Game-Theoretic Solutions for Constrained Geo-Social Event Organization

Lefteris Ntaflos  
HKUST  
entaflos@ust.hk

George Trimponias  
Huawei Noah’s Ark Lab  
g.trimponias@huawei.com

Dimitris Papadias  
HKUST  
dimitris@cs.ust.hk

ABSTRACT

In Geo-Social Event Organization (GSEO), each user of a geo-social network is assigned to an event, so that the distance and social costs are minimized. Specifically, the distance cost is the total distance between every user and his assigned event. The social cost is measured in terms of the pairs of friends in different events. Intuitively, users should be assigned to events in their vicinity, which are also recommended to their friends. Moreover, the events may have constraints on the number of users that they can accommodate. GSEO is an NP-Hard problem. In this paper, we utilize a game-theoretic framework, where each user constitutes a player that wishes to minimize his own social and distance cost. We demonstrate that the Nash Equilibrium concept is inadequate due to the capacity constraints, and propose the notion of pairwise stability, which yields better solutions. In addition, we develop a number of optimization techniques to achieve efficiency. Our experimental evaluation on real datasets demonstrates that the proposed methods always outperform the state-of-the-art in terms of solution quality, while they are up to one order of magnitude faster.

CCS CONCEPTS

• Information systems → Spatial-temporal systems;

KEYWORDS

Geo-Social Networks, Graph Partitioning, Game Theory

1 INTRODUCTION

Let \( G = (V, E, W) \) be an undirected graph, where the set \( V \) of nodes represents the users of a geo-social network, the set \( E \) of edges denotes their friendships, and \( W \) is the set of edge weights. Given a set of events \( P \), with minimum and maximum capacity constraints \( \min_p, \max_p, \forall p \in P \), Geo-Social Event Organization (GSEO) assigns each user to a single event according to the following conditions:

- The capacity constraints are satisfied:

\[
\min_p \leq |p| \leq \max_p, \forall p \in P
\]

where \(|p|\) is the number of users assigned to event \( p \). Constraints capture real-life situations e.g., an event may require a minimum number of participants, or may be imposed by the available budget that an event advertiser is willing to spend for its campaign.

- Objective Function (1) is minimized:

\[
\alpha \sum_{v \in V} d(v, p_v) + (1 - \alpha) \sum_{(v, f) \in E} w(v, f) \tag{1}
\]

where \( p_v \) is the event assigned to user \( v \), \( d(v, p_v) \) is the distance between \( v \) and \( p_v \), and \( w(v, f) \) is the weight of the edge \((v, f) \in E\). The first sum measures the quality of a solution in terms of the total distance cost between users and their assigned events. The second sum is the social cost, and equals the total weight of the edges between friends assigned to different events. The parameter \( \alpha \) adjusts the relative importance of the two factors.

Objective Function (1) implies that users should be assigned to events in their vicinity, which are also recommended to their friends. Unconstrained GSEO constitutes an instance of Uniform Metric Labeling (UML) [6], a well-known NP-Hard problem, which is a generalization of Multiway Cut [5]. Several vision problems such as stereo matching [4], photomontage [2] and interactive photo segmentation [10] can also be modeled as UML. Correspondingly, those problems are solved by approximations such as graph cuts [4], generalized belief propagation [12], and tree reweighted message passing [11]. However, these algorithms aim at graphs with up to a few hundred of nodes.

For very large graphs, commonly found in social networks, Armenatzoglou et al. [3] study GSEO without considering capacity constraints, and propose a best-response algorithm that always converges to a Nash equilibrium (i.e. a local minimum). Their algorithm, however, is inapplicable in the presence of capacity constraints. Li et al. [8] propose algorithms for Social Event Organization (SEO), a utility maximization problem similar to constrained GSEO. In SEO, (i) some events may be left empty despite the existence of sufficient users, (ii) some users may remain unassigned despite the fact that the maximum capacities are large enough to accommodate all users. On the contrary, in GSEO we consider that, if the number of users is adequate and the maximum capacities are sufficient, no event will be left empty, and no user will be left unassigned.

