This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

Conflict-Aware Event-Participant Arrangement and Its Variant for Online Setting

Jieying She, Yongxin Tong, *Member, IEEE*, Lei Chen, *Member, IEEE*, and Caleb Chen Cao

Abstract—With the rapid development of Web 2.0 and Online To Offline (O2O) marketing model, various *online event-based social networks* (EBSNs) are getting popular. An important task of EBSNs is to facilitate the most satisfactory event-participant arrangement for both sides, i.e. events enroll more participants and participants are arranged with personally interesting events. Existing approaches usually focus on the arrangement of each single event to a set of potential users, or ignore the conflicts between different events, which leads to infeasible or redundant arrangements. In this paper, to address the shortcomings of existing approaches, we first identify a more general and useful event-participant arrangement problem, called *Global Event-participant Arrangement with Conflict and Capacity* ($GEACC$) problem, focusing on the conflicts of different events and making event-participant arrangements in a global view. We find that the GEACC problem is NP-hard due to the conflicts among events. Thus, we design two approximation algorithms with provable approximation ratios and an exact algorithm with pruning technique to address this problem. In addition, we propose an online setting of GEACC, called OnlineGEACC, which is also practical in real-world scenarios. We further design an online algorithm with provable performance guarantee. Finally, we verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.

Index Terms—Event-Based Social Networks, Event Arrangement

1 Introduction

The prevalence of Web 2.0 and Online To Offline (O2O) marketing model has led to the boom of various *online event-based social networks* (EBSNs) [1]. For example, Groupon[1] collects group purchase events and recommends these group discounts to users, and Meetup[2] receives information on recruitment of attendees in offline events, such as gatherings, sports activities, etc., and sends such information to users. Such EBSNs facilitate organizing social events and ease the recruitment of group activity participants. Note that "participant" and "user" are used interchangeably in this paper.

However, most existing EBSNs only provide a public/open event information sharing platform [1], where strategic organization and global event-participant arrangement are absent. Imagine the following scenario. Bob is a sport enthusiast and usually attends sports activities organized on Meetup. In a Saturday evening, Bob faces a dilemma since Meetup recommends him three conflicting sport activities on Sunday: a hiking trip from 8:00 a.m. to 12:00 p.m., a badminton game from 9:00 a.m. to 11:00 a.m., and a basketball game from 11:30 a.m. to 1:30 p.m. on a basketball court that is one-hour away by car from the badminton stadium. Though Bob is interested in all three sports, he can only attend at most one of them. In fact, many users usually encounter the same problem: *they have to confront with a confusing choice from many conflicting events*.

Besides resolving conflicts of events, it is appealing to have an event-participant arrangement strategy that globally optimizes the benefits of both event organizers and users, e.g. for organizing a carnival or a film festival. Particularly, [2] [3] [4] [5] are the recent studies on such event arrangement problem in static scenarios, a.k.a offline scenarios, i.e. information of events and users is fully given.

In addition to the above static setting, the online setting of EBSN platforms, which has not yet been studied, is also important. That is, *users can dynamically login EBSN platforms at any time and register for events in a first-come, first-served way*. Since an EBSN platform cannot know in advance whether users who are more interested in a certain event will come later or not, it has to make decisions solely for the current user. Imagine the following scenario. Carol is music fan and wants to attend a concert. At the time she logs in, only a pop and a jazz music concert are available with one spot left, respectively. Carol decides to join the pop music concert. Two days later, David, who is only interested in pop music, logs in the platform. However, since the pop music concert has no quota left, David cannot find any interesting event to attend.

The aforementioned scenario depicts another practical real-world setting of event arrangement on EBSNs: with users logging in at any time, how to make proper arrangement for users solely based on the information available so that to maximize the overall satisfaction of both sides. As offline algorithms cannot solve the online setting since full information of events and users is no longer available, *the challenge of the online setting is to design a quality-guaranteed algorithm to make online event arrangement on EBSNs*.

To further illustrate the motivation, we go through a toy example as follows.

- *J. She, L. Chen and C.C. Cao are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: {jshe, leichen,caochen}@cse.ust.hk.*
- *Y. Tong is with the SKLSDE Lab, School of Computer Science and Engineering, Beihang University, PR China. E-mail: yxtong@buaa.edu.cn.*

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

TABLE 1: Interestingness and Conflicts between Events and Users

	u_1 (3)	u_2 (1)	u_3 (1)	u_4 (2)	u_5 (3)	Conflicts
v_1 (5)	**0.93**	0.43	**0.84**	0.64	**0.65**	v_3
v_2 (3)	0	0.35	0.19	**0.21**	**0.4**	NA
v_3 (2)	0.86	**0.57**	0.78	**0.79**	0.68	v_1

Example 1. *Suppose we have three events $v_1 - v_3$ and five users $u_1 - u_5$ in an EBSN. We assume that each event/user is associated with a profile, which is represented by a multi-dimensional attribute vector and can show its preferences towards the participants/events. We can then calculate a user's interest in an event based on the similarity of their attributes. TABLE 1 presents the interestingness values between each pair of event and user, as well as the conflicts between events. In addition, each event/user has a capacity. For an event, the capacity is the maximum number of participants, and for a user, the capacity is the maximum number of assigned events. In this example, $v_1 - v_3$ have capacities of 5, 3, and 2, and $u_1 - u_5$ have capacities of 3, 1, 1, 2, and 3 respectively (in brackets). Events v_1 and v_3 are conflicting. Notice that u_1 is the most interested user in both v_1 and v_3. However, u_1 can only be assigned to one of v_1 and v_3. Existing methods do not consider conflicts of events and thus yield an infeasible arrangement. A feasible and also optimal arrangement that we want to achieve is shown in bold font in TABLE 1, whose total interestingness values add up to 4.39.*

For the online setting, suppose the users arrive at the platform in the order of $u_5, u_4, \cdots, u_1$. In this case, one possible arrangement is that u_5 chooses to attend v_2 and v_3 when s/he arrives as these two events are the most interesting to her/him and are non-conflicting, u_4 then attends v_2 and v_3, u_3 attends v_1, u_2 can only attend v_1 but not v_3 since v_3 has been out of quota at the time u_2 arrives, and u_1 finally attends v_1, whose total interestingness values add up to 4.28, which is less than the offline optimal one.

As discussed above, we propose a new event-participant arrangement strategy, called Global Event-participant Arrangement with Conflict and Capacity (GEACC). Specifically, given a set of events and a set of users, each one is associated with a capacity to its type, which is the allowable maximum number for its counterpart, and some events are conflicting. Users have preferences to different events, each of which is measured as a non-negative "interestingness value". The GEACC problem is to find an event-participant arrangement, such that the sum of the interestingness values over all the assigned pairs of event and user is maximized, while the capacity and conflicting constraints are satisfied. We also study an online version of the GEACC problem, called OnlineGEACC, which is identical to GEACC except that users in OnlineGEACC arrive at the EBSN platform one by one in an online way and we have to make decision for a newly arrived user solely based on the information that is already available before the next user arrives.

In summary, we make the following contributions. Note that different to our preliminary work [3], we make new contributions by proposing the online setting of GEACC and also an online algorithm.

- We identify a new event-participant arrangement problem with extensive real-life applications, and propose a formal definition of Global Event-participant Arrangement with Conflict and Capacity (GEACC) problem.
- We prove that GEACC is NP-hard and design two approximation algorithms, MinCostFlow-GEACC and Greedy-GEACC. MinCostFlow-GEACC has $\frac{1}{\alpha}$ approximation ratio, where α is the maximum of users' capacities, but is not scalable due to its quartic time complexity. We further develop a more efficient greedy-based approximation algorithm, which guarantees $\frac{1}{1+\alpha}$ worst-case approximation ratio. We also present an exact algorithm that utilizes an effective pruning rule to reduce redundant search space.
- We propose an online setting of the GEACC problem and design an online algorithm, OnlineGreedy-GEACC, with provable competitive ratio.
- We verify the effectiveness and efficiency of the proposed offline and online methods with extensive experiments on real and synthetic datasets.

TABLE 2: Summary of Symbol Notations

Notation	Description
V (U)	The set of events (The set of users)
$\boldsymbol{l}_v$ ($\boldsymbol{l}_u$)	Attribute vector of v (u)
d	Dimension of $\boldsymbol{l}_v/\boldsymbol{l}_u$
T	Maximum value of an element of $\boldsymbol{l}_v/\boldsymbol{l}_u$
c_v (c_u)	Capacity of v (u)
$sim(v,u)$	Interestingness value between v and u
CF	A set of conflicting event pairs
M	An arrangement for the events and the users

The rest of the paper is organized as follows. We define GEACC and OnlineGEACC in Section 2. Section 3 presents two offline approximation algorithms. An exact solution with pruning is presented in Section 4. Section 5 presents an online algorithm with competitive ratio analysis. Extensive experiment results are discussed in Section 6. We review related works in Section 7 and conclude in Section 8.

2 Problem Statement

In this section, we first formally define GEACC, the offline scenario. We then formulate the online version of GEACC, OnlineGEACC.

2.1 Offline Scenario of GEACC

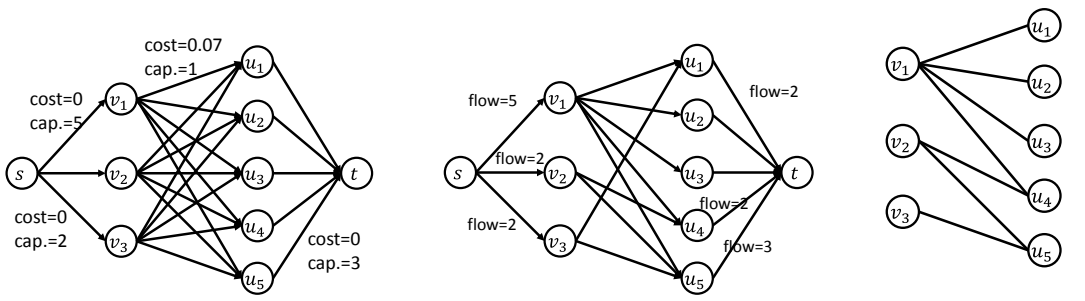
We first introduce two basic concepts, event and user, and then formally define conflicting event pairs.

Definition 1 (Event). *An event is defined as $v =< \boldsymbol{l}_v, c_v >$, where $\boldsymbol{l}_v =< l_v^1, l_v^2, ..., l_v^d >$ with $l_v^i \in [0,T], \forall 1 \leq i \leq d$ is a d-dimensional attribute vector, the values of whose elements are in the range of $[0,T]$, and c_v is the capacity of the event, namely the maximum number of attendees of the event.*

Definition 2 (User). *A user is defined as $u =< \boldsymbol{l}_u, c_u >$, where $\boldsymbol{l}_u =< l_u^1, l_u^2, ..., l_u^d >$ with $l_u^i \in [0,T], \forall 1 \leq i \leq d$ is a d-dimensional attribute vector, the values of whose elements are in the range of $[0,T]$, and c_u is the capacity of the user, namely the maximum number of arranged events for the user.*

Basically, two events are conflicting if users cannot attend them at the same time. For example, their timetables may overlap, or their locations may be too far away for users who attend one of them to catch the other one. And we have the following definition.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering



(a) Flow graph constructed (b) Minimum cost flow result (c) Final arrangement

Fig. 1: Illustrated example of MinCostFlow-GEACC.

