Conflict-Aware Event-Participant Arrangement

Jieying She, Yongxin Tong, Lei Chen, Caleb Chen Cao
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong SAR, PR China
{jshe,yxtong,leichen,caochen}@cse.ust.hk

***Abstract*—With the rapid development of Web 2.0 and Online To Offline (O2O) marketing model, various *online event-based social networks* (EBSNs), such as Meetup and Whova, 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, and 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. Though it is useful, unfortunately, we find that the GEACC problem is NP-hard due to the conflict constraints among events. Thus, we design two approximation algorithms with provable approximation ratios and an exact algorithm with pruning technique to address this problem. Finally, we verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.**

I. 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. Taking another example, Meetup[2] receives information on recruitment of attendees in group events, such as gatherings, sports activities, etc., and sends the event information to users. Such EBSNs facilitate organizing social events and ease the recruitment of group activity participants compared with that in the Web 1.0 age. Note that "participant" and "user" stand for the same meaning and they are used interchangeably in this paper.

Although existing EBSNs can improve effectiveness and efficiency of organizing social activities, most of them 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 the evening of Saturday, 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 these three sports, he can only attend at most one of them. In fact, many users in EBSNs usually encounter the same problem: *they have to confront with a confusing choice from many conflicting events.*

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

Furthermore, in EBSNs, event organizers tend to enroll more participants and participants look for arrangements with personally interesting events. However, existing approaches usually focus on the arrangement of each single event to a set of potential users. In such cases, apart from the aforementioned conflicting scenarios, the global satisfaction for both sides may not be optimized, which can be measured as the total interest scores between each event-participant pair of a given arrangement. In other words, *most existing EBSNs do not support event-participant arrangement in a global view.*

Therefore, it is appealing to have a new event-participant arrangement strategy that satisfies all conflicting constraints and globally optimizes the arrangement benefits, especially when arrangements are paid. To further illustrate this motivation, we go through a toy example as follows.

Example 1: Suppose we have three events v_1, v_2, v_3 and five users u_1, u_2, u_3, u_4, u_5 in an EBSN. We assume that each event/user is associated with a profile, which consists of a list of attributes. For events, some attributes can represent their preferences towards the participants, such as ages or gender, and the corresponding attributes of users characterize the users themselves. Likewise, some attributes can also represent users' preferences towards different events. Thus, each event/user is represented by a multi-dimensional attribute vector. Then, we can calculate a user's interest in an event based on the similarity between their attributes. TABLE I presents the interestingness values between each pair of event and user, as well as the conflicts between different events. In addition, each event/user may have 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 event capacities of 5, 3, and 2, and $u_1 - u_5$ have participant 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 for both v_1 and v_3. However, u_1 can only be assigned to one of v_1 and v_3 due to conflicts. 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 I, whose total interestingness values add up to 4.39.

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

TABLE I: 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

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 interestingness values over all the assigned pairs of event and user is maximized, while the capacity and conflicting constraints are satisfied.

It turns out that when there is no conflicting event constraint and all capacity values are set to one, the GEACC problem can be reduced to a classical problem, weighted bipartite graph matching[2][3]. Although the GEACC problem is quite related to traditional assignment or matching problems, we differ from them in that we introduce capacity to each event/user and thus study a many-to-many weighted matching problem. More importantly, we introduce the concept of conflicting events in our problem, which is new compared with traditional problems. Especially, as discussed later, the GEACC problem is actually NP-hard after introducing the conflicting event constraint, which is the main challenge to solve the problem. To the best of our knowledge, this is the first work that studies the GEACC problem. Therefore, we should design efficient algorithms specifically for our problem. We make the following contributions.

- 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 the GEACC problem 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. MinCostFlow-GEACC is not scalable for large datasets due to its quartic time complexity. Therefore, we further develop a greedy-based approximation algorithm, which is more efficient than MinCostFlow-GEACC and guarantees $\frac{1}{1+\alpha}$ worst-case approximation ratio. We also present an exact algorithm, which utilizes an effective pruning rule to reduce redundant search space.
- We verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.

The rest of the paper is organized as follows. In Section II, we formally formulate our problem and prove its NP-hardness. In Section III, we present two approximation algorithms with theoretically guaranteed approximation ratios. An exact solution with pruning is presented in Section IV. Extensive experiments on both synthetic and real datasets are presented in Section V. We review previous works in Section VI. We finally conclude this paper in Section VII.

II. Problem Statement

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 vector used to record attribute values of the event, and c_v is the capacity of the event, namely the maximum number of attendees of the event.

Similar to the definition of events, users are formally defined as follows.

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 vector to represent attribute values of the user, 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.

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 hidden attributes $\boldsymbol{l}_v$ of v and the hidden attributes $\boldsymbol{l}_u$ of u.

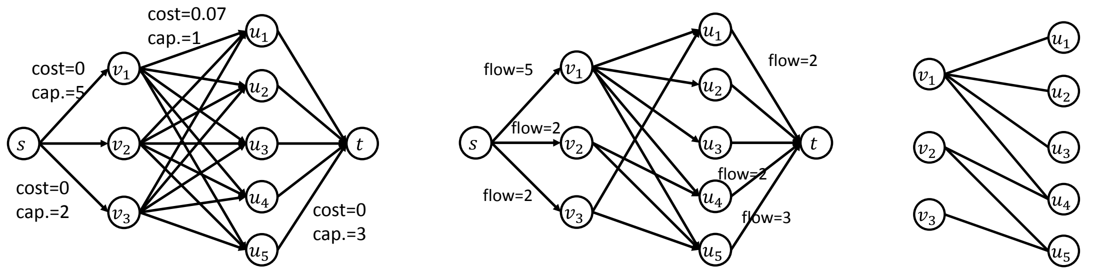
Particularly, we use Equation (1) as our similarity function in evaluation, where $\sqrt{dT^2}$ is the furthest Euclidean distance possible between any pair of $\boldsymbol{l_v}, \boldsymbol{l_u}$. Note that other similarity functions are applicable to our problem. We assume that $\max_u sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0, \forall v \in V$ and $\max_v sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0, \forall u \in U$.

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

We finally define our problem as follows.

