SOCIAL EVENT ORGANIZATION

Behrad Farsi

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

June 2017

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Behrad Farsi**
Entitled: **Social Event Organization**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair
Dr. O. Ormandjieva

_____ Examiner
Dr. T. Fevens

_____ Examiner
Dr. A. Harutyunyan

_____ Supervisor
Dr. L. Narayanan

Approved _____
Chair of Department or Graduate Program Director

_____ _____
Dr.Amir Asif, PhD, PEng
Dean, Faculty of Engineering and Computer Science

Abstract

Social Event Organization

Behrad Farsi

In recent years, services such as Meetup, Plancast, and Eventbrite have provided platforms for planning and organizing live events. Social event organization (SEO), the problem of finding an assignment of users to events by considering their interests and social connections, is a problem that has received growing attention after being introduced by [24]. Given a weighted bipartite graph specifying the interest of every user in every event, and a social network between the users, the main goal of the SEO problem is to assign users to events so as to maximize a social welfare function, while respecting the minimum and maximum cardinality bounds associated with events. The problem is shown to be NP-complete, and in fact, hard to approximate [24].

First, we review the previous solutions proposed in [24] and discuss some problems in these algorithms. Then, we propose the Second-Chance Dynamic Greedy (SCDG) and Community-Aware Static Greedy (CASG) algorithms to enhance the quality of the results produced by the existing algorithms. Our experiments using both synthetic and datasets from Meetup and Plancast show that our algorithms obtain

a social welfare up to 60% better than that obtained by Phantom-Aware Dynamic Greedy (PADG), the best algorithm in [24].

Second, we propose the personality-oriented objective function that takes into account a user's willingness participate in large events, or events with unknown people. We adapt PADG, SCDG, and CASG so as to optimize this new objective function. Our experiments show that the personality-oriented version of SCDG improves the social welfare by up to 100% over the adaptation of PADG.

Acknowledgments

First and foremost, I would like to express my sincere gratitude and utmost thanks to my supervisor, Professor Lata Narayanan, for her continuous support, helpful guidance and insightful comments throughout this research. I would like to extend my sincere gratitude to my family. This thesis would not have been be possible without all their support. Also, I am grateful to my dear friend, mr. Shahab Harrafi for his enthusiastic encouragement.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

CASG	Community-Aware Static Greedy
GAP	Generalized Assignment Problem
GAP-MQ	GAP with Minimum Quantities Problem
MKP	Multidimensional Knapsack Problem
NRMP	National Resident Matching Problem
P	Community-Aware Static Greedy
ROL	Rank Order Lists
SAP	Seminar Assignment Problem
SCDG	Second Chance Dynamic Greedy
SEO	Social Event Organization
SMP	Stable Marriage Problem

Chapter 1

Introduction

1.1 Introduction

With the emergence of the World Wide Web and the proliferation of social networking services such as Facebook and Twitter, new platforms with novel objectives have been established to enable interactions between people all around the world in recent years. The success and evolution of social media platforms have led them to offer more tangible and non-virtual experiences via providing new services for organizing *live events* in addition to online interaction/communication services.

Facebook Events[1], Meetup[2] and Eventbrite[3] are a just handful of the event-based social networks (ESBN) that provide rich infrastructures for planning, promoting and

[1] http://www.events.fb.com
[2] http://www.meetup.com
[3] http://www.eventbrite.com

managing live events. EBSN platforms enable online users to create, arrange and organize offline events [**?**] such as gatherings, conventions, meetings, ceremonies, concerts or competitions. The process of event organizing consists of different tasks that have to be addressed such as cost estimation, organizing transportation, scheduling and marketing, which the aforementioned platforms aim to facilitate for organizers. Data supplied by social networks facilitate the automation of event management. Information about users' interests, their interactions and connections between them acquired through social networks such as Facebook provides the knowledge base required for making tailored recommendations and advertisements toward users. In the context of event management services, this information can be exploited to suggest, promote and advertise events, match users, find the target audience and many other activities regarding the event management and organization.

For example, Meetup is an event management platform founded in 2002 which lets event *organizers* create groups which are associated with location, schedule, event type, topics and description. Also, Meetup lets *users* to set their location and pick their favorite topics. Once a group is established, Meetup announces it to the interested nearby users so that they can join the group and Events are advertised to group members who can choose to join them. Users can also establish social connections with other group members via an instant messaging service provided by Meetup. As of June 2017, Meetup claimed to have 30.30 million members and 272,203 groups [1].

The Automation of the process of finding the users who are interested in the group events and also, in other participants can be regarded as a motivation for this research.

In this thesis, we study one of the problems regarding event management, called *Social Event Organization (SEO)*, originally proposed by [24], that is a combinatorial optimization problem which falls under the scope of data mining. The goal of the SEO problem is to maximize overall satisfaction by assigning users to events, according to their innate interest in the event and social affinities to each other, while respecting the event capacity bounds.

As we will discuss in detail in Chapter 2, this problem has similarities to *the Seminar Assignment Problem* in which the goal is to assign students to seminars with given profits, constraints on minimum enrollment and maximum capacities for seminars such that the aggregated profit is maximized [23]. Also, it shares similarities with recommendation systems which aim to opt for the best available option based on a vector of criteria.

The reasoning behind targeting overall satisfaction instead of individual satisfaction is that live events may impose contradictory constraints; for example a user full satisfaction may result in leaving other users discontented. Also, some users may rather to go to a less preferred event while being accompanied by their friends. Moreover, satisfying event cardinality constraints mandate a global solution for a finite number of events.

In the context of the social event organization problem, a person's satisfaction with an assignment is measured by his or her intrinsic interest in the event plus the social experience achieved through enjoying the event in the company of people from his or her social circle. Basically, innate affinity expresses the event's appeal to a user which can be attained either manually through a questionnaire or it can be deduced based on user favorite hobbies, interests, previous activities, location of the event and many other items of information that are available on social networks.

Another important factor that plays a major role in the quality of a live experience for a participant is his/her interest in other attendees, as having more friends on board will generally enhance the experience. The social network, defined as pairwise interests (or social affinities) between all the users, will let us estimate the quality of a solution proposed for an instance of SEO problem. For each event attendee, this quality is simply measured by his/her overall interest in other participants. There are different parameters and information that can be employed to gauge the interest between users. For example, for a pair of users this value can be estimated based on the evaluation of their interaction through social media, the number of common friends or liked pages on social networks, closeness of their locations etc. In this thesis, we assume that both innate and social affinities are provided as numerical input to the problem.

The last component of the problem is the event cardinality bounds, which specifies

for each event, the minimum number of users that is required so that the event could be held and the capacity of the event which is the maximum number of users that can be enrolled for it. A feasible solution of SEO problem respects the cardinality bounds for all the events, meaning that for each event, the number of assigned users is either zero (the event is canceled), or it falls between the minimum and maximum cardinality bounds.

1.2 Problem Definition

Social event organization is the problem of assigning a group of users to a set of events to maximize the overall social welfare, evaluated by an utility function, while respecting the feasibility condition of the problem, i.e. the number of users assigned to an event is either zero or it falls between minimum and maximum cardinality bounds associated with events. The social welfare objective function has been defined in [24] as a combination of interest of users to events that they are assigned to (innate affinity) and the interest of users who are assigned to the same event (social affinity), aggregated through the produced assignments.

Innate affinity is represented by the function $\sigma : U \times A \rightarrow \mathbb{R}_+$, which maps a user-event pair $<u, a>$ to a positive value that indicates the level of affinity that user u has for the event a. The value of $\sigma(u, a)$ can be inferred from a combination of different metrics such as the distance of the event location from the user's location,

number of times that the user u has attended a similar event, a questionnaire filled out by users, etc.

Social affinity is represented by a social network graph $G = (U, E, w)$ defined on U, where E represents connections between users and the function $w : U \times U \to \mathbb{R}_+$ maps a user-user pair to a value that measures the interest of two users in each other. The value $w(u, v)$ for two arbitrary users u and v can be measured using various criteria which indicate the level of social affinity between user u and user v, and it that can be estimated through various criteria such as the number of common hobbies, number of common events they have been to, their friendship status on online social networks, etc. We define N_u as the set of users who are interested in u, i.e. $N_u = \{v : w_{u,v} > 0\}$. We assume that the relationship between users is symmetric, that is, $w(u, v)$ is equal to $w(v, u)$. Also, we normalize user-user affinity values to be non-negative, where $w(u, v) = 0$ implies that there is no connection between the users.

1.2.1 Feasible Assignment

A feasible assignment for an instance of the SEO problem is a partial function M from users in U to events in A, represented by $M(u)$, which respects the event cardinality constraints. For every event $a \in A$, the maximum cardinality bound δ_a limits the number of users that can be assigned to event a and the minimum cardinality bound γ_a indicates the minimum number of users required for event a to be held. Also, it

should be noted that γ_a and δ_a are both non-negative values with $0 \leq \gamma_a \leq \delta_a$.

1.2.2 Utility Function

Assuming that the set of users S is assigned to event a, then the utility function $\mu(S, a)$ is defined as follows:

$$\mu(S, a) = (1 - \alpha) \sum_{u \in S} \sigma_{u,a} + \alpha \sum_{u,v \in S, u \neq v} w(u, v) \tag{1}$$

The utility function μ is a linear combination of the social and innate affinities. The social affinity term is a summation over user-user affinities gained by the pairs of users in S and the innate affinity term is a summation over the user-event affinities gained by the users in S_a. Also, this function employs a coefficient $\alpha \in [0, 1]$ in order to specify the relative effect of social and innate affinities on overall welfare and it is provided to the problem instance as input. It is trivial to see that by setting α to 0.5 there will be equal emphasis on both terms. Any value higher than 0.5 gives more importance to social affinity.

As we observe, the social affinity term is acquired by the participation of people with cross interest in one another, so it can be expected that having a group of friends attending to the same event will lead to a larger social affinity gain than when they are scattered among different events. On the other hand, innate affinity is solely

accumulated by innate interest of users in the event that they are assigned to.

Let $M^{-1}(a) = \{u \in U | M(u) = a\}$. Observe that in the case where there are no users assigned to event a, $M^{-1}(a) = \emptyset$. Finally, for a feasible assignment M, i.e. $\forall a \in A, M^{-1}(a) \neq \emptyset \rightarrow \gamma_a \leq |M^{-1}(a)| \leq \delta_a$, the total social welfare is defined by the objective function $\omega(M)$ as follows:

$$\omega(M) = \sum_{a \in A} \mu(M^{-1}(a), a) \tag{2}$$

We can rewrite the total social welfare function as follows:

$$\omega(M) = \sum_{a \in A} [(1-\alpha) \sum_{u \in M^{-1}(a)} \sigma_{u,a} + \alpha \sum_{u,v \in M^{-1}(a), u \neq v} w(u,v)] \tag{3}$$

The total social welfare function operates on a set of assignments made to the set of events A, simply by adding up the social welfare gains obtained though event assignments. As we mentioned earlier, $M^{-1}(a)$ returns the set of users assigned to a (and an empty set if no user is assigned to a), thus $\mu(M^{-1}(a), a)$ will calculate the social welfare gained by the users assigned to a.

