Real-Time Multi-Criteria Social Graph Partitioning: A Game Theoretic Approach

Nikos Armenatzoglou[1] Huy Pham[2] Vasilis Ntranos[2] Dimitris Papadias[1] Cyrus Shahabi[2]

Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{nikos, dimitris}@cse.ust.hk

University of Southern California, USA
Los Angeles
{huyvpham, shahabi, ntranos}@usc.edu

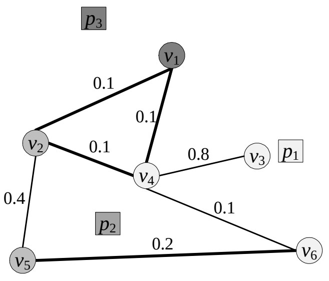
ABSTRACT

Graph partitioning has attracted considerable attention due to its high practicality for real-world applications. It is particularly relevant to social networks because it enables the grouping of users into communities for market analysis and advertising purposes. In this paper, we introduce RMGP, a type of real-time multi-criteria graph partitioning for social networks that groups the users based on their connectivity and their similarity to a set of input classes. We consider RMGP as an on-line task, which may be frequently performed for different query parameters (e.g., classes). In order to overcome the serious performance issues associated with the large social graphs found in practice, we develop solutions based on a game theoretic framework. Specifically, we consider each user as a player, whose goal is to find the class that optimizes his objective function. We propose algorithms based on best-response dynamics, analyze their properties, and show their efficiency and effectiveness on real datasets under centralized and decentralized scenarios.

1. INTRODUCTION

In this paper, we introduce RMGP a novel framework for Real-time Multi-criteria Graph Partitioning. RMGP partitions a social network into a set of input classes, so that users in the same class are socially connected, and at the same time they have high similarity to the class' properties. Formally, let $G = (V, E, W)$ be a social graph, where V is the set of users, E is the set of edges (i.e., social connections), and W is the set of edge weights (denoting the strength of social connections). The edges may be directed (e.g., representing the *follow* relationship in Twitter), and the weights may be binary (i.e., indicating simply the presence or absence of a friendship). Let P be a set classes given at query time, and $c : V \times P \to \mathbb{R}^+$ a function that measures the cost of assigning a user to a class, i.e., $c(v, s_v)$ is the cost of assigning $v \in V$ to $s_v \in P$. Then, RMGP returns an assignment of each user v to a single class s_v that minimizes the following function:

$$RMGP(G,P,\alpha) = \alpha \cdot \sum_{v \in V} c(v, s_v) + (1-\alpha) \cdot \sum_{\substack{e=(v,f)\in E \wedge \\ s_v \neq s_f}} w_e \quad (1)$$



	$\|\|v_i, p_1\|\|$	$\|\|v_i, p_2\|\|$	$\|\|v_i, p_3\|\|$
v_1	0.48	0.6	0.27
v_2	0.8	0.34	0.44
v_3	0.1	0.54	0.67
v_4	0.47	0.2	0.54
v_5	0.94	0.3	0.8
v_6	0.34	0.67	0.99

Figure 1: Running Example

The first sum in Equation 1 represents the *assignment cost*, while the second one corresponds to the *social cost*, i.e., the total weight of edges crossing classes. The *preference* parameter $\alpha \in (0, 1)$ adjusts the relative importance of the two factors (i.e., if $a > 0.5$, the partition process should aim more at minimizing the assignment cost at the expense of the social aspect). Each class becomes a partition, so that the weighted sum of assignment and social costs is minimized. Conversely, the similarities between users and their assigned classes, as well as the social connectivity inside classes are maximized.

EXAMPLE 1. *Location-Aware Graph Partitioning (LAGP). Assume that a geo-social network wishes to promote upcoming events to users based on proximity and social connectivity. Each event corresponds to a class, and the cost of assigning a user to a class is his/her current distance, or travel time, to the respective event.* RMGP *assigns each user to an event that minimizes (i) the distance/travel time between the user and the event, and (ii) the social connectivity between users assigned to different events. Consequently, a user is assigned to an event that is nearby and, at the same time, it is recommended to several of his friends.*

We illustrate LAGP through the example of Figure 1, with users $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and a set of three locations $P = \{p_1, p_2, p_3\}$ (each location corresponds to an event/class). The edge weights indicate the strength of social connections. The table in Figure 1 shows the Euclidean distance $||v_i, p_j||$ between each user $v_i \in V$ and location $p_j \in P$, which serves as the cost $c(v_i, p_j)$ (the larger the distance, the higher the cost of assigning v_i to class p_j). Assuming $\alpha = 0.5$, RMGP assigns v_1 to event p_3, v_2 and v_5 to event p_2, and v_3, v_4, v_6 to event p_1 (users have the same color as the event they are assigned to). Observe that user v_4 is assigned to p_1, and not to the closest location p_2, because his friends v_3 and v_6 are also assigned to p_1. The bold edges are the ones that contribute to the social cost, i.e., connecting friends in different events.

EXAMPLE 2. *Topic-Aware Graph Partitioning (TAGP). Consider that an on-line discussion forum wishes to place an advertisement for each user, in a manner that maximizes the "word of mouth" effect. In the corresponding* RMGP *task, each potential advertisement constitutes a class. The assignment cost of a user is based on some (dis-)similarity measure (e.g., tf-idf) between his current discussions and the advertisement topic. The social connectivity between users is based on the number of common discussions in which they have participated. Thus, users are likely to be assigned advertisements that match their own interests, and those of their frequent discussion co-participants.*

In general, RMGP refers to the partition of a graph according to connectivity, and one or more additional criteria. Specifically, in addition to social connectivity, we have the travel cost in LAGP and the text similarity in TAGP. A combination of multiple criteria can also be supported. For example, if each user has a profile, the assignment cost could take into account both the distance of each user and his preference to an event (e.g., based on textual similarity between the profile and the event description). Note that the combination of different criteria is application-dependent, and orthogonal to RMGP; e.g., the assignment cost $c(v, s_v)$ of Equation 1 could be a linear combination (or any other scoring function) of the distance and the preference of user v to event s_v.

We consider RMGP as an on-line process because input parameters change frequently. Specifically, in LAGP, locations of users maybe updated through check-ins, while new events may appear frequently. Therefore, RMGP recommendations should be efficiently generated in order to accommodate the fast-pace changes. Moreover, for some tasks only a subset of the network, determined at query time, may participate in RMGP. For instance, if a geo-social network wishes to advertise events at a certain area, only the users who recently checked-in that area, and the corresponding induced sub-graph, are relevant.

Another important issue in RMGP, and all tasks that partition a graph according to connectivity and additional criteria, is how to set the relative importance of the social aspect in the total cost. For instance, in LAGP the distance-based assignment cost can be in the order of thousands (of meters), whereas the edge weights may be in the range $[0, 1]$. In general, vast differences in the assignment and social costs render the value of the preference parameter α (and all types of relative weights) irrelevant. *Normalization* is crucial for adjusting the importance of the assignment and social costs as expressed by α, and eliminating the effects of factors such as the extent of the data space.

Graph partitioning has been studied extensively in social networks, where users are classified into groups or communities for advertising and analysis purposes. However, existing approaches focus on different versions of the problem, where the graph is partitioned based on connectivity and/or similarities between node attributes [32][28], but not between nodes and input classes. On the other hand, RMGP can be considered as an instance of Uniform Metric Labeling (UML) [12]. UML is a well-known NP-Hard problem that has only been studied theoretically in the context of approximation algorithms. Thus, all existing (approximate) solutions aim at small instances, and are not applicable to the large graphs commonly found in social networks. Moreover, they do not consider the real-time and decentralized nature of the problem, which is fundamental in our setting.

To effectively solve RMGP in real world social networks, we model the problem as a *game*, where each user corresponds to a player: his goal is to find the class that minimizes his assignment and social costs. As we show, the game reaches an equilibrium, i.e., a local minimum where no user has incentive to deviate. Modeling and solving RMGP as a game has several advantages. First, the game mimics the behavior of individual *real-world* users who aim at optimizing their own objectives, independently of those of the others. Consequently it yields recommendations (i.e., solutions) that are likely to be followed by the users. On the other hand, optimization methods that minimize the global cost, without considering the individual objectives, could lead to recommendations that are not followed by numerous users (e.g., in LAGP, users assigned to far events), rendering the solution meaningless. Moreover, the resulting algorithms are fast, allowing the real-time execution of queries, and facilitate distributed processing.

We first develop a baseline algorithm based on *best-response dynamics*, and analyze its properties. Our experimental evaluation shows that even the baseline algorithm is many orders of magnitude faster than state-of-the art UML methods and yields solutions that are very close to the optimal approximation. Then, we propose optimizations that enhance pruning of the search space by pre-computing a set of valid strategies per user and utilizing parallelism. Moreover, we apply our algorithms in decentralized scenarios (where the social graph is distributed at different servers), using an implementation that reduces the communication costs between processing units. Finally, we discuss and solve normalization issues that arise in RMGP due to the different values/measurements used for the assignment and social costs. Our contributions are summarized as follows:

- We introduce RMGP, and model the problem based on a game theoretic framework.
- We propose highly optimized algorithms that can efficiently solve RMGP tasks for large social networks in real-time.
- We apply our techniques to decentralized implementations of social networks.
- We show how to solve normalization issues in RMGP.
- We illustrate the effectiveness of our techniques through extensive experiments with real datasets under centralized and decentralized settings.

The rest of the paper is organized as follows. Section 2 overviews the related work and presents background on game theory. Section 3 proposes the baseline algorithm and analyzes its behavior. Section 4 describes multiple optimizations. Section 5 introduces a framework for applying RMGP in decentralized environments. Section 6 includes our experiments, and Section 7 concludes the paper with directions for future work.

2. RELATED WORK

Section 2.1 presents related work on graph partition algorithms and Section 2.2 provides background on game theory.

2.1 Graph Algorithms

Existing work on graph partitioning can be grouped into three main categories: i) attribute-based, ii) connectivity-based, and iii) attribute and connectivity-based. Attribute-based methods partition the graph based only on the similarity of node attributes, without considering the social connectivity [26]. Connectivity-based approaches partition the graph based only on connectivity, e.g., normalized cut [25], structural density [31], and modularity [23]. Partitioning can also be based on multiple objective functions on the graph's structural properties. For instance, [3] minimizes both the weight of edges that cross partitions, and the shortest path length connecting two nodes in different partitions.