Definition 5 (GEACC Problem): Given a set of events V, each v of which with maximum attendee capacity c_v and hidden attributes $\boldsymbol{l}_v$, a set of users U, each u of which with maximum number of assigned events c_u and hidden 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

- $\sum_u m(v, u) \leq c_v, \forall v \in V$ and $\sum_v m(v, u) \leq c_u, \forall u \in U$
- $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0, \forall \{v, u\} \in M$
- 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 $\{v_i, v_j\} \in CF$



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

Fig. 1: Illustrated example of MinCostFlow-GEACC.

Note that we assume that $\max_v c_v \leq |U|$ and $\max_u c_u \leq |V|$. Also note that "matching" and "arrangement" are used interchangeably in this paper. Example 1 in Section I explains the above definition. We next show the NP-hardness of the GEACC problem.

Theorem 1: The GEACC problem is NP-hard.

Proof: The maximum flow problem with conflict graph (MFCG) is NP-hard even if the network consists of only disjoint paths of length smaller or equal to three[4]. We reduce the MFCG with disjoint paths of length three (MFCGS) to the GEACC problem. The following is an instance of the MFCGS problem. We are given a directed connected graph $G = (N, A)$ with source node s, sink node t, and m disjoint paths $P_i : s \to p_{i,1} \to p_{i,2} \to t$, where $p_{i,1}, p_{i,2} \neq s, t$ and each arc (n_i, n_j) has capacity r_{n_i,n_j}. A conflict graph $H = (A, E)$ is also given with vertices corresponding to arcs of G s.t. if $(a, a\prime) \in E$, at most one of a and $a\prime$ can carry flow in a feasible solution. Without loss of generality, we only consider the instances of MFCGS where any two arcs $(a, a\prime) \in E$ are in two different paths $P, P\prime$, otherwise we can safely remove any path containing conflicting arcs without affecting the solution of the problem. The decision problem of MFCGS is to decide if there is a feasible flow F s.t. $|F| = k$.

We then construct an instance of the GEACC problem from the instance of the MFCGS problem accordingly:

(1) Each node in $\{p_{i,2}\}$ corresponds to an event in V, where the capacity of each event in V is set to 1.

(2) For any $p_{i,2}, p_{j,2}$, if there exist arcs $a_i \in P_i$ and $a_j \in P_j$ such that $(a_i, a_j) \in E$, the events v_i and v_j corresponding to $p_{i,2}$ and $p_{j,2}$ are conflicting with each other.

(3) The set of nodes $\{p_{i,1}\}$ correspond to U in the following way. For any $p_{i,1}, p_{j,1}$, if the events v_i and v_j corresponding to $p_{i,2}$ and $p_{j,2}$ are conflicting as in (2), $p_{i,1}$ and $p_{j,1}$ correspond to the same user u, namely $p_{i,1}$ and $p_{j,1}$ share u, the capacity of which is set to the number of $p_{i,1}$'s that share u. Each of the remaining $p_{i,1}$'s that do not share users with any other node corresponds to one user, the capacity of which is set to 1.

(4) Let the capacity r_{P_i} of each path P_i be $\min\{r_{s,p_{i,1}}, r_{p_{i,1},p_{i,2}}, r_{p_{i,2},t}\}$, and $R = \sum_i r_{P_i}$. The interestingness value between a pair of event and user is set to $\frac{r_{P_i}}{R}$ if the corresponding vertices in N are connected in P_i in G. Otherwise, the interestingness value is set to 0.

In this instance of the GEACC problem, we want to decide if we can find a feasible matching such that its $MaxSum$ is $\frac{k}{R}$. It is easy to see that the MFCGS instance is YES if and only if the GEACC instance is YES, which completes the proof. ■

III. Approximate Solutions for GEACC

In this section, we present two approximation algorithms for the GEACC problem. The first one has a larger theoretical approximation ratio, but is not scalable for large datasets. Thus to overcome the scalability issue, we propose the second more efficient approximation algorithm that guarantees a slightly lower approximation ratio.

A. MinCostFlow-GEACC Approximation Algorithm

The idea of the first approximation algorithm MinCostFlow-GEACC is to first ignore the conflict condition and try to find a matching with maximum sum of interestingness values, and then resolve the conflict issues in the matching 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. To resolve the conflicting events after obtaining a temporary matching that may contain conflicts, 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 $a_F(u, t).capacity = c_u$. 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

```
Algorithm 1: MinCostFlow-GEACC
  input : $V, U, \{c_v\}, \{c_u\}, \{\boldsymbol{l}_v\}, \{\boldsymbol{l}_u\}, CF$
  output: A feasible arrangement $M$
1 construct $G_F = (N_F, A_F)$;
2 $m_\emptyset(v,u) \leftarrow 0, \forall v \in V, u \in U$;
3 foreach $\Delta \leftarrow \Delta_{min}$ to $\Delta_{max}$ do
4     $F^\Delta \leftarrow$ MinCostFlow($G_F, \Delta$);
5     construct $M_\emptyset^\Delta$ accordingly;
6     if $MaxSum(M_\emptyset^\Delta) > MaxSum(M_\emptyset)$ then
7         $M_\emptyset \leftarrow M_\emptyset^\Delta$;
8 $m(v,u) \leftarrow 0, \forall v \in V, u \in U$;
9 foreach $u \in U$ do
10    $L \leftarrow$ sorted list of $\{v | m_\emptyset(v,u) = 1\}$ in
      non-increasing order of $sim(\boldsymbol{l}_v, \boldsymbol{l}_u)$;
11    for $i \leftarrow 1$ to $|L|$ do
12        $v_i^L \leftarrow$ the $i$-th element of $L$ ;
13        if $v_i^L$ does not conflict with u's matched events
          in $M$ then
14            $m(v,u) \leftarrow 1$;
15 return $M$
```

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 $CF = \emptyset$.

After obtaining $M_\emptyset$, our second step of MinCostFlow-GEACC is then to resolve the conflicts in $M_\emptyset$. For each $u \in U$, our task is to 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[5]. Therefore, instead of finding the optimal independent set, 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 of MinCostFlow-GEACC is illustrated in Algorithm 1. In lines 1-7, 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 matching $M_\emptyset$ for the GEACC instance without considering conflicting events. In the second step, we obtain a feasible matching M by resolving the conflicting events for each u in lines 8-14. Particularly, at each iteration, we greedily add the most similar pair possible by checking its conflicts with the selected pairs in lines 12-14.