The social event organization problem has been proposed and formally defined by [24] as follows:

Problem 1 *(SOCIAL EVENT ORGANIZATION (SEO)). Given a set U of users with a social graph $G = (U, E)$, a set A of events where each $a \in A$ has a minimum*

and maximum cardinality bound, denoted $\gamma_a \in \mathbb{R}$ *and* $\delta_a \in \mathbb{R}$ *respectively* ($\gamma_a \leq \delta_a$)*, an innate affinity function* $\sigma(\cdot,\cdot)$*, and a social affinity function* $\omega(\cdot,\cdot)$*, produce a feasible assignment* $M : U \rightarrow A$ *that has the maximum overall social welfare* $\omega(M)$*, i.e., find*

$$M^* = arg \quad max\{\omega(M)|\ M \text{ is feasible}\}.$$

1.3 Thesis contributions

The contributions of this thesis are listed below:

- We propose SCDG algorithm, a variation of the PCADG algorithm, which increment the overall social welfare gain by addressing the shortcomings of PCADG algorithm [24].
- We introduce the CASG algorithm that achieves a very good trade-off between efficiency and solution quality in practice.
- We propose a new objective function that takes into account the personality of users with respect to their extroversion, when evaluating a feasible assignment.
- We adapt PADG and SCDG to consider the abovementioned personality-oriented objective function.
- We conducted experiments to evaluate the studied and proposed algorithms.

Our experiments show that our new algorithms outperform the SEO algorithms introduced by [24] in terms of the overall social welfare and average regret ratio.

1.4 Outline of Thesis

In Chapter 2, we have conducted a comparative study of the problems that share similarities with social event organization problem proposed in [24], including the follow-up research that have been done in recent years. In the second chapter, we provide the notations and preliminaries, then we proceed to examine the existing solutions for SEO problem while proposing new algorithms. In Chapter 4, we present the results of experiments that we have conducted on both real and synthetic datasets. We conclude in Chapter 5, and give directions for future research.

Chapter 2

Related Work

The social event organization problem studied in this thesis is a specific type of assignment problem originally proposed by [24], in which the goal is to maximize a linear objective function by assigning users to events while respecting the cardinality constraints imposed on the events. The SEO problem has similarities to a wide range of combinatorial optimization problems which we will review in this chapter. Also, we will compare and contrast these problems to provide a deeper insight into the techniques that have been employed to conquer assignment problems. At the end of the chapter, we will give a conclusion to our review of the literature.

2.1 Knapsack Problem

The SEO problem is a descendant of the classic problem of Knapsack which falls under the topic of combinatorial optimization. The Knapsack problem is generally defined as picking out a subset of items out of a finite collection, such that the chosen items maximize an objective linear function while respecting a linear inequality constraint [22].

There are a number of variations of the knapsack problem that have been studied. One of the popular variations that can be regarded as one of the predecessors of SEO, is the 0-1 knapsack problem which is defined as follows:

Problem 2 (0-1 Knapsack problem) *An instance of the problem consists of a knapsack with a given capacity $C \in \mathbb{N}$ and n items. Each item, namely i, consumes a given volume from the container S_i and has a given value V_i. The aim of algorithm is choose a subset of the n items that maximizes the total value while ensuring that the total volume of the chosen items does not exceed the knapsack capacity* [33].

Fractional knapsack problem is a variation of the problem in which a fraction of each item can be taken to be put into the knapsack. The fractional knapsack problem can be solved using a simple greedy algorithm that fill the knapsack with fractions of items in decreasing order of the ratio their value to their size as long as the summation of the chosen items respects the knapsack capacity [10]. The 0-1 knapsack problem is known

to be NP-complete, however assuming that all of the item sizes are non-negative, it can be solved using a dynamic programming approach in pseudo-polynomial time of $O(nS)$ (n is number of items and S is the size of largest item).

Our problem is more similar to the former variation, in that sense that an item is either assigned to an event or it is excluded from the solution. Unlike 0-1 Knapsack, in the SEO problem there is an array of "containers" to which the items can be assigned. In this sense, our problem is close to the multidimensional Knapsack problem defined below:

Problem 3 (Multidimensional Knapsack Problem) *Given n items with profits $p_1, ..., p_n$ and weights $w_{i,j} \geq 0, i \in M = \{1, ..., m\}, j \in \{1, ..., n\}$ and m knapsacks with capacities $c_1, ..., c_m$, maximize the function $\sum_{j \in N} p_j.x_j$ such that the summation over the weight of items assigned to each knapsack does not exceed its capacity, i.e. $\sum_{j \in N} w_{i,j}.x_j \leq c_i, \forall i \in M$ and $x_j \in \{0, 1\}$ ($x_j = 1$ if the item j is placed in the knapsack, and $x_j = 0$ otherwise)* [6].

The multidimensional knapsack problem is NP-hard and similar to SEO, for each item there are multiple options and choosing one of them affects the rest of the assignments. The main difference between SEO and MKP is that in the knapsack problem, no pairwise profit is gained by assigning two items to the same knapsack. Various algorithms have been proposed to solve the multidimensional knapsack problem using different techniques such as dynamic programming [15], combination of dynamic

programming and heuristics [36] and linear programming-based branch and bound [31]. See the survey by Puchinger et al. for more details.

2.2 Generalized Assignment Problem

Another clan of related problems that has been widely studied in the so-called Generalized Assignment problem (GAP) [8, 12, 23, 32].

2.2.1 Bin Packing Problem

The Bin Packing problem is defined by [21] as follows:

Problem 4 (Bin Packing Problem) *Given n items with sizes $S_1, ...S_n$ such that $0 < S_i \leq 1$ and $S_i \in \mathbb{Q}, \forall i \in \{1, ..., n\}$. The goal is to pack the items into a minimum number of bins of capacity 1.*

The Bin packing problem is NP-hard and approximation approaches have been proposed to solve the problem in linear time. One of the approximations proposed for the problem, The first Fit algorithm, uses a simple greedy approach in which for each item, the first bin that the item fits in is chosen from a set of bins, and if no such a bin is found then a new bin is added to the set and the item is placed in it.

2.2.2 GAP with Minimum Quantities

A special case of GAP, called GAP with minimum quantities (GAP-MQ), sets a minimum constraint on the number of items that is required to be assigned to a bin. The problem is defined as follows:

Problem 5 (GAP with minimum quantities problem(GAP-MQ)) *Assume a setting in which we are given n items and m bins with capacities $B_1, ..., B_m \in \mathbb{N}$ such that each bin requires a minimum quantity $q_1, ..., q_m \in \mathbb{N}$ (for all bins $q_i \leq B_i$). Also, for each item i and bin j, we are given a size $s_{i,j} \in \mathbb{N}$ and a profit $p_{i,j} \in \mathbb{N}$ that will be gained through assigning i to j. The goal is to maximize the total profit by assigning items to bins while respecting capacity of the bins. Also, each bin b_i is either left empty or at least B_i items is assigned to it* [23].

The GAP-MQ problem is proven to be NP-hard but for the special case of fixed number of bins, a pseudo-polynomial time dynamic programming solution and a polynomial time dual approximation algorithm is proposed by [23]. Both the SEO and GAP-MQ problems are maximization problems in which the main task is to assign items to containers (or bins) in such a way as to maximize the profit that is gained through the assignment. In SEO the size of items are all equal to 1, and it is more similar to a specific case of GAP-MQ, called the Seminar Assignment problem (SAP) which is defined by [23] as follows:

Problem 6 (Seminar Assignment problem) *Assume m seminars with minimum quantities $q_1, ..., q_m \in \mathbb{N}$ and maximum capacities $B_1, ..., B_m \in \mathbb{N}$ and n students. For each student i $(1 \leq i \leq n)$ and seminar j $(1 \leq j \leq m)$, a profit $p_{i,j}$ is accumulated through assigning student i to seminar j. The problem is defined as maximizing the total profit by assigning students to seminars while respecting the minimum quantity (in case of opening a seminar by assigning students to it) and maximum capacity of the seminars* [23].

It is proven in [23] that SAP is not only NP-hard but it does not have a PTAS using a reduction from the 3-bounded 3-dimensional matching problem. Also, it is shown that for a special case of SAP where the number of m is constant, the problem can be solved in polynomial time by reducing it to an instance of minimum cost flow problem.

Similar to SAP, the SEO problem can be partially modeled by a bipartite graph that consists of a set of users and a set of events and the profit of assigning a user to an event is represented by the weighted edge drawn from the user to event. Although SEO and SAP are quite similar, the pairwise interest between users which is modeled as a social network in SEO, differentiates the two problems. The social network of interests between users makes the SAP solutions unusable for the SEO problem.

It is trivial to see that a special case of SEO in which no social network is defined on the users is the same as the SAP problem. It follows that SEO is NP-hard.

2.3 Matching Problems

Another special case of SEO is the setting in which the minimum and maximum capacity of events are set to one. In this case the social network defined on users is irrelevant to the problem definition and it maintains slight similarities to the Stable Marriage problem, which is defined as follows:

Problem 7 (Stable Marriage problem) *Assume a bipartite graph consisting of a set of men $M_1, ..., M_n$ and a set of women $W_1, ..., W_n$ and a set of weighted edges E that connects each person to all members of the opposite gender, such that each weight from person i to person j expresses the interest of i in j. The objective of the problem is to assign each man to a woman so that there is no pair of a man and a woman who both prefer the other to their actual match.* [7]

Gale and Shapley in [14] propose an algorithm that solves the problem in time $O(n^2)$. The algorithm works through rounds of assigning men to the women who they are interested in the most. The women reply back to the proposed men by marking the matched pair as engaged. In the next round, the unengaged men propose to the women they prefer the most, and the women, revise their choices by getting engaged to a new man if he maintains a higher rank. The process continues as long as there is an available person.

The similarity between the dynamic greedy solution proposed for SEO (Section

3.3.2, Chapter 3) and Gale-Shapley is that, they both employ a revisioning strategy to find the best possible solution. The dynamic Greedy approach (discussed in Section 3.3.2) follows the same iterate in which users "apply" for the event they prefer the most and this process continues until the minimum number of users required by an event is acquired. The resemblance in both algorithms is that an item (a man in SMP or a user in SEO) greedily choose the best option that he has in mind at the first round and in the next rounds of algorithm, when he is given more options, he revises them to find the most promising one.

2.3.1 National Resident Matching Program

One of the motivations to study SEO is the National Resident Matching Problem, that is matching graduating applicants with residency programs considering the given preference ordering of applicants to programs and programs to applicants (both are called Rank Order Lists or ROLs) while respecting the residency programs' capacities (number of open positions offered by each program) [29].

Here we investigate the similarities and differences between NRMP and SEO, assuming that applicants in NRMP are equivalent of users in SEO and residency programs in NRMP are equivalent of events in SEO.