Methods in the third category partition a graph based on both connectivity and attribute similarity [32]. For example, Van Gennip et al. [29] create groups of users in a geo-social graph so that users in the same group are nearby and socially connected. [32] defines a distance function between two nodes that combines their social connectivity and attribute similarity. A conventional clustering algorithm, such as K-Medoids [11], can then be adapted to partition the graph using the distance function. Even though the above techniques partition a graph using multiple objectives none of them can be used for RMGP. In particular, besides the social connectivity, we consider the similarity of each node to a set of input classes, instead of the similarity between two nodes based on their attribute values.

On the other hand, RMGP is closely related to the Uniform Metric Labeling (UML) problem. The input of UML is (i) an undirected edge-weighted graph $G = (V, E, W)$, (ii) a set L of k labels, (iii) a function $c(v, l)$ that denotes the cost of assigning label $l \in L$ to node $v \in V$, and (iv) a uniform function $d(l, l')$ that returns 1 if $l \neq l'$; 0 otherwise. UML assigns a label l to every node v (the assignment is denoted as v_l), such that the following objective function is minimized.

$$\sum_{v \in V} c(v, v_l) + \sum_{e=(u,v) \in E} w_e \cdot d(v_l, u_l) \tag{2}$$

Comparing the objective function of UML with Equation 1, it is easy to verify that they are equivalent. Specifically, the labels of UML correspond to classes in RMGP. Moreover, Equation 2 can incorporate the preference parameter α by simply modifying $c(v, v_l)$ and w_e.

UML can be formulated as an integer linear program (ILP), which is NP-hard. Existing solutions are based on linear programming (LP) relaxations [12] [6] and focus on providing theoretical approximation guarantees to bound the integrality gap between the optimal (ILP) and the rounded (LP) solutions that can be obtained in polynomial time. Even though both LP methods provide a 2-approximation algorithm for UML, their respective complexities, $O\left((|E| + k \cdot |V|)^{3.5}\right)$ and $O\left((k(V + kE)^{3.5})\right)$, are still prohibitive for practical applications. To avoid linear programming, Bracht et al. [4] propose a greedy approach that runs in $O(k \cdot |V|^{3.6})$, but guarantees a much looser approximation ratio $8 \log |V|$. Furthermore, the algorithm requires extensive graph transformations; i.e., for each class it generates a new graph that connects the class to all nodes.

Due to their high complexity, UML algorithms have only been experimentally evaluated on small graphs with up to a few hundreds of nodes. In this paper, we compare our techniques to the above approaches and show that our solutions can be computed several orders of magnitude faster and are very close to the theoretical 2-approximation obtained by LP. Moreover, previous UML algorithms do not take into consideration the real-time and decentralized nature of the problem, which is fundamental in our setting.

Several platforms, such as Giraph [2], Pregel [18], GraphLab [17] have been used for implementing graph partitioning algorithms. These approaches aim at offline tasks as they were designed for graph analytics. A recent paper [16] studies LAGP tasks assuming that events have minimum and maximum participation constraints that cannot be violated (i.e., events that cannot reach the minimum number of participants are canceled). Finally, Metis [10], a well-known graph partitioning tool, focuses on k-way graph partitioning, i.e., it creates k clusters that minimize the cut based on connectivity only. In our experimental evaluation, we design a benchmark solution based on Metis.

2.2 Game Theory

In *strategic games*, players compete with each other over the same resources in order to optimize their individual objective functions. Under this framework, a player chooses a *strategy* that minimizes his own cost without taking into account the effect of his choice on other players' objectives. Formally, a strategic game is a tuple $< V, \{S_v\}_{v \in V}, \{C_v : \times_{u \in V} S_u\}_{v \in V} \to \mathbb{R} >$, where S_v represents all the possible decisions that player v can take during the game to optimize his function C_v. The optimization of C_v depends on v's own strategy, as well as the strategies of the other players. Thus, the objective function of each player takes as input the *strategic space* $S = S_1 \times S_2 \times \cdots \times S_{|V|}$, which is the Cartesian product of the strategies of all players. A strategic game has a *pure Nash equilibrium* [22], if there exist a specific choice of strategies $s_v \in S_v$ such that the following condition is true for all $v \in V$:

$$C_v(s_1, ..., s_v, ..., s_{|V|}) \leq C_v(s_1, ..., s'_v, ..., s_{|V|}), \ \forall s'_v \in S_v$$

In other words, no player has incentive to deviate from his current strategy. Figure 2 illustrates a common framework for studying the dynamic process of decision-making in a strategic game. Initially, a random strategy is assigned to every player. Then, in an iterative manner, each player selects his best strategy in response to the current strategies of all the other players; v chooses $s^*_v = \arg\min_{s_v \in S_v} C_v(s_1, ..., s_v, ..., s_{|V|})$, i.e, the strategy that minimizes his own cost function. This decision is called the *best-response*, and the process *best-response dynamics* [7].

```
Input: Strategic game $< V, \{S_v\}_{v \in V}, \{C_v : \times_{u \in V} S_u\}_{v \in V} \to \mathbb{R} >$
Output: Nash equilibrium

1. Assign a random strategy to each player
2. Repeat
3.     For each player $v \in V$
4.         compute $v$'s best strategy wrt the other players' strategies
5.         let $v$ follow his best strategy
6. Until Nash equilibrium //no player has incentive to change his strategy
7. Return the strategy of each player
```

Figure 2: Best-Response Dynamics

Issues related to the framework of Figure 2 involve: (i) whether a Nash equilibrium exists, (ii) how fast it can be found (speed of convergence), (iii) how good is the resulting solution (quality). Three measures have been widely used to evaluate the quality of equilibria : (i) *social optimum* (OPT), (ii) *price of stability* (PoS), and (iii) *price of anarchy* (PoA). The social optimum is the solution that yields the optimal values to all the objective functions, so that their total cost (or utility) is minimum (or maximum). The PoS of a strategic game is the ratio between the best value among its equilibria and the social optimum i.e., PoS = best equilibrium / OPT. On the other hand, the PoA of a game is the ratio between the worst value among its equilibria and the social optimum i.e., PoA = worst equilibrium / OPT. Intuitively, PoS corresponds to the best solution, while PoA to the worst possible solution found.

Potential games constitute special class of strategic games, in which a single function $\Phi : \times_{v \in V} S_v \to \mathbb{R}$, called the *potential function*, can be used to express the objective functions of all the players [19]. Let $\overline{s_v}$ denote the set of strategies followed by all players except v, i.e., $\overline{s_v} = \{s_1, ..., s_{v-1}, s_{v+1}, ..., s_{|V|}\}$. A potential game is *exact*, if there exist a potential function Φ, such that for all s_i and all possible combinations of $\overline{s_i} \in \times_{j \in N \setminus \{i\}} S_j$, the following condition holds:

$$C_v(s_v, \overline{s_v}) - C_v(s'_v, \overline{s_v}) = \Phi(s_v, \overline{s_v}) - \Phi(s'_v, \overline{s_v})$$

For potential games, the best-response dynamics of Figure 2 always converge to a pure Nash equilibrium [19]. Therefore, we can use the local search algorithm of Figure 2 to obtain solutions for any combinatorial problem that falls in the above framework. In most cases the potential function Φ can be directly obtained by the objective function of the corresponding optimization problem and C_v can be defined accordingly. Since Φ represents the objective functions of all players, the set of Nash equilibria can be found by locating the local minima of Φ.

Game theory has been used for solving numerous well-known graph problems, e.g., minimum spanning tree and community detection [8][24][21]. To the best of our knowledge, there is no work that considers a game theoretic approach for the UML problem.

3. RMGP: GAME THEORETIC APPROACH

In order to model RMGP as a game, we assume that each user is a player, whose goal is to join a class with low assignment cost that contains many of his closest friends. Initially, players are randomly assigned to classes, and then they start changing classes according to their best-responses until they reach a Nash equilibrium. In the rest of the section, we describe the algorithmic framework, and we analyze its properties.

3.1 Baseline Algorithm

The RMGP game is a tuple $< V, \{S_v\}_{v \in V}, \{C_v : \times_{u \in V} S_u \rightarrow \mathbb{R}\}_{v \in V} >$, where each user/player $v \in V$ has a set of strategies S_v and a cost function C_v. Let $s_v \in S_v$ be a specific strategy of player v, which represents the class that v is assigned to, and $c(v, s_v)$ be its cost. Given s_v and the strategies $\overline{s_v}$ of the other players, the total cost $C_v(s_v, \overline{s_v})$ of v is the weighted sum of (i) the *assignment cost* $c(v, s_v)$, and (ii) the *social cost*, i.e., half[1] of the total weight of edges that connect v with the subset of his friends in $adj(v)$, who are assigned to different classes. The goal of each player $v \in V$ is to find the class s_v that minimizes his own total cost as expressed by Equation 3. Similar to Equation 1, parameter α adjusts the relative importance of the assignment and social costs.

$$C_v(s_v, \overline{s_v}) = \alpha \cdot c(v, s_v) + (1-\alpha) \cdot \sum_{\substack{f \in adj(v) \wedge \\ s_f \neq s_v}} \frac{1}{2} w_{(v,f)} \quad (3)$$

It is important to note that the RMGP objective function in Equation 1 is equal to the sum of all individual user costs, i.e., $RMGP(G, P, \alpha) = \sum_{v \in V} C_v(s_v, \overline{s_v})$. This decomposition of the RMGP objective into a sum of individual (per user) costs functions provides a natural motivation for modeling RMGP as a game, since users' own goals are considered for reaching a global solution. Consequently, recommendations generated by the RMGP game are likely to be followed by the users.