We model GSEO as a game, where users are players that choose their most preferred event. The objective is then to obtain an assignment, where no player has an incentive to deviate from his current event. Our contributions are summarized as follows: (i) We introduce the notion pairwise stability and develop a novel type of dynamics that generates high quality assignments. (ii) We propose various optimizations and data structures for fast real-time performance. (iii) We experimentally demonstrate that our framework achieves superior solutions than the current state-of-the-art, while substantially dropping the execution time. Section 2 introduces the game-theoretic framework for the GSEO problem. Section 3 presents a concrete implementation of our framework and the associated data structures. Section 4 contains the experimental evaluation. Section 5 concludes the paper.
2 GAME-THEORETIC FRAMEWORK

We model GSEO as the game \( G = \langle V, (P_v), (c_v) \rangle \), where the set of players \( V \) corresponds to the users and the strategy set \( P_v \) of user \( v \) coincides with the set of events \( P \). Each event \( p \in P \) may have capacity constraints \( min_p \leq |p| \leq max_p \), i.e., the number of users assigned to event \( p \) must be between \( min_p \) and \( max_p \). Equation (2) describes the individual cost of user \( v \) for assignment \( p_v \), given the strategies \( P_v \) of the other users. It consists of the weighted sum of the distance cost, i.e., the distance between \( v \) and event \( p_v \), and the social cost, i.e., half of the total weight of the edges connected to \( v \) friends assigned to different events. Since each edge \((v, f)\) is considered in the cost of both \( v \) and \( f \), by summarizing the individual costs of all users, we obtain Objective Function (1).

\[
c_c(p_v, \overline{p_v}) = \alpha \cdot d(v, p_v) + (1 - \alpha) \cdot \sum_{(v, f) \in E^\backslash p_v, p_v} \frac{1}{2} \cdot w(v, f) \tag{2}
\]

An assignment is a mapping from the set of users \( V \) to the set of events \( P \), which is (1) total, i.e., every user must be assigned to exactly one event, assuming that the total event capacity can accommodate all users, and (2) feasible, i.e., it satisfies the upper and lower capacity constraints. We consider that \( \sum_{p \in P} min_p \leq |V| \leq \sum_{p \in P} max_p \), so that all events can reach their minimum capacity, and all users can be assigned to some event.

From a game-theoretic perspective, the goal of each player is to select the event that minimizes his own cost, as expressed by Equation (2). Player \( v \) has an incentive to perform a unilateral deviation from his assigned event \( p_v \) to another one \( p_v' \), if \( p_v' \) yields a smaller cost for \( v \). However, this swap may violate the minimum capacity of \( p_v \), or the maximum capacity of \( p_v' \). Now consider two users \( v \) and \( u \), assigned to \( p_v \) and \( p_u \) respectively, and that both \( p_v, p_u \) are at their minimum (or maximum) capacities. Also, assume that both \( v \) and \( u \) would benefit by swapping events. However, this is not possible by individual unilateral deviations because it would violate the constraints. In order to cover such cases, we introduce the concept of pairwise stability: an assignment is stable, if for all pairs of users \((v, w) \in V\), exchanging their events will not decrease the cost of both \( v \) and \( w \).

Figure 1 presents the general framework. Line 1 computes an initial assignment. During that step, all users are assigned to some event, and all events reach their minimum capacity, but do not exceed their maximum capacity. The outer loop (Lines 2-9) corresponds to a super-round. Each super-round performs rounds of unilateral deviations (Lines 3-5), until reaching a Nash equilibrium, i.e., a local minimum where no user can decrease his cost without violating some capacity constraint. Then, rounds of bilateral deviations allow pairs of users to swap events, until reaching a stable assignment (Lines 6-8). Observe that after a bilateral deviation, a user \( v \) may be assigned to an event \( p \), which was not allowed by a unilateral deviation because \( p \) was full. This may create new opportunities for \( v \) to further drop his cost in subsequent unilateral deviations.