First of all, NRMP aims to find a stable matching between applicants and residency programs based on their two-sided preferences (the same as SMP), while SEO's

objective is to assign users to events based on their interest in events and there is no preferential ranking of the users considered from event organizers' point of view (similar to SAP). Secondly, both NRMP and SEO define maximum cardinality constraint on the number of accepted applicants/users per program/event, although in SEO, in addition to maximum bound, we are given a minimum bound per event (similar to SAP).

Finally, in NRMP applicants may apply in groups of two, therefore the algorithm will assign them to the same program. On the other hand, in SEO decisions for users are made independently and the social network defined on the users is considered to increase the opportunity of assigning people with interest in each other to the same event.

The NRMP algorithm given in [29] iterates though applicants and for each applicant A, the most preferable program with available positions or the program with a tentative match of a less preferable applicant B, is chosen. In the latter case, applicant B's rank order list of programs is examined and the next most preferred open program (program with vacant position) or the one with a tentative match who is less preferable than B, namely C, is chosen. The tentative matching of C will be canceled and B will be displaced. This process will continue until either no more positions have been left or all the applicants are matched with the programs [29].

2.4 Social Event Organization Problem

As mentioned in Chapter 1, our focus in this thesis is on the problem introduced by [24], that is the problem of providing a set of event assignments for a group of users according to their interest in the events and also their interest in each other. The problem statement also includes the minimum and maximum cardinality bound constraints associated with events which a feasible solution to the problem should respect.

The proposed solutions utilize a greedy approach while updating the heuristics as the assignments are produced. Since our algorithms are the algorithms in [24] as a starting point, we will describe their algorithms in detail in the next chapter.

2.4.1 Bottleneck-aware arrangement over event-based social networks

A follow-up research on the social event organization problem has been done by Tong et al., in which the employed objective function assigns users to events based on their *distance to the event location*, number of shared attributes between events and users, and finally the relationships among users [34].

Here, unlike SEO, the social network graph merely determines whether two users are mutually connected or not, and the problem states that a user-event assignment

requires that there should be at least one of the user's friend to be assigned to the same event. While this approach reduces the complexities of having a weighted social network graph involved in the assignment process, it reduces the flexibility, in the sense that a user may prefer to attend an event merely due to his/her interest, regardless of other attendees. Also, this approach fails to consider the impact of the presence of people from the user's social circle on the preference of an event over another. Finally, the objective function employed by SEO is more comprehensive in the sense that the shared similarities, the distance between user and events, and many other conditions can be included while evaluating a user's interest in an event.

The research done by [34], proposes a greedy approach in which the list of events is traversed and for each event, a list of available users is examined in the non-ascending order of gain, produced by the suggested objective function. Then, if there is a friend already assigned to that event, the assignment is finalized.

2.4.2 Event Organization Scheme

In one of the latest works done on the social organization problem, Huang et al. have defined a new scheme in which the input is the same as SEO but a user will receive a set of recommended events instead of being assigned to only one event. Also, a user-event connection, called *willingness strength*, is calculated by employing a ranking based method and a probabilistic approach. The result is a weighted bipartite graph,

consisting of users, events and weighted edges between them, such that weights of edges represent a recommendation score. Finally, the edge extraction is applied by using a greedy approach [19].

The definition of social organization problem and the solutions proposed by [19] falls into the category of recommendation systems, rather than the assignment problem which is the subject of this thesis.

2.4.3 Utility-Aware Social Event-Participant Planning

In another paper, She et al. propose the problem of planning a non-conflicting schedule of events, which is proven to be NP-hard [30]. The problem is defined as follows: Given a group of users, a set of events , the event time-tables, event capacities, a user-event utility vector, the cost of attending events and the locations of both users and events, produce a feasible and non-conflicting event participating schedule such that the total cost is minimized (the required budget and the distance to the event location) while the overall satisfaction defined by user to event utility values is maximized [30].

The proposed solutions to the problem use a profit/cost ratio to evaluate the events and similar to SEO, [30] uses greedy approaches to conquer the problem. In another effort, the problem is to broken into subproblems which are solved separately using dynamic programming techniques and at the end, the conflicting cases are eliminated.

The UASEP problem demonstrates major differences to the SEO problem, including the absence of a social network defined between users which enables it to employ a bottom-up approach. Also, this problem allows a user to be assigned to multiple events (as long as it does not violate the feasibility condition) which makes it inherently different from the SEO problem.

2.5 Conclusion

The majority of the assignment and matching problems reviewed in this chapter, unlike SEO, can be modeled as a weighted bipartite graph consisting of a set of items and a set of bins. Despite their similarities to SEO, the affiliation network defined between users, outlaws the possibility of employing the approaches studied in the chapter to solve the SEO problem.

The main difference between the discussed problems and SEO problem comes from the fact that the objective function employed by SEO includes the pairwise interest between users which is accumulated through assigning users with social interest in each other to the same event, meaning that once a user is assigned to an event or a user is removed from it, the interest of other users to that event will change. The solutions proposed for SEO in [24] will be explained in detail in the next chapter.

Chapter 3

Algorithms

In this chapter, we describe our algorithms for the SEO problem. We start with some definitions and conventions used in all algorithms we describe. As discussed in Chapter 1, the SEO problem is a type of assignment problem, where the objective is to maximize the profit by assigning objects with unit capacity to containers entailing minimum and maximum capacities, where the profit of loading objects to containers is defined by a weighted graph induced on objects and a weighted bipartite graph between objects and containers.

The original paper [24] employs a user-event analogy in which users are the objects and events represents the containers. We will use the same terminology throughout our research. The output of algorithms proposed for SEO is a feasible assignment that is a global assignment that respects the cardinality constraints on the occupied

events.

As discussed in Chapter 1, Section 1.2.1, a feasible assignment should respect the cardinality constraints associated with the events. During the execution of the SEO algorithms, users are assigned to events. Initially, all events have no users assigned to them, but as the algorithm proceeds, events can be in the following states with respect to the number of the users (indicated by $|S_a|$):

1. Phantom: When the number of users assigned to an event falls below the minimum cardinality constraint, i.e. $|S_a| < \gamma_a$, we mark the event as "phantom" to denote the fact that minimum cardinality constraint γ_a has been violated via the assignment.

2. Real: If the number of users assigned to an event reaches the minimum cardinality, i.e. $\gamma_a \leq |S_a| \leq \delta_a$, we mark the event as "real".

3. Open: As long as the number of users assigned to an event falls below the maximum cardinality constraint, the event as is marked as "open", i.e. $|S_a| < \delta_a$. An open event can be real or phantom.

4. Closed: We mark the event as "closed" if the maximum cardinality constraint has been met, i.e. $|S_a| = \delta_a$, this implies that no more users can be assigned to the event. A closed event is real.

In the social welfare evaluation, we only consider real events and phantom events

are disregarded through the evaluation process. Depending on the input and the SEO algorithm employed, a user may be assigned to an event or it may be left undecided, which means the algorithm hasn't assigned the user to any event. The assignment function $M : U \to A$ assigns users to their finalized event in A. For every user $u \in U$ there are two cases for $M(u)$:

1. $M(u) = a$: User u assignment is committed to event a. At this time the user is regarded as $Unavailable$.

2. $M(u) = \emptyset$: User u is not assigned to any event. At this time the user is regarded as $Available$.

The SEO algorithms presented by [24] may attempt different assignments for a user and these practices do not have any impact on $M(u)$, but as soon as the decision is made the $M(u)$ will be allocated to the finalized event and this decision will not be reversed at any further point of the algorithm. On the other hand algorithms proposed by this thesis (including CASG, PCASG and SCDG) have a post-processing phase in which the assignments given by M may be changed if they improve the overall social welfare.

3.1 Hardness Results

As discussed in [24], the SEO problem is NP-hard, even for two special cases where there is only innate interest between users (i.e. $\forall <u, v> \in U \times U : w_{u,v} = 0$) and events and where there is only social interest between users (i.e. $\forall <u, a> \in A \times U : \sigma_{u,a} = 0$).

We provide three theorems from [24] to make these cases clearer:

Theorem 1 *The decision version of Social Event Organization is NP-complete.*

Theorem 2 *It is NP-hard to approximate SEO-Innate within a factor of* $(1 - 1/n + \epsilon)(\forall \epsilon > 0)$ *in polynomial time, where* n *is the number of users.*

Theorem 3 *Assuming the Unique Games with Small Set Expansion Conjecture, it is NP-hard to approximate SEO-Social within any constant factor in polynomial time.*

In light of the above hardness and inapproximability results, in the rest of this chapter we study heuristics and techniques to solve the problem.

3.2 Outline

In this section, we outline the structure of the SEO algorithms studied in this thesis. The SEO algorithms utilize Dynamic Greedy strategy and Static Greedy strategy,

both of which employ a greedy approach in which users are assigned to events by criteria based on the social and innate affinities. Regardless of the employed algorithm, the entire assignment process can be described in four phases:

Initialization: At the initialization phase we create a list $\mathcal{L}$ of all possible $<user - event>$ pairs, i.e. $<u, a> \in U \times A$. The list $\mathcal{L}$ represents all the possible assignments. Algorithms that use dynamic greedy strategy assign a lookahead value to each $<u, a>$ pair which is the social welfare that is accumulated by assigning user u to event a. At the initialization phase of the algorithms proposed by [24], for a pair of user-event $\{<u, a>|u \in U, a \in A\}$ the utility value is merely the innate interest of user u in event a. We will propose a new approach regarding the utility value initialization in this research. The list $\mathcal{L}$ is sorted in descending order of the utility value.

Assignment: The assignment process consists of picking a user-event pair out of list $\mathcal{L}$ and attempting an assignment if it respects cardinality constraints of the event that user is assigning to and also it meets the regarded criteria that will be explained later in this chapter. All the algorithms described here choose the pair with highest lookahead value to maximize the social welfare gained by the solution.

The assignment process stops when there is no more pair to inspect, or in other

words when the list $\mathcal{L}$ is empty. At the end of assignment process the mapping function M will contain the permanent assignments that are decided for users in U.

Update: The algorithms which employ the dynamic greedy approach update the utility gain whenever an assignment is made using the heuristics described bellow:

Baseline Heuristic: The dynamic greedy algorithms proposed by [24], update the interest of users in events every time that an assignment is made. When a user x is assigned to an event a, i.e. $S_a \leftarrow S_a \cup x$, the interest of available users toward a will increase depending on their interest in users who have been assigned to a. Thus, we will update tuples of event a in $\mathcal{L}$ using the function that follows:

$$g(u, a|S_a) = (1 - \alpha)(\sigma_{u,a}) + \alpha \sum_{v \in S_a} w(u, v) \tag{4}$$

Community-Aware Heuristic: Community in the context of a social network defined on a user set U, essentially is a subset of the users $C \subseteq U$ such that the internal connections between users of C are stronger than their association with the rest of network [13]. In other words, the users in a community tend to have more interaction with other people from the

same community; therefore they can be expected to participate in social events together.