Definition 3 (Conflicting Event Pair)**.** *A pair of events $\{v_i, v_j\}$ are conflicting if a user can attend at most one of the two events but not both.*

Thus, in any feasible arrangement M of events and users, no user can be assigned to conflicting events simultaneously. We denote $m(v,u) = 1$ or $\{v,u\} \in M$ as user u is assigned to event v, and $m(v,u) = 0$ or $\{v,u\} \notin M$ as u is not assigned to v. We then define users' interest in events as follows.

Definition 4 (Interestingness Value)**.** *A user u's interest (interestingness value) in event v is measured by a similarity function $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \in [0,1]$ based on the attributes $\boldsymbol{l}_v$ of v and the attributes $\boldsymbol{l}_u$ of u.*

Particularly, we use Equation (1) as our similarity function in the experiments, where $\sqrt{dT^2}$ is the furthest Euclidean distance possible between any pair of $\boldsymbol{l_v}, \boldsymbol{l_u}$ and thus $\frac{\|\boldsymbol{l}_v - \boldsymbol{l}_u\|_2}{\sqrt{dT^2}}$ is the normalized Euclidean distance between $\boldsymbol{l_v}$ and $\boldsymbol{l_u}$. Note that other similarity functions or even matrices denoting the interestingness values are applicable to our problem. We assume that $\max_u sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0, \forall v \in V$ and $\max_v sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0, \forall u \in U$ so that no event/user is negligible (events that are not interesting to any user or users who are not interested in any event).

$$sim(\boldsymbol{l}_v, \boldsymbol{l}_u) = 1 - \frac{\|\boldsymbol{l}_v - \boldsymbol{l}_u\|_2}{\sqrt{dT^2}} \tag{1}$$

We finally define our problem as follows.

Definition 5 (GEACC Problem)**.** *Given a set of events V, each v of which with capacity c_v and attributes $\boldsymbol{l}_v$, a set of users U, each u of which with capacity c_u and attributes $\boldsymbol{l}_u$, a set of conflicting event pairs CF and a similarity function, find an arrangement M among events and users to maximize $MaxSum(M) = \sum_{v \in V, u \in U} m(v,u) sim(\boldsymbol{l}_v, \boldsymbol{l}_u)$ such that*

- $\sum_u m(v,u) \le c_v, \forall v \in V$ *and* $\sum_v m(v,u) \le c_u, \forall u \in U$
- $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0, \forall \{v,u\} \in M$
- *There does NOT exist a triple v_i, v_j, u_k such that $m(v_i,u_k) = 1$, $m(v_j,u_k) = 1$, and $\{v_i, v_j\} \in CF$*

Note that we assume that $\max_v c_v \le |U|$ and $\max_u c_u \le |V|$. Also note that "matching" and "arrangement" are used interchangeably in this paper. For example, Example 1 in Section 1 illustrates the concepts of capacity and interestingness values and an optimal arrangement that satisfies the capacity and conflict constraints is shown. Table 2 summarizes the symbol notations. We next show the NP-hardness of the GEACC problem.

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

Proof. Please see the Appendix. □

2.2 Online Scenario of GEACC

We finally formally define the online setting of the GEACC problem as follows.

Definition 6 (OnlineGEACC Problem)**.** *Given a set of events V, each v of which has capacity c_v and attributes $\boldsymbol{l}_v$, a set of users U, each u of which arrives at the EBSN platform one by one with capacity c_u and attributes $\boldsymbol{l}_u$, which are unknown before u appears, a set of conflicting event pairs CF and a similarity function, find an arrangement M among events and users to maximize $MaxSum(M) = \sum_{v \in V, u \in U} m(v,u) sim(\boldsymbol{l}_v, \boldsymbol{l}_u)$ such that*

- *The arrangement for a new-coming u must be decided before the next user appears and cannot be revoked.*
- *The three constraints of GEACC are satisfied.*

Notice that the most important constraint of OnlineGEACC is that decisions for a new-coming user should be made immediately and cannot be revoked. For example, Example 1 in Section 1 also explains the online scenarios that the users arrive one by one in the order of $u_5, u_4, \cdots, u_1$ and how the online decision for each user is made.

3 Approximate Solutions for GEACC

In this section, we present two approximation algorithms for the GEACC problem. The first one is based on the minimum cost flow problem but is not scalable for large datasets. We then propose the second more efficient approximation algorithm.

3.1 MinCostFlow-GEACC Algorithm

The idea of MinCostFlow-GEACC is to first ignore the conflict condition and try to find a matching with maximum sum of interestingness values, and then resolve conflicts afterwards. Without considering conflicts, i.e. $CF = \emptyset$, GEACC can be reduced to the minimum cost flow (MCF) problem as explained shortly. Thus, we transform a GEACC instance to an MCF instance and obtain a temporary matching based on the solution of the transformed MCF instance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

To resolve the conflicts in the temporary matching, we use a greedy method to select the most interesting non-conflicting events for each user. We first explain the first step in detail.

Given an instance of GEACC with $CF = \emptyset$, we construct a flow network $G_F = (N_F, A_F)$ as follows. $N_F = V \cup U \cup \{s, t\}$, where s is a source node and t is a sink node. For every pair $v \in V, u \in U$ (including those with $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) = 0$), there is a directed arc $a_F(v, u) \in A_F$ from v to u with $a_F(v, u).cost = 1 - sim(\boldsymbol{l}_v, \boldsymbol{l}_u)$ and $a_F(v, u).capacity = 1$. For every $v \in V$, there is a directed arc $a_F(s, v) \in A_F$ from s to v with $a_F(s, v).cost = 0$ and $a_F(s, v).capacity = c_v$. For every $u \in U$, there is a directed arc $a_F(u, t) \in A_F$ from u to t with $a_F(u, t).cost = 0$ and $a_F(u, t).capacity = c_u$. The flow of an arc is integer-valued. We then send different amounts of flows from s to t. Specifically, for each $\Delta \in \{\Delta_{min}, \Delta_{min}+1, \cdots, \Delta_{max}\}$, where $\Delta_{min} = \min\{|V|, |U|\}$ and $\Delta_{max} = \min\{\sum_v c_v, \sum_u c_u\}$, we send an amount of Δ flows and calculate its corresponding minimum cost flow $F^\Delta = \{flow^\Delta(a_F)\}$. We also obtain an arrangement $M_\emptyset^\Delta$ corresponding to Δ by letting $m_\emptyset^\Delta(v, u) = 1$ iff $flow^\Delta(v, u) = 1$ and $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0$. Finally, we select the arrangement $M_\emptyset$ from $\{M_\emptyset^{\Delta_{min}}, M_\emptyset^{\Delta_{min}+1}, \cdots, M_\emptyset^{\Delta_{max}}\}$ with the maximum $MaxSum$ as the arrangement for the GEACC instance with $CF = \emptyset$. Particularly, we use the Successive Shortest Path Algorithm (SSPA) to calculate the minimum cost flow since it is the one suitable for large-scale data and many-to-many matching with real-valued arc costs as pointed out by [6].

After obtaining $M_\emptyset$, our second step is to resolve the conflicts. For each $u \in U$, we select a set of non-conflicting events from the ones assigned to u in $M_\emptyset$ such that the sum of the interestingness values between u and the selected events is maximized. Note that such selection procedure is identical to the maximum-weight independent set problem by regarding non-conflicting events as independent to each other and taking the similarity between u and an event as the weight of the event. The maximum-weight independent set problem is NP-hard [7]. Therefore, we find a set in a greedy way by iteratively selecting the most similar unselected pair that does not conflict with any pair that is already selected.

The whole procedure is illustrated in Algorithm 1. In lines 1-6, we first construct a flow network G_F and calculate the minimum cost flow on G_F with different amounts of flows as described previously, and then obtain a temporary matching $M_\emptyset$. In the second step, we obtain a feasible matching M by resolving the conflicting events for each u in lines 7-12. Particularly, at each iteration, we greedily add the most similar pair possible in lines 9-12.

Example 2. *Back to our running example in Example 1. Fig. 1a shows the flow network G_F. Fig. 1b shows the minimum cost flow corresponding to $M_\emptyset$, where each presented arc has flow at least one. The arcs with flows larger than one are marked, and the others have flow of one. Notice that u_1 is assigned to conflicting events v_1 and v_3 simultaneously in $M_\emptyset$. Since $sim(\boldsymbol{l}_{v_1}, \boldsymbol{l}_{u_1}) > sim(\boldsymbol{l}_{v_3}, \boldsymbol{l}_{u_1})$, only v_1 is assigned to u_1 in the final result. Similarly, for u_5, we remove $\{v_1, u_5\}$ and assign v_3 to u_5. Fig. 1c shows the final arrangement result, which has $MaxSum = 4.13$.*

Approximation Ratio. Next, we study the approximation ratio of MinCostFlow-GEACC.

Algorithm 1: MinCostFlow-GEACC

```
   input : V, U, {c_v}, {c_u}, {l_v}, {l_u}, CF
   output: A feasible arrangement M
1  construct G_F = (N_F, A_F);
2  foreach Δ ← Δ_min to Δ_max do
3      F^Δ ← MinCostFlow(G_F, Δ);
4      construct M_∅^Δ accordingly;
5      if MaxSum(M_∅^Δ) > MaxSum(M_∅) then
6          M_∅ ← M_∅^Δ;
7  foreach u ∈ U do
8      L ← sorted list of {v|m_∅(v, u) = 1} in
       non-increasing order of sim(l_v, l_u);
9      for i ← 1 to |L| do
10         v_i^L ← the i-th element of L ;
11         if v_i^L does not conflict with u's matched events in
           M then
12             m(v, u) ← 1;
13 return M
```

Lemma 1. *The $M_\emptyset$ obtained from the minimum cost flows of G_F is optimal for the GEACC instance with $CF = \emptyset$.*

Proof. Please see the Appendix. □

Corollary 1. *Let $M_{OPT} = \{m_{OPT}\}$ denote the optimal feasible matching. It holds that $MaxSum(M_{OPT}) \leq MaxSum(M_\emptyset)$.*

Theorem 2. *For the matching M returned by MinCostFlow-GEACC, it holds that $MaxSum(M) \geq \frac{MaxSum(M_{OPT})}{\max c_u}$, i.e. $MaxSum(M)$ is at least $\frac{1}{\max c_u}$ of the optimal result.*

Proof. Please see the Appendix. □

Complexity Analysis. The first step takes $O((\Delta_{max}^2 - \Delta_{min}^2)|V||U|\log(|V| + |U|))$ time. For the second step, the time complexity is $O(|U|((\max c_u)\log(\max c_u) + (\max c_u)^2)) = O(|U|(\max c_u)^2)$, where $O((\max c_u)\log(\max c_u))$ is the cost of line 8 and $O((\max c_u)^2)$ is the cost of lines 9-12. Since $\max c_u$ is relatively small compared to the other parameters, the major time consumption of MinCostFlow-GEACC comes from computing the minimum cost flow. In summary, the total time cost is $O((\Delta_{max}^2 - \Delta_{min}^2)|V||U|\log(|V| + |U|) + |U|(\max c_u)^2)$.