Figure 3 depicts the baseline algorithm $RMGP_b$, which applies the framework of Figure 2 to RMGP. Initially, $RMGP_b$ assigns every player to a random class and computes the maximum social cost $maxSC_v$, assuming that all friends of v are assigned to a different class. Then, it starts the best-response procedure (Lines 4-14). Each iteration of the repeat-loop corresponds to a *round*. A round computes, for each player v, the cost of assigning v to every class. Initially the cost of a class is set as the assignment cost plus $maxSC_v$ (Lines 8-9). Then, for each strategy s_f followed by a friend f of v, the social cost of s_f is reduced according to the weight of the edge (v, f) (Lines 9-10). Lines 11-13 assign v to the strategy with the minimum cost. The algorithm terminates when there is no class change for any user during a round. The output is the final class s_v assigned to each player v.

[1]The social cost of assigning two users v, f in different classes is evenly divided between v and f, i.e., for an edge $(v, f) \in E$, only $\frac{1}{2} w_{(v,f)}$ is attributed to the cost C_v.

```
Input: Social Graph $G = (V, E, W)$, Classes $P$
Output: Nash equilibrium

1.  For each player $v \in V$
2.      assign $v \in V$ to a random class/strategy
3.      $maxSC_v = (1-\alpha) \cdot \sum_{f \in adj(v)} \frac{1}{2} \cdot w_{(v,f)}$
4.  Repeat
5.      For each player $v \in V$
6.          $minCost = \infty$
7.          For each class $p \in P$
8.              $cost_v[p] = \alpha \cdot c(v, p) + maxSC_v$
9.          For each friend $f$ of $v$
10.             $cost_v[s_f] = cost_v[s_f] - (1-\alpha) \cdot \frac{1}{2} \cdot w_{(v,f)}$
11.         For each class $p \in P$
12.             If $cost_v[p] < minCost$
13.                 $minCost = cost_v[p]$, $s_v = p$
14. Until Nash equilibrium
15. Return players' strategies
```

Figure 3: $RMGP_b$ Algorithm

Table 1 shows all the execution steps of $RMGP_b$ for our running example with $\alpha = 0.5$. Initially (first row), we randomly assign each player to a location/class $p \in P$, e.g., players v_1 and v_4 to location p_1. The value of the cost function for each player is shown inside the parenthesis next to his id (e.g., $C_{v_1} = 0.265$). Next, we iterate over the players, starting with v_1. The first column contains the cost for each of the three classes, and the best response is underlined (e.g., the best-response of v_1 is p_3 with cost 0.185). The pair of gray cells per row indicates strategy deviations; the cell with a bold value represents the new class. For instance, v_1, following his best response, switches from p_1 to p_3. A round terminates when all players have been examined. At the first round v_1, v_3 and v_6 change strategies. During the second round, there are no strategy deviations (i.e., there are no gray cells). Thus, $RMGP_b$ has reached an equilibrium and it terminates, returning the final assignments (v_3, v_4, v_6 to p_1, v_2, v_5 to p_2, and v_1 to p_3).

The performance of $RMGP_b$ can be improved by some simple heuristics. Specifically, in Line 2, instead of a random initialization, we can assign each player to the class that yields the minimum assignment cost (e.g., the closest event). Moreover, instead of selecting players at random in each round (Line 5), we can consider them in decreasing order of their degrees. The intuition behind this choice is that strategy changes of highly connected users (community leaders) will propagate fast to the other players, decreasing the number of rounds. We examine the effects of these heuristics in the experimental evaluation.

$RMGP_b$ can perform graph partitioning by combining connectivity with arbitrary criteria. For instance, in LAGP (running example), the assignment cost $c(v, s_v)$ in Line 8 could take into account both the distance of each user and his preference to an event (e.g., based on textual similarity between the profile and the event description). Moreover, it can easily handle cases where only a subset of the graph participates in the game (e.g., only users in a certain geographic area) by adding appropriate conditions in Lines 5 and/or 9. Finally, if multiple executions are required (e.g., in a scenario sending location-based advertisements every hour), the solution of the last execution can be used as the seed of the next one, in order to minimize the number of rounds.

Steps		$\boldsymbol{p_1}$	$\boldsymbol{p_2}$	$\boldsymbol{p_3}$
Initialization		$v_1(0.265), v_4(0.485)$	$v_2(0.22), v_5(0.15), v_6(0.36)$	$v_3(0.535)$
Round 1	$v_1(0.265, 0.325, \underline{0.185})$	$v_4(0.485)$	$v_2(0.22), v_5(0.15), v_6(0.36)$	$\boldsymbol{v_1(0.185)}, v_3(0.535)$
	$v_2(0.525, \underline{0.22}, 0.345)$	$v_4(0.485)$	$\boldsymbol{v_2(0.22)}, v_5(0.15), v_6(0.36)$	$v_1(0.185), v_3(0.535)$
	$v_3(\underline{0.05}, 0.47, 0.535)$	$\boldsymbol{v_3(0.05)}, v_4(0.485)$	$v_2(0.22), v_5(0.15), v_6(0.36)$	$v_1(0.185)$
	$v_4(\underline{0.31}, 0.325, 0.52)$	$v_3(0.05), \boldsymbol{v_4(0.31)}$	$v_2(0.22), v_5(0.15), v_6(0.36)$	$v_1(0.185)$
	$v_5(0.62, \underline{0.15}, 0.55)$	$v_3(0.05), v_4(0.31)$	$v_2(0.22), \boldsymbol{v_5(0.15)}, v_6(0.36)$	$v_1(0.185)$
	$v_6(\underline{0.22}, 0.36, 0.0.57)$	$v_3(0.05), v_4(0.31), \boldsymbol{v_6(0.22)}$	$v_2(0.22), v_5(0.15)$	$v_1(0.185)$
Round 2	$v_1(0.265, 0.325, \underline{0.185})$	$v_3(0.05), v_4(0.31), v_6(0.22)$	$v_2(0.22), v_5(0.15)$	$\boldsymbol{v_1(0.185)}$
	$v_2(0.525, \underline{0.22}, 0.345)$	$v_3(0.05), v_4(0.31), v_6(0.22)$	$\boldsymbol{v_2(0.22)}, v_5(0.15)$	$v_1(0.185)$
	$v_3(\underline{0.05}, 0.47, 0.535)$	$\boldsymbol{v_3(0.05)}, v_4(0.31), v_6(0.22)$	$v_2(0.22), v_5(0.15)$	$v_1(0.185)$
	$v_4(\underline{0.285}, 0.35, 0.52)$	$v_3(0.05), \boldsymbol{v_4(0.285)}, v_6(0.22)$	$v_2(0.22), v_5(0.15)$	$v_1(0.185)$
	$v_5(0.57, \underline{0.2}, 0.55)$	$v_3(0.05), v_4(0.285), v_6(0.22)$	$v_2(0.22), \boldsymbol{v_5(0.2)}$	$v_1(0.185)$
	$v_6(\underline{0.22}, 0.36, 0.57)$	$v_3(0.05), v_4(0.285), \boldsymbol{v_6(0.22)}$	$v_2(0.22), v_5(0.2)$	$v_1(0.185)$

Table 1: $RMGP_b$ **Example**

3.2 Analysis

We first analyze the running time of $RMGP_b$ and then investigate the quality of its solutions. Let $k = |P|$ be the number of input classes, and $|V|$, $|E|$ be the node and edge cardinality respectively.

LEMMA 1. *The complexity of each round is* $\Theta(k \cdot |V| + |E|)$.

PROOF. The best response of a user v requires initializing the costs for k strategies (Lines 7-8), each involving constant time. Lines 10-11 reduce the cost of classes that contain the d_v friends of v, where d_v is the degree of v. Finally, Lines 11-13 find, among the k classes, the one incurring the minimum cost for v. Summarizing, the total cost for a single user is $2 \cdot k + d_v$. Repeating the same process for all users yields: $\sum_{v \in V}(2 \cdot k + d_v) = 2 \cdot k \cdot |V| + \sum_{v \in V} d_v = 2 \cdot k \cdot |V| + 2 \cdot |E| = \Theta(k \cdot |V| + |E|)$. □

In order to obtain the number of rounds, we first show that RMGP is an exact potential game (see Section 2.2). Specifically, the players' best responses according to their own objective function (Equation 3) reduce the value of the potential function $\Phi(S)$ given below. Note that $\Phi(S)$ is *not* the same as the RMGP objective function given in Equation 1 (due to the $\frac{1}{2}$ factor in the social cost) and it will only be used in the following as a tool for our analysis.

$$\Phi(S) = \alpha \cdot \sum_{v \in V} c(v, s_v) + (1-\alpha) \cdot \sum_{e=(v,f) \in E \wedge s_v \neq s_f} \frac{1}{2} w_e \qquad (4)$$

THEOREM 1. RMGP *constitutes an exact potential game.*

PROOF. Recall from Section 2.2, that it suffices to show that for every player v, who changes his strategy from the current one s_v to the best-response s'_v, and for all possible combinations of the other players' strategies $\overline{s_v}$ it holds that: $C_v(s_v, \overline{s_v}) - C_v(s'_v, \overline{s_v}) = \Phi(s_v, \overline{s_v}) - \Phi(s'_v, \overline{s_v})$. Indeed, we have $\Phi(s_v, \overline{s_v}) - \Phi(s'_v, \overline{s_v}) = \alpha \cdot (c(v, s_v) - c(v, s'_v)) + (1-\alpha) \cdot (\sum_{f \in adj(v) \wedge s_v \neq s_f} \frac{1}{2} w_{(v,f)} - \sum_{f \in adj(v) \wedge s'_v \neq s_f} \frac{1}{2} w_{(v,f)}) =$
$\alpha \cdot c(v, s_v) + (1-\alpha) \cdot \sum_{f \in adj(v) \wedge s_v \neq s_f} \frac{1}{2} w_{(v,f)} - (\alpha \cdot c(v, s'_v) + (1-\alpha) \cdot \sum_{f \in adj(v) \wedge s'_v \neq s_f} \frac{1}{2} w_{(v,f)}) =$
$C_v(s_v, \overline{s_v}) - C_v(s'_v, \overline{s_v})$ □

Since RMGP is an exact potential game, and the set of strategic configurations S is finite, players need to change their strategies a finite number of times before a Nash equilibrium is reached. In order to provide an upper bound for the number of rounds required for convergence of $RMGP_b$, we consider a scaled version of the problem where the objective function takes integer values. More specifically, we assume an equivalent game with potential function $\Phi_{\mathbb{Z}}(S) = d \cdot \Phi(S)$, where d is a positive multiplicative factor chosen such that $\Phi_{\mathbb{Z}}(S) \in \mathbb{Z}$, $\forall S$. Notice that d is a constant that depends on the choice of $c(v, s_v)$, w_e and α, and does not scale with the size of the problem. We should further point out that this scaling only serves the purpose of proving the following theorems and the actual execution of the algorithm can be performed on decimal values.