As shown in the long version of this paper [9], the framework of Figure 1 always converges to a solution, which is both a Nash equilibrium and pairwise stable. [9] also contains details about the price of stability and the price of anarchy that provide bounds on the quality of the solutions achieved.

---

\[\text{Input: Strategic game } G = \langle V, (P_v), (c_v) \rangle > \text{ with capacity constraints} \]

\[\text{Output: Nash equilibrium and pairwise stable assignment}\]

1. Compute a feasible initial assignment
2. repeat
   3. repeat
      4. perform legal unilateral deviations
      5. until Nash equilibrium
      6. repeat
         7. perform bilateral deviations
         8. until pairwise stability
      9. until Nash equilibrium and pairwise stability
   10. return the event assigned to each user \( v \in V \)

---

3 ALGORITHMS

We first propose an INIT algorithm for generating initial assignments. INIT involves two phases: Phase 1 fills all events to their minimum capacity, whereas Phase 2 assigns all the users left unassigned during Phase 1. Both phases adopt an iterative approach based on sampling. At each iteration, a sample of users is randomly selected and the cost of their possible assignments is computed. The assignment with the lowest cost is then performed. Figure 2 illustrates the pseudocode of INIT. Lines 1-6 perform a precomputation step that calculates the cost of assigning each user \( v \) to every event \( p \) according to Equation (2). These costs are stored in a \([|V||P|]\) array, called the cost table. Since initially all users are unassigned, we set a maximum social cost per user \( v \) and event \( p \), assuming that all the friends of \( v \) are at events other than \( p \). For a user \( v \), the costs for all events are stored in a min-heap \( H_v \) of size \(|P|\); the event \( p^* \) with the lowest cost for \( v \) is at the top of \( H_v \). Since there is a heap per user, the total number of user heaps is \(|V|\).

Lines 7-24 implement Phase 1. Let \( P_{op} \) be the set of \( \textit{open} \) events that have not reached their minimum capacity, and \( V_{un} \) be the set of unassigned users (initially, \( P_{op} = P \) and \( V_{un} = V \)). While there are still open events, INIT selects a random set \( S \) of distinct users from \( V_{un} \). For each user \( v \) in \( S \) it obtains the event \( p^* \) with the lowest cost among events in \( P_{op} \), which is at the top of \( H_v \) (Lines 13-14). Let \( v' \) be the user with the minimum lowest cost and \( p' \) the corresponding event; \( v' \) is assigned to \( p' \) and removed from \( V_{un} \) (Line 17). Lines 18-20 decrease the costs of the friends of \( v' \) for \( p' \), to reflect the new assignment, and update the corresponding heaps. Finally, if the user cardinality \( |P' \) of \( p' \) reaches its minimum capacity \( min_{p'} \), \( p' \) is excluded from \( P_{op} \), and the event closes (i.e., it will not receive more assignments at Phase 1). Lines 23-24 remove closed events from the heaps of all unassigned users (the rest of the users will not be re-assigned during INIT and their heaps will not be used again). At the end of Phase 1, since all events close, all heaps become empty. Phase 2 (Lines 25-26) repeats the process with the following differences: (i) The heaps are re-built using the user/event costs computed during Phase 1 and stored in the cost table; (ii) \( P_{op} \) now contains events that have reached their minimum, but are below their maximum capacity (initially, \( P_{op} = P \)); (iii) the loop is repeated while there are unassigned users (\(|V_{un}| > 0\) in Line 7); and (iv) an event \( p \) from \( P_{op} \) closes when it becomes full (\(|p'| = max_{p'}\) in Line 21).