3.2 Greedy-GEACC Algorithm

MinCostFlow-GEACC could be inefficient when the scale of data is large. In this subsection, we present a more efficient algorithm, Greedy-GEACC. The main idea of Greedy-GEACC is to greedily add the most similar unmatched pair $\{v, u\}$ that does not conflict with existing matched pairs into the current matching at each iteration. Unlike MinCostFlow-GEACC that resolves conflicts after obtaining a temporary result, Greedy-GEACC avoids conflicts from the first beginning.

Specifically, we maintain a heap H to store the most similar pair candidates between $v \in V$ and $u \in U$ and extract the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

most similar one from H at each iteration. We initialize H as follows. For each $v \in V$, we find its first nearest neighbor (NN) $u_{nn} \in U$, i.e. $sim(\boldsymbol{l}_v, \boldsymbol{l}_{u_{nn}}) \geq sim(\boldsymbol{l}_v, \boldsymbol{l}_{u'}), \forall u' \in U$. We call u_{nn} a *visited neighbor* of v, and the others *unvisited neighbors*. Each such pair $\{v, u_{nn}\}$ is pushed into H. Note that $sim(\boldsymbol{l}_v, \boldsymbol{l}_{u_{nn}}) > 0$ according to our problem definition. Similarly, for each $u \in U$, we also find its first NN $v_{nn} \in V$. We also call v_{nn} a *visited neighbor* of u. For each such pair $\{v_{nn}, u\}$, if it is not yet in H, we push it into H. Thus, NO pair is pushed into H for more than once. After the initialization step, we enter the iteration of greedily adding the most similar pair in H into the current matching, which is empty initially.

We then iterate as follows. At each iteration, we pop the pair $\{v, u\}$ with $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'}), \forall \{v', u'\} \in H$ from H, which we call a *visited pair*. Thus, pairs that have not yet been pushed into H or those that are still in H are called *unvisited*. If neither v nor u is full in capacity, and $\{v, u\}$ does not conflict with existing matched pairs, we can safely add $\{v, u\}$ into the current matching by setting $m(v, u) = 1$ and decreasing the available capacities of v and u by one respectively. Whether $\{v, u\}$ is added to the current matching or not, we then update H as follows. For v, if it is not yet fully occupied, we find its next *feasible unvisited NN* $u_{nn} \in U$, i.e. $sim(\boldsymbol{l}_v, \boldsymbol{l}_{u_{nn}}) \geq sim(\boldsymbol{l}_v, \boldsymbol{l}_{u'}), \forall$ feasible unvisited neighbor u' of v, where we call an unvisited neighbor u' feasible if $sim(\boldsymbol{l}_v, \boldsymbol{l}_{u'}) > 0$ and $\{v, u'\}$ satisfies the capacity and conflict constraints if it is to be added to the matching. Note that u_{nn} may not exist as there may be no more feasible unvisited neighbors in U for v. In such case, we do nothing to H. Otherwise, $\{v, u_{nn}\}$ is pushed into H if it is not yet in H. We call v a *finished node* if u_{nn} cannot be found for v. Similarly, for u, we also find its next feasible unvisited NN $v_{nn} \in V$ if u is not yet fully occupied. If v_{nn} exists and $\{v_{nn}, u\}$ is not yet in H, we push $\{v_{nn}, u\}$ into H. We also call u a finished node if v_{nn} cannot be found for u. $u_{nn}(v_{nn})$ becomes $v(u)$'s visited neighbor if it exists. After updating H, we proceed to the next iteration. The iteration procedure terminates when H becomes empty.

The procedure of Greedy-GEACC is illustrated in Algorithm 2. In lines 1-9, we initialize the heap H by pushing each $v(u)$ and its first NN in $U(V)$ into H. In lines 11-23, we iteratively pop the most similar pair $\{v, u\}$ from H and add it to the current matching if possible. Lines 13-15 check the feasibility of the pair before adding it to the matching. Then in lines 16-23, we push $v(u)$ paired with its next feasible unvisited NN in $U(V)$ into H if possible. The whole iteration terminates when H becomes empty.

Example 3. *We continue to use Example 1 for illustration of Greedy-GEACC. Fig. 2a shows the state of H after the first iteration, where $\{v_1, u_1\}$ is popped from H and added to the matching. The next feasible unvisited NN of v_1 is u_3 but $\{v_1, u_3\}$ is already in H, so we do not push $\{v_1, u_3\}$ into H. As the next feasible unvisited NN of u_1 cannot be found, u_1 becomes a finished node. Then in the second iteration, as shown in Fig. 2b, we pop $\{v_3, u_1\}$ from H. Note that v_3 conflicts with v_1, which is already matched to u_1, so we cannot add $\{v_3, u_1\}$ to the matching. The next NN of v_3 is u_4, and $\{v_3, u_4\}$ is already in H, so we do not push $\{v_3, u_4\}$ into H. Then during the third iteration, $\{v_1, u_3\}$ is popped from H, which can be added to the matching. The next NN of v_1 is u_5, and we push $\{v_1, u_5\}$ into H (in bold). Note that u_3 has been fully occupied, so we do not find the next NN of u_3. Subsequent iterations are omitted for brevity. Fig. 2d shows the final iteration, where H becomes empty and we have a final arrangement with $MaxSum$ of 4.28.*

Algorithm 2: Greedy-GEACC

```
   input : V, U, {c_v}, {c_u}, {l_v}, {l_u}, CF
   output: A feasible arrangement M
1  H ← ∅;
2  foreach v ∈ V do
3      u_nn ← v's first NN in U;
4      push {v, u_nn} into H;
5  foreach u ∈ U do
6      v_nn ← u's first NN in V;
7      if {v_nn, u} ∉ H then
8          push {v_nn, u} into H;
9  heapify H;
10 m(v, u) ← 0, ∀v ∈ V, u ∈ U;
11 while H ≠ ∅ do
12     extract the most similar pair {v, u} from H;
13     if (c_v > 0) and (c_u > 0) and (v does not conflict with
       u's matched events) then
14         m(v, u) ← 1;
15         decrease c_v, c_u by 1;
16     if c_v > 0 then
17         u_nn ← v's next feasible unvisted NN;
18         if u_nn∃ and {v, u_nn} ∉ H then
19             push {v, u_nn} into H;
20     if c_u > 0 then
21         v_nn ← u's next feasible unvisited NN;
22         if v_nn∃ and {v_nn, u} ∉ H then
23             push {v_nn, u} into H;
24 return M
```

We next show some properties and the correctness of Greedy-GEACC.

Lemma 2. *For every $v(u)$, if $v(u)$ is not a finished node, at least one pair incident to $v(u)$ is in H before the next iteration.*

Proof. Please see the Appendix. □

Lemma 3. *At each iteration of Greedy-GEACC, the most similar unvisited pair $\{v, u\}$ possible is popped from H, i.e. $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'})$ for all feasible unvisited $\{v', u'\} \in \{\{v', u'\} | v', u' \text{ are unfinished}\}$.*

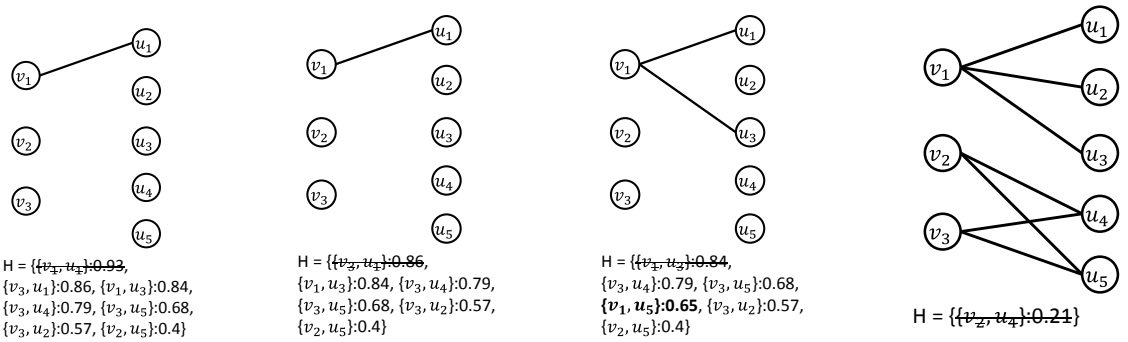
Proof. Please see the Appendix. □

Corollary 2. *At each iteration of Greedy-GEACC, for the popped pair $\{v, u\}$, it holds that $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \leq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'}), \forall$ visited $\{v', u'\}$.*

Lemma 4. *Greedy-GEACC terminates after a finite number of iterations.*

Proof. Please see the Appendix. □

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering



(a) 1st iteration (b) 2nd iteration (c) 3rd iteration (d) Final iteration

Fig. 2: Illustrated example of Greedy-GEACC.

Lemma 5. *When Greedy-GEACC terminates, no more unmatched pair $\{v, u\}$, i.e. $m(v, u) = 0$, can be added to the current matching.*

Proof. Please see the Appendix. □

Lemmas 2 to 5 ensure that Greedy-GEACC adds the most similar unvisited pair possible into the matching at each iteration and terminates when the current matching can no more be improved by adding new unmatched pairs.

Approximation Ratio. We next study the approximation ratio of Greedy-GEACC.

Theorem 3. *For the matching M returned by Greedy-GEACC, it holds that $MaxSum(M) \geq \frac{MaxSum(M_{OPT})}{1+\max c_u}$, i.e. $MaxSum(M)$ is at least $\frac{1}{1+\max c_u}$ of the optimal result.*

Proof. Please see the Appendix. □

Complexity Analysis. Without limiting ourselves to using specific index, let $\sigma(S)$ denote the time to find a k-th NN in a set S. Note that a number of index techniques can be used in our problem, such as iDistance [8] and VA-File [9]. It follows that the time cost of the initialization step is $O(|V|\sigma(V) + |U|\sigma(U) + |V| + |U|)$, where $O(|V| + |U|)$ is cost of building H. In the second step, we have at most $O(|V||U|)$ iterations, each of which takes $O(\log(|V| + |U|))$ to pop a pair from H and $O(\sigma(V)+\sigma(U)+\log(|V|+|U|))$ to push new pairs into H. In summary, Greedy-GEACC takes $O(|V||U|(\sigma(V) + \sigma(U) + \log(|V| + |U|)))$ time in the worst case.