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.

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

Proof: Suppose to the contrary, there exists another matching $M'_\emptyset$ for the GEACC instance with $CF = \emptyset$ such that $MaxSum(M'_\emptyset) > MaxSum(M_\emptyset)$.

Let $M_\emptyset^\star$ initially be a copy of $M'_\emptyset$. We modify $M_\emptyset^\star$ as follows. If $\sum_{v \in V, u \in U} m_\emptyset^\star(v,u) < \Delta_{min}$, there must exist some pair $\{v', u'\}$ such that neither v' nor u' is fully occupied and $m_\emptyset^\star(v', u') = 0$. For each of such pairs, we assign v' to u' in $M_\emptyset^\star$, i.e. changing $m_\emptyset^\star(v', u')$ to one, regardless of whether $sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'}) = 0$ or not. Therefore, we obtain a $M_\emptyset^\star$ such that $\Delta = \sum_{v \in V, u \in U} m_\emptyset^\star(v,u) \geq \Delta_{min}$ and $MaxSum(M_\emptyset^\star) \geq MaxSum(M'_\emptyset)$. Notice that it must hold that $\Delta \leq \Delta_{max}$. Recall that $M_\emptyset^\Delta$ is the matching obtained from the minimum cost flow with amount Δ. Since $MaxSum(M_\emptyset) \geq MaxSum(M_\emptyset^\Delta)$, it holds that $MaxSum(M_\emptyset^\star) > MaxSum(M_\emptyset^\Delta)$.

Now we construct a flow $F^\star = \{flow^\star(a_F)\}$ on G_F from $M_\emptyset^\star$ as follows. Let $flow^\star(v,u) = m_\emptyset^\star(v,u)$ for each $\{v,u\}$ pair, $flow^\star(s,v) = \sum_{u \in U} m_\emptyset^\star(v,u)$ for each $\{s,v\}$ pair, and $flow^\star(u,t) = \sum_{v \in V} m_\emptyset^\star(v,u)$ for each $\{u,t\}$ pair. It is easy to see that $F^\star$ is feasible with a total amount of Δ flows. Thus,

$$\begin{aligned}
F^\star.cost &= \sum_{v \in V, u \in U} flow^\star(v,u) \times (1 - sim(\boldsymbol{l}_v, \boldsymbol{l}_u)) \\
&= \Delta - \sum_{v \in V, u \in U} m_\emptyset^\star(v,u) \times sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \\
&< \Delta - \sum_{v \in V, u \in U} m_\emptyset^\Delta(v,u) \times sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \\
&= \Delta - \sum_{v \in V, u \in U, sim(\boldsymbol{l}_v, \boldsymbol{l}_u) > 0} flow^\Delta(v,u) \times sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \\
&\quad - \sum_{v \in V, u \in U, sim(\boldsymbol{l}_v, \boldsymbol{l}_u) = 0} flow^\Delta(v,u) \times sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \\
&= F^\Delta.cost
\end{aligned}$$

We obtain a feasible flow $F^\star$ with cost less than F^Δ, contradicting with the fact that F^Δ is the minimum cost flow on G_F with amount of Δ flows. It follows that $M_\emptyset$ is optimal for the GEACC instance with $CF = \emptyset$. ■

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: For each u that is matched to at least one event in M, let v_u^{max} denote the most interesting event matched to her/him, i.e. $v_u^{max} = \arg\max_v \{sim(\boldsymbol{l}_v, \boldsymbol{l}_u) | m(v,u) = 1\}$. Obviously, $sim(\boldsymbol{l}_{v_u^{max}}, \boldsymbol{l}_u) \times c_u \geq \sum_{v | m_\emptyset(v,u)=1} sim(\boldsymbol{l}_v, \boldsymbol{l}_u)$. Notice that $m(v_u^{max}, u) = m_\emptyset(v_u^{max}, u) = 1$ as v_u^{max} is always selected in Algorithm 1 if u is matched in M (and thus is also matched in $M_\emptyset$). Thus, it holds that

$$\begin{aligned} MaxSum(M) &= \sum_{u|v_u^{max}\exists} \sum_{v|m(v,u)=1} sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \\ &\geq \sum_{u|v_u^{max}\exists} sim(\boldsymbol{l}_{v_u^{max}}, \boldsymbol{l}_u) \\ &\geq \sum_{u|v_u^{max}\exists} \frac{\sum_{v|m_\emptyset(v,u)=1} sim(\boldsymbol{l}_v, \boldsymbol{l}_u)}{c_u} \\ &\geq \frac{\sum_{u|v_u^{max}\exists} \sum_{v|m_\emptyset(v,u)=1} sim(\boldsymbol{l}_v, \boldsymbol{l}_u)}{\max c_u} \\ &= \frac{MaxSum(M_\emptyset)}{\max c_u} \geq \frac{MaxSum(M_{OPT})}{\max c_u} \end{aligned}$$

■

Complexity Analysis. For the first step of MinCostFlow-GEACC, numerous algorithms have been proposed for the MCF problem. [6] pointed out that Successive Shortest Path Algorithm (SSPA) is the one suitable for large-scale data and many-to-many matching with real-valued arc costs. Then the first step takes $O((\Delta_{max}^2 - \Delta_{min}^2)((|V| \times |U| + (|V| + |U|)) \log(|V| + |U|)))$ time. For the second step, the time complexity is $O(|U|((\max c_u) \log(\max c_u) + (\max c_u)^2))$, where $O((\max c_u) \log(\max c_u)$ is the cost of line 10 and $O((\max c_u)^2)$ is the cost of lines 11-14. 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 $O((\Delta_{max}^2 - \Delta_{min}^2)((|V| \times |U| + (|V| + |U|)) \log(|V| + |U|)) + |U|((\max c_u) \log(\max c_u) + (\max c_u)^2))$.

