hong Kong	18048.14	318.45	401.65	17956.24	5.33	407.98
Singapore	52417.64	11620.78	2788.79	52745.78	38.78	2998.47
Vancouver	70574.78	87144.54	13407.55	70741.34	121.52	14002.46

the conflict between events and made arrangements for users that avoid conflicts. Authors of [6] took into account the users' travel budget and extended the problem from simply matching the users with events to scheduling reasonable travels for users to participate in events. Cheng et al. [2] further considered the lower bound of the number of participants in the events and the issue of dynamic planning. The existing works only considered the preference of users, none of them studied on the bilateral preference stable planning problem. The authors of [20] studied a kind of stable matching problem over EBSNs; however, our work studies an incremental problem which is different from it.

Studies on Stable Matching Problem. The stable marriage (SM) problem was first proposed by Gale and Shapley [21] in 1962. Suppose a set of man and a set of woman with the same size; a stable marriage for these men and women is that everyone is matched with a partner and there are no such two couples



(a) Total utility versus $|E|$ (b) Total utility versus $|U|$ (c) Total utility versus $|E|$

(d) Total utility versus $|U|$ (e) Total utility versus $|E|$ (f) Total utility versus $|U|$

(g) Total utility versus $|E|$ (h) Total utility versus $|U|$

Figure 3: Efficiency on synthetic datasets.

that the man and the woman prefer each other than his/her partner. Reference [22] proved that the stable marriage always exists. There are some extensions of stable marriage problem, SM with incomplete preference lists, SM with preference lists with ties, and SM with incomplete preference lists with ties. In SM with incomplete preference lists, each person's preference list may be incomplete. Authors of [23] partitioned the set of men (women) into two sets: one is the set of men (women) who have partners in all stable matching, and the other is the set of men (women) who are single in all stable matching. In SM with preference lists with ties, one can include two or more persons with the same preference in a tie. Authors of [24] found a weakly stable matching in polynomial time. The last extension allows both incompleteness and ties in preference list. There series of approximation algorithm [25–28] and the approximation ratio reached 1.8 in [28]. The definition of stable matching problem is different from ours and the attributes are not changed dynamically; the existing studies cannot be applied.

6. Conclusions

In this paper, we define the Incremental Bilateral Preference Stable Planning (IBPSP) problem, which dynamically make plans for all the users to participate in suitable events. In this problem, we consider the upper bound of participants, event time conflicts, and the travel budget of users and dynamically update the plans when the constraints change. We propose several algorithms to update the plans when the changes come. We verify the effectiveness, efficiency, and scalability of the proposed methods through extensive experiments on both read and synthetic datasets.