Next, we discuss unilateral deviations, corresponding to Lines 3-5 of the general framework in Figure 1. Figure 3 illustrates the
Input: Geo-social Graph $G = (V, E, W)$, set of events $P$ with constraints $\min p, \max p, \forall p \in P$, $V_{un} = V$, $P_{op} = P$, $S = \emptyset$, $|S|$

Output: Nash equilibrium assignment

1. for each user $v$
   2. for each event $p$
   3. $c(v, p) \leftarrow a \cdot d(v, p)$
   4. for each friend $f$ of $v$
   5. $c(v, p) \leftarrow c(v, p) + \frac{1}{2}(1 - a) \cdot w(v, f)$
   6. $H_v = \min$-heap containing $c(v, p)$ for each event $p$
   7. while $|P_{op}| > 0$
   8. $c_m \leftarrow \infty$
   9. $\delta \leftarrow \{|S| \text{ random distinct users from } V_{un}\}$
   10. $S \leftarrow \delta$
   11. else
   12. $S \leftarrow V_{un}$
   13. for each user $v \in S$
   14. $p' \leftarrow \top(H_v)$ ($p'$ is event with min cost for $v \in P_{op}$)
   15. if $c(v, p') < c_m$
   16. $c_m \leftarrow c(v, p')$, $\delta' \leftarrow v, p' \leftarrow p'$
   17. $\delta' \leftarrow p', V_{un} \leftarrow V_{un} - \delta'$
   18. for each friend $f \in V_{un}$ of $\delta'$
   19. $c(f, p') \leftarrow c(f, p') - \frac{1}{2}(1 - a) \cdot w(v', f)$
   20. decrease key($H_f, p', c(f, p')$)
   21. if $|\delta'| = |\delta'|$
   22. $P_{op} \leftarrow p'$
   23. for each user $v \in V_{un}$
   24. remove $p'$ from $H_v$
   25. re-build heaps; $P_{op} = P$

Repeat 7-25, except Line 7; $P_{op} \leftarrow V_{un}$; Line 21: $\min p' \rightarrow \max p'$

Figure 2: INIT Function

pseudo-code of the UNIF function, which takes as input the initial assignment generated by INIT, and performs a series of rounds until reaching a point where no player can deviate from his current assignment. A unilateral deviation for user $v$ is allowed only if its current event $p \in P_{op}$ exceeds its minimum capacity $\min p$ (Line 4). In this case, the event $p' \neq p$ with the minimum cost $c(v, p')$ for $v$ is retrieved from his heap $H_v$. If $p'$ is full, UNIF retrieves the next cheaper event, until finding one that can receive more users, or until encountering $p$ (in which case all non-full events have cost higher than the current assignment, and there is no unilateral deviation for $v$ at this round). If a deviation from $p$ to $p'$ occurs, Lines 9-11 update the costs of each friend $f$ of $v$: specifically the cost $c(f, p)$ increases by $\frac{1}{2}(1 - a) \cdot w(v, f)$ due to the departure of $v$, while $c(f, p')$ decreases by the same amount.

Figure 4 illustrates BI, the function that implements stability. Given that in practice $|P| \ll |V|$, to enhance efficiency, we perform bilateral deviations for pairs of events, instead of pairs of users. Consequently, in addition to the user heaps, utilized by INIT and UNIF, BI uses event pair heaps. Specifically, for each pair of events $p_i, p_j$ there is a min-heap $EP_{op}$, which contains, for every user $v$ currently assigned to $p_i$, the cost change incurred by moving to $p_j$: $\delta c(v, p_i, p_j) = c(v, p_i) - c(v, p_j)$. The top contains the user with the minimum $\delta c(v, p_i, p_j)$, which may be positive or negative (if the swap benefits $v$). Each user $v$ exists in a single heap $EP_{op}$, which contains $|P|$ entries with the assignment costs of $v$ to all events. Therefore, the total space requirement for all the event heaps is $|V||P|$. For each pair of events $p_i, p_j$, we obtain the users $v$ and $u$ at the top of the heaps $EP_{op}$ and $EP_{op}$, respectively. Let $\Delta v \leftarrow \delta c(v, p_i, p_j)$, and $\Delta u \leftarrow \delta c(u, p_j, p_i)$. Swapping the events of $v$ and $u$ benefits both, if $\Delta v < 0$ and $\Delta u < 0$ (Line 9). However, if $v$ and $u$ are friends, $\Delta v + \Delta u$ must increase by $\frac{1}{2}(1 - a) \cdot w(v, u)$ to take into consideration the departure of $u$ (from $p_j$). If the swapping occurs, Lines 11-16 update the costs of the friends of $v$ and $u$ for both events. Finally, if $v$ and $u$ are friends, we also have to update their costs for both events (Lines 17-21); e.g., $c(v, p_j)$ decreases because of the inclusion of $u$ in $p_j$.

4 EXPERIMENTS

For our experimental evaluation we use the Gowalla and Foursquare datasets. Gowalla [1] contains 12,748 users, connected through 48,419 edges, who checked-in at Austin and Dallas during a weekend in February 2009. For the same time and place, we collected 128 social events from Eventbrite. Foursquare [7] contains 2,153,371
users and 1,143,092 events/venues, over the world in September 2013. The number of edges is 27,098,490. In both datasets the weight of all friendships is equal to 1. When we fix the number of events, we randomly select a subset from the corresponding dataset. In all experiments we set the parameter $\alpha$ of Equation (1) to 0.5, so that the distance and social costs have equal weights, and apply the normalization technique of [3] to make the distance and social costs comparable. All the algorithms were implemented in C++ under Linux Ubuntu, and executed on an Intel Xeon E5-2660 2.20GHz with 16GB RAM.

Next we compare the proposed GAME framework against SEOG, the best algorithm for the social event organization (SEOG) problem [8]. SEOG is a greedy technique that maintains all (user, event) pairs in a Fibonacci heap, with key the distance between every user and event. Each time a pair $(v, p)$ is popped, if $v$ is unassigned and the capacity constraint of $p$ is satisfied, $v$ is assigned to $p$. For SEOG, only events with at least one user need to respect their capacity constraints. Consequently, SEOG may leave some events empty and some users unassigned, even though there are enough users and sufficiently large capacities. To measure the number of such users and events we performed some experiments (see [9]). For instance, in Foursquare, when $|P| = 8$, only about 47% of the users are assigned and just 25% of the events have a number of users within their capacity constraints.

In order to avoid the problems of unassigned users and empty events of SEOG, for the following experiments we set the minimum capacity constraints of all the events to zero. We generate 30 problem instances with the same events, but different maximum capacities, following the methodology of [8], and report the mean over all instances. Figure 5 plots the solution quality of GAME against SEOG, as a function of the number $|P|$ of events ranging from 8 to 128. In all cases, GAME generates the best solutions. An interesting observation is that the total cost increases with the number of events. This can be explained by the fact that, when there are numerous events, friends are more likely to be divided, increasing the total social cost.

Figure 6 shows the running time versus the number of events. GAME outperforms SEOG by a wide margin in all settings. For example, in the largest problem instance (Foursquare, $|P| = 128$), GAME terminates in 16 minutes, whereas SEOG requires 3.5 hours. This is because SEOG uses a large heap of size $|V| : |P|$ that needs to be updated numerous times. In contrast, GAME uses smaller heaps (for each user and pair of events) that simplify the update process.

Summarizing the experimental evaluation, in terms of effectiveness, GAME always achieves the maximum number of assignments, by respecting the minimum capacities of all classes and not leaving empty events. Regarding solution quality, GAME yields up to 15% improvement with respect to SEOG, while it is never inferior. Finally, GAME is significantly faster than SEOG, with the performance gains exceeding an order of magnitude for some settings.

5 CONCLUSION

Geo-social Event Organization (GSEO) is becoming increasingly important with the proliferation of geo-social networks and related services. In this paper, we propose an effective and efficient framework for the capacitated GSEO based on a game-theoretic approach. To address the limitations of the Nash equilibrium concept, we introduce pairwise stability, which allows users to swap events if this leads to better solutions. The proposed algorithms outperform the state-of-the-art in effectiveness, solution quality and efficiency.

REFERENCES