B. Greedy-GEACC Approximation Algorithm

MinCostFlow-GEACC could be inefficient when the scale of data is large. In this subsection, we present a more efficient algorithm, Greedy-GEACC, with a slightly lower approximation ratio compared with that of MinCostFlow-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 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 $\{v, u\}$

Algorithm 2: Greedy-GEACC

```
   input : $V, U, \{c_v\}, \{c_u\}, \{\boldsymbol{l}_v\}, \{\boldsymbol{l}_u\}, CF$
   output: A feasible arrangement $M$
1  $H \leftarrow \emptyset$;
2  foreach $v \in V$ do
3      $u_{nn} \leftarrow v$'s first NN in $U$;
4      push $\{v, u_{nn}\}$ into $H$;
5  foreach $u \in U$ do
6      $v_{nn} \leftarrow u$'s first NN in $V$;
7      if $\{v_{nn}, u\} \notin H$ then
8          push $\{v_{nn}, u\}$ into $H$;
9  heapify $H$;
10 $m(v, u) \leftarrow 0, \forall v \in V, u \in U$;
11 while $H \neq \emptyset$ 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         $m(v, u) \leftarrow 1$;
15         decrease $c_v, c_u$ by 1;
16     if $c_v > 0$ then
17         $u_{nn} \leftarrow v$'s next feasible unvisted NN;
18         if $u_{nn}\exists$ and $\{v, u_{nn}\} \notin H$ then
19             push $\{v, u_{nn}\}$ into $H$;
20     if $c_u > 0$ then
21         $v_{nn} \leftarrow u$'s next feasible unvisited NN;
22         if $v_{nn}\exists$ and $\{v_{nn}, u\} \notin H$ then
23             push $\{v_{nn}, u\}$ into $H$;
24 return $M$
```

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.

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: In the initialization step, for each v, exactly one pair $\{v, u_{nn}\}$ incident to v is pushed into H. For each u, either one pair $\{v_{nn}, u\}$ incident to u is pushed into H or $\{v_{nn}, u\}$ is already in H. Therefore, Lemma 2 holds before entering the iteration procedure of Greedy-GEACC. Then at each iteration, whenever a pair $\{v, u\}$ is popped from H, either a new pair $\{v, u_{nn}\}(\{v_{nn}, u\})$ for $v(u)$ is already in or pushed into H, or $v(u)$ becomes a finished node. It follows that at least one edge incident to $v(u)$ is in H before proceeding to the next iteration if $v(u)$ is unfinished. ∎

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 $\{v', u'\} \in \{\{v', u'\} | v', u'$ are unfinished$\}$.

Proof: For any feasible unvisited pair $\{v', u'\}$ whose v', u' are both unfinished, it is either in H or has not yet been pushed into H. Obviously, if $\{v', u'\} \in H$, $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'})$. It remains to analyze the case when $\{v', u'\}$ has not yet been pushed into H. In such case, Lemma 2 indicates that at least one pair $\{v', u'_{nn}\}(\{v'_{nn}, u'\})$ incident to $v'(u')$ is in H. It is easy to see that $sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'_{nn}}) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'})$ and $sim(\boldsymbol{l}_{v'_{nn}}, \boldsymbol{l}_{u'}) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'})$ from the way we push pairs into H. Thus, $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'_{nn}}) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'})$ and $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq sim(\boldsymbol{l}_{v'_{nn}}, \boldsymbol{l}_{u'}) \geq sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'})$. ∎

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: First, it is easy to see that each pair $\{v, u\}$ is pushed into H for at most once. Since each $v(u)$ has $|U|(|V|)$ neighbors in $U(V)$, it follows that a finite number of pairs are pushed into H. It follows that H becomes empty after a finite number of iterations. ∎

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: For each unmatched pair $\{v, u\}$, it is either visited or has never been pushed into H. In the first case, $\{v, u\}$ is unmatched either because v or u or both are fully occupied, or because $\{v, u\}$ conflicts with the existing matching. Thus, $\{v, u\}$ cannot be added to the current matching. In the latter case, since $\{v, u\}$ has never been pushed into H, it follows that u is NOT a feasible unvisited neighbor of v or v is NOT a feasible unvisited neighbor of u. Thus, $\{v, u\}$ cannot be added to the current matching. It follows that Lemma 5 holds. ∎

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.

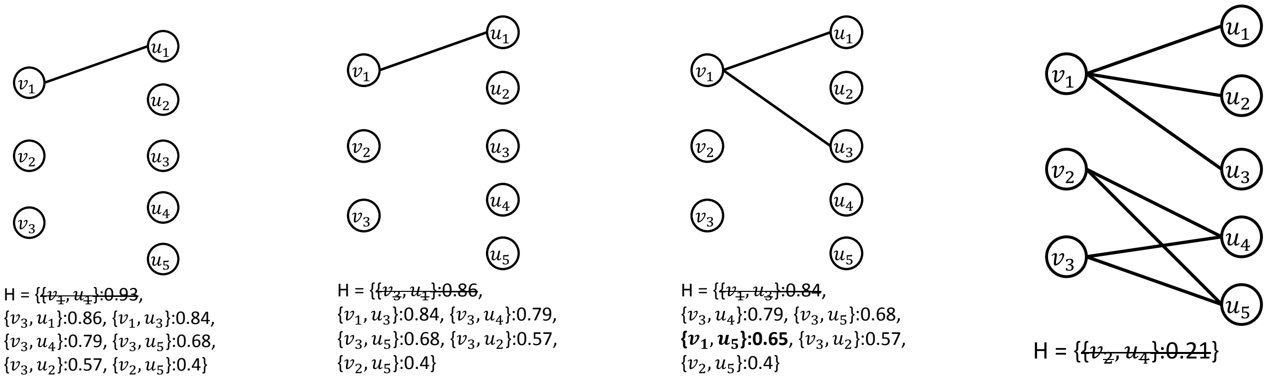
Approximation Ratio. We next study the approximation ratio of Greedy-GEACC.