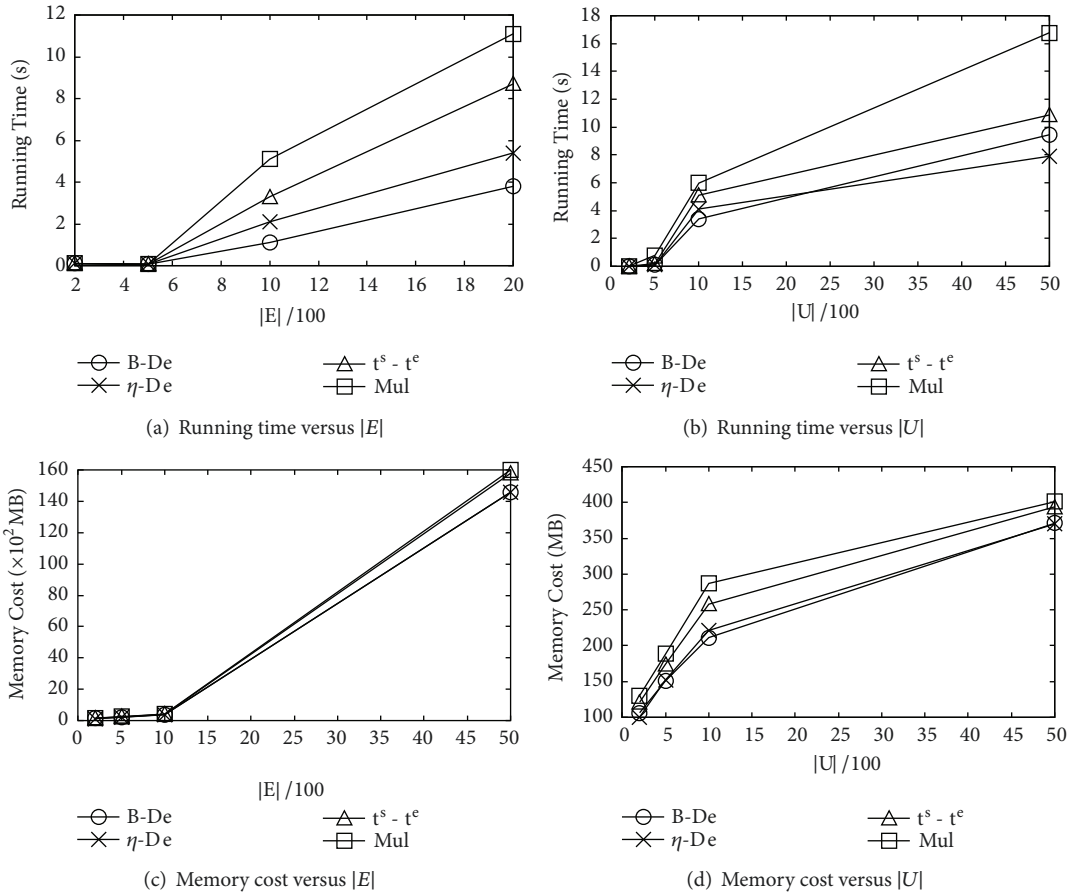


Figure 4: Effectiveness on synthetic datasets.

Data Availability

The Meetup data supporting the finding of this study are from previously reported studies and datasets, which have been cited as [1]. The processed data are available from the corresponding author upon request.

Conflicts of Interest

The authors declared that they have no conflicts of interest to this work

Acknowledgments

The work is supported by the National Key R&D Program of China (Grant no. 2016YFC1401900), the National Natural Science Foundation of China (Grants nos. 61332006, 61332014, 61328202, U1401256, 61572119, 61622202, 61572121, and 61702086), the Fundamental Research Funds for the Central Universities (Grants nos. N150402005, N171604007, and N171904007), the Natural Science Foundation of Liaoning Province (Grant no. 20170520164), and the China Postdoctoral Science Foundation (Grant no. 2018M631806).

References

[1] X. Liu, Q. He, Y. Tian, W. Lee, J. McPherson, and J. Han, "Event-based social networks: linking the online and offline social worlds," in *Proceedings of the 18th ACM SIGKDD International Conference*, pp. 1032–1040, Beijing, China, August 2012.

[2] Y. Cheng, Y. Yuan, L. Chen, C. Giraud-Carrier, and G. Wang, "Complex event-participant planning & its incremental variant," in *Proceedings of the 33rd IEEE International Conference on Data Engineering, ICDE 2017*, pp. 859–870, April 2017.

[3] J. She, Y. Tong, L. Chen, and C. C. Cao, "Conflict-aware event-participant arrangement," in *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE)*, pp. 735–746, Seoul, South Korea, April 2015.

[4] J. She, Y. Tong, L. Chen, and C. C. Cao, "Conflict-aware event-participant arrangement and its variant for online setting," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 9, pp. 2281–2295, 2016.

[5] K. Li, W. Lu, S. Bhagat, L. V. Lakshmanan, and C. Yu, "On social event organization," in *Proceedings of the 20th ACM SIGKDD international conference*, pp. 1206–1215, August 2014.

[6] J. She, Y. Tong, and L. Chen, "Utility-aware social event-participant planning," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2015*, pp. 1629–1643, June 2015.

[7] S. Qiao, N. Han, J. Zhou, R.-H. Li, C. Jin, and L. A. Gutierrez, "SocialMix: A familiarity-based and preference-aware location suggestion approach," *Engineering Applications of Artificial Intelligence*, vol. 68, pp. 192–204, 2018.

[8] K. Feng, G. Cong, S. S. Bhowmick, and S. Ma, "In search of influential event organizers in online social networks," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD 2014*, pp. 63–74, USA, June 2014.