Let $C^* \triangleq d \cdot \sum_{v \in V} \max_{p \in P} c(v, p)$, and $W^* \triangleq \frac{d}{2} \cdot \sum_{e \in E} w_e$. Intuitively, C^* corresponds to the (scaled) maximum assignment cost, assuming that each user is assigned to his "worst" strategy. Similarly, W^* is the (scaled) maximum social cost of any solution, assuming that all edges cross classes. The following lemma describes an upper bound on the number of rounds required until $RMGP_b$ converges to a Nash equilibrium. The main idea behind the proof of the lemma is that in each round $\Phi_{\mathbb{Z}}(S)$ will decrease by at least 1 and hence the total number of rounds cannot be more that the difference between its maximum and the minimum values.

LEMMA 2. *The number of rounds required until $RMGP_b$ converges to an equilibrium is upper bounded by* $\max\{C^*, W^*\}$.

PROOF. We can easily see that the scaled version of $RMGP_b$ with potential function $\Phi_{\mathbb{Z}}(S) = d \cdot \Phi(S) = \alpha \cdot \sum_{v \in V} c'(v, s_v) + (1-\alpha) \cdot \sum_{e=(v,f) \in E \wedge s_v \neq s_f} w'_e$, where $c'(v, s_v) = d \cdot c(v, s_v)$ and $w'_e = d \cdot w_e/2$, will converge to a Nash equilibrium in the same number of rounds as $RMGP_b$. The cost reduction in $\Phi_{\mathbb{Z}}$ after each strategy change of a player is at least 1, i.e., $\Phi_{\mathbb{Z}}(s_v, \overline{s_v}) - \Phi_{\mathbb{Z}}(s'_v, \overline{s_v}) \geq 1$ for all players $v \in V$ and all possible strategies S. This is because i) a player deviates only if his cost decreases according to his best response, ii) the improvement is at least 1 as $\Phi_{\mathbb{Z}}$ takes integer values, and iii) the improvement of a user's cost is the same as the improvement of the global potential function due to the property of exact potential game. Consequently, the total number of rounds is at most the maximum possible value of $\Phi_{\mathbb{Z}}$, denoted as Φ_{max}, minus the minimum possible value, denoted as Φ_{min}. In the worst case, $\Phi_{min} = 0$ and $\Phi_{max} \leq \alpha \cdot C^* + (1-\alpha) \cdot W^* \leq \max\{C^*, W^*\}$. Therefore, the number of rounds to reach an equilibrium is upper-bounded by $\max\{C^*, W^*\}$, as stated by the lemma. □

Next, we continue our discussion with some theoretical results on the quality of the solutions of $RMGP_b$.

THEOREM 2. *In $RMGP_b$, the price of stability (PoS) is bounded by 2, and the price of anarchy (PoA) is bounded by*
$PoA \leq 1 + \frac{(1-\alpha)}{\alpha} \frac{deg_{avg} \cdot w_{avg}}{2 \cdot c^*_{avg}}$, *where deg_{avg} is the average degree, $w_{avg} = \frac{1}{|E|} \sum_{e \in E} w_e$ is the average edge weight and $c^*_{avg} = \frac{1}{|V|} \sum_{v \in V} \min_{s_v \in P} c(v, s_v)$ is the average minimum per user cost.*

PROOF. Let $\mathcal{C}(S)$ denote the summation of all players' costs, $\mathcal{C}(S) \triangleq \sum_{v \in V} C_v(s_v, \overline{s_v})$, and recall that $\mathcal{C}(S)$ is equal to the RMGP objective function of given in Equation 1. From the definition of the potential function $\Phi(S)$ (Equation 4) we can directly see that for any strategic configuration S, we have that

$$\frac{1}{2}\mathcal{C}(S) \leq \Phi(S) \leq \mathcal{C}(S). \tag{5}$$

Let $\mathcal{S}^*$ be the globally optimal set of strategies that minimize $\mathcal{C}(S)$, and let $OPT = \mathcal{C}(\mathcal{S}^*)$. Further, let $\mathcal{S}^{**}$ be the set of strategies that yields the minimum of the potential function $\Phi(S)$, i.e., the best Nash equilibrium. From the bound in (5) and since $\mathcal{C}(S^*) \leq \mathcal{C}(S), \forall S$ and $\Phi(S^{**}) \leq \Phi(S), \forall S$, we have:
$\mathcal{C}(S^{**}) \leq 2\Phi(S^{**}) \leq 2\Phi(S^*) \leq 2\mathcal{C}(S^*) = 2 \cdot OPT$, and hence the PoS is bounded by 2 as stated by the Theorem.

Now, let $S^{\#}$ be the strategic configuration of any Nash equilibrium (e.g., the one obtained by $RMGP_b$). Since $S^{\#}$ is a Nash equilibrium, all the user cost functions are locally minimized and hence we have that $C_v(s_v^{\#}, \overline{s_v}^{\#}) \leq C_v(s_v', \overline{s_v}^{\#})$, for all $v \in V$ and all $s_v' \in P$. Choosing s_v' to be the strategy with the smallest assignment cost for user v (i.e., $s_v' = \arg\min_{s_v \in P} c(v, s_v)$) we obtain $\mathcal{C}(S^{\#}) = \sum_{v \in V} C_v(s_v^{\#}, \overline{s_v}^{\#}) \leq \sum_{v \in V} C_v(s_v{}', \overline{s_v}^{\#}) =$

$$\begin{aligned}
&= \alpha \sum_{v \in V} \min_{s_v \in P} c(v, s_v) + (1-\alpha) \sum_{v \in V} \sum_{\substack{f \in adj(v) \wedge \\ s_v' \neq s_f^{\#}}} \frac{1}{2} w_{(f,v)} \\
&\leq \alpha \sum_{v \in V} \min_{s_v \in P} c(v, s_v) + (1-\alpha) \sum_{v \in V} \sum_{f \in adj(v)} \frac{1}{2} w_{(f,v)} \\
&= \alpha \sum_{v \in V} \min_{s_v \in P} c(v, s_v) + (1-\alpha) \sum_{e \in E} w_e \\
&= \alpha |V| c_{avg}^* + (1-\alpha)|E| w_{avg} \qquad (6)
\end{aligned}$$

Since $\mathcal{C}(S^*) \geq \alpha \cdot |V| \cdot c_{avg}^*$, we have that $PoA = \mathcal{C}(S^{\#})/\mathcal{C}(S^*)$ $\leq 1 + \frac{1-\alpha}{\alpha} \cdot \frac{|E| \cdot w_{avg}}{|V| \cdot c_{avg}^*}$, and since $|E| = |V| \cdot deg_{avg}/2$ we obtain the required bound. □

3.3 Normalization Issues

In several applications, the assignment and social costs may not be comparable. For instance, in LAGP the assignment cost per user is defined on distance and can be in the order of thousands (of meters), whereas edge weights may be in the range $[0, 1]$. On the other hand, for TAGP the assignment cost is based on similarity and is usually in the range $[0, 1]$, while the social cost (e.g., based on the number of common discussions threads) may be in the order of thousands. Even for the same type of task, costs may exhibit huge fluctuations depending on the problem characteristics; e.g., distances in LAGP may be defined in a unit space (i.e., a square with extent 1 per axis), or may be given in meters, miles etc.

In such cases, the direct application of RMGP algorithms, or any partition method that combines social with other criteria, may be meaningless; if distances are very large compared to edge weights, most users are likely to be assigned to the closest event independently of their friends because the total cost is dominated by the assignment cost (i.e., distances to events). On the other hand, for high edge weights (compared to distances), users are likely to be grouped based only on their social connections, independently of their distance to events. To avoid the effect of the specific problem characteristics and measurements, we introduce the *normalized version* RMGP$^{\text{N}}$ of multi-criteria graph partitioning.

The aim of RMGP$^{\text{N}}$ is to regularize the assignment and social costs so that they are comparable: when $\alpha = 0.5$ the two sums of Equation 1 should have similar values; other values of α should indeed reflect the input preferences (e.g., $\alpha = 0.9$ should mean that the assignment overhead is about 9 times more important that the social aspect). As a first step, recall that the objective function of conventional RMGP (Equation 1) can be written in terms of the per user costs (Equation 3) as $RMGP(G, P, \alpha) = \sum_{v \in V} C_v(s_v, \overline{s_v})$. If AC_v (and SC_v) denote the *average* assignment (and social) cost per user, $RMGP(G, P, \alpha)$ can be rewritten as

$$RMGP(G, P, \alpha) = \alpha \cdot |V| \cdot AC_v + (1-\alpha) \cdot \frac{1}{2} \cdot |V| \cdot SC_v$$

In RMGP$^{\text{N}}$ for $\alpha = 0.5$, the average assignment and social costs must be equal; i.e., it should hold that:

$$|V| \cdot AC_v = \frac{1}{2} \cdot |V| \cdot SC_v \Rightarrow AC_v = \frac{1}{2} \cdot SC_v$$

To achieve this, the costs should be normalized, by re-adjusting the assignment cost or the edge weights. In the following we focus on the normalization of assignment cost. Let the *normalization constant* be $c_N = \frac{SC_v}{2 \cdot AC_v}$; then, RMGP$^{\text{N}}$ aims at minimizing the function of Equation 7:

$$RMGP^N(G, P, \alpha) = c_N \cdot \alpha \cdot \sum_{v \in V} c(v, s_v) + (1-a) \cdot \sum_{\substack{e=(v,f) \in E \wedge \\ s_v \neq s_f}} w_e \tag{7}$$

The value of c_N depends on the problem characteristics. We focus on LAGP before discussing other applications. Let $||v_i, p_j||$ be the distance between user $v_i \in V$ and event $p_j \in P$. Equation 7 essentially normalizes this distance to $c_N \cdot ||v_i, p_j||$; if $c_N > 1$, the data space is expanded, whereas if $c_N < 1$ the data space contracts. However, we cannot compute the exact value of c_N because it requires AC_v and SC_v, which can only be obtained after solving the problem. Instead, we approximate c_N using estimations of AC_v and SC_v based on two heuristic approaches.