As we mentioned in Section 1.2, the SEO problem consists of a graph $G = (U, E, w)$ defined on a set of users U and the term w represents weight of edges E between users. The community aware SEO algorithms try to assign people that have a strong bond between them to the same event. In this way, we will increase the social affinity accumulated through the user-event assignments.

Community aware algorithms proposed by [24] employ a heuristic function that presumes the remaining spots in an event will be occupied by the friends of users who already are assigned to it, i.e. given user-event pair $<u, a>$, and regarding the partial assignment S_a that has been made to event a, th utility function takes an average on all unassigned users social affinity to u and multiplies it by the empty spots in a.

Given the user-event pair $<u, a>$, the assignment set S_a and the set of users that hasn't assigned to any event yet, i.e. $V = \{u | u \in U \wedge M(u) =$

$\emptyset \wedge u \notin \cup_{a \in A} S_a\}$, the gain in utility is defined as follows:

$$g(u, a|S_a) = (1 - \alpha)(\sigma_{u,a}) + \alpha \sum_{v \in S_a} w(u, v)$$
$$+\alpha(\delta_a - |S_a|) \sum_{v \in V} w(u, v)/(|V| - 1) \quad [24, p.6] \tag{5}$$

Reinitialization: When the assignment process is over, it is probable that there will be user(s) in U that are not assigned to any event, i.e. $\exists u \in U \quad s.t. \quad M(u) = \emptyset$. In the reinitialization phase, the available users along with the events that are still open (have room for more users to be assigned to) are inserted into list $\mathcal{L}$. After the reinitialization phase, the assignment phase will be executed again. We will run reinitialization-assignment cycle as long as there are available users and open events left.

As we mentioned earlier the list $\mathcal{L}$ is initialized with all possible user-event pairs. In the reinitialization strategy employed by algorithms proposed by [24], the list $\mathcal{L}$ is reloaded with all the possible pairs consists of the currently available users and real open events, i.e. $\mathcal{L} = <u, a> \in U \times A \quad s.t. \quad M(u) = \emptyset \quad and \quad \gamma_a \leq |S_a| < \delta_a$. The assignment-reinitialization cycle terminates when there is no user available or no open event.

In this thesis we propose a new reinitialization strategy which includes a subset of phantom events along with real open events, i.e. $A' \subseteq \{a \in A \quad s.t. \quad |S_a| <$

$\delta_a\}$. This approach will give users another chance to be assigned to events that they intended to attend but it turns out that there are not enough users assigned to them at the end of an assignment round (when list $\mathcal{L}$ becomes empty). The problem with this approach is that the set of A' could remain either in the same size or its size will decrease slowly in the reinitialization-assignment rounds. In fact, it is possible the algorithm does not terminate. To address this issue we will introduce a minimum improvement threshold to decide when to terminate the reinitialization-assignment cycle.

Post-Processing: We introduce a new post-process phase in the algorithms proposed by this research to improve the social welfare gained through the previous stages. Once the assignment process is over we will adjust the assignments through the procedures discussed below:

• **Event Switch:** In the event switch phase, we swap the events assigned to users if it increases the total social welfare gain. We iterate through all the pairs of unavailable users, i.e. $<u_1, u_2> \in U \times U \quad s.t. \quad M(u_1) = \emptyset \quad and \quad M(u_2) = \emptyset$ and test to see if switching the events to which they are assigned, i.e. $M(u_1) \rightleftarrows M(u_2)$, will lead to a greater social welfare. It is trivial to see that in this process, the number of users assigned to each event will remain the same, but the social welfare will increase.

• **User Substitution:** During the assignment process a number of pairs in list $\mathcal{L}$ are ignored due to the capacity constraints of events. In the user substitution process, for each user-event pair $<u, a>$, we try to assign u to a by substituting it with one of the users who have been already assigned to a during the assignment process, such that the substitution leads to a higher utility gain. To find the best candidate to remove from a, we use the utility function 1 such that the value of $\mu(S, a)$ after substitution gives the highest utility gain among all the options.

We demonstrate the algorithm template using the above 4 phases in the following Algorithm:

Algorithm 1 Template$(U, A, \sigma, \omega, \alpha)$

```
1: Initialize L with user-event pairs
2: while |L| > 0 do
3:     Pop a user-event pair <u, a> from L
4:     if <u, a> respects the cardinality constrains of event a then
5:         Assign u to a
6:     end if
7:     Update list L if necessary
8:     if |L| = 0 then
9:         Reinitialize L with available users and open events
10:    end if
11: end while
12: Post process of assignments
13: return Produced assignments
```

All of the algorithms we describe in The subsequent sections follow the template of 1 but differ in the specific strategies represented in lines 3, 7, 9, 12 which we

will discuss in detail in the following sections.

3.3 Algorithms Review

In this section we study the algorithms for the SEO problem proposed by [24]. Furthermore, we discuss their shortcomings and problems.

3.3.1 Static Pairwise Greedy Algorithm

- **Initialization:** Static Pairwise Greedy (SPG) algorithm assigns two users to an event at a time. In the initialization phase, SPG loads the list $\mathcal{L}$ with $<user_1, user_2, event>$ tuples (represented in Algorithm 3). The pairwise user-event tuples represent the possible assignments where two users in a tuple are assigned to the same event. Each tuple is accompanied with a lookahead value which is defined by the gain function g as follows:

$$g((u,v),a) = (1-\alpha)(\sigma_{u,a}) + 2\alpha w(u,v) \quad [24, p.5] \tag{6}$$

Finally the list $\mathcal{L}$ is ordered by gain values in non-increasing order.

Algorithm 2 SPG$(U, A, \sigma, \omega, \alpha)$

$\mathcal{L}$ = SPG_Initialization$(U, A, \sigma, \omega, \alpha)$
while $< u, v, a, g(u, a \mid S_a) > \leftarrow \mathcal{L}.pop()$ **do**
  if $M(u) = \perp$ *and* $M(v) = \perp$ *and* $|S_a| + 1 < \delta_a$ **then**
    SPG_Assignment$(u, v, a, U, A, \sigma, \omega, \alpha, \mathcal{L})$
  end if
  if $\mathcal{L}.empty()$ **then**
    SPG_Reinitialization$(U, A, \sigma, \omega, \alpha, \mathcal{L})$
  end if
end while
return M

Algorithm 3 SPG_Initialization$(U, A, \sigma, \omega, \alpha)$

$M(u) \leftarrow \perp, \forall u \in U; S_a \leftarrow \emptyset, \forall a \in A$
for `each` $(u, v, a) \in U \times U \times A$ **do**
  $\mathcal{L}.insert(< u, v, a >)$
end for
Sort list $\mathcal{L}$ by gain values in non-increasing order

- **Assignment:** In assignment phase, we pass through the list $\mathcal{L}$, examining each tuple $<u, v, a>$ to assign both u and v to event a if both u and v are available and event a has two free slots, i.e. $\delta_a - |S_a| \geq 2$. If the availability and cardinality constraint conditions are not met then the tuple is simply ignored. If the assignment is made, both users are marked as unavailable.

  Once a phantom event $a \in \mathcal{P}$ receives the required minimum number of users, i.e. $|S_a| = \gamma_a$, all these users will be permanently assigned to it, i.e. $\forall u \in S_a : M(u) = a$ and the event is marked as real by removing it from phantom event list, i.e. $\mathcal{P} \leftarrow \mathcal{P} \backslash a$. When user u is permanently assigned to an event a,

it is removed from all the events that it was previously chosen for, i.e. $\forall e \in A \quad s.t. \quad u \in S_e : S_e \leftarrow S_e - \{u\}$. For the next user-event pairs that concern the real event a, the user will be permanently assigned to it as long as event a has not reached its capacity (i.e. $|S_a| < \delta_a$).

Algorithm 4 SPG_Assignment$(u, v, a, U, A, \sigma, \omega, \alpha, \mathcal{L})$

```
1: S_a ← S_a ∪ {u, v}
2: if |S_a| > γ_a then
3:     M(u) ← a
4:     M(v) ← a
5:     for all a' ∈ A s.t. a' ≠ a ∧ (u ∈ S_a' ∨ v ∈ S_a') do
6:         S_a' ← S_a' − {u}
7:         S_a' ← S_a' − {v}
8:     end for
9: end if
10: if |S_a| = γ_a then
11:     for all r ∈ S_a do
12:         M(r) ← a
13:         for all a' ∈ A s.t. a' ≠ a ∧ r ∈ S_a' do
14:             S_a' ← S_a' − {r}
15:         end for
16:     end for
17: end if
```

- **Update:** This algorithm tries each possible assignment only once and it does not update the potential gain values as the assignments are made.

- **Reinitialization** At the end of assignment process, in the reinitialization phase, $\mathcal{L}$ is reloaded with open events and available users in the same manner as in the initialization phase. The Assignment-Reinitialization cycle will continue until there is no open event or all the users are already assigned.

Algorithm 5 SPG_Reinitialization$(U, A, \sigma, \omega, \alpha)$

1: **for** `each` $(u, v, a) \in U \times A \;\; s.t. \;\; M(u) = \perp \wedge M(v) = \perp \wedge \gamma_a \leq |S_a| < \delta_a$ **do**
2: $\quad \mathcal{L}.insert(< u, v, a, g(u, v, a \mid S_a) >)$
3: **end for**

- **Post-Processing:** This algorithm does not perform post-processing.

- **Time Complexity**: It is trivial to see the time complexity for going the loop without reinitialization is $O(|\mathcal{L}| \log(|\mathcal{L}|) + |\mathcal{L}|) = O(|U||A| \log(|U||A|) + |U||A|)$, as the list $\mathcal{L}$ is sorted at initialization phase in $O(|\mathcal{L}| \log(|\mathcal{L}|)$ and subsequently the list $\mathcal{L}$ is traversed only once through pop operations while each pop operation takes $O(1)$. After each reinitialization, at least one user must be assigned an event; if not, we terminate the algorithm. Thus, there can be at most $|U|$ re-initializations, and the total time required in the worst case is $O(|U|^2|A| \log(|U||A|) + |U|^2|A|)$.

3.3.2 Dynamic Greedy Algorithm

The Dynamic Greedy (DG) algorithm, demonstrated in Algorithm 6, retains the same structure as SPG, except that DG considers one user at a time while SPG assigns a pair of users to an event. Moreover, DG performs update operation (discussed in Section 3.2) while SPG does not.

Algorithm 6 DG$(U, A, \sigma, \omega, \alpha)$

$\mathcal{L}$ = Initialization$(U, A, \sigma, \omega, \alpha)$
while $< u, a, g(u, a \mid S_a) > \leftarrow \mathcal{L}.pop()$ **do**
  if $M(u) = \perp$ *and* $|S_a| < \delta_a$ **then**
    DG_Assignment$(U, A, \sigma, \omega, \alpha, \mathcal{L})$
  end if
  DG_Update$(U, A, \sigma, \omega, \alpha)$
  if $\mathcal{L}.empty()$ **then**
    DG_Reinitialization$(U, A, \sigma, \omega, \alpha, \mathcal{L})$
  end if
end while
return M

- **Initialization:** In the initialization phase (represented in Algorithm 7), we create a list of user-event pairs that are associated with a potential gain value calculated using Equation 4.