[9] X. Zhang, J. Zhao, and G. Cao, "Who will attend? - predicting event attendance in event-based social network," in *Proceedings of the 16th IEEE International Conference on Mobile Data Management, MDM 2015*, pp. 74–83, USA, June 2015.

[10] W. Zhang, J. Wang, and W. Feng, "Combining latent factor model with location features for event-based group recommendation," in *Proceedings of the 19th ACM SIGKDD International Conference*, p. 910, August 2013.

[11] 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 *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp 2014*, pp. 425–434, 2014.

[12] T.-A. N. Pham, X. Li, G. Cong, and Z. Zhang, "A general graph-based model for recommendation in event-based social networks," in *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering*, pp. 567–578, 2015.

[13] Xiaoyang Liu, Chao Liu, and Xiaoping Zeng, "Online Social Network Emergency Public Event Information Propagation and Nonlinear Mathematical Modeling," *Complexity*, vol. 2017, Article ID 5857372, 7 pages, 2017.

[14] W. Hu, H. Wang, C. Peng, H. Liang, and B. Du, "An event detection method for social networks based on link prediction," *Information Systems*, vol. 71, pp. 16–26, 2017.

[15] J. She, Y. Tong, L. Chen, and T. Song, "Feedback-Aware social event-participant arrangement," in *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data, SIGMOD 2017*, pp. 851–865, May 2017.

[16] Y. Tong, J. She, and R. Meng, "Bottleneck-aware arrangement over event-based social networks: the max-min approach," *World Wide Web*, vol. 19, no. 6, pp. 1151–1177, 2016.

[17] G. Li, Y. Liu, B. Ribeiro, and H. Ding, "On group popularity prediction in event-based social networks," in *Proceedings of the 12th International AAAI Conference on Web and Social Media*, pp. 644–647, June 2018.

[18] J. Cao, Z. Zhu, L. Shi, B. Liu, and Z. Ma, "Multi-feature based event recommendation in event-based social network," *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, pp. 618–633, 2018.

[19] Z. Wang, P. He, L. Shou, K. Chen, S. Wu, and G. Chen, "Toward the new item problem: context-enhanced event recommendation in event-based social networks," in *Proceedings of the European Conference on Information Retrieval*, vol. 9022 of *Lecture Notes in Computer Science*, pp. 333–338, Springer International Publishing, 2015.

[20] Y. Cheng, G. Wang, B. Li, and Y. Yuan, "Bilateral preference stable planning over event based social networks," *Journal of Software*, vol. 30, pp. 573–588, 2019.

[21] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[22] A. E. Roth, "Deferred acceptance algorithms: history, theory, practice, and open questions," *International Journal of Game Theory*, vol. 36, no. 3-4, pp. 537–569, 2008.

[23] D. Gale and M. Sotomayor, "Some remarks on the stable matching problem," *Discrete Applied Mathematics: The Journal of Combinatorial Algorithms, Informatics and Computational Sciences*, vol. 11, no. 3, pp. 223–232, 1985.

[24] R. W. Irving, "Stable marriage and indifference," *Discrete Applied Mathematics: The Journal of Combinatorial Algorithms, Informatics and Computational Sciences*, vol. 48, no. 3, pp. 261–272, 1994.

[25] K. Iwama, S. Miyazaki, and K. Okamoto, "A (2-clog N/N)-approximation algorithm for the stable marriage problem," in *Proceedings of the 2004 Scandinavian Workshop on Algorithm Theory*, vol. 3111 of *Lecture Notes in Computer Science*, pp. 349–361, Springer, Berlin, 2004.

[26] K. Iwama, S. Miyazaki, and N. Yamauchi, "A 2-c$(1/\sqrt{n})$)-approximation algorithm for the stable marriage problem," in *Proceedings of the 2005 International Symposium on Algorithms and Computation*, vol. 3827 of *Lecture Notes in Computer Science*, pp. 902–914, Springer, Berlin, 2005.

[27] K. Iwama, S. Miyazaki, and N. Yamauchi, "A 1.875-approximation algorithm for the stable marriage problem," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pp. 288–297, ACM, New Orleans, La, USA.

[28] S. Khuller, "Problems column," *ACM Transactions on Algorithms*, vol. 3, no. 3, 2007.