The *optimistic* approach considers that among the k events, every user is assigned to the closest one. Therefore, $AC_v = dist_{min}$, where $dist_{min}$ is the average minimum distance between every user and any event, and it can be derived by cost models [27], or it can be computed at an initialization phase. Regarding SC_v, let deg_{avg} be the average node degree and w_{avg} be the average edge weight, both of which are query-independent and available apriori. We assume that among the deg_{avg} friends of a user, only a fraction $\frac{1}{\sqrt{k}}$ is assigned to different events, so that $SC_v = \frac{deg_{avg} \cdot w_{avg}}{\sqrt{k}}$ (e.g., if there are 16 events, 1/4 of the friends of a user are expected to be in other classes). Thus, the estimation for the normalization constant is $C_{Nopt} = \frac{deg_{avg} \cdot w_{avg}}{2 \cdot dist_{min} \cdot \sqrt{k}}$.

The *pessimistic* approach considers that each user is assigned to the event with the median distance, i.e., $AC_v = dist_{med}$, where $dist_{med}$ is the average median distance between every user and event. For the estimation of SC_v, we assume that users are uniformly assigned to k events, so that there are $\frac{deg_{avg} \cdot (k-1)}{k}$ edges connecting a user with friends at different events. Accordingly, $SC_v = \frac{deg_{avg} \cdot (k-1) \cdot w_{avg}}{k}$ and $C_{Npes} = \frac{deg_{avg} \cdot (k-1) \cdot w_{avg}}{2 \cdot dist_{med} \cdot k}$. C_{Npes} may be larger or smaller than C_{Nopt} depending on the values of the above parameters.

According to either the optimistic or the pessimistic approach, the *normalization constant* c_N, involves graph-dependent parameters deg_{avg}, w_{avg}, and application-dependent parameters $dist_{min}$, $dist_{med}$. In other applications, the values of $dist_{min}$, $dist_{med}$ should be computed accordingly. For instance, in TAGP, a pessimistic approach would set $dist_{med}$ as the average median dissimilarity between every user and advertisement. In case of multiple criteria (e.g., both distance and text dissimilarity), the values of $dist_{min}$, $dist_{med}$ should take into account all criteria. In general, the computation of application-dependent parameters is orthogonal to the normalization framework. Since normalization only involves multiplication of the assignment cost by c_N, RMGP^N preserves all the properties of RMGP presented in this section. It can also easily accommodate the optimizations of the next section. Moreover, our experimental evaluation confirms that normalization is indeed necessary in order to obtain meaningful results.

4. OPTIMIZATIONS

In this section we present optimizations that improve the performance of the baseline algorithm without compromising the convergence guarantee. Section 4.1 describes the elimination of strategies that can never be followed, regardless of the decisions of other players. Section 4.2 investigates how independent strategies can be utilized to achieve parallelism. Section 4.3 enhances efficiency at the expense of space by exploring a global table of strategies. The proposed optimizations are orthogonal and can be applied in any combination.

4.1 Pruning by Strategy Elimination

$RMGP_b$ computes the cost functions of all players for each strategy in every round. However, there are strategies that cannot be assigned to some users. For example, in LAGP, a user with a small number of friends will be assigned to an event in his vicinity. Consequently, we can reduce the strategic space, i.e., prune the events that are far from the user, and avoid redundant computations.

Let $s_{v,min}$ be the strategy with the minimum assignment cost for player v, and W_v be half the sum of the edge weights incident to v, i.e., $s_{v,min} = \arg\min_{s_v \in S_v} c(v, s_v)$ and $W_v = \sum_{f \in adj(v)} \frac{1}{2} \cdot w_{v,f}$, respectively. In the worst case (i.e., assuming that none of his friends follows $s_{v,min}$), the total cost of assigning v to $s_{v,min}$ is:

$$\alpha \cdot c(v, s_{v,min}) + (1-\alpha) \cdot W_v$$

The *reduced strategic space* of v, denoted as S_v^r, contains only the possible strategies that v may follow. A strategy $p \in S_v^r$ if and only if the lowest possible total cost of assigning p to v does not exceed $\alpha \cdot c(v, s_{v,min}) + (1-\alpha) \cdot W_v$. The lowest cost for p is achieved when all friends of v follow p, i.e., $\alpha \cdot c(v,p) + (1-\alpha) \cdot 0 = \alpha \cdot c(v,p)$. Consequently, p belongs to S_v^r if:

$$\alpha \cdot c(v,p) \leq \alpha \cdot c(v, s_{v,min}) + (1-\alpha) \cdot W_v$$

The *valid region* VR_v of player v is a bound such that no strategy with assignment cost higher than VR_v can be assigned to v:

$$VR_v = \frac{\alpha \cdot c(v, s_{v,min}) + (1-\alpha) \cdot W_v}{\alpha}$$

Equivalently, the reduced strategic space S_v^r of v contains only those strategies, whose assignment cost does not exceed VR_v. In the running example of Figure 1, if $\alpha = 0.5$, then $VR_{v_1} = 0.37$. $S_{v_1}^r$ contains only p_3, since $c(v_1, p_1) = 0.48$, $c(v_1, p_2) = 0.6$, and $c(v_1, p_3) = 0.27$. Therefore, we can directly remove v_1 from the game, and directly assign v_1 to p_3. Similarly, we can eliminate v_6 and prune p_1 from $S_{v_2}^r$.

We refer to the version of the algorithm that includes strategy elimination as $RMGP_{se}$. In a initialization step, $RMGP_{se}$ computes the valid regions for all users and eliminates the strategies that can never be followed. If during this process, it discovers a user v that can only be assigned to a single strategy (i.e., the one with the minimum assignment cost), it directly assigns v and removes him from the game. Since $RMGP_{se}$ never prunes best responses, it has the same convergence guarantees as $RMGP_b$.

4.2 Parallelism with Independent Strategies

In each round, $RMGP_b$ computes the best response of all users sequentially, in a round-robin fashion. This is a fundamental requirement in best response dynamics: if multiple players change strategies simultaneously their decisions may be based on "outdated" information and there is the chance that the overall potential function increases. However, in our setting, if two users are *not socially connected*, the strategic deviations of one will not affect the best-response of the other; i.e., these two users can select their best responses *simultaneously*. Based on this observation, we can partition the users in N_g groups such that no two users in the same group share an edge. In the running example of Figure 1 we could have three (or more groups), e.g., $\mathcal{G}_1 = \{v_1, v_5\}$, $\mathcal{G}_2 = \{v_2, v_3, v_6\}$, and $\mathcal{G}_3 = \{v_4\}$. The best responses of the users in the same group (e.g., v_2, v_3 and v_6) are computed in parallel, either by the same machine through multi-threading, or by different machines.

We refer to this game as $RMGP_{is}$. As shown in Figure 4, $RMGP_{is}$ examines each group in a round-robin manner. Since there are implementation limits on the number of threads that can be created, the algorithm takes as input a parameter T that defines the maximum threads that can run simultaneously. For each group $\mathcal{G}_i$, $RMGP_{is}$ creates T sets of users, each consisting of at most $\lceil |\mathcal{G}_i|/T \rceil$ users (Lines 3-6). The threads are executed in parallel, computing the best response of the users in the corresponding group sequentially (Line 7). $RMGP_{is}$ waits for all threads to finish in order to continue to the next group (Line 8). The algorithm terminates when none of the users in any group changes strategy.

```
Input: Social Graph $G = (V, E, W)$, Classes $P$, Colored Groups $\mathcal{G}_i$,
       Number of threads $T$
Output: Nash equilibrium

1.  assign each player $v \in V$ to a class/strategy
2.  Repeat
3.      For $i \in [1, N_g]$
4.          users per thread $upt = \lceil |\mathcal{G}_i|/T \rceil$
5.          For each $t \in [1, T]$
6.              $\mathcal{U}_t$ = choose at most $upt$ users from $\mathcal{G}_i$
7.              start thread: compute best responses of users in $\mathcal{G}_i^t$
8.              $\mathcal{G}_i = \mathcal{G}_i \setminus \mathcal{U}_t$
9.          wait for all threads to finish
10. Until Nash equilibrium
11. Return players' strategies
```

Figure 4: $RMGP_{is}$ Algorithm

Groups can be computed by a graph coloring algorithm. In particular, graph coloring assigns colors to nodes such that no two incident nodes share the same color. The problem of coloring a graph using the minimum number of colors is NP-Hard. However, there are polynomial greedy approaches (e.g., $O(|V|^2)$, [30]) that use at most $d_{max} + 1$ colors, where d_{max} is the maximum degree. We apply such an algorithm off-line to create groups of nodes with the same color.

4.3 Scheduling with Global Table

Depending on the strategy deviations, numerous redundant computations can be avoided with proper book-keeping. In particular, for a class/strategy p and a user v, if the set of v's friends, who followed p does not change, then the cost of assigning v to p remains the same. Even if some of v's friends have switched strategies, it is possible that v is not affected, and the costly examination of his strategies can be avoided. Based on these observations we propose the $RMGP_{gt}$ optimization.

$RMGP_{gt}$ randomly assigns each player v to a strategy and computes his cost for each strategy. This information is stored in a $|V| \cdot k$ array (k is the class cardinality), referred to as the *global table*, which is kept updated in the following manner. A Boolean variable h_v indicates whether user v is happy, i.e., if his current strategy has the minimum total cost among the strategies. The method only examines those players for which h_v is $false$. In more detail, assume that the algorithm considers player v, where s_v is his current strategy, and s'_v is his best-response. First, v updates his strategy from s_v to s'_v, and sets $h_v = true$. Then, he informs his friends so that they will update their costs for the corresponding strategies. Specifically, for each friend f of v, the cost of s_v increases by $(1-\alpha) \cdot \frac{1}{2} \cdot w_{(v,f)}$ due to the class change of v; on the other hand, the cost of s'_v decreases by $(1-\alpha) \cdot \frac{1}{2} \cdot w_{(v,u)}$. If, after the changes, the current strategy of f is the one with the lowest cost, h_f is set to $true$. The algorithm terminates when all players are happy.