Algorithm 7 DG_Initialization$(U, A, \sigma, \omega, \alpha)$

$M(u) \leftarrow \perp, \forall u \in U; S_a \leftarrow \emptyset, \forall a \in A$
$p \leftarrow 0, DP \leftarrow \emptyset$
for `each` $(u, a) \in U \times A$ **do**
  $\mathcal{L}.insert(< u, a >)$
end for

- **Assignment:** In the assignment process, we traverse the list $\mathcal{L}$ in non-increasing order of gain value; for the pair user-event $<u, a>$, we will assign user u to event a if no cardinality constraint is violated by the assignment. Assigning u to a is done by calling DG_Assignment procedure (demonstrated in Algorithm 8).

Algorithm 8 DG_Assignment$(U, A, \sigma, \omega, \alpha, \mathcal{L})$

$S_a \leftarrow S_a \cup \{u\}$
if $|S_a| > \gamma_a$ **then**
    $M(u) \leftarrow a$
    for all $a' \in A \ \ s.t. \ \ a' \neq a \wedge u \in S_{a'}$ **do**
        $S_{a'} \leftarrow S_{a'} - \{u\}$
    end for
end if
if $|S_a| = \gamma_a$ **then**
    for all $v \in S_a$ **do**
        $M(v) \leftarrow a$
        for all $a' \in A \ \ s.t. \ \ a' \neq a \wedge v \in S_{a'}$ **do**
            $S_{a'} \leftarrow S_{a'} - \{v\}$
        end for
    end for
end if

- **Update:** DG performs an update operation on potential gain values associated with affected user-event pairs in list $\mathcal{L}$ every time an assignment is made. Update procedure is represented in 9.

Algorithm 9 DG_Update$(U, A, \sigma, \omega, \alpha)$

for all $v : (w(u, v) > 0 \wedge M(v) = \perp)$ **do**
    $\mathcal{L}.update(< v, a, g(v, a \mid S_a) >)$ ▷ Using Eq.4
end for

Suppose the pair $<u, a>$ is chosen at iteration i when $|S_a| < \delta_a$. After tentative assignment of u to a, potential gain values assigned to pairs consisting the available users that are interested in user u and the event a, i.e. $\{<v, a'>|a' = a \wedge v \in U \wedge w(u, v) > 0 \wedge M(v) = \emptyset\}$, will be update according to the new assignment. In this way it is assured that when a user is assigned to an event

(either tentatively or permanently), his friends are aware of this assignment so they can make a more informed decision.

- **Reinitialization:** Once the list $\mathcal{L}$ is empty the reinitialization process is executed by calling DG_Reinitialization (demonstrated in 10). In this process the user-event pairs consisting of the available users and the open events are inserted to the list $\mathcal{L}$.

Algorithm 10 DG_Reinitialization$(U, A, \sigma, \omega, \alpha)$

1: **for** `each` $(u, a) \in U \times A \;\; s.t. \;\; M(u) = \perp \wedge \gamma_a \leq |S_a| < \delta_a$ **do**
2: $\quad \mathcal{L}.insert(< u, a, g(u, a \mid S_a) >)$
3: **end for**

- **Post-Processing:** This algorithm does not perform post-processing.

- **Time Complexity:** DG time complexity is the same as PCADG and it is discussed in Section 3.3.4.

3.3.3 Phantom-Aware Dynamic Greedy

The Phantom-Aware Dynamic Greedy (PADG) algorithm proposed by [24], is a variation of DG that employs the preemptive phantom-aware strategy to minimize the number of phantom events. PADG (demonstrated in Algorithm 11) perform the same steps as DG, except that the assignment process is executed by calling PADG_Assignment (represented in Algorithm 12).

Algorithm 11 PADG$(U, A, \sigma, \omega, \alpha)$

$\mathcal{L}$ = DG_Initialization$(U, A, \sigma, \omega, \alpha)$
while $< u, a, g(u, a \mid S_a) > \leftarrow \mathcal{L}.pop()$ **do**
  if $M(u) = \perp$ *and* $|S_a| < \delta_a$ **then**
    PADG_Assignment$(U, A, \sigma, \omega, \alpha, \mathcal{L})$
  end if
  DG_Update$(U, A, \sigma, \omega, \alpha)$
  if $\mathcal{L}.empty()$ **then**
    DG_Reinitialization$(U, A, \sigma, \omega, \alpha, \mathcal{L})$
  end if
end while
return M

- **Initialization:** The initialization phase is identical to DG algorithm.

- **Assignment:** For a user-event $<u, a>$, such that $|S_a| = 0$ (meaning that no user has been assigned to a yet), if the number of available users $|V|$ is less than the minimum capacity γ_a then the pair u, a and also any pairs targeted to event a will be ignored. This is because the event a is certain to be a phantom event at the end of the assignment phase. Instead, we prefer to allow the users in these user-event pairs to be assigned to real events.

Algorithm 12 PADG Assignment$(U, A, \sigma, \omega, \alpha)$

```
1: if |S_a| = 0 then
2:     if deficit + γ_a ≤ |V| then
3:         deficit ← deficit + γ_a − 1
4:         P ← P ∪ {a}
5:     else
6:         Continue
7:     end if
8: else
9:     if a ∈ P then
10:        deficit ← deficit − 1
11:    end if
12: end if
13: S_a ← S_a ∪ {u}
14: V ← V\{u}
15: if |S_a| > γ_a then
16:    M(u) ← a
17:    for all a′ ∈ A s.t. a′ ≠ a ∧ u ∈ S_a′ do
18:        S_a′ ← S_a′ − {u}
19:    end for
20: end if
21: if |S_a| = γ_a then
22:    P ← P\{a}
23:    for all v ∈ S_a do
24:        M(u) ← a
25:        for all a′ ∈ A s.t. a′ ≠ a ∧ v ∈ S_a′ do
26:            S_a′ ← S_a′ − {v}
27:        end for
28:    end for
29: end if
```

Phantom-Aware Strategy is implemented by maintaining an accumulative $deficit$ value throughout the assignment process to avoid assigning users to events that have no chance of acquiring minimum number of users at the end of assignment process. In other words, $deficit$ is the number of users that is required to realize

the phantom events that at least one user is assigned to them. Assuming that $\mathcal{P}$ is the set of phantom events, i.e. $\mathcal{P} = \{a \in A \quad | \quad 0 < |S_a| < \gamma_a\}$, the deficit value can be calculated as follows:

$$deficit = \sum_{a \in \mathcal{P}} \gamma_a - |S_a| \tag{7}$$

In assignment process, for an event a that there is no assignment made yet, i.e. $|S_a| = 0$, if the number of available users $|V|$ is less than $deficit$, then we ignore all the user-event pairs that is targeted to a. By employing this strategy, we prevent assigning users to events that will remain in phantom state by the end of assignment process and thus, the overall social welfare will increase.

- **Update:** The update procedure is identical to DG algorithm.
- **Reinitialization:** The reinitialization procedure is identical to DG algorithm.
- **Post-Processing:** This algorithm does not perform post-processing.

3.3.4 Community-Aware Phantom-Aware Dynamic Greedy

PCADG is a version of PADG algorithm that employs the community-aware utility gain function introduced in Section 3.2. This method assumes that as a user is assigned to an event, the rest of the empty spots in the event will be occupied by its

friends via increasing the potential gain value of their corresponding tuples in $\mathcal{L}$ that is aimed to user's event.

Time and Space Complexity: As stated in [24], for each user-event pair in list $\mathcal{L}$, $|N_u|$ update operations with constant time of execution takes place. As for every user the update operations are executed only one time (when they are permanently assigned to an event), the whole update operations take $O(|E|)$, which $|E|$ is an upper bound to the number of edges between pair of users (u, v) such that $w(u, v) > 0$. Also, as every pair in $\mathcal{L}$ one pop operation takes place and each pop operation is executed in $O(\mathcal{L})$, the overall algorithm takes $O(|\mathcal{L}|^2 + |E|) = O(|U|^2|A|^2 + |E|)$. Finally, by having $|E| = |U|(|U|-1)/2) = O(|U|^2)$, and considering the fact that there may be $|U|$ reinitialization phases, the overall time complexity of $PCADG$ is $O(|U|^3|A|^2 + |U|^3)$. Also, the space complexity of PCADG is $\Omega(\mathcal{L} + \omega(E))$ [24].

3.4 Problems with the described approaches

There are a number of problems with approaches in [24] which we will discuss below:

Problem with Community-Aware Gain Function The authors of [24] claim the community-aware gain function "optimistically assumes that the remaining spots in the event will be filled with friends of u." We describe three problems with this function:

1. Consider an instance of the problem where $|S_{a_1}| = 0$, $\delta_{a_1} = 2$, $\sigma_{u_1,a_1} = 1$ and a user-event pair $<u_1, a_1>$ is being updated during the assignment process. Also, consider the available users u_2, u_3 and u_4 with social interests 6,0 and 0 in user u_1 respectively. By updating the gain of assigning u_1 to a_1 using Equation 5 we will have:

$$\begin{aligned} g(u_1, a_1|S_{a_1}) &= (1-\alpha)(\sigma_{a_1}) + \alpha(\delta_{a_1})(\sum_{v \in V} w(u_1, v))/(|V|-1) \\ &= (1-\alpha) + \alpha(2)(6+0+0)/(3) \\ &= (1-\alpha) + 4\alpha \end{aligned}$$

It is observed that the community-aware function pessimistically estimates 4α additional gain (resulting from the third term in Equation 5), while the actual gain resulting from assigning u_2 to a_1 gives us 6α which is higher than the estimated gain. This example shows community-aware utility gain proposed in [24] actually *underestimates* the potential gain in social welfare.

2. Now consider the same example, except that $\delta_a = 4$, then we will have:

$$g(u_1, a_1|S_{a_1}) = (1-\alpha)(\sigma_{a_1}) + \alpha(\delta_{a_1})(\sum_{v \in V} w(u_1, v))/(|V|-1)$$
$$= (1-\alpha) + \alpha(4)(6+0+0)/(3)$$
$$= (1-\alpha) + 12\alpha$$

   In this case, the community-aware function *over-estimates* the additional utility gain to be 12α although it is 6α in fact.

3. During the update process for a pair $<u, a>$, the community-aware gain function (Equation 5) includes all the available users and we do not check if they are friends of user u. Moreover, some of the available users may not have innate affinity towards event a at all. Thus, it may be unrealistic to assume that all these users will be assigned to event a. A solution to this problem is to only consider available users with social affinity for u and innate affinity for a. however, this will not completely solve the problem as one set of available users may increase the utility gain of multiple user-event pairs that are targeted to different events, although that user can be assigned to only one of those events. In fact the function outlined here maybe is too optimistic, and not realistic.