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: For any $\{v, u\} \in M$, i.e. $m(v, u) = 1$, either (1) $m_{OPT}(v, u) = 1$, or (2) $m_{OPT}(v, u) = 0$, and there are at most $\max c_u$ pairs $\{v'_1, u\}, \{v'_2, u\}, \cdots, \{v'_k, u\}, 0 \leq k \leq \max c_u$ that are matched in M_{OPT} but unmatched in M because of $\{v, u\}$ (due to conflicts with $\{v, u\}$ or capacity constraint as $\{v, u\}$ occupies one capacity of u and $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq sim(\boldsymbol{l}_{v'_i}, \boldsymbol{l}_u), 1 \leq i \leq k$), and there is at most one pair $\{v, u'\}$ that is matched in M_{OPT} but unmatched in M due to $\{v, u\}$ (since $\{v, u\}$ occupies one capacity of v and $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq sim(\boldsymbol{l}_v, \boldsymbol{l}_{u'})$), and thus there are at most $1 + \max c_u$ pairs that are matched in M_{OPT} but unmatched in M because of $\{v, u\}$. It follows that $sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \geq \frac{1}{1+\max c_u}(sim(\boldsymbol{l}_v, \boldsymbol{l}_{u'}) + \sum_i sim(\boldsymbol{l}_{v'_i}, \boldsymbol{l}_u))$. Notice that for each $\{v', u'\} \in M_{OPT} \setminus M$, there must be at least one pair $\{v, u\} \in M \setminus M_{OPT}$ that is "responsible for" the "unmatched case" of $\{v', u'\}$ in M due to (2), otherwise $\{v', u'\}$ would have been added into M in Greedy-GEACC. Then we have

$$\begin{aligned}
MaxSum(M) &= MaxSum(M \cap M_{OPT}) + MaxSum(M \setminus M_{OPT}) \\
&= MaxSum(M \cap M_{OPT}) + \sum_{\{v,u\} \in M \setminus M_{OPT}} sim(\boldsymbol{l}_v, \boldsymbol{l}_u) \\
&\geq MaxSum(M \cap M_{OPT}) \\
&\quad + \frac{1}{1+\max c_u} \sum_{\{v',u'\} \in M_{OPT} \setminus M} sim(\boldsymbol{l}_{v'}, \boldsymbol{l}_{u'}) \\
&= MaxSum(M \cap M_{OPT}) \\
&\quad + \frac{1}{1+\max c_u} MaxSum(M_{OPT} \setminus M) \\
&> \frac{1}{1+\max c_u} (MaxSum(M \cap M_{OPT}) \\
&\quad + MaxSum(M_{OPT} \setminus M)) \\
&= \frac{1}{1+\max c_u} MaxSum(M_{OPT})
\end{aligned} \tag{2}$$

∎



v_1 v_2 v_3 u_1 u_2 u_3 u_4 u_5
H = {~~$\{v_1, u_1\}$:0.93~~, $\{v_3, u_1\}$:0.86, $\{v_1, u_3\}$:0.84, $\{v_3, u_4\}$:0.79, $\{v_3, u_5\}$:0.68, $\{v_3, u_2\}$:0.57, $\{v_2, u_5\}$:0.4}
H = {~~$\{v_3, u_1\}$:0.86~~, $\{v_1, u_3\}$:0.84, $\{v_3, u_4\}$:0.79, $\{v_3, u_5\}$:0.68, $\{v_3, u_2\}$:0.57, $\{v_2, u_5\}$:0.4}
H = {~~$\{v_1, u_3\}$:0.84~~, $\{v_3, u_4\}$:0.79, $\{v_3, u_5\}$:0.68, **$\{v_1, u_5\}$:0.65**, $\{v_3, u_2\}$:0.57, $\{v_2, u_5\}$:0.4}
H = {~~$\{v_2, u_4\}$:0.21~~}

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

Fig. 2: Illustrated example of Greedy-GEACC.

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[7] and VA-File[8]. 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 the time cost in 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 $O(|V|\sigma(V)+|U|\sigma(U)+|V|+|U|+|V||U|(\sigma(V)+\sigma(U)+\log(|V|+|U|)))$ time in worst case.

IV. 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 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

$$sum_{max}(v_i^L, u_{i,j}) = MaxSum(M_{visited}) + \sum_{i+1\le k\le|V|} s_{v_k^L} \times c_{v_k^L} + sim(\boldsymbol{l}_{v_i^L}, \boldsymbol{l}_{u_{i,j}}) \times c_{v_i^L, remain} \quad (3)$$

Algorithm 3: Prune-GEACC

```
  input : $V, U, \{c_v\}, \{c_u\}, \{\boldsymbol{l}_v\}, \{\boldsymbol{l}_u\}, CF$
  output: A feasible arrangement $M$
1 $M \leftarrow$ Greedy-GEACC();
2 foreach $v \in V$ do
3     $u_{v,1} \leftarrow$ 1-NN of $v$;
4     $s_v \leftarrow sim(\boldsymbol{l}_v, \boldsymbol{l}_{u_{v,1}})$;
5 $L \leftarrow$ sorted list of $v$ in non-increasing order of $s_v \times c_v$;
6 $sum_{remain} \leftarrow \sum_{2\le k\le|V|} s_{v_k^L} \times c_{v_k^L}$;
7 $m_c(v, u) \leftarrow 0, \forall v \in V, u \in U$;
8 Search-GEACC(1, 1);
9 return $M$
```

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

Proof: Let M_{remain} denote the partial matching on the remaining unvisited pairs. Note that M_{remain} has no effect on $M_{visited}$. Thus, for matching $M' \supseteq M_{visited}$, $MaxSum(M') = MaxSum(M_{visited}) + MaxSum(M_{remain})$. It is easy to see that for each $v_k^L (i + 1 \le k \le |V|)$, it holds that $s_{v_k^L} \times c_{v_k^L} \ge \sum_u m_{remain}(v_k^L, u) \times sim(\boldsymbol{l}_{v_k^L}, \boldsymbol{l}_u)$. For v_i^L, it holds that $sim(\boldsymbol{l}_{v_i^L}, \boldsymbol{l}_{u_{i,j}}) \times c_{v_i^L, remain} \ge \sum_{j\le l\le|U|} m_{remain}(v_i^L, u_{i,l}) \times sim(\boldsymbol{l}_{v_i^L}, \boldsymbol{l}_{u_{i,l}})$. Therefore, it follows that $MaxSum(M_{best}) \ge sum_{max}(v_i^L, u_{i,j}) \ge MaxSum(M_{visited}) + MaxSum(M_{remain}) = MaxSum(M')$. Since $MaxSum(M_{best}) \le MaxSum(M_{OPT})$, it follows that $MaxSum(M') \le MaxSum(M_{OPT})$. ■