Figure 5 depicts the pseudocode of $RMGP_{gt}$. Initially, $RMGP_{gt}$ computes the global table GT and sets v as happy or unhappy accordingly (Lines 3-6). Then, it processes the game described above until reaching a Nash equilibrium. $RMGP_{gt}$ performs the same number of rounds as $RMGP_b$, assuming that both use the same initial assignments. However, $RMGP_{gt}$ is faster since it only schedules unhappy users that will switch strategies, as opposed to all players in $RMGP_b$. Therefore, the cost of each round in $RMGP_{gt}$ is lower than that of $RMGP_b$, and it gradually decreases. The trade-off is that $RMGP_{gt}$ needs to maintain in main memory, for each user v, the cost of every strategy. The space requirement can be reduced if, during the construction of the global table, we apply the optimization of Section 4.1 to eliminate strategies that can never be followed. Finally, $RMGP_{gt}$ follows the definitions of $RMGP_b$ for cost and potential functions; therefore, convergence is guaranteed and the PoS and PoA remain the same.

Input: Social Graph $G = (V, E, W)$, Classes P
Output: Nash equilibrium

```
1.  initialize array $GT$ of size $|V| \times k$
2.  assign each player $v \in V$ to a class/strategy
3.  For each player $v \in V$
4.      For each class/strategy $p \in P$
5.          $GT[v][p] = C_v(p, \overline{s_v})$
6.      If $s_v = \arg\min_{p \in P} GT[v][p]$ then $h_v = true$; $false$ otherwise
7.  Repeat
8.      For each player $v \in F$
9.          If $h_v = false$
10.             $s'_v = \arg\min_{p \in P} GT[v][p]$, $h_v = true$
11.             For each friend $f \in adj(v)$
12.                 $GT[f][s'_v] = GT[f][s'_v] - (1-\alpha) \cdot \frac{1}{2} \cdot w_{(v,f)}$
13.                 $GT[f][s_v] = GT[f][s_v] + (1-\alpha) \cdot \frac{1}{2} \cdot w_{(v,f)}$
14.                 If $s_f = \arg\min_{p \in P} GT[f][p]$
15.                     $h_f = true$; $false$ otherwise
16. Until Nash equilibrium
17. Return players' strategies
```

Figure 5: $RMGP_{gt}$ Algorithm

5. DECENTRALIZED GAME

Commercial social networks usually distribute the social graph across multiple servers in order to improve content delivery and accommodate the geographically diverse nature of data generation and demand. Computation in decentralized storage systems is usually performed through a distributed data access interface (API) between the application and the corresponding servers. One such example is Facebook's TAO [5]. Following a similar approach, one could perform RMGP on a distributed social graph by fetching the data over the network through the API to a master processing unit and executing the algorithm locally. We refer to this approach as fetch-and-execute (FaE). However, for large problem instances, transferring the data to a server can be a major bottleneck in performance. Moreover, a single server may not be able to store the entire graph due to memory limitations.

To avoid these shortcomings, we propose a framework for performing RMGP in a decentralized manner. Our framework assumes that the graph is distributed over several *slave nodes* that have processing capabilities. The partitioning scheme used for assigning the data to the slaves is orthogonal to our problem. A *master node* M acts as a game coordinator that manages the data exchange between the slave nodes. We call this framework DG, for decentralized game. DG requires that the social graph has been colored using a distributed graph coloring technique such as [13]. Although we assume that the slaves can only communicate through M, DG can be easily extended to handle direct data exchange between slaves.

We describe DG through an LAGP scenario, e.g., we wish to assign users, who are currently checked-in within the bounds of a specific geographical area to k events. Figure 6 depicts the processing method and communication steps between M and the slave nodes. First, M sends an initialization command to all slave nodes, which contains the query (i.e., events, α, area of interest) along with a strategy initialization method, e.g., random or closest event. Then, each slave: i) determines the users who are stored locally and will participate in the game, i.e., are within the area of interest, ii) initializes players' strategies based on the method decided by M, and iii) sends the strategic vector of the local users, referred to as the local strategic vector LSV, along with the distinct colors of the players to M. M integrates the union of all LSVs into a global strategic vector GSV, and sends it to the slaves that will participate to the game (slaves that do not contain users within the area of interest are excluded). Once a slave node receives GSV, it i) stores it locally, ii) performs additional initialization tasks, e.g., computes the global table if processing is based on $RMGP_{gt}$, and iii) sends an acknowledgment to M, denoting that it is ready to start the game.

After M has received an acknowledgment from *all* slaves, it starts the execution of the game. Each iteration of the repeat loop (Lines 14-25) corresponds to a round of the game. In every round, M sends to the slave nodes a command to compute the best response of all users with a specific color. Upon receiving the command, each slave computes the best responses and sends back to M only the strategy changes. Next, M updates GSV and redistributes the changes. When a slave node receives such a message, it updates the local GSV and the global table if required, and sends an acknowledgment to M. The round finishes when all colors are processed. The algorithm terminates if M receives no strategic updates in a round.

Compared to FaE, DG improves the overall performance in two ways. First, it lessens the communication overhead by allowing slave nodes to locally process their data and exchange only strategy changes through the network. This is an important feature since data transfers may significantly burden the performance of decentralized systems. Second, DG reduces the query response time by enabling slaves to work in parallel.

```
1.  $M$ sends $q$ and initialization parameters to the all slave nodes
2.  Each slave node
3.      determines the local players
4.      initializes the strategies of the local players
5.      sends the strategies and the distinct colors of the local players' to $M$
6.  let $D$ denote all slave nodes that participate to the game
7.  let $CL$ be the set containing the distinct colors of players
8.  $M$ computes $GSV$
9.  $M$ distributes $GSV$ to all slave nodes in $D$
10. Each slave node in $D$
11.     stores GSV locally
12.     performs additional initialization tasks (if required)
13.     sends ACK to $M$
14. Repeat
15.     For each color $c \in CL$
16.         $M$ sends "compute $c$" command to all slave nodes in $D$
17.         Each slave node in $D$
18.             computes the best responses of the users in group $c$
19.             sends the strategic changes to $M$
20.         $M$ updates GSV
21.         $M$ distributes the changes to the slave nodes in $D$
22.         Each slave node in $D$
23.             updates its local strategic vector and players' scores
24.             sends ACK to $M$
25. Until no strategic changes
```

Figure 6: Decentralized Game

6. EXPERIMENTS

We evaluate the proposed approaches assuming a LAGP task, where the users of a geo-social network are assigned to events based on proximity and social aspects (i.e., the running example). The graph and the locations of users are stored in two main-memory hash tables where the user IDs are used as keys. In the social hash table, for each user there is an adjacency list of pairs (friend id, edge weight). The location hash table maintains the coordinates of the last check-in of each user. All algorithms were implemented in C++, under Linux Ubuntu and executed on an Intel Xeon E5-2660 2.20GHz with 16GB RAM.

We use two real datasets: Gowalla and Foursquare. Gowalla [1] consists of i) 12,748 users in the Dallas and Austin metropolitan area, ii) their check-ins over a weekend in February 2009, and iii) 48,419 edges/friendships among them. 128 different social events that took place during the same weekend in Dallas and Austin, were obtained from Eventbrite [2]. Foursquare [15], collected in September 2013, consists of 2,153,471 users and their check-ins, 27,098,490 friendships, and 1,143,092 events/venues. In both datasets, all edge weights equal to 1. For experiments, where we need to reduce the size of the social graph, we use the Forest Fire sampling technique [14]. For decreasing the event cardinality, we randomly select the required number of events.

Section 6.1 compares the performance of $RMGP_b$ with algorithms for related problems. Section 6.2 investigates the effect of normalization on the quality of the solutions. Section 6.3 evaluates the optimizations of the baseline approach in a centralized setting. Section 6.4 studies RMGP in a decentralized setting.

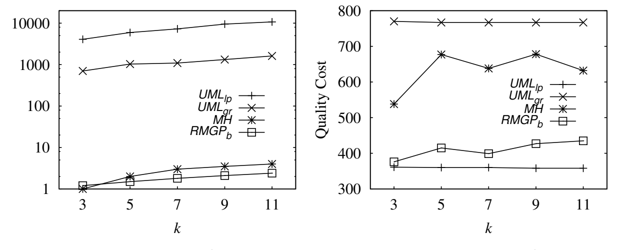
6.1 Comparison with Other Approaches

Since as discussed in Section 2 existing graph partitioning methods focus on different problems, in order to evaluate the proposed techniques we use the following benchmarks. (i) The Metis-Hungarian approach (MH), which initially computes the minimum unbalanced k-way social cut using METIS, and then it assigns every partition (i.e., group of users) to an event using the Hungarian method [20], so that each partition is assigned to a different event and the total assignment cost is minimized. (ii) The linear programming UML algorithm (UML_{lp}) [12], which guarantees an approximation ratio 2. (iii) The faster greedy UML approach (UML_{gr}) [4], which guarantees a looser approximation ratio $8 \log |V|$.

[2]http://www.eventbrite.com

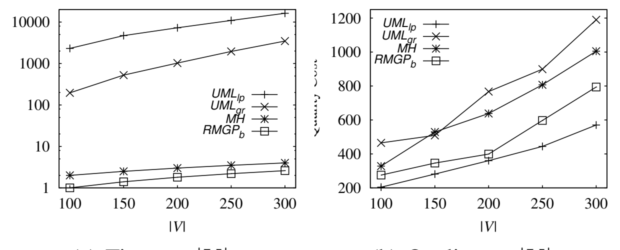
We compare the above techniques with the *unoptimized* $RMGP_b$ in terms of efficiency and solution quality. $RMGP_b$ applies random initial assignment and random order for considering users in each round. Since UML methods aim at small datasets, we reduce the size of Gowalla through Forest Fire. Moreover, we pre-compute the costs (distances) of the assignments because they are required by UML_{lp} and UML_{gr} as input. In order to solve UML_{lp}, we use CVX [9].