False Promise Problem This problem can arise when a user, namely u, is

temporarily assigned to multiple phantom events, which we will refer to as the event set β_u. Now assume a situation in which there are other users assigned to events in β_u because of their social interest in u.

As soon as one of the phantom events from set β_u, namely a', receives the minimum number of users, user u will be removed from all other phantom events' assignment lists and it will be permanently assigned to event a' by having $M(u) = a'$. Now, all of u's friends will be still assigned to their events, because of the false promise of having u on board. In this situation there is a chance that in the absence of user u, they prefer other events rather than their current ones. We will address this issue with our SCDG algorithm which tries to give users another chance.

Conflicting Heuristic Functions Problem PCADG employs two different heuristic functions which conflict with each other. In the initialization phase the algorithm only includes innate interest to calculate heuristic values of user-event pairs, while during the assignment phase a user-event, namely $<u, a>$, the heuristic value calculated by function $g(u, a|S_a)$ (Equation 5) includes the average social interest of available users to the event a, assuming that the rest of free spots in the event will be occupied by available users who are interested in u. We will address this problem in our SCDG algorithm by employing community-aware initialization.

3.5 New Algorithms

In this section we propose two new algorithms SCDG and CASG which aims to improve total welfare over the algorithms discussed in the previous section. Also, we propose the personality-oriented utility function at the end of chapter.

3.5.1 Second Chance Dynamic Greedy Algorithm

As discussed in the previous section, the PCADG algorithm has some problems that we address in our Second Chance Dynamic Greedy (SCDG) algorithm, a variation to PCADG. One of the major issues with PCADG is the false promise problem, explained in Section 3.4. SCDG's approach to this problem is to remove all the temporarily assigned users from an event whenever one of them is permanently assigned to *another* event. Then, all the user-event pairs consisting of these users and the event that they were eliminated from are put back to the list of possible assignments $\mathcal{L}$. This operation will give these users another chance to be assigned to the event that they have the highest interest in as well as the chance of being assigned to the same event as the users whom they are interested in.

Algorithm 13 SCDG$(U, A, \sigma, \omega, \alpha)$

$\mathcal{L}$ = Community_Aware_Initialization$(U, A, \sigma, \omega, \alpha)$
Sort $\mathcal{L}$ in non-increasing order of gain values
while $< u, a, g(u, a \mid S_a) > \leftarrow \mathcal{L}.pop()$ **do**
    if $M(u) = \perp$ *and* $|S_a| < \delta_a$ **then**
        SCDG_Assignment$(U, A, \sigma, \omega, \alpha, \mathcal{L})$
    end if
    SCDG_Update$(U, A, \sigma, \omega, \alpha)$
    if $\mathcal{L}.empty()$ **then**
        SCDG_Reinitialization$(U, A, \sigma, \omega, \alpha, \mathcal{L})$
    end if
end while
User_Substitute$(U, A, \sigma, \omega, \alpha, T, M, DP)$
Row_Switch$(U, A, \sigma, \omega, \alpha, T, M)$
return M

- **Initialization:** We demonstrate SCDG in Algorithm 13. In initialization phase (represented in *Community_Aware_Initilization*) the user-event pair list $\mathcal{L}$ is created and for each pair, we employ community aware initialization introduced in 11 to estimate the potential gain value. The initialization method takes the accumulated interest of a user's social circle into account, in order to increase the chance of having friends to be assigned to the same event.

Algorithm 14 Community_Aware_Initialization$(U, A, \sigma, \omega, \alpha)$

$M(u) \leftarrow \perp, \forall u \in U; N(u) \leftarrow 0, \forall u \in U$
$S_a \leftarrow \emptyset, \forall a \in A; P \leftarrow \emptyset$
$deficit(a) \leftarrow 0, \forall a \in A; V \leftarrow U$
for `each` $(u, a) \in U \times A$ *s.t.* $\sigma_{u,a} > 0$ **do**
    $g(u, a \mid S_a) \leftarrow (1 - \alpha)\sigma_{u,a} +$
    $\mathcal{L}.insert(< u, a, g(u, a \mid S_a) >)$
end for

- **Assignment:** We execute the assignment process by iterating through the list $\mathcal{L}$, following the same assignment process as described in PCADG algorithm (represented in Algorithm 15). Once a user is permanently assigned to an event, namely event e, it should be removed from all other events that it has applied for. We call this process *Commitment*. Here is the point at which we remove the users who have applied for the event e from the assignment list S_e and subsequently we re-add their corresponding user-event pairs (i.e. user-event pairs that we just removed from S_e) to list $\mathcal{L}$. We use list $\mathcal{H}$ to store all the affected events those which have lost users during the commitment process. Once an event becomes real, i.e. $\gamma_a < |S_a|$, all the users who applied for it are permanently assigned to the event right away and then commitment process is executed.

Algorithm 15 SCDG_Assignment$(U, A, \sigma, \omega, \alpha, \mathcal{L})$

```
1: if |S_a| = 0 then
2:     if deficit + γ_a ≤ |V| then
3:         deficit ← deficit + γ_a − 1
4:         P ← P ∪ {a}
5:     else
6:         Return
7:     end if
8: else
9:     if a ∈ P then
10:        deficit ← deficit − 1
11:    end if
12: end if
13: S_a ← S_a ∪ {u}
14: N(u) ← N(u) + 1
15: V ← V\{u}
16: if |S_a| > γ_a then
17:    M(u) ← a
18:    for all a' ∈ A s.t. a' ≠ a ∧ u ∈ S_a' do
19:        S_a' ← S_a' − {u}
20:        N(u) ← N(u) − 1
21:        H.insert(a')
22:    end for
23: end if
24: if |S_a| = γ_a then
25:    P ← P\{a}
26:    for all v ∈ S_a do
27:        M(v) ← a
28:        for all a' ∈ A s.t. a' ≠ a ∧ v ∈ S_a' do
29:            S_a' ← S_a' − {v}
30:            N(v) ← N(v) − 1
31:            H.insert(a')
32:        end for
33:    end for
34: end if
```

- **Update:** When the assignment process for a user-event pair is over we will

update the affected pairs in list assignments $\mathcal{L}$ by calling Algorithm 16. The update process consists of removing users from the temporary assignments made to events that were involved in the commitment process. Once the users are removed from assignment lists, they are stored in temporarily list $\mathcal{H}'$ so that we can put them back to $\mathcal{L}$ later on. This strategy will give users another chance to be assigned to the event that they are interested in the most. As there will be no user assigned to the affected event, we will remove it from the phantom events list and the $deficit$ value is decreased by the number of users required to make the affected event real.

Algorithm 16 SCDG_Update$(U, A, \sigma, \omega, \alpha)$

```
1: H' ← ∅
2: n ← |S_a|
3: while a' ← H.pop() do
4:     for all u' ∈ S_a' do
5:         H'.insert(< u', a' >)
6:         S_a' ← S_a'/{u'}
7:         N(u') ← N(u') − 1
8:         if N(u') = 0 then
9:             V ← V ∪ {u'}
10:        end if
11:    end for
12:    P ← P\{a'}
13:    deficit ← deficit − γ_a' − n
14: end while
15: while < u', a' >← H'.pop() do
16:    L.insert(< u', a', g(u', a' | S_a') >)
17:    for all v : (w(u', v) > 0 ∧ M(v) = ⊥) do
18:        L.update(< v, a', g(v, a' | S_a') >)
19:    end for
20: end while
21: for all v : (w(u, v) > 0 ∧ M(v) = ⊥ ∧ v ∉ S_a ∧ |S_a| < δ_a) do
22:    L.update_or_add(< v, a, g(v, a | S_a) >)
23: end for
```

Subsequent to undoing the side effects of assigning users to events in $\mathcal{H}$, we make pairs out of de-assigned users and the events that they are taken from. We put back these pairs in the list $\mathcal{L}$ and then we update the interest of people in their social circle (who have social interest in them) so that they know the assignments made to events in $\mathcal{H}$ has been repealed. At the end we execute *update_or_add* on the pairs containing friends of the user who has been assigned to an event during the assignment process. This subroutine will adjust lookahead values

according to the recent assignment as there might be an increase of interest toward an event because of the people who are already assigned to it.

- **Reinitialization:** When list $\mathcal{L}$ is empty, we execute the reinitialization subroutine (discussed in Section 3.2) to make the algorithm continue running as long as there are possible assignments to be made. During this process the users who has been assigned to phantom events and the open events are recycled into the list $\mathcal{L}$. We have tweaked this process to include a subset of phantom events in addition to open events to increase the profit gain out of the next iterate of the assignment-reinitialization cycle.

- **Post-Processing:** Once the assignment phase is over, user-event pairs which have been ignored during the assignment process (due to cardinality constraints) are processed through $User\_Substitute$ procedure as demonstrated in Algorithm 17. This procedure will replace the available users with assigned users if the substitution leads to a higher social welfare gain.

  For each user-event pair, namely $<u, a>$, such that a is real and u has not been assigned to any event, we temporarily replace the users assigned to a with u (line 5) to check if the social welfare gained by a would increase. By iterating through the users in S_a, we find the best substitute (stored in $ChosenUser$) which would give us the highest social welfare increase. At the end, in line 13,

we will permanently replace $ChosenUser$ with u.

Algorithm 17 User_Substitute$(U, A, \sigma, \omega, \alpha, T, M, DP)$

for each $<u, a> \in DP$ *s.t.* $(M(u) = \perp$ and $\gamma_a \leq |S_{M(u)}| \leq \delta_a)$ **do**
  $ChosenUser \leftarrow u$
  $GreatestUtility \leftarrow \mu(S_a, a)$
  for each $v \in S_a$ **do**
    $S_a \leftarrow S_a \backslash \{v\} \cup \{u\}$
    if $GreatestUtility < \mu(S_a, a)$ **then**
      $GreatestUtility \leftarrow \mu(S_a, a)$
      $ChosenUser \leftarrow v$
    end if
    $S_a \leftarrow S_a \backslash \{u\} \cup \{v\}$
  end for
  if $ChosenUser \neq u$ **then**
    $S_a \leftarrow S_a \backslash \{ChosenUser\} \cup \{u\}$
    $M(ChosenUser) \leftarrow \perp$
    $M(u) \leftarrow a$
  end if
end for
return M

At the end of algorithm, we perform the event switch process (introduced in Section 3.2). Assume M is the global assignment made on (U, A) and M' : $M(u_1) \rightleftarrows M(u_2)$, then $M \leftarrow M'$ if $\omega(M) < \omega(M')$; meaning that the users do an event swap if it increases the social welfare. Assuming S_{a_1} and S_{a_2} are the sets that users u_1 and u_2 are being assigned to, and S'_{a_1} and S'_{a_2} are the sets after swapping users' events respectively, then we can calculate the social welfare difference resulting from switching events between users u_1 and u_2 as:

$$\mathcal{D}(u_1, u_2) = (\mu(S'_{a_1}, a_2) + \mu(S'_{a_2}, a_1)) - (\mu(S_{a_1}, a_1) + \mu(S_{a_2}, a_2)) \tag{8}$$

By expanding each term in Equation 8 we have

$$\begin{aligned}\mathcal{D}(u_1,u_2) &= (1-\alpha)((\sigma_{u_1,a_2}+\sigma_{u_2,a_1})-(\sigma_{u_1,a_1}+\sigma_{u_2,a_2}))\\ +\alpha[(&\sum_{u_1\in S_{a_2},u\neq u_1} w(u,u_1)+w(u_1,u)+\sum_{u_2\in S'_{a_1},u\neq u_2} w(u,u_2)+w(u_2),u)\\ -(&\sum_{u_1\in S_{a_1},u\neq u_1} w(u,u_1)+w(u_1,u)+\sum_{u_2\in S_{a_2},u\neq u_2} w(u,u_2)+w(u_2,u))]\end{aligned} \tag{9}$$

Once all the users try to swap their events with each other, it is possible that there are still users who are willing to switch their events with one another, therefore we run the swapping process as long as the social welfare growth passes a given threshold $T \quad s.t. \quad 0 < T \leq 1$. After realizing the assignments M (before a swapping round) and M' (after a swapping round) we try the test $(1-\omega(M)/\omega(M') > T)$ to decide whether to go for another swapping round or to terminate the algorithm. The SCDG algorithm executes the event switch process only once the assignments are decided, thus there will be no need to keep list $\mathcal{L}$ and other auxiliary variables, this will give us the ability to parallelize the swapping process which we discuss by introducing *Row_Switch* procedure.