Lemma 6 indicates that when we are about to visit a pair $\{v_i^L, u_{i,j}\}$ during the recursion process, if $sum_{max}(v_i^L, u_{i,j}) \le MaxSum(M_{best})$, 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

Algorithm 4: Search-GEACC

```
   input : $v_{id}$, $u_{id}$
1  $v \leftarrow v^L_{v_{id}}$;
2  $u \leftarrow u_{id}$-NN of $v$;
3  if $c_v > 0$ and $c_u > 0$ and $v$ does not conflict with $u$'s matched events then
4      $m_c(v, u) \leftarrow 1$;
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             $sum_{remain} \leftarrow sum_{remain} - s_{v^L_{v_{id}+1}} \times c_{v^L_{v_{id}+1}}$;
12             Search($v_{id} + 1$, 1);
13             recover $sum_{remain}$;
14     else
15         $u_{nn} \leftarrow (u_{id} + 1)$-NN of $v$;
16         if $MaxSum(M_c) + sum_{remain} + sim(\boldsymbol{l}_v, \boldsymbol{l}_{u_{nn}}) \times c_v > MaxSum(M)$ then
17             Search($v_{id}$, $u_{id} + 1$);
18     $m_c(v, u) \leftarrow 0$;
19     increase $c_v, c_u$ by 1;
20 Same as lines 6-17;
```

calculate $sum_{max}(v^L_i, u_{i,j})$ by summing over all unvisited pairs. We maintain $sum_{remain} = \sum_{i+1 \le k \le |V|} s_{v^L_k} \times c_{v^L_k}$, and $sum_{max}(v^L_i, u_{i,j}) = MaxSum(M_{visited}) + sum_{remain} + sim(\boldsymbol{l}_{v^L_i}, \boldsymbol{l}_{u_{i,j}}) \times c_{v^L_i, remain}$. sum_{remain} is maintained as follows. Initially, $sum_{remain} = \sum_{2 \le k \le |V|} s_{v^L_k} \times c_{v^L_k}$. Whenever we are about to visit $\{v^L_i, u_{i,1}\}$, we simply subtract $s_{v^L_i} \times c_{v^L_i}$ 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 $v^L_{v_{id}+1}$ in L and enumerate the states of pair $v^L_{v_{id}+1}$ 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.

TABLE II: Real Dataset

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

V. Evaluation

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 a set of tags and a location. Each event in the dataset is also associated with a location. The events are not explicitly associated with tags. Note that each event is created by a "group" on Meetup, which can be viewed as a community, and each group is associated with a set of tags. Thus, for each event, we use the tags of the group who creates it as the tags of the event itself. Since the tags are created by users, some of them have misspellings and different tags referring to the same thing are used by users. To address this problem, we merge the tags with the same meaning and select 20 most popular tags as attributes of users/events, where each initial attribute value is the number of original tags associated with the user/event that refer to the same merged tag. For example, if a user/event is tagged with "outdoor-activities" and "outdoor-lovers-and-travel-lovers", both of which refer to the same merged tag "outdoor", then the user/event has initial value of 2 for attribute "outdoor". We further normalize each attribute value by the total number of original tags associated with the user/event. For example, if the user/event with initial value of 2 for attribute "outdoor" is associated with 10 original tags in total, the final value of attribute "outdoor" will be 0.2 for the user/event. Notice that it is unlikely for a user living in a city to attend a meet-up event held in another city. Therefore, 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 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 II 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 III, where we mark our default settings in bold font. Note that all generated capacity values are converted into integers.

Baselines. We use two random algorithms as baselines. For the first baseline, Random-V, we iterate over each $v \in V$, and

TABLE III: Synthetic Dataset

Factor	Setting
$\lvert V \rvert$	20, 50, **100**, 200, 500
$\lvert U \rvert$	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: $\mu = 2$, $\sigma = 1$
$\frac{\lvert CF \rvert}{\lvert V \rvert(\lvert V \rvert-1)/2}$	0, **0.25**, 0.5, 0.75, 1
Scalability	$\lvert V \rvert = 100, 200, 500, 1000$ $\lvert U \rvert =$ 10K, 25K, 50K, 75K, 100K

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.

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. 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.

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 outperforms in every aspect. Greedy-GEACC consumes as less as running time and space as the baselines do, and returns matchings with the largest $MaxSum$. Though Greedy-GEACC has a theoretically lower approximation ratio than that of MinCostFlow-GEACC, in practice Greedy-GEACC can outperform MinCostFlow-GEACC in terms of $MaxSum$. The reason is that the worst case of Greedy-GEACC as analyzed in Section III.B can rarely happen. Thus, Greedy-GEACC can perform much better in practice. 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 little 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.

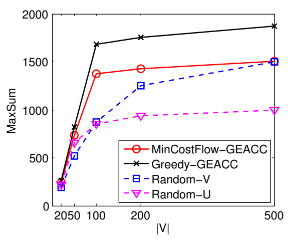
Effect of capacity. We next study the effect of capacity of events and users. We first study the results when c_v varies, which are shown in the first column of Fig. 4. The values of c_v are generated uniformly in range [1, $\max c_v$], where $\max c_v$ varies in our experiment. Thus, when $\max c_v$ 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.

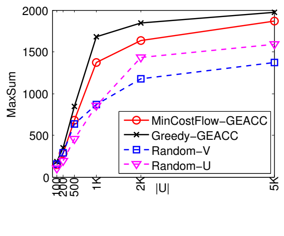
Effect of distribution. The third column of Fig. 4 shows the results when we generate the synthetic data according to different distributions. Specifically, we present the results when the attribute values are generated following Zipf distribution and the capacity values are generated following Normal distribution. We observe that the general patterns of data generated by different distributions are similar in every aspect. Therefore, 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 respectively 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.