Figure 7(a) assesses the execution time (in ms) as a function of the number k of events, for $|V| = 200$. $RMGP_b$ is more than 3 orders of magnitude faster than UML_{gr} and UML_{lp} because both UML algorithms have high complexity and require graph transformations at execution time. MH is slightly slower than $RMGP_b$. Figure 7(b) shows the solution quality (i.e., total assignment cost according to Equation 1) versus k. UML_{lp} yields the lowest cost because in the worst case it generates a 2-approximation of the optimal solution. Actually, in most settings, the linear relaxation gave integral solutions, which means that the output of UML_{lp} is optimal (this is not expected for larger graph instances). The assignments of $RMGP_b$ have higher cost, but still comparable to UML_{lp}. However, the solutions of UML_{gr} and MH are significantly worse; UML_{gr} is bounded by a higher (worse) approximation ratio $8 \log |V|$, while MH minimizes the social cost (i.e., k-way cut) but yields high assignment costs.



(a) Time vs. k (b) Quality vs. k

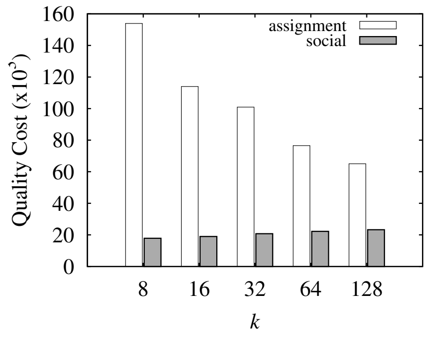
Figure 7: $RMGP_b$, MH, UML_{lp} and UML_{gr} vs. k



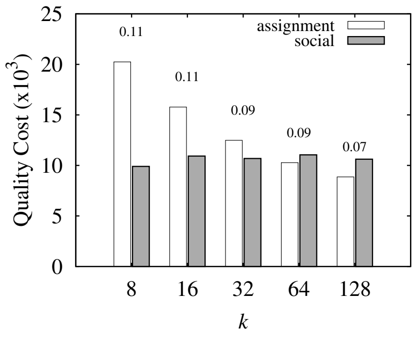
(a) Time vs. $|V|$ (b) Quality vs. $|V|$

Figure 8: $RMGP_b$, MH, UML_{lp} and UML_{gr} vs. $|V|$

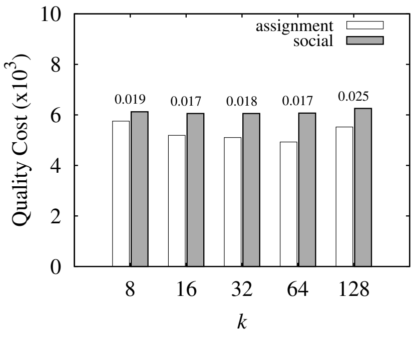
Figure 8 plots the running time and quality cost versus the node cardinality $|V|$, for $k = 7$. Similar to the previous experiment, $RMGP_b$ is the most efficient algorithm while the quality of its solutions remains close to UML_{lp} (which in most cases is optimal). Note that the quality cost grows with the size of the graph, because there are more users to assign. We restrict the maximum number of nodes to 300, because otherwise UML_{lp} and UML_{gr} would be too slow. In the remaining experiments, we focus on game theoretic



(a) No normalization



(b) Optimistic normalization



(c) Pessimistic normalization

Figure 9: Effect of Normalization, Cost vs. *k* ($\alpha = 0.5$)

algorithms using the complete real datasets. We exclude UML_{lp} and UML_{gr} because they are inefficient for large graphs, and MH since it generates solutions of low quality.

6.2 Effect of Normalization

To demonstrate the effect of normalization, we use Gowalla. Figure 9(a) shows the assignment and social costs as a function of the number of events k for the original (non-normalized) RMGP. Figures 9(b) and 9(c) illustrate the corresponding diagrams for RMGP^N according to the optimistic and pessimistic normalization, respectively. In all cases, the preference parameter α is set to 0.5 so that the assignment and social costs are considered equally important. The solutions are obtained by $RMGP_b$, where initially all users are assigned to the closest event.

According to Figure 9(a), the assignment dominates the social cost for all values of k before the normalization. This happens because the average distance between a user and an event is above 100 (kms) for all values of k, whereas the edge weights are equal to 1 and the average node degree is 7.6. Consequently, even if all friends of a user are assigned to different events, the social cost is likely to be a small fraction of the assignment cost, and most users stay at the closest event. For instance, if $k = 8$, only 1434 users are re-assigned to a new event, while 11314 remain at their initial assignment; the social gain of moving to a further event attended by friends cannot compensate for the increase in the distance.

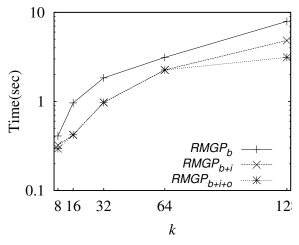
On the other hand, the normalized versions of the problem exhibit a much more balanced behavior, especially with the pessimistic approach, for which the assignment and social costs are similar for all values of k (recall that α is set to 0.5). For $k = 8$, in the optimistic (pessimistic) approach 3459 (6583) users are re-assigned to events that reduce their social cost. The numbers on top of the columns for RMGP^N show the values of the normalization constant c_N used for the specific value of k (see Section 3.3). Finally, note that the overall costs in the three diagrams are not directly comparable since they are based on different formulae (Equation 1 for RMGP and Equation 7 for RMGP^N), or different approximations of c_N (for the two versions of RMGP^N). Moreover, they correspond to different solutions.

Since the solutions of the non-normalized version of RMGP are futile, in all the following experiments we assume RMGP^N normalized according to the pessimistic approach.

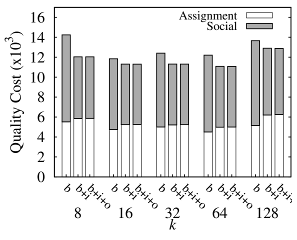
6.3 Centralized RMGP

We examine the behavior of $RMGP_b$ assuming a single server, which maintains the Gowalla dataset and performs query processing. Recall that $RMGP_b$ initializes users to random events, and each round considers users in a random order. We slightly modify the baseline to create two versions: i) $RMGP_{b+i}$ that initializes users to their closest event, and ii) $RMGP_{b+i+o}$, which in addition considers users in decreasing order of their degree.

Figure 10(a) illustrates the CPU time of the three variants as a function of the number k of events, for $\alpha = 0.5$. $RMGP_{b+i}$ improves significantly the performance of $RMGP_b$, whereas the additional effect of $RMGP_{b+i+o}$ is evident only for $k = 128$. Figure 10(b) shows the quality of the final solutions produced by the algorithms, for the same settings. For each value of k, the first column corresponds to the cost of $RMGP_b$, the second to the cost $RMGP_{b+i}$, and the third to the cost $RMGP_{b+i+o}$. The white part of a column indicates its assignment (i.e., distance) cost, whereas the gray part is the social component. Observe that the two components have comparable costs because $\alpha = 0.5$ and the problem is normalized. Usually $RMGP_{b+i}$ and $RMGP_{b+i+o}$ reach the same solution since they start with the same initialization, and their differences in terms of quality are negligible. On the other hand, the solutions produced by $RMGP_b$ are inferior in all settings.

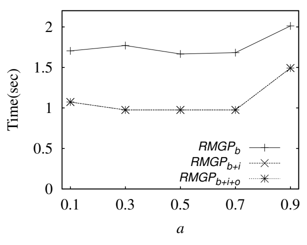


(a) Time vs. k

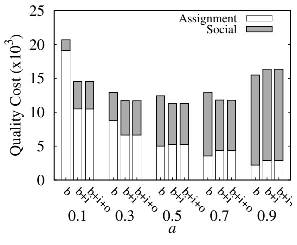


(b) Quality vs. k

Figure 10: Behavior of $RMGP_b$ ($\alpha = 0.5$)



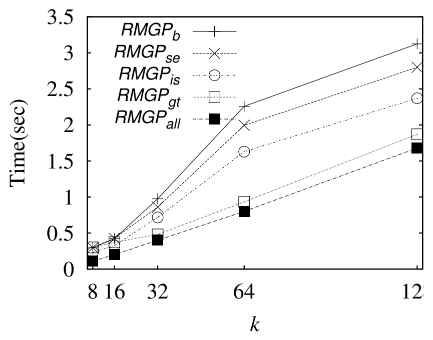
(a) Time vs. α



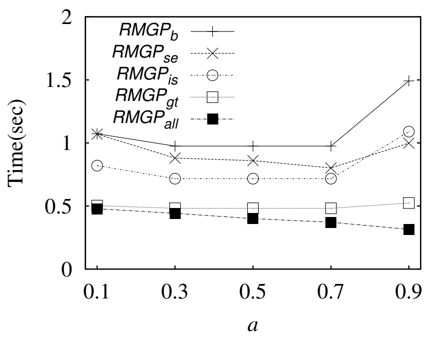
(b) Quality vs. α

Figure 11: Behavior of $RMGP_b$ ($k = 32$)

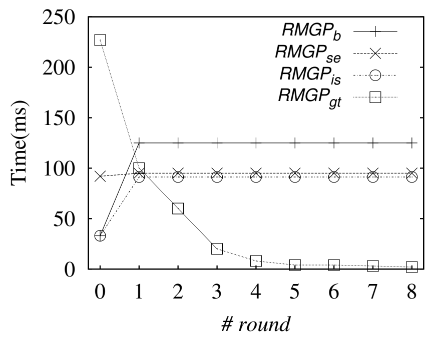
Figure 11(a) studies the effect of α on the running time, for $k = 32$. Similar to Figure 10(a), $RMGP_{b+i}$ and $RMGP_{b+i+o}$ have almost identical performance, and they are both better than



(a) Time vs. k ($\alpha = 0.5$)



(b) Time vs. α ($k = 32$)



(c) Time vs. round

Figure 12: Effect of Optimizations

$RMGP_b$. The value of α does not have a significant impact, and in all cases the heuristic versions require between 5 and 8 rounds to terminate, whereas the original algorithm needs between 9 and 11 rounds. Figure 11(b) measures the effect of α on the quality of solutions. Small values of α indicate low importance of the assignment cost, and the algorithms aim at minimizing mainly the social cost. This explains the small contribution of the gray (i.e., social) component in the leftmost columns of the diagram. As α increases, the assignment cost becomes more important; for $\alpha = 0.9$, the social component dominates the total cost because the algorithms tend to assign users to their closest events. Similar to Figure 10(b), $RMGP_{b+i}$ and $RMGP_{b+i+o}$ yield, in general, better solutions than $RMGP_b$. Since $RMGP_{b+i+o}$ has the best overall performance, we use it as the baseline algorithm for subsequent experiments.