4 Exact Solution for GEACC

In this section, we present an exact solution for the GEACC problem. Since GEACC is NP-hard, it seems that the only way to find an optimal solution is to enumerate all possible matchings and select the optimal one. Without pruning, the search space will be as large as $2^{|V|\times|U|}$. To improve efficiency, we propose a pruning technique to reduce the search space. We name this exact algorithm Prune-GEACC.

Basically, in any matching, each pair $\{v, u\}$ has two states: matched or unmatched. Thus, we search possible matchings by enumerating different combinations of states of all pairs in a recursive way. Specifically, for each v, let $u_{v,j}$ be its j-NN in U and s_v be $sim(\boldsymbol{l}_v, \boldsymbol{l}_{u_{v,1}})$. Let L be the sorted list of v in non-increasing order of $s_v \times c_v$, where the i-th element is v_i^L. We visit each element in L in order, and

Algorithm 3: Prune-GEACC

input : $V, U, \{c_v\}, \{c_u\}, \{\boldsymbol{l}_v\}, \{\boldsymbol{l}_u\}, CF$
output: A feasible arrangement M

```
M ← Greedy-GEACC();
foreach v ∈ V do
    u_{v,1} ← 1-NN of v;
    s_v ← sim(l_v, l_{u_{v,1}});
L ← sorted list of v in non-increasing order of s_v × c_v;
sum_remain ← Σ_{2≤k≤|V|} s_{v_k^L} × c_{v_k^L};
m_c(v, u) ← 0, ∀v ∈ V, u ∈ U;
Search-GEACC(1, 1);
return M
```

enumerate each pair $\{v, u\}$ incident to v in non-increasing order of $sim(\boldsymbol{l}_v, \boldsymbol{l}_u)$. In other words, we visit $|U|$ NNs of v in order. Let $\{v_i^L, u_{i,j}\}$ be the pair that will be visited next, $M_{visited}$ be the partial matching determined by the states of the visited pairs, and M_{best} be the best complete matching found so far. And we have the following lemma.

Lemma 6. *Let*

$$sum_{max}(v_i^L, u_{i,j}) = MaxSum(M_{visited}) + \sum_{i+1\leq k\leq|V|} s_{v_k^L} \times c_{v_k^L} + sim(\boldsymbol{l}_{v_i^L}, \boldsymbol{l}_{u_{i,j}}) \times c_{v_i^L,remain} \quad (2)$$

where $c_{v_i^L,remain}$ is the remaining capacity of v_i^L after being assigned partially in $M_{visited}$. If $sum_{max}(v_i^L, u_{i,j}) \leq MaxSum(M_{best})$, for any matching $M' \supseteq M_{visited}$, it holds that $MaxSum(M') \leq MaxSum(M_{OPT})$, where M_{OPT} is the optimal matching.

Proof. Please see the Appendix. □

Lemma 6 indicates that when we are about to visit a pair $\{v_i^L, u_{i,j}\}$ during the recursion process, if $sum_{max}(v_i^L, u_{i,j}) \leq MaxSum(M_{best})$, we can safely prune at $\{v_i^L, u_{i,j}\}$ as no matching better than the current one could be found by visiting the remaining unvisited pairs. Therefore, in Prune-GEACC, we maintain $sum_{max}(v_i^L, u_{i,j})$ and prune at a certain search node whenever Lemma 6 holds. First note that actually we do not need to repetitively calculate $sum_{max}(v_i^L, u_{i,j})$ by summing over all unvisited pairs. We maintain $sum_{remain} = \sum_{i+1\leq k\leq|V|} s_{v_k^L} \times c_{v_k^L}$, and $sum_{max}(v_i^L, u_{i,j}) = MaxSum(M_{visited}) + sum_{remain} +$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

Algorithm 4: Search-GEACC

```
   input : v_id, u_id
1  v ← v^L_{v_id};
2  u ← u_id-NN of v;
3  if c_v > 0 and c_u > 0 and v does not conflict with u's
   matched events then
4      m_c(v, u) ← 1;
5      decrease c_v, c_u by 1;
6      if u_id = |U| or c_v = 0 then
7          if v_id = |V| then
8              if MaxSum(M_c) > MaxSum(M) then
9                  update M to M_c;
10         else if
           MaxSum(M_c) + sum_remain > MaxSum(M)
           then
11             sum_remain ← sum_remain − s_{v^L_{v_id+1}} × c_{v^L_{v_id+1}};
12             Search(v_id + 1, 1);
13             recover sum_remain;
14     else
15         u_nn ← (u_id + 1)-NN of v;
16         if MaxSum(M_c) + sum_remain +
           sim(l_v, l_{u_nn}) × c_v > MaxSum(M) then
17             Search(v_id, u_id + 1);
18     m_c(v, u) ← 0;
19     increase c_v, c_u by 1;
20 Same as lines 6-17;
```

$sim(\boldsymbol{l}_{v_i^L}, \boldsymbol{l}_{u_{i,j}}) \times c_{v_i^L,remain}$. sum_{remain} is maintained as follows. Initially, $sum_{remain} = \sum_{2 \le k \le |V|} s_{v_k^L} \times c_{v_k^L}$. Whenever we are about to visit $\{v_i^L, u_{i,1}\}$, we simply subtract $s_{v_i^L} \times c_{v_i^L}$ from sum_{remain}. Also note that initially, our best matching is empty, which could reduce the efficiency of Prune-GEACC at the beginning of recursion. Therefore, we run Greedy-GEACC first before running Prune-GEACC, and use the matching found by Greedy-GEACC as the best matching found so far so that to prune poor matchings from the first beginning.

The main procedure of Prune-GEACC is illustrated in Algorithm 3. In line 1, we find an initial matching M by running Greedy-GEACC. In lines 2-4, we find the 1-NN $u_{v,1}$ in U for each v and obtain s_v. In lines 5-7, we initialize L, sum_{remain} and M_c. We enter recursion by visiting the first element in L and its 1-NN in line 8.

Algorithm 4 illustrates the Search recursion procedure of Prune-GEACC. At each depth of recursion, we enumerate the two states of a particular pair $\{v, u\}$, where v is the v_{id}-th element in L, and u is the u_{id}-NN of v. If $\{v, u\}$ satisfies certain constraints (line 3), we enumerate the state of $\{v, u\}$ as matched in lines 4-19. In lines 4-5, we add $\{v, u\}$ to the current matching M_c, and decrease the capacities of v, u properly. If u_{id} is $|U|$ or v is fully occupied, we proceed to the next element $v_{v_{id}+1}^L$ in L and enumerate the states of pair $v_{v_{id}+1}^L$ and its 1-NN in U (lines 7-13). Otherwise, we proceed to the next NN of v and enumerate the states between it and v (lines 15-17). In lines 7-9, we check whether all pairs have been enumerated and update the best matching found so far (lines 8-9). Otherwise, we check whether enumerating the remaining pairs could yield a better matching (line 10). If finding a better matching is possible, we update sum_{remain} and proceed to enumerating the next pair (lines 11-13). Similarly, we check whether finding a better matching is possible in line 15 if we are to proceed to enumerating the pair of v and its next NN. In line 20, we enumerate the state of $\{v, u\}$ as unmatched, the procedure of which is the same as lines 6-17.

5 Solution to OnlineGEACC

In this section, we present the solution to the online scenario of GEACC, i.e. OnlineGEACC. Notice that the offline solutions mentioned above cannot solve OnlineGEACC. The reason is that offline solutions need full information of events and users, which is no longer available in online scenarios. For example, MinCostFlow-GEACC needs all attribute and capacity values of users to construct the flow network, and Greedy-GEACC needs to know the pair of event and users with the globally largest similarity value repetitively. However, in the online scenarios, since users come one by one and information of subsequent users cannot be known in advance, no way can the offline solutions be applicable. Therefore, we present an algorithm called OnlineGreedy-GEACC specifically for the OnlineGEACC problem, which is based on the framework of [10]. The difference between our algorithm and the framework in [10] is that [10] does not address the conflicts of nodes.

The main idea of the OnlineGreedy-GEACC algorithm is to greedily match pairs of events and users whose interestingness values are above a randomly chosen threshold: we first randomly pick a threshold on the weights of the edges, i.e. the interestingness values of the pairs, to be matched, and then for each new coming user u, we find all the events that can be feasibly matched to v and have interestingness values with u no less than the picked threshold and greedily arrange at most c_u events to u that are of the largest interestingness values with u.

Specifically, in the OnlineGreedy-GEACC algorithm, we first sample a k uniformly from $\{1, 2, ..., \lceil -\log_2 w_{min} \rceil\}$, where w_{min} is the minimum possible interestingness value of a user towards an event other than zero, which can be learned from historical values and can be regarded as an arbitrarily small real number. We then pick the threshold as $\theta = \frac{1}{2^k}$. Then whenever a new user u comes, we first find all the events v in V that are not yet full in capacity and have interestingness values no less than the threshold, i.e. $sim(v, u) \ge \theta$, which result in V'. If V' is empty, no event will be arranged to u. Otherwise, we visit each event in V' in non-increasing order of their interestingness values with u and assign a visited event to u if it does not conflict with any event that is already assigned to u until u is full of capacity or all the events in V' have been visited, whichever occurs first. Notice that the process of visiting events in V' can be regarded as selecting at most c_u non-conflicting events that are most interesting to u from V'.

The procedure is illustrated in Algorithm 5. In lines 1-2, we randomly pick a threshold θ. We then keep looping lines 3-9 to make arrangement for each new-coming user u. Specifically, we first construct the set V' whose events are non-full and have interestingness values with u no less

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

Algorithm 5: OnlineGreedy-GEACC

input : $V, U, \{c_v\}, \{c_u\}, \{\boldsymbol{l}_v\}, \{\boldsymbol{l}_u\}, CF$
output: A feasible arrangement M
1 sample $k \leftarrow \{1, 2, ..., \lceil -\log_2 w_{min} \rceil\}$ uniformly;
2 $\theta \leftarrow \frac{1}{2^k}$;
3 **foreach** *new-arrival* u **do**
4 $V' \leftarrow \{v \in V | c_v > 0$ and $sim(v, u) \geq \theta\}$;
5 **foreach** $v \in V'$ *in non-increasing order of* $sim(v, u)$ **do**
6 **if** *v does not conflict with u's matched events* **then**
7 $m(v, u) \leftarrow 1$;
8 decrease c_v, c_u by 1;
9 break if c_u becomes 0;
10 **return** M

than the threshold in line 4. We then visit each $v \in V'$ in non-increasing order of $sim(v, u)$ and assign the valid ones to u and update their left-over capacity accordingly until u becomes full in capacity in lines 5-9.