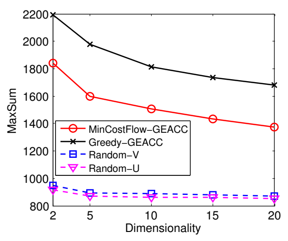
Real dataset. The last column of Fig. 4 shows the results on real dataset (Auckland) when the capacity values are generated following Uniform distribution. Notice that the results on real dataset have similar patterns to those of the synthetic data. Similar patterns are observed on the other two real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



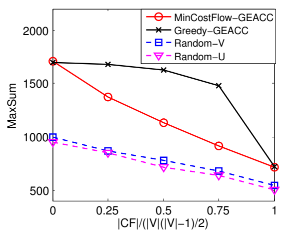
(a) MaxSum of varying $|V|$



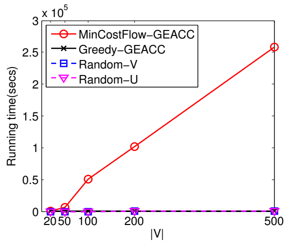
(b) MaxSum of varying $|U|$



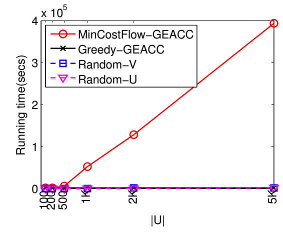
(c) MaxSum of varying d



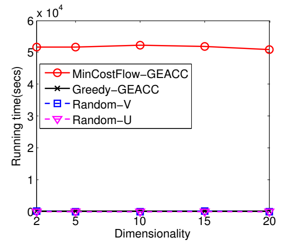
(d) MaxSum of varying $|CF|$



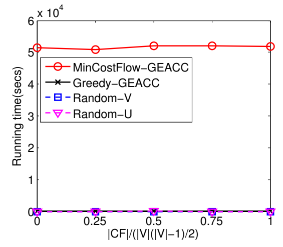
(e) Runtime of varying $|V|$



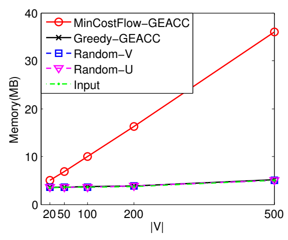
(f) Runtime of varying $|U|$



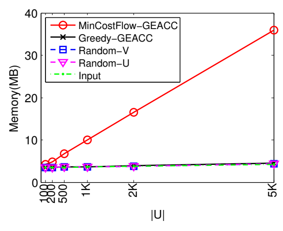
(g) Runtime of varying d



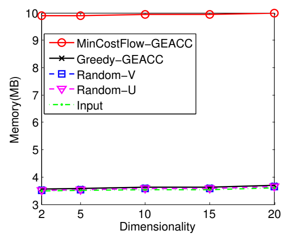
(h) Runtime of varying $|CF|$



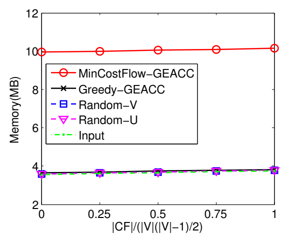
(i) Memory of varying $|V|$



(j) Memory of varying $|U|$



(k) Memory of varying d



(l) Memory of varying $|CF|$

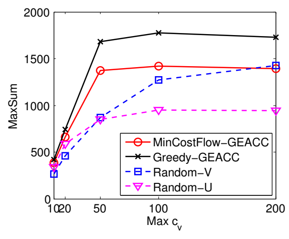
Fig. 3: Results on varying cardinality, dimensionality and the size of conflict set.

and when the capacity values are generated following Normal distribution, and we omit the results due to limited space.

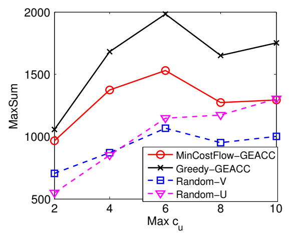
Scalability. MinCostFlow-GEACC is not efficient enough according to our previous experiment results. Thus, we study the scalability of Greedy-GEACC in this part. The results are shown in Fig. 5a and 5b. Specifically, we set $|V| = 100, 200, 500, 1000$ respectively, and vary the size of $|U|$. Since $|U|$ 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 $c_v \sim$ Uniform$[1, 10]$. The other parameters are set to default. In Fig. 5c, we compare the approximated $MaxSums$ 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 $MaxSums$ returned by Greedy-GEACC are quite close to the optimal ones, indicating that Greedy-GEACC returns quite good results in practice though with a theoretically lower approximation ratio. 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.

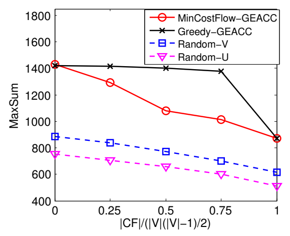
Effectiveness of pruning. We finally study the effectiveness of our pruning technique, whose results are shown in Fig. 6. In Fig. 6a, we present the averaged depth of recursion when a pruning takes place in Prune-GEACC. Specifically, we set $c_v \in [1, 10]$ and $|V| = 5$, $|U| = 10$ and $|V| = 5$, $|U| = 15$ 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 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,



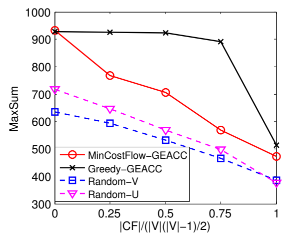
(a) MaxSum of varying c_v



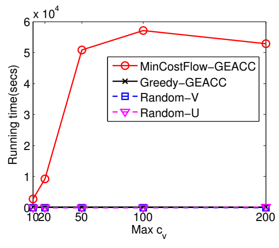
(b) MaxSum of varying c_u



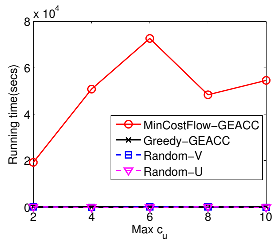
(c) MaxSum of varying distribution



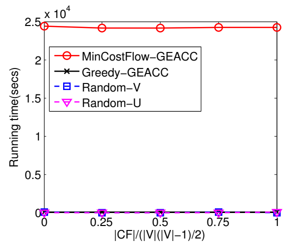
(d) MaxSum of real dataset (AKL)