It is trivial to see that for a pair of events, namely a_1, a_2, this process should be done sequentially, as each user's decision may affect the decision of others'. For example, when an event switch between user u_1 from a_1 and u_2 from a_2 is made, the remaining users' interest in e_1 and e_2 will be updated according to

their social interest in u_1 and u_2 but during the execution of swapping process for the users assigned to a_1, a_2, the swap process for the rest of users can be done in parallel.

Furthermore, as it can be observed in Equation 8, the only parameters involved in an event switch decision are the users who are assigned to the switching events. Thus, a set of event switches consisting of distinct event pairs can be decided concurrently. In order to parallelize the event switch process, we simply partition the set of events A into pairs, and run Row_Switch (demonstrated in 18) in parallel for every pair of events.

Algorithm 18 Row_Switch$(U, A, \sigma, \omega, \alpha, T, M)$

$K \leftarrow \omega(M)$
do
  for each $(a_1, a_2) \in A \times A$ $s.t.$ $\gamma_{a_1} \leq |a_1| \leq \delta_{a_1}$ **and** $\gamma_{a_2} \leq |a_2| \leq \delta_{a_2}$ **do**
    for each (u, v)
      $s.t.$ $u \in S_{a_1}$ **and** $v \in S_{a_2}$ **and** $M(u) \neq \perp$ **and** $M(v) \neq \perp$ **do**
      if $\mathcal{D}(u_1, u_2) > 0$ **then**
        $S_{a_1} \leftarrow S_{a_1} \backslash \{u\} \cup \{v\}$
        $S_{a_2} \leftarrow S_{a_2} \backslash \{v\} \cup \{u\}$
        $M(u) \leftarrow a_2$
        $M(v) \leftarrow a_1$
      end if
    end for
  end for
  $K' \leftarrow \omega(M)$
while $1 - K/K' > T$ ▷ T: A threshold value set between 0 and 1.
return M

- **Time Complexity**: We provide a rough estimation of a worst case scenario

to explain the time complexity of SCDG algorithm, which is more complicated than PCADG algorithm complexity due to its more convoluted update operation presented in Algorithm 16.

Let $\mathcal{I}_u$ denote the set of phantom events from which user u, who is the first user who meets the minimum cardinality bound, is removed, being permanently assigned. During the assignment process, every user can be assigned to at most $|A|$ events, thus $|\mathcal{I}_u| \leq |A| - 1$. When u is removed from events $\in \mathcal{I}_u$, all other users who are already assigned to events $\in \mathcal{I}_u$ will be removed and reinserted as user-event pairs to the list $\mathcal{L}$. Let Γ_A denote the highest minimum cardinality of events in A, then the upper bound of number of these pairs is $\Gamma_A(|A| - 1)$. For every permanent assignment that takes place, the size of $\mathcal{I}_u$ decreases by one. For the event set A, we define Φ_A to be the summation of the number of these reinserted user-event pairs as follows:

$$\Phi_A = \sum_{x=1}^{|A|-1} x\Gamma_A = \Gamma_A(|A| - 1)|A|/2 = O(\Gamma_A|A|^2) \tag{10}$$

Therefore, the number of pop operations that can take place during the assignment process is $O(|\mathcal{L}| + \Phi_A) = O(|\mathcal{L}| + \Gamma_A|A|^2)$. As every pop operation takes $O(|\mathcal{L}|)$ (finding the pair with maximum gain from $\mathcal{L}$), one round of the assignment process takes $O(|\mathcal{L}|^2 + \Gamma_A|A|^2)$. In post-processing, one round of

Row-Switch performs $O(|U|^2)$ swap operations in the worst case. In the swap operation, we use Equation 9 which calculates a summation over the users assigned to both events involved. Let Δ_A denote the highest maximum cardinality of events in A, then the switching process takes $O(|U|^2\Delta_A)$ time in the worst case scenario.

Finally, each User_Substitute round performs at most $O(|U||A|)$ swap operations. Also, each User_Substitute swap operation tries every user that has already assigned to the event, thus a User_Substitute round takes $O(|U||A|\Delta_A)$. The number of Row-Switch and User-substitute rounds can be specified by the algorithm. In our implementation, we chose to make the number of Row_Switch rounds and User_Substitute rounds depend on the parameter T, which specifies a minimum improvement in social welfare. We denote the number of rounds of Row_Switch or User_Substitute for threshold T by $\mathcal{R}_\mathcal{T}$ and therefore, the total execution time of post-processing is $O(\mathcal{R}_\mathcal{T}\Delta_A(|U|^2 + |U||A|))$. In our experiments, the number of event-switch rounds R_T was generally 1 or 2.

Therefore, SCDG time complexity is $O(|\mathcal{L}|^2+\Gamma_A|A|^2|\mathcal{L}|+\mathcal{R}_\mathcal{T}\Delta_A(|U|^2+|U||A|))$. It should be noted that by taking the reinitialization phase into account, the assignment process may be executed $|U|$ times in the worst case, as in each round, at least one user should be assigned to an event or the assignment-reinitialization cycle is terminated. By multiplying the time complexity of the

assignment phase by $|U|$, we have $O(|\mathcal{L}|^2|U| + \Gamma_A|A|^2|\mathcal{L}||U| + \mathcal{R}_\mathcal{T}\Delta_A(|U|^2 + |U||A|))$.

3.5.2 Community-Aware Static Greedy

In this section we propose a minimal and effective solution to the SEO problem that we call the Community-Aware Static Greedy algorithm. CASG is composed of a simple assignment process in which no update operation on user-event potential gain is performed, similar to SPG (represented in Algorithm 19), except that it considers a single user-event pair in each iteration, instead of assigning pair of users to an event at each iteration of assignment process. Also, similar to SPG the list $\mathcal{L}$ is ordered only once in initialization phase (demonstrated in 20).

Algorithm 19 CASG$(U, A, \sigma, \omega, \alpha)$

```
1: L = SCDG_Initialization(U, A, σ, ω, α)
2: while < u, a, g(u, a | S_a) > ← L.pop() do
3:     if M(u) = ⊥ and |S_a| < δ_a then
4:         CASG_Assignment(U, A, σ, ω, α, L)
5:     end if
6:     if L.empty() then
7:         CASG_Reinitialization(U, A, σ, ω, α, L)
8:     end if
9: end while
10: User_Substitute(U, A, σ, ω, α, T, M, DP)
11: Event_Switch(U, A, σ, ω, α, T, M)
12: return M
```

- **Initialization:** While SPG employs the utility function provided in Equation

6, in CASG the list $\mathcal{L}$ is ordered in in non-increasing order by the community aware potential gain which is define as follows:

$$g_{CA}(u,a) = (1-\alpha)(\sigma_{u,a}) + \alpha\delta_a \sum_{v \in U} w(u,v)/(|U|-1) \tag{11}$$

Algorithm 20 CASG_Initialization$(U, A, \sigma, \omega, \alpha)$

1: $M(u) \leftarrow \perp, \forall u \in U; S_a \leftarrow \emptyset, \forall a \in A$
2: **for** `each` $(u,a) \in U \times A$ **do**
3: $\quad \mathcal{L}.insert(< u, a, g_{CA}(u,a) >)$
4: **end for**
5: Sort list $\mathcal{L}$ by gain values in non-increasing order

- **Assignment:** In assignment phase (demonstrated in 11), the pairs of user-event $<u,a>$ are processed in the non-increasing order of g and if $|S_a| \leq \delta_a$ the user u will be permanently assigned to a, i.e. $M(u) = a$. Also, for every pair that violates the event cardinality constraint, the pair is inserted into list DP. This list which will be processed at the end of algorithm by calling $User\_Substitute$.

Algorithm 21 CASG_Assignment$(U, A, \sigma, \omega, \alpha, \mathcal{L})$

```
1: if M(u) = ⊥ and |S_a| < δ_a then
2:     S_a ← S_a ∪ {u}
3:     if |S_a| > γ_a then
4:         M(u) ← a
5:         for all a' ∈ A s.t. a' ≠ a ∧ u ∈ S_a' do
6:             S_a' ← S_a' − {u}
7:         end for
8:     end if
9:     if |S_a| = γ_a then
10:        P ← P\{a}
11:        for all v ∈ S_a do
12:            M(u) ← a
13:            for all a' ∈ A s.t. a' ≠ a ∧ v ∈ S_a' do
14:                S_a' ← S_a' − {v}
15:            end for
16:        end for
17:    end if
18: else
19:    DP.insert(< u, a >)
20: end if
21: p ← p + 1
22: if |L| = 0 then
23:    for each (u, a) ∈ RS(U, A, p, M|S) do
24:        L.insert(< u, a >)
25:    end for
26: end if
```

- **Update:** No update operation is performed in this algorithm.
- **Reinitialization:** When the $\mathcal{L}$ is empty the reinitialization phase is executed as discussed in Section 3.2 which may lead to another Assignment-Reinitialization cycle depending on availability of users and open events.
- **Post-Processing:** Once the assignment-reinitialization cycle is terminated we

execute the post-processing phase, similar to SCDG, by calling the User_Substitution (Algorithm 18) and Row_Switch (Algorithm 17) procedures.

Time Complexity: One round of the initialization and assignment phases is simply a sort and a traversal of the list $\mathcal{L}$, which takes $|\mathcal{L}|log(|\mathcal{L}| + |\mathcal{L}|$ as each pop operation takes $O(1)$. Once the assignment process is over we execute the post-processing procedures which takes $O(\mathcal{R}_{\mathcal{T}}\Delta_A(|U|^2 + |U||A|))$ as discussed in Section 3.5.1, and therefore, the total execution time of algorithm is $O(|\mathcal{L}|log(|\mathcal{L}| + |\mathcal{L}| + \mathcal{R}_{\mathcal{T}}\Delta_A(|U|^2 + |U||A|))$. Moreover, by taking the reinitialization phase into account, the assignment process may be executed $|U|$ times in the worst case, as in each round, at least one user should be assigned to an event or the assignment-reinitialization cycle is terminated. Therefore, the whole algorithm takes $O(|\mathcal{L}|log(|\mathcal{L}||U| + |\mathcal{L}||U| + \mathcal{R}_{\mathcal{T}}\Delta_A(|U|^2 + |U||A|))$.

3.5.3 Personality-Oriented Social Event Organization

Authors Li et al. [24] propose an abstract and a generalized way of evaluating the user-event assignment quality, in which calculation of parameters of social and affinity interests are left out of the discussion. Variations of objective functions for the SEO problem have been defined by others, such as [34], where the willingness of going to an event has been regarded as a function of distance between user and event, the number of shared properties among them and also, the event expenses as suggested

by [30].

In this section we consider how the personality of a user may affect his or her interest in attending an event by introducing a new objective function which focuses on a different aspect of a live experience, that is the effect of extroversion-introversion traits of users on their interest in participating social events. Extroversion is characterized by demanding sociability, gatherings and many friends, where introversion is identified as being reflective, reserved and showing solitary behavior [26]. Moreover, the ambiversion personality falls in the middle of extroversion-introversion continuum [9].

We employ an extroversion index to tune the impact of presence of users friends on their interest in an event. We assume that for every user in U the extroversion index, denoted by $\mathcal{E}(u)$ such that $0 \leq \mathcal{E}(u) \leq 1$, is provided as a normalized input to the problem. The lower values of $\mathcal{E}(u)$ indicate that user u interest in attending an event is highly influenced by other attendees, rather than his/her sheer interest in that event.

In order to formulate the effect of $\mathcal{E}$ on the gain of assigning two arbitrary users u and v to event a, denoted by $f(u, v, a)$, we provide Table 1 which consists of their extroversion indices, their innate interest in a and their social interest in each other. The set of possible inputs for each parameter is denoted by values of VL, L, ML, M, MH and H (standing for very low, low, medium low, medium, medium high and high

respectively), indicating a summarized range of values. Finally, $F(u, v, a)$ column represents the gain of assigning user u to event a.

Table 1: Personality-oriented gain function12

	$\mathcal{E}(u)$	$\mathcal{E}(v)$	$\sigma_{u,a}$	$\sigma_{v,a}$	$w(u, v)$	$f(u, v, a)$
1	H	H	L	L	L	L
2	H	H	L	L	H	L
3	H	H	L	H	L	M
4	H	H	L	H	H	M
5	H	H	H	H	L	H
6	H	H	H	H	H	H
7	L	L	L	L	L	VL
8	L	L	L	L	H	L
9	L	L	L	H	L	ML
10	L	L	L	H	H	M
11	L	L	H	H	L	MH
12	L	L	H	H	H	H
13	L	H	L	L	L	VL
14	L	H	L	L	H	L
15	L	H	H	L	L	ML
16	L	H	H	L	H	M
17	L	H	L	H	L	ML
18	L	H	L	H	H	M
19	L	H	H	H	L	MH
20	L	H	H	H	H	H

Table 1 consists of three sets of rows and each set corresponds to one possible combination for $\mathcal{E}(u)$ and $\mathcal{E}(v)$ values as follows:

- **Both Extroverted:** The first set consists of the case where both of the users are highly extroverted (lines 1 to 6). In this case, the gain of assigning users u and v to event a, denoted by the function $f(u, v, a)$, is only affected by their

innate interests in event a. This is justified by the fact that extroverted people are more interested in meeting new people and expanding their social circle [26], so their interest in other participants does not impact their willingness to participate in an event. In the lines 1 and 2, both users have low interest in event a and increasing their social interest in one another in line 2 does not have any impact on the low value of $f(u, v, a)$. On the other hand, in the lines 2 and 3, value of $f(u, v, a)$ increases from low to medium as the innate interest of user v in event a ($\sigma_{v,a}$) increases from low to high. Moreover, the values of function $f(u, v, a)$ in the lines 3 and 4 are both medium, regardless of the values of the social interest $w(u, v)$. The same logic applies to the lines 5 and 6. Finally, the utility function increases from M to H from line 4 to line 5 (and line 6) as both of the innate interests $\sigma_{u,a}$ and $\sigma_{v,a}$ are high.

- **Both Introverted:** For the case where both users are highly introverted, that is, $\mathcal{E}(u) = L$ and $\mathcal{E}(v) = L$ (line 7 to 12), the utility gain of assigning users u and v to event a is affected by *both* their social interest in one another and their innate interests in the event a. Just as in the case where users u and v are extroverted, the innate interest of the two users determines whether the value of $f(u, v, a)$ is in the low, medium, or high *category* that is, if both have low innate interest, then $f(u, v, a)$ is in the low category, if one user has high innate interest and the other has low innate interest then $f(u, v, a)$ in the medium

category, and if both have high innate interest, then $f(u, v, a)$ is in the high category. However, unlike the case when u and v are extroverted, here, if u and v don't know each other (i.e $w(u, v) = L$) the value of $f(u, v, a)$ is downgraded somewhat, while staying in the same category that is, if both have low innate interest, then $f(u, v, a)$ is low if $w(u, v)$ is high (u and v know each other very well) and it is very low if $w(u, v)$ is low (u and v do not know each other). Also, if one user has high innate interest and the other has low innate interest then $f(u, v, a)$ is medium if $w(u, v)$ is high and it is medium low if $w(u, v)$ is low. Finally, if both users have high innate interest, then $f(u, v, a)$ is high if $w(u, v)$ is high and it is medium high if $w(u, v)$ is low.

- **Both Introverted:** Finally the third set consists of pairs where one user is highly introverted and the other one is highly extroverted (lines 13 to 20). We consider this case to be the same as the case where both users are introverted, as the introverted user (user u in our example) is affected by both the innate interest and social interest, and therefore, it will affect the overall gain of assigning users u and v to event a.

3.5.3.1 Personality-Oriented Utility Function

We define the function $f(u, v, a)$ such that it respects all the cases provided in Table 1. Let $\mathcal{Q}$ denote the maximum interest value between users in a graph $G = (U, E, w)$,

i.e. $\mathcal{Q} = Max(w(u,v))$. Also, let $\beta_{u,v}$ denote the lower extroversion index between users u and v, i.e. $\beta_{u,v} = Min(\mathcal{E}(u), \mathcal{E}(v))$. First, we define the pairwise interest of users in events by introducing the function $f(u,v,a)$, by which the gain of having users u and v assigned to the event a is defined as follows:

$$\begin{aligned} f(u,v,a) &= (1-\beta_{u,v})(\alpha(\sigma_{u,a}+\sigma_{v,a})+w(u,v)) + \beta_{u,v}(\alpha(\sigma_{u,a}+\sigma_{v,a})+\mathcal{Q}) \\ &= \alpha(\sigma_{u,a}+\sigma_{v,a}) + (1-\beta_{u,v})w(u,v) + \beta_{u,v}\mathcal{Q} \end{aligned} \tag{12}$$

We use a coefficient α to balance the total interest of u and v in the event a over their social interest in each other, as the former term is a summation of two values and the latter is only one value. Now we define $\mu(S,a)$ as the utility gain of assigning the set of users S to the event a as follows:

$$\sigma(S,a) = \sum_{u,v \in S} f(u,v,a) + \sum_{u \in S} (\mathcal{E}(u) - 0.5)|S| \tag{13}$$

The first term of Equation 13 is a summation over pairwise gain of users attending event a and the second term is the per user effect of number of people assigned to event a which can be described in three cases:

- $\mathcal{E}(u) > 0.5$: This case indicates that user u is extrovert and thus the number of attendees has a positive impact on user u. The positive value of $\mathcal{E}(u) - 0.5$ states this positive effect.

- $\mathcal{E}(u) = 0$: In this case user u is ambiverted (a person with both extrovert and introvert tendencies) and therefore, neutral to the number of attendees.

- $\mathcal{E}(u) < 0.5$: For introvert users the number of attendees has a negative impact on the live experience. The negative value of $\mathcal{E}(u) - 0.5$ implicitly states that the assignment S is not interesting to an introvert person.

3.5.4 Personality-Oriented SG

We introduce POPADG by incorporating the personality-oriented objective function (discussed in Section 3.5.3.1) in the PADG algorithm (discussed in Section 3.3.3). The algorithms follows the same steps as in Algorithm 11 while employing the personality-oriented initialization function which is defined as follows:

$$\begin{aligned} g_{po}(u,a) &= f(u,a) + (\mathcal{E}(u) - 0.5) \\ &= \alpha(\sigma_{u,a}) + (\mathcal{E}(u) - 0.5) \end{aligned} \tag{14}$$

The $g_{po}(u,a)$ denotes that gain of assigning user u to event a. We utilize this function in the initialization phase as represented in Algorithm 22.

Algorithm 22 POPADG_Initialization$(U, A, \sigma, \omega, \alpha)$

1: $M(u) \leftarrow \perp, \forall u \in U; S_a \leftarrow \emptyset, \forall a \in A$
2: **for** `each` $(u,a) \in U \times A$ **do**
3: $\quad \mathcal{L}.insert(< u, a, g_{PO}(u,a) >)$
4: **end for**

3.5.5 Personality-Oriented SG

We introduce POSG by employing CASG algorithm (discussed in Section 3.5.2) and the personality-oriented objective function (discussed in Section 3.5.3.1). Similar to POPADG, we employ Equation 14 in initialization phase, in order to calculate the gain values. Once the assignment process is over we run the post process by calling User_Substitution and Row_Switch procedures to increase the overall social welfare gain calculated by the personality-oriented gain function (Equation 13).

3.5.6 Personality-Oriented SCDG

We propose POSCDG by adapting SCDG algorithm (introduced in 3.5.1) for the personality-oriented objective function (discussed in 3.5.3). Similar to POCADG, we use Equation 14 to initialize the list of user-event pairs $\mathcal{L}$. Also, during the update process of list $\mathcal{L}$ we use Equation 13 to calculate the gain of assigning a user to an event during the update process. It should be noted that for a pair user-event $<u, a>$, this utility function considers the gain of assigning the set $S_a \cup u$ to a, however the utility function employed in SCDG (discussed in 4) only considers the additional gain of assigning u to a.

Finally, in post process phase we employ Equation 13 for both User_Substitution and Row_Switch procedures to perform the process of switching events between users.

Chapter 4

Experiments

In this chapter we provide the results of experiments that we have conducted on the SEO algorithms CASG (Section 3.5.2), PADG (Section 