Example 4. *Back to our running example. Suppose again the arrival order of users is $u_5, u_4, \cdots, u_1$. Notice that the minimum interestingness value other than zero is 0.19, and thus $\lceil -\log_2 w_{min} \rceil = 3$. Suppose k is sampled as 2 and thus $\theta = 0.25$. Then when u_5 first comes, $V' = \{v_1, v_2, v_3\}$ and we assign the most interesting combination v_3 and v_2 to it. When u_4 comes, $V' = \{v_1, v_3\}$ and only the most interesting v_3 is assigned to u_4 since v_1 conflicts with v_3. Similarly, v_1 is assigned to each of u_3, u_2, u_1, respectively, when they come, and the $MaxSum$ value is 4.07.*

Competitive ratio. We next study the competitive ratio of OnlineGreedy-GEACC, the analysis of which is based on the framework in [10]. The difference between our analysis and the framework in [10] is that [10] does not need to consider the conflicts or capacities of nodes.

Theorem 4. *OnlineGreedy-GEACC has competitive ratio of* $\frac{1}{2\lceil -\log_2 w_{min} \rceil (\max c_u + 2)}$.

Proof. Let O denote the offline optimal matching result. Suppose k is sampled as i. First, consider the set $S = O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})} \setminus M_{\geq \frac{1}{2^i}}$, where $O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})}$ denotes the subset of edges in O whose weights, i.e. interestingness values, lie in the interval $[\frac{1}{2^i}, \frac{1}{2^{i-1}})$ and $M_{\geq \frac{1}{2^i}}$ denotes the subset of edges in the matching returned by OnlineGreedy-GEACC when $k = i$. Notice that when $i = 1$, $S = O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}}]} \setminus M_{\geq \frac{1}{2^i}}$. For brevity, we also denote as $O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})}$ in the subsequent proof even when $i = 1$.

For each edge (v, u) in S, it is unmatched in $M_{\geq \frac{1}{2^i}}$ due to three possible cases: (1) v is already full in capacity when u arrives; (2) v conflicts with an event matched to u; (3) u is matched to other events and is already full in capacity when visiting v. Notice that all the three cases can "blame" a certain edge in $M_{\geq \frac{1}{2^i}}$, and each edge in $M_{\geq \frac{1}{2^i}}$ is "responsible" for at most $\max c_u + 1$ edges in S. Therefore, we have

$$|O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})} \setminus M_{\geq \frac{1}{2^i}}| \leq (\max c_u + 1)|M_{\geq \frac{1}{2^i}}| \quad (3)$$

and

TABLE 3: Real Dataset

City	$\|V\|$	$\|U\|$	c_v	c_u	$\frac{\|CF\|}{\|V\|(\|V\|-1)/2}$
VA	225	2012	Uni.: [1, 50]	Uni.: [1, 4]	0, 0.25,
Auckland	37	569	Nor.: $\mu = 25$,	Nor.: $\mu = 2$,	0.5, 0.75,
Singapore	87	1500	$\sigma = 12.5$	$\sigma = 1$	1

$$\begin{aligned} |O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})}| &= |O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})} \cap M_{\geq \frac{1}{2^i}}| + |O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})} \setminus M_{\geq \frac{1}{2^i}}| \\ &\leq |M_{\geq \frac{1}{2^i}}| + (\max c_u + 1)|M_{\geq \frac{1}{2^i}}| \\ &= (\max c_u + 2)|M_{\geq \frac{1}{2^i}}| \end{aligned} \quad (4)$$

Since $MaxSum(O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})}) \leq \frac{1}{2^{i-1}}|O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})}|$, we have

$$\begin{aligned} MaxSum(M_{\geq \frac{1}{2^i}}) &\geq \frac{1}{2^i}|M_{\geq \frac{1}{2^i}}| \\ &\geq \frac{1}{2^i(\max c_u + 2)}|O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})}| \\ &\geq \frac{1}{2(\max c_u + 2)} MaxSum(O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})}) \end{aligned} \quad (5)$$

Now consider O. Let $N \underset{N}{=} \lceil -\log_2 w_{min} \rceil$. We have

$$MaxSum(O) = \sum_{i=1}^{N} MaxSum(O_{([\frac{1}{2^i}, \frac{1}{2^{i-1}}))}) \quad (6)$$

Then

$$\begin{aligned} E(MaxSum(M)) &= \sum_{i=1}^{N} \frac{1}{N} MaxSum(M_{\geq \frac{1}{2^i}}) \\ &\geq \frac{\sum_{i=1}^{N} MaxSum(O_{[\frac{1}{2^i}, \frac{1}{2^{i-1}})})}{2N(\max c_u + 2)} \\ &= \frac{MaxSum(O)}{2N(\max c_u + 2)} \\ &= \frac{MaxSum(O)}{2\lceil -\log_2 w_{min} \rceil (\max c_u + 2)} \end{aligned} \quad (7)$$

□

Complexity analysis. For each new-coming event, we spend at most $O(|V|)$ time to construct the set V' whose size is at most $|V|$. We then spend $O(|V'| \log |V'|)$ time to sort V' in order and at most $O(c_u)$ time to check conflicts when visiting an event in V'. Therefore, the time complexity is $O(|V| \log |V| + |V| c_u)$ for each user, and the space complexity is $O(|V|)$.

6 Evaluation

6.1 Experiment Setup

We use both real and synthetic datasets for experiments. We use the Meetup dataset from [1] as real dataset. In the Meetup dataset, each user is associated with some tags and a location. The events are not explicitly associated with tags, but each event is organized by a "group" on Meetup, and each group is associated with some tags. Thus, for each event, we use the tags of the group who creates it as the tags of the event itself. To remove misspelled and redundant tags, we merge the tags with the same meaning and select 20 most popular tags as attributes of users/events, and use the normalized frequencies of the tags as attribute values. Since it is unlikely for a user living in a city to attend a meet-up event held in another city, we cluster events and users based on their locations and focus on the events/users located in the same city. We select three popular cities, Vancouver, Auckland, and Singapore, and extract events and users

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

TABLE 4: Synthetic Dataset

Factor	Setting
$\|V\|$	20, 50, **100**, 200, 500
$\|U\|$	100, 200, 500, **1000**, 2000, 5000
d	2, 5, 10, 15, **20**
l_v^i, l_u^i	($T = 10000$) **Uniform: [0, T]**, Zipf: 1.3 Normal: $\mu = T/4$, $\sigma = T/4$; $\mu = 3T/4$, $\sigma = T/4$
c_v	Uniform: [1, 10], [1, 20], **[1, 50]**, [1, 100], [1, 200] Normal: $\mu = 25$, $\sigma = 12.5$
c_u	Uniform: [1, 2], **[1, 4]**, [1, 6], [1, 8], [1, 10] Normal: $\mu = 2$, $\sigma = 1$
$\frac{\|CF\|}{\|V\|(\|V\|-1)/2}$	0, **0.25**, 0.5, 0.75, 1
Scalability	$\|V\| = 100, 200, 500, 1000$ $\|U\| = 10$K, 25K, 50K, 75K, 100K

located within the area around each city. Since capacity and conflict information is not given in the dataset, we generate capacity of events/users following Uniform and Normal distribution, and randomly select a subset of event pairs as conflicting pairs. TABLE 3 presents the statistics and configuration. For synthetic data, we generate attribute values and capacity of events/users following Uniform, Normal and Zipf distributions respectively. Statistics and configuration of synthetic data are illustrated in TABLE 4, where we mark our default settings in bold font. Note that all generated capacity values are converted into integers.

We mainly evaluate our algorithms in terms of $MaxSum$, running time and memory cost, and study the effect of varying parameters on the performance of the algorithms. Notice that some figures are plotted in log scale for clearness. For OnlineGEACC, since the arrival orders of users are random, we randomly run 50 different orders for each setting and run 5 times for each order as OnlineGreedy-GEACC is randomized. We report the average $MaxSum$ values and memory costs over all $50 \times 5 = 250$ tests and also the average running time for a user over all 250 tests. For example, if processing all 1000 users takes 1 second on average over 250 tests, we report the running time as $1/1000 = 1$ms per user on average. w_{min} is set to 0.0001. The algorithms are implemented in C++, and the experiments were performed on a machine with Intel i7-2600 3.40GHZ 8-core CPU and 8GB memory.

6.2 Evaluation for GEACC

Baselines. We use two random algorithms as baselines. For the first baseline, Random-V, we iterate over each $v \in V$, and at each iteration add each pair $\{v, u\}, \forall u \in U$ into M with probability $\frac{c_v}{|U|}$ if $\{v, u\}$ satisfies all the constraints. For the second baseline, Random-U, we iterate over each $u \in U$, and at each iteration add each pair $\{v, u\}, \forall v \in V$ into M with probability $\frac{c_u}{|V|}$ if $\{v, u\}$ satisfies all the constraints.

Effect of cardinality. We first show the effect of varying cardinality of V and U. The first column of Fig. 3 shows the results on varying $|V|$, where the other parameters are set to default. We have the following observations. First, Greedy-GEACC performs the best. Greedy-GEACC consumes as less as space as the baselines do while is slightly inefficient than the baselines, and returns matchings with the largest $MaxSum$. Second, MinCostFlow-GEACC achieves larger MaxSum than the baselines do but is much less efficient in both time and space. Third, $MaxSum$ increases when $|V|$ becomes larger, but the increase becomes smaller when $|V|$ gets large. This is because when $|V|$ is larger, users generally have more matching options and there may be more matched pairs. However, when $|V|$ becomes too large, users' capacity will become saturated and thus the $MaxSum$ will increase slower. Finally, the running time and memory cost increases (slightly for Greedy-GEACC) as $|V|$ increases, which is natural as the data size becomes larger.

The second column of Fig. 3 shows the results on varying $|U|$, which have similar patterns to those when $|V|$ varies.

Effect of dimensionality. We next show the results of varying the dimensionality d of the attribute space in the third column of Fig. 3. We can observe that $MaxSum$ decreases as d increases since the attribute space becomes sparser when d increases, which leads to the larger averaged distance between attribute vectors. Also, d has slight effect on both the time and space consumption of the algorithms.

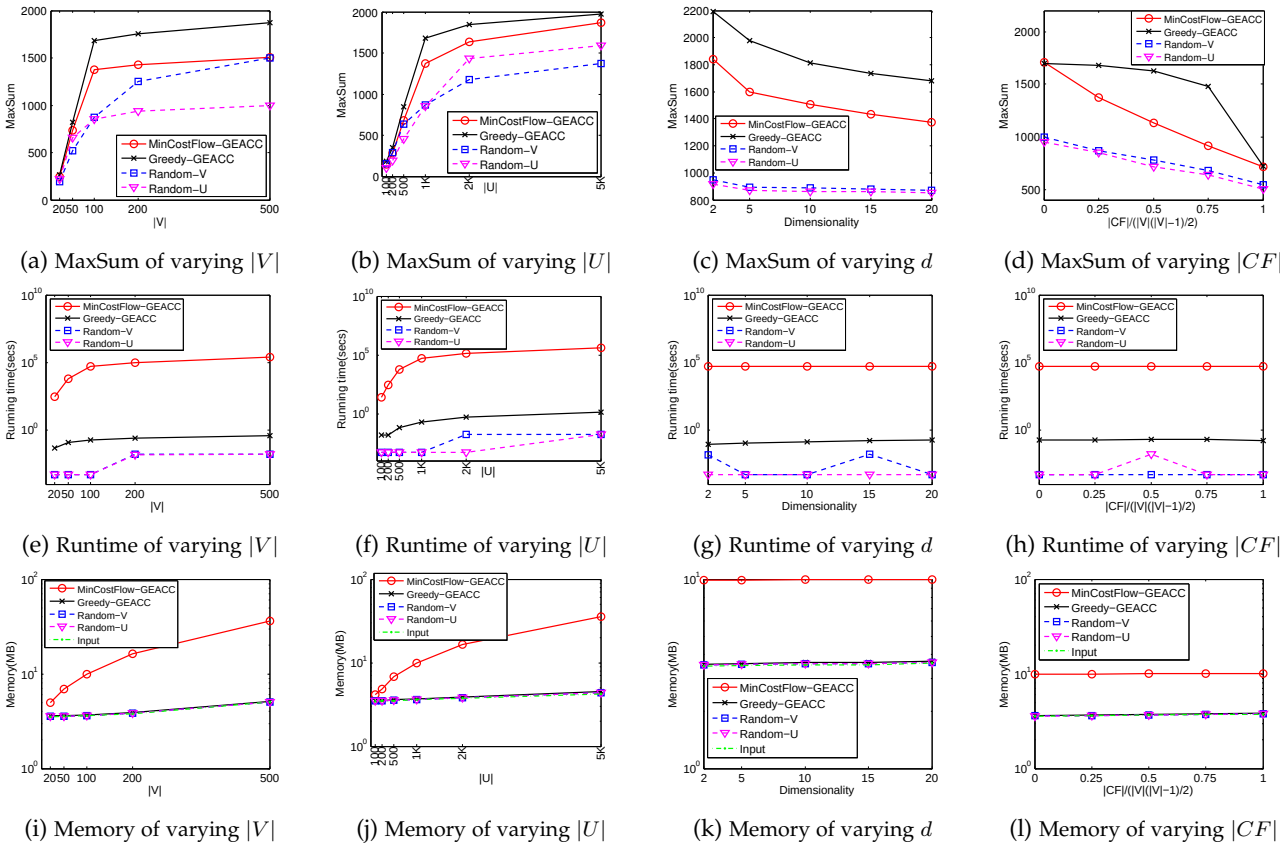
Effect of conflict set size. The last column of Fig. 3 shows the results of varying $|CF|$, where we vary the size of CF w.r.t. the size of event pairs, i.e. $|V|(|V|-1)/2$. Notice there are two extreme cases: when $|CF|/(|V|(|V|-1)/2) = 0$, i.e. $CF = \emptyset$, and when $|CF|/(|V|(|V|-1)/2) = 1$, i.e. every pair of events are conflicting. The other parameters are set to default. We have the following observations. First, when $CF = \emptyset$, MinCostFlow-GEACC has a slightly better $MaxSum$ than Greedy-GEACC does, which is reasonable as MinCostFlow-GEACC returns an optimal matching in this case. Second, $MaxSum$ decreases when the relative size of CF increases. This is reasonable as the number of possible matched pairs decreases as $|CF|$ increases. Finally, the varying size of CF has little effect on the running time of the algorithms, as the cost of the algorithms mainly depends on the size of V and U.

Effect of capacity. We next study the effect of capacity of events and users. We first study the results when c_v varies, which are shown in the first column of Fig. 4. The values of c_v are generated uniformly in range [1, $\max c_v$], where $\max c_v$ varies in our experiment. Thus, when $\max c_v$ increases, the overall capacity of v increases too. We have the following observations. First, $MaxSum$ generally increases as c_v becomes larger. This is reasonable as events can accommodate more users who are interested in them when their capacity increases. Second, increasing c_v results in larger time cost of MinCostFlow-GEACC, but has little effect on Greedy-GEACC and the baselines. This is because when c_v increases, the number of iterations for calculation of minimum cost flow in MinCostFlow-GEACC also increases, leading to larger time consumption of MinCostFlow-GEACC. Notice that when c_v is large w.r.t. $|U|(= 1000)$, the increase of time cost of MinCostFlow-GEACC becomes slighter since the amount of flow in such cases is determined by c_u (remember that $\Delta_{max} = \min\{\sum c_v, \sum c_u\}$). Finally, varying c_v has little effect on the memory cost of all the algorithms.

The second column of Fig. 4 shows the results of varying c_u. Similary, the values of c_u are generated uniformly in range [1, $\max c_u$] and $\max c_u$ varies in our experiment. We observe that the results have similar patterns as those of varying c_v though with some fluctuation due to the small gap between consecutive $\max c_u$'s.

Effect of distribution. The third column of Fig. 4 shows the results when we generate the synthetic data according

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering



(a) MaxSum of varying $|V|$ (b) MaxSum of varying $|U|$ (c) MaxSum of varying d (d) MaxSum of varying $|CF|$

(e) Runtime of varying $|V|$ (f) Runtime of varying $|U|$ (g) Runtime of varying d (h) Runtime of varying $|CF|$

(i) Memory of varying $|V|$ (j) Memory of varying $|U|$ (k) Memory of varying d (l) Memory of varying $|CF|$

Fig. 3: Results on varying cardinality, dimensionality and the size of conflict set.

to different distributions. Specifically, we present the results when the attributes are generated following Zipf distribution and the capacities are generated following Normal distribution. We observe that the general patterns of data generated by different distributions are similar in every aspect. It indicates that we do not lose generality by studying the other experiments on data generated uniformly.

We also study the results when the attribute values are generated following Uniform, Normal and Zipf distributions and the capacity values are generated following Uniform and Normal distributions. The results have similar trending patterns, and we do not present them for brevity.

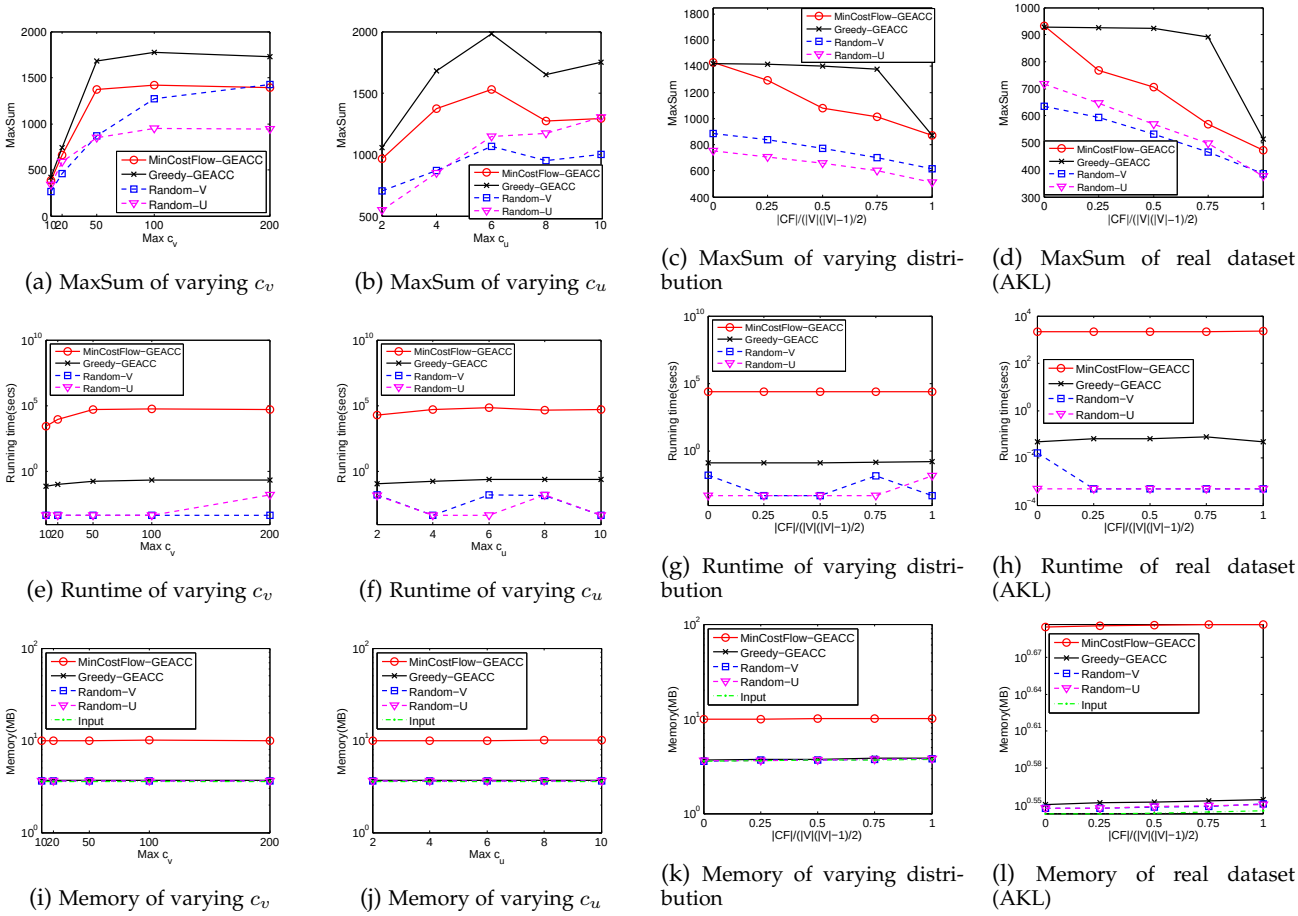
Real dataset. The last column of Fig. 4 shows the results on real dataset (Auckland) when the capacity values are generated following Uniform distribution. Notice that the results on real dataset have similar patterns to those of the synthetic data. Similar patterns are observed on the other two real datasets and when the capacity values are generated following Normal distribution, and we omit the results due to limited space.

Scalability. MinCostFlow-GEACC is not efficient enough according to our previous experiment results. Thus, we study the scalability of Greedy-GEACC in this part. The results are shown in Fig. 5a and 5b. Specifically, we set $|V| = 100, 200, 500, 1000$ respectively, and vary the size of $|U|$. Since $|U|$ is relatively large, we set $\max c_v$ to 200. The other parameters are set to default. We observe that the memory cost of Greedy-GEACC grows linearly with the size of data and is relatively small subtracting those consumed by input data. Also, the time cost of Greedy-GEACC grows nearly linearly with the size of data. The results show that Greedy-GEACC is scalable in both time and space.

Effectiveness of approximate solutions. We next study the effectiveness of our approximate solutions, whose results are presented in Fig. 5c and 5d. Notice that since we need to find the exact solutions in this part of evaluation and Prune-GEACC is infeasible on large dataset, we set $|V| = 5$, $|U| = 15$ and $c_v \sim \text{Uniform}[1, 10]$. The other parameters are set to default. In Fig. 5c, we compare the approximated $MaxSum$s returned by MinCostFlow-GEACC and Greedy-GEACC with the optimal $MaxSum$. We first observe that when $|CF| = \emptyset$, MinCostFlow-GEACC returns the optimal matching, which is reasonable. We also observe that the $MaxSum$s returned by Greedy-GEACC are quite close to the optimal ones, indicating that Greedy-GEACC returns quite good results in practice. Fig. 5d shows the running time of different algorithms. The results indicate that the two approximate solutions are very efficient compared with the exact solution. Therefore, in overall, our approximate solutions are both effective and efficient.

Effectiveness of pruning. We finally study the effectiveness of our pruning technique, whose results are shown in Fig. 6. In Fig. 6a, we present the averaged depth of recursion when a pruning takes place in Prune-GEACC. Specifically, we set $c_v \in [1, 10]$ and $|V| = 5$, $|U| = 10$ and $|V| = 5$, $|U| = 15$ respectively, and set the other parameters to default. The dash lines indicate the largest depths of recursions in the two settings, which is 50 when $|V| = 5$, $|U| = 10$ and 75 when $|V| = 5$, $|U| = 15$. We observe that the averaged depth pruned by Prune-GEACC

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering

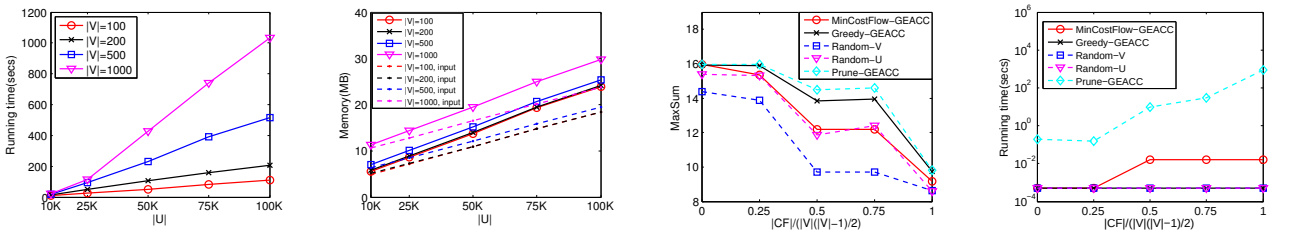


(a) MaxSum of varying c_v (b) MaxSum of varying c_u (c) MaxSum of varying distribution (d) MaxSum of real dataset (AKL)

(e) Runtime of varying c_v (f) Runtime of varying c_u (g) Runtime of varying distribution (h) Runtime of real dataset (AKL)

(i) Memory of varying c_v (j) Memory of varying c_u (k) Memory of varying distribution (l) Memory of real dataset (AKL)

Fig. 4: Results on varying capacity, distribution and on real dataset.



(a) Runtime of scalability test (b) Memory of scalability test (c) MaxSum of various methods (d) Runtime of various methods

Fig. 5: Study of scalability and effectiveness of approximate solutions.

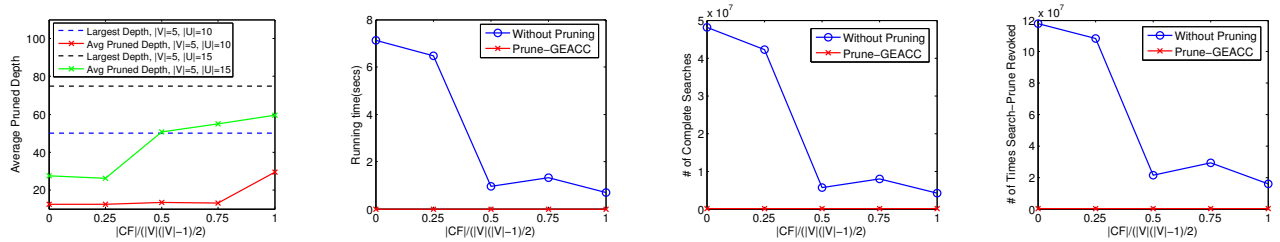
is quite small compared with the largest depth, indicating the effectiveness of pruning. In Fig. 6b, we present the running time of Prune-GEACC and that of exhaustive search without pruning, with $|V| = 5$, $|U| = 10$ and $c_v \in [1, 10]$. We can observe that Prune-GEACC is much more efficient than exhaustive search without pruning. In Fig. 6c, we present the number of complete searches, i.e. the number of times when the recursion reaches the largest depth possible and finds a complete matching. We observe that the number of complete searches of Prune-GEACC is much smaller than that of exhaustive search without pruning, indicating that many partial matchings are pruned by Prune-GEACC during recursion. Finally, in Fig. 6d, we present the number of times Search-GEACC is revoked, i.e. the number of times we enter a level of recursion. We observe again that Prune-GEACC revokes Search-GEACC much less often. In summary, the results in this part indicate that our pruning technique is quite effective.

Conclusion. Greedy-GEACC and MinCostFlow-GEACC are both efficient compared with the exact solution, and they yield acceptable approximate results. Greedy-GEACC outperforms MinCostFlow-GEACC in every aspect of $MaxSum$, running time and memory cost. Finally, Greedy-GEACC is effective and also scalable in both terms of time and space in practice.

6.3 Evaluation for OnlineGEACC

We next evaluate the online algorithm OnlineGreedy-GEACC. In particularly, we compare OnlineGreedy-GEACC with an online baseline algorithm, OnlineRand-GEACC, and the two offline approximate algorithms. OnlineRand-GEACC is basically OnlineGreedy-GEACC, but instead of picking events for a single user in a greedy way, it picks events randomly. In other words, OnlineRand-GEACC replaces line 5 of Algorithm 5 by visiting each $v \in V'$ in a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering



(a) Averaged pruned depth (b) Runtime (c) # of complete searches (d) # of times search revoked

Fig. 6: Performance of Prune-GEACC against the exact solution without pruning.

random-shuffled order. Note that OnlineRand-GEACC has the same competitive ratio with OnlineGreedy-GEACC.

Effect of cardinality. The first two columns of Fig. 7 show the results when varying $|V|$ and $|U|$. Similar to offline GEACC, we can observe that the $MaxSum$ values increase with increasing $|V|$ and $|U|$. Particularly, we can see that OnlineGreedy-GEACC, which performs much better than the online baseline algorithm, is nearly as good as the offline approximate algorithm Greedy-GEACC in terms of $MaxSum$. It indicates that our online algorithm can make quite good decisions in a dynamic environment. In Fig. 7e, we show the running time (in ms) and memory cost (in MB) of OnlineGreedy-GEACC and OnlineRand-GEACC when varying $|V|$, where the values of running time are indicated on the left and the ones of memory cost are on the right. We can observe that both the running time and memory cost increase as $|V|$ increases, which is reasonable as more events need to be considered and stored. Fig. 7f shows the running time and memory results when $|U|$ varies. One interesting observation is that the running time decreases as $|U|$ increases. The reason is as fewer events are available for users who come late, it takes much less time to arrange events for late-comers. Therefore, as $|U|$ increases, more users (the ones who come later) will need less time to get processed, and the average running time per user will decrease. Compared with OnlineRand-GEACC, we can see that OnlineGreedy-GEACC is slightly slower since it spends more time to visit the events in a sorted order.

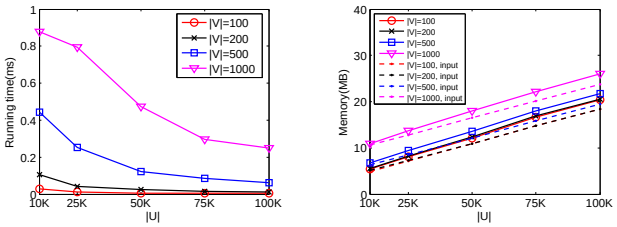
Effect of dimensionality. The third column of Fig. 7 shows the results when varying d. For $MaxSum$, we can observe that OnlineGreedy-GEACC is again nearly as good as the offline approximate algorithm Greedy-GEACC. As for running time and memory cost, we can observe that the values increase slightly as it takes more time to calculate the interestingness value and more space to store the attribute values when d increases.

Effect of conflict set size. The last column of Fig. 7 present the results when varying the number of conflicting event pairs. We can observe that similar to the offline scenario, the $MaxSum$ value of OnlineGreedy-GEACC decreases with increasing number of conflicting event pairs. Again, we can see that OnlineGreedy-GEACC can compete with the best offline approximate algorithm in terms of $MaxSum$ values. Particularly, we observe that the online algorithms are event much better than the offline approximate algorithm MinCostFlow-GEACC when there are a large number of conflicting event pairs. As for running time and memory cost, we can observe that they both increase when there are more conflicting event pairs, which is reasonable as when more events are conflicting, more information of conflicting event pairs needs to be stored and it is more difficult to find c_u events that are non-conflicting with each other and thus we need to visit more events for a user in both online algorithms.

Effect of capacity. The results when varying c_v and c_u are presented in the first two columns of Fig. 8. We can observe similar patterns of $MaxSum$ values of OnlineGreedy-GEACC as those of the offline algorithms. Fig. 8e shows the running time and memory cost results when varying c_v. We can observe that the online algorithms spend more time and memory when c_v increases. This is reasonable as many events are still available for late comers when c_v is large, and thus it still takes quite a long time to process late comers. When c_u increases, however, we can observe decreasing running time as Fig. 8f shows. The reason is similar to that of Fig. 7f, which is that less events are available for late comers as c_u increases and thus it takes less time to process late comers. Note that the fluctuation of memory cost is probably due to system performance fluctuation as the changes of values are insignificant.

Effect of distribution. In the Fig. 8c and 8g, we show the results when the attribute values are generated following Zipf distribution and the capacity values are generated following Normal distribution. We can observe that the patterns are similar to those when the values are generated following Uniform distribution.

Real dataset. The last column of Fig. 8 shows the results on the Auckland dataset. We can observe that OnlineGreedy-GEACC performs nearly as good as Greedy-GEACC in terms of $MaxSum$ values. As for running time and memory, we can observe similar patterns as Fig. 7h and 8g in general.

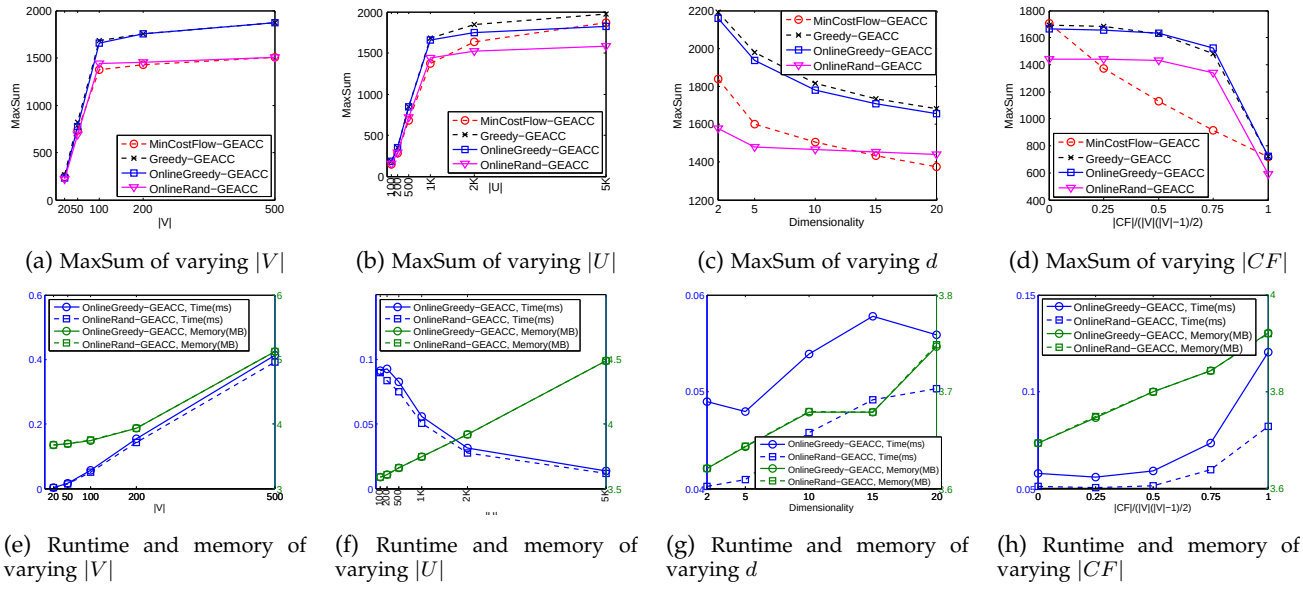


(a) Runtime of scalability test (b) Memory of scalability test

Fig. 9: Study of scalability of OnlineGreedy-GEACC.

Scalability. We finally study the scalability of OnlineGreedy-GEACC in Fig. 9. For running time, we can again observe that it decreases as $|U|$ increases. On the other hand, we can also see that OnlineGreedy-GEACC is

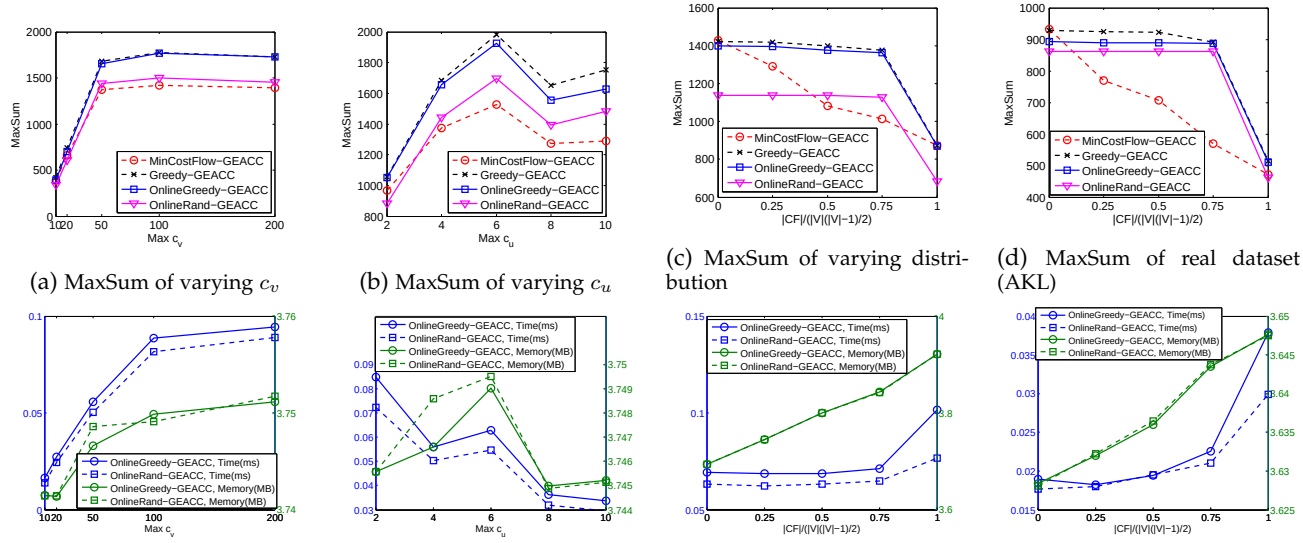
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2565468, IEEE Transactions on Knowledge and Data Engineering



(a) MaxSum of varying $|V|$ (b) MaxSum of varying $|U|$ (c) MaxSum of varying d (d) MaxSum of varying $|CF|$

(e) Runtime and memory of varying $|V|$ (f) Runtime and memory of varying $|U|$ (g) Runtime and memory of varying d (h) Runtime and memory of varying $|CF|$

Fig. 7: Results of OnlineGreedy-GEACC on varying cardinality, dimensionality and the size of conflict set.



(a) MaxSum of varying c_v (b) MaxSum of varying c_u (c) MaxSum of varying distribution (d) MaxSum of real dataset (AKL)

(e) Runtime and memory of varying c_v (f) Runtime and memory of varying c_u (g) Runtime and memory of varying distribution (h) Runtime and memory of real dataset (AKL)

Fig. 8: Results of OnlineGreedy-GEACC on varying capacity, distribution and on real dataset.

quite fast and scalable as it takes less than 1ms to process a user. As for memory cost, we can see that OnlineGreedy-GEACC takes quite small memory in addition to the input memory. The results show that OnlineGreedy-GEACC is quite scalable in both time and memory.

Conclusion. OnlineGreedy-GEACC is quite effective, even compared with the offline approximate algorithms which have full information of users. Also, OnlineGreedy-GEACC is very efficient and scalable. Finally, though OnlineGreedy-GEACC and OnlineRand-GEACC share the same competitive ratio, OnlineGreedy-GEACC is much better in practice due to its greedy strategy.

7 Related Work

Location and activity/event recommendation. This topic has been studied a lot in recent years due to the rising popularity of location-based social network (LBSN) and EBSN [11] [12] [1] [13] [14] [15] [16] [17] [18] [19]. However, such works focused on user-oriented recommendation. In other words, they focused on mining interests of each user in certain items (locations/events) and made recommendation in a single user's view. For instance, [18] recommended top-k recommend top-k POIs for a users query point in a given time interval so that the weighted sum of the spatial distance to the query point and the temporal aggregate in the time interval is minimized. Also, they did not consider conflicts and capacity of events/users. Our work is distinct from them in that we support event-participant arrangement in a globally systematic way so that to satisfy the interests of most users and consider conflicts and capacity of events/users. In addition, based on the classical influence maximization problem [20], [21] studied the problem of discovering influential event organizers in EBSNs. Different from their work that only considered event organizers,

our work focuses on how to make a globally satisfactory arrangement towards both sides of event organizers and users.

Bipartite matching and its variants. Assignment on bipartite graph has been a hot research topic for decades. The branch of bipartite matching problems most related to ours is maximum weighted bipartite matching [22] [23]. However, the original problem does not consider conflicts between nodes or capacity of nodes. Recent works [6] [24] [25] introduced capacity to nodes, but still they did not take conflicts of nodes into consideration. Notice that without the conflict constraint, the maximum weighted bipartite matching and the stable marriage problems with/without capacity constraints can be solved in polynomial time. However, our problem differs from previous works since our problem is much harder (NP-hard) due to the conflict constraints of nodes. Furthermore, the online scenario of the maximum weighted bipartite matching problem has also been studied [26] [10] [27]. However, these works also do not consider conflict constraints of nodes.

Event arrangement on EBSNs. Recently, the event arrangement problem has been studied a lot [2] [3] [4] [5], which all aim to find an arrangement among a set of users and a set of events to maximize the total or minimum satisfaction of users. Particularly, [2] considers interest and social relationship of users, [3] is the conference version of this paper, [4] considers spatio and temporal factors, and [5] try to maximize the satisfaction of the least satisfied user. However, all these works only consider offline scenarios.

8 Conclusion

In this paper, we identify a novel event-participant arrangement problem called *Global Event-participant Arrangement with Conflict and Capacity* (*GEACC*). We first analyze our differences compared with traditional matching problems and prove the NP-hardness of our problem. Then, we design an exact algorithm and two approximation algorithms. The exact algorithm is efficient for small datasets by means of a pruning rule. The MinCostFlow-GEACC approximation algorithm is not scalable to large datasets, and the Greedy-GEACC approximation algorithm runs significantly faster than MinCostFlow-GEACC while guarantees the same order of approximation ratio. In addition to the offline setting, we also study the online scenario of GEACC, called OnlineGEACC, where users arrive on the EBSN platforms in an online way. We further propose a competitive-ratio-guaranteed online algorithm for OnlineGEACC, called OnlineGreedy-GEACC. We conduct extensive experiments which verify the efficiency, effectiveness and scalability of the proposed approaches.

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 *KDD'12*.

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

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

[4] J. She, Y. Tong, and L. Chen, "Utility-aware social event-participant planning," in *SIGMOD'15*.

[5] Y. Tong, J. She, and R. Meng, "Bottleneck-aware arrangement over event-based social networks: the max-min approach," *World Wide Web*, 2015.

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

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

[8] H. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "idistance: An adaptive b+-tree based indexing method for nearest neighbor search," *ACM Transactions on Database Systems (TODS)*, 2005.

[9] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB'98*.

[10] H. Ting and X. Xiang, "Near optimal algorithms for online maximum edge-weighted b-matching and two-sided vertex-weighted b-matching," *Theoretical Computer Science*, 2015.

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

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

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

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

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

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

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

[18] Y. Sun, J. Qi, Y. Zheng, and R. Zhang, "K-nearest neighbor temporal aggregate queries," in *EDBT'15*.

[19] J. Qi, R. Zhang, L. Kulik, D. Lin, and Y. Xue, "The min-dist location selection query," in *ICDE'12*.

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

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

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

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

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

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

[26] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for on-line bipartite matching," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, 1990.

[27] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE'16*.

[28] U. Pferschy and J. Schauer, "The maximum flow problem with disjunctive constraints," *Journal of Combinatorial Optimization*, 2013.

Jieying She is currently a PhD student in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. Her major research interest is social network data management.

Yongxin Tong received the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology in 2014. He is currently an associate professor in the School of Computer Science and Engineering, Beihang University. He is a member of CCF, ACM, and IEEE. His research interests include crowdsourcing, uncertain data mining and management and social network analysis.

Lei Chen received the PhD degree in computer science from the University of Waterloo in 2005. He is currently an associate professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include crowdsourcing, social networks, probabilistic and uncertain data, cloud data processing, and graph data. He is a member of the ACM, the IEEE, and the IEEE Computer Society.

Caleb Chen Cao received the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology in 2014. His major research interest is crowdsourcing.