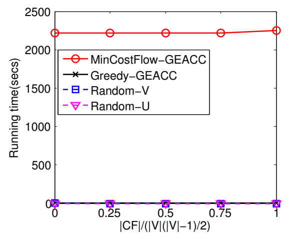
(e) Runtime of varying c_v



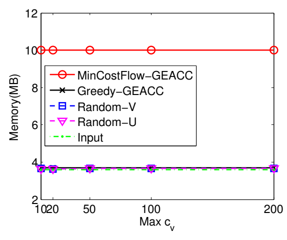
(f) Runtime of varying c_u



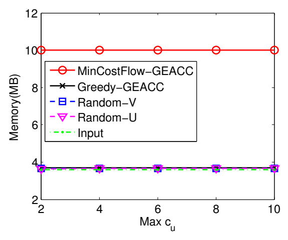
(g) Runtime of varying distribution



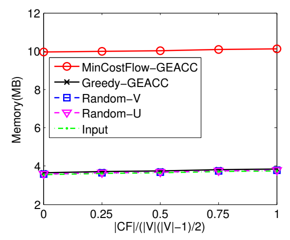
(h) Runtime of real dataset (AKL)



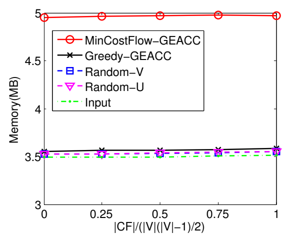
(i) Memory of varying c_v



(j) Memory of varying c_u

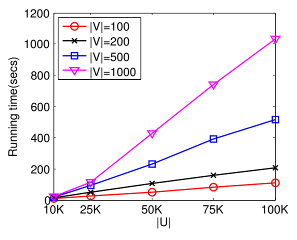


(k) Memory of varying distribution

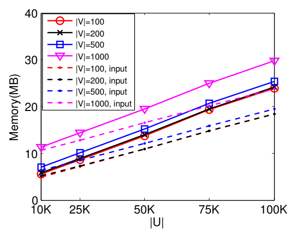


(l) Memory of real dataset (AKL)

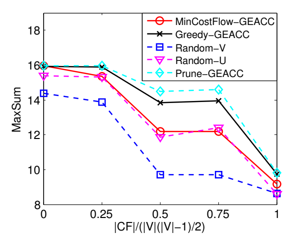
Fig. 4: Results on varying capacity, distribution and on real dataset.



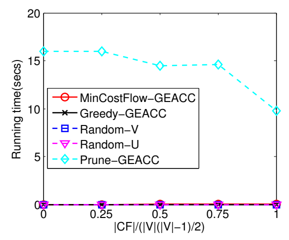
(a) Runtime of scalability test



(b) Memory of scalability test



(c) MaxSum of various methods



(d) Runtime of various methods

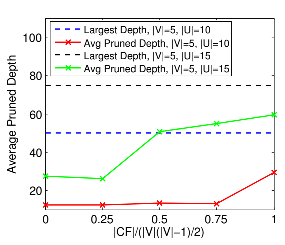
Fig. 5: Study of scalability and effectiveness of approximate solutions.

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.

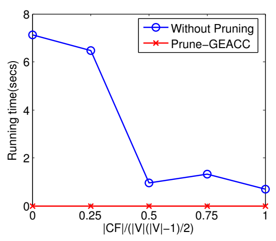
Conclusion. Greedy-GEACC and MinCostFlow-GEACC are both efficient compared with the exact solution, and they yield acceptable approximate results. Greedy-GEACC, though with a theoretically lower approximation ratio than that of MinCostFlow-GEACC due to its worst-case approximation ratio, outperforms 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.

VI. Related Work

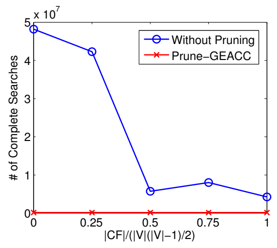
Location and activity/event recommendation. This topic has been studied a lot in recent years due to the rising popularity of location-based social network (LBSN) and EBSN [9][10][1][11][12][13][14][15]. However, such works focused on user-oriented recommendation. In other words, they focused on mining interests of each user in certain items (locations/events) and made recommendation in a single user's view. 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



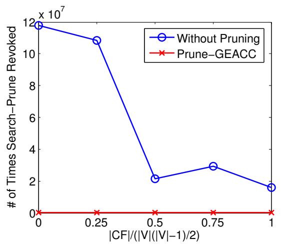
(a) Averaged pruned depth



(b) Runtime



(c) # of complete searches



(d) # of times search revoked

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

conflicts and capacity of events/users. In addition, based on the classical influence maximization problem[16], [17] 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.

Travel route recommendation. Travel route recommendation [18][19] is a technique that recommends one or a series of landmarks to travelers, which is related to our problem since conflicts between landmarks should be considered. However, we differ from such works in that we study a global arrangement problem for all events and users instead of mining interests of users and making recommendation to a single user.

Bipartite graph matching. 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 [2][3]. However, the original problem does not consider conflicts between nodes or capacity of nodes. Recent works [6][20] 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 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.

VII. 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 obtains a tight approximation ratio but is not scalable to large dataset due to its quartic time complexity. In order to enhance the scalability, we propose the Greedy-GEACC approximation algorithm which runs significantly faster than MinCostFlow-GEACC and guarantees a slightly lower approximation ratio. We conduct extensive experiments which verify the efficiency, effectiveness and scalability of the proposed approaches.

Acknowledgment

We are grateful to anonymous reviewers for their constructive comments on this work. This work is supported in part by the Hong Kong RGC Project N_HKUST637/13, National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, NSFC Grant No. 61232018, Microsoft Research Asia Gift Grant, Google Faculty Award 2013, and Microsoft Research Asia Fellowship 2012.

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

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

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

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

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

[7] 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," *TODS'05*.

[8] 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*.

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

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

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

[12] 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*.

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

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

[15] 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*.

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

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

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

[19] H.-P. Hsieh, C.-T. Li, and S.-D. Lin, "Exploiting large-scale check-in data to recommend time-sensitive routes," in *UrbComp'12*.

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