Next we evaluate the optimizations of Section 4, namely $RMGP_{se}$, $RMGP_{is}$, $RMGP_{gt}$. Recall that i) $RMGP_{se}$ prunes events that cannot be assigned to a user (independently of the friend assignments) during an initialization round, ii) $RMGP_{is}$ exploits parallelization through independent strategies, and iii) $RMGP_{gt}$ utilizes a global table to only examine users who deviate (i.e., are re-assigned to a different event). $RMGP_{All}$ applies all three optimizations. Since the algorithms start with the same initial assignments (users are assigned to the closest events initially), they reach similar solutions. Thus, in the following, we exclude experiments on the quality of the solutions and only focus on efficiency.

Figure 12(a) measures the running time as a function of the number of events k, for $\alpha = 0.5$. The time of all approaches increases with k because the number of possible assignments per user grows linearly. $RMGP_{gt}$ achieves the best gains among individual optimizations because it only schedules unhappy users that will switch strategies. Therefore, the cost of each round in $RMGP_{gt}$ is lower than that of the other optimizations, and it gradually decreases (see also Figure 12(c)).

Figure 12(b) examines the effect of α on the running time, for $k = 32$. As α increases the spatial component starts dominating the total cost and users tend to be assigned to the closest events. Accordingly, the pruning power of $RMGP_{se}$ increases since the valid regions shrink (eliminating more users), and for $\alpha = 0.9$ it outperforms $RMGP_{is}$. $RMGP_{All}$ has the best performance in all cases as it incorporates all optimizations.

Figure 12(c) shows the running time per individual round for a random query with $k = 32$ and $\alpha = 0.5$. All algorithms terminate after 8 rounds. Round 0 constitutes the initialization step, which involves the cost of sorting the users in decreasing order of their degree and assigning them to their closest event (i.e., the two basic heuristics). In addition, for $RMGP_{se}$ it includes the computation of valid regions, and for $RMGP_{gt}$ it includes the generation of the global table. Thus, Round 0 is more expensive for these methods. Subsequent rounds have fixed cost for $RMGP_b$, $RMGP_{se}$ and $RMGP_{is}$ because they incur exactly the same computations. The savings of $RMGP_{se}$ and $RMGP_{is}$ with respect to $RMGP_b$ are due to strategy elimination and parallelism respectively. On the other hand, in the case of $RMGP_{gt}$ the running time per round gradually decreases because the the number of users who are reassigned drops as the algorithm approaches convergence.

6.4 Decentralized RMGP

In this section we evaluate DG, the framework presented in Section 5, versus FaE, which involves fetching the data to a server and executing the query locally. We use three identical servers (Intel Xeon E5-2660 2.20GHz with 16GB RAM) that communicate using an 100Mbps Ethernet connection. The Foursquare dataset is distributed to two of the servers. In DG, the third server acts as the master node, while in FaE it receives the entire dataset and performs all computations. Both DG and FaE employ $RMGP_{all}$ as the underlying algorithm since it has been shown to be the most efficient in the previous experiments.

Figure 13 depicts the execution time (in sec) versus k, for $\alpha = 0.5$. The columns represent the cost of FaE, which is split into transfer (gray) and execution (white) time. The first factor corresponds to the time required for the processing server to receive the social graph from the other servers, and is query-independent. For values of k up to 128 it dominates the total cost of FaE. DG avoids this cost and can start the game earlier. On the other hand, the execution time of FaE is comparable to DG, although the latter involves communication with the slaves during the game, while FAE executes $RMGP_{all}$ locally. This can be explained by the high cost of initialization (Round 0). For instance, if $k = 1024$ more than 2.2 billion computations of euclidean distances among users and events are required. In DG, slaves perform, in parallel, initializations for users in their partition, compensating for the additional data transfers during the rounds. For both FaE and DG the execution time increases linearly with k, mainly due to the cost of the initialization step. Observe that the running time (up to 5 minutes for the largest instance) is small compared to the size of the problem (2,153,471 users, 27,098,490 edges, up to 1,024 events), especially considering that we only use two slaves. Recall that UML methods require comparable times for graphs of a few hundred nodes.

Figure 14 plots the processing time (left axis) and data transferred (right axis) per round by DG for a random query with $k = 256$. The game terminates in 17 rounds. Round 0 incurs 58 seconds due to the initialization. The processing time of subsequent rounds gradually decreases because fewer users change strategies;

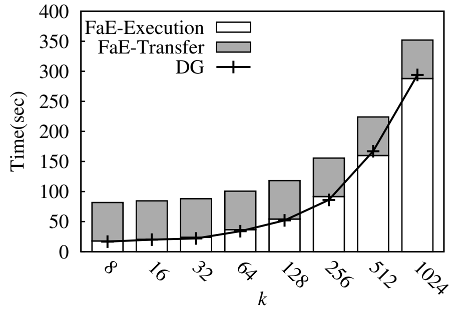


Figure 13: Decentralized Game: time versus k

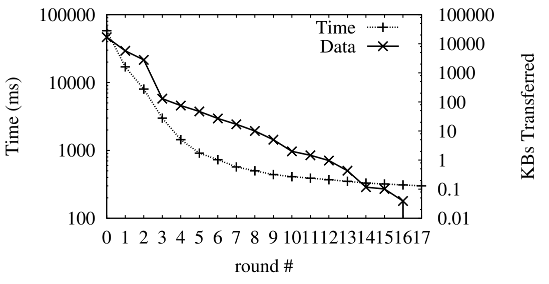


Figure 14: Decentralized Game: bytes transferred per round

this results in reduced data transfers and update operations at the master and slave nodes. Note that even the last round consumes some time because of the network communication (e.g., command and acknowledgment messages). The maximum amount of data (16MBs) is transferred at Round 0 because each slave obtains an up-to-date version of the entire strategic vector. Subsequent rounds only communicate strategy changes, and the amount of data transferred diminishes in accordance with the processing time.

7. CONCLUSION

This paper studies a type of multi-criteria graph partitioning, which assigns users of a social network to a set of input classes so that their assignment and social costs are minimized. To achieve efficiency, we model the problem as a game, develop a best-response algorithm, and propose several optimizations to enhance its performance. We implement our methods in both centralized and decentralized settings, where the social graph is distributed at different servers. In addition, we apply normalization to resolve issues that arise due to large differences in the assignment and social costs. Finally, we demonstrate the effectiveness of the proposed techniques with extensive experiments on real datasets. Given that our methods significantly outperform existing UML algorithms, they could be used to provide solutions to UML problems in real-time, even for large graphs.

Acknowledgments

Nikos Armenatzoglou and Dimitris Papadias were supported by GRF grant 16207914 from Hong Kong RGC. Huy Pham and Cyrus Shahabi were supported by NSF grants IIS-1115153, IIS-1320149, and CNS-1461963.

8. REFERENCES

[1] Stanford large network dataset collection,. `https://snap.stanford.edu`.

[2] Apache Giraph. `http://http://giraph.apache.org/`, 2014.

[3] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis. Multi-objective circuit partitioning for cutsize and path-based delay minimization. In *ICCAD*, 2002.

[4] E. C. Bracht, L. A. Meira, and F. K. Miyazawa. A greedy approximation algorithm for the uniform metric labeling problem analyzed by a primal-dual technique. *Journal of Experimental Algorithmics (JEA)*, 10:2–11, 2005.

[5] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. C. Li, et al. Tao: Facebook's distributed data store for the social graph. In *USENIX*, 2013.

[6] C. Chekuri, S. Khanna, J. S. Naor, and L. Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *SODA*, 2001.

[7] D. Fudenberg and T. Jean. *Game Theory*. MIT Press, Cambridge, 1991.

[8] D. Granot and G. Huberman. Minimum cost spanning tree games. *Mathematical Programming*, 21(1):1–18, 1981.

[9] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. Lecture Notes in Control and Information Sciences. 2008.

[10] G. Karypis and V. Kumar. METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System. `www.cs.umn.edu/karypis/metis`, 1995.

[11] L. Kaufman and P. Rousseeuw. Clustering by means of medoids. *Statistical Data Analysis based on the L1 Norm*, pages 405 – 416, 1987.

[12] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Journal of the ACM (JACM)*, 49(5):616–639, 2002.

[13] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *ACM PODC*, 2006.

[14] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *KDD*, 2006.

[15] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. Lars: A location-aware recommender system. In *ICDE*, 2012.

[16] K. Li, W. Lu, S. Bhagat, L. V. Lakshmanan, and C. Yu. On social event organization. In *SIGKDD*, 2014.

[17] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *PVLDB*, 5(8):716–727, 2012.

[18] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, 2010.

[19] D. Monderer and L. S. Shapley. Potential games. *Games and economic behavior*, 14(1):124–143, 1996.

[20] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial & Applied Mathematics*, 5(1):32–38, 1957.

[21] R. Narayanam and Y. Narahari. A game theory inspired, decentralized, local information based algorithm for community detection in social graphs. In *ICPR*, 2012.

[22] J. F. Nash. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[23] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[24] N. Nisan. *Algorithmic game theory*. Cambridge University Press, 2007.

[25] J. Shi and J. Malik. Normalized cuts and image segmentation. *TPAMI*, 22(8):888–905, 2000.

[26] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, 2007.

[27] Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. An efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces. *TKDE*, 16(10):1169–1184, 2004.

[28] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, 2008.

[29] Y. van Gennip, B. Hunter, R. Ahn, P. Elliott, K. Luh, M. Halvorson, S. Reid, M. Valasik, J. Wo, G. E. Tita, et al. Community detection using spectral clustering on sparse geosocial data. *SIAM Journal on Applied Mathematics*, 73(1):67–83, 2013.

[30] D. J. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[31] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger. Scan: a structural clustering algorithm for networks. In *KDD*, 2007.

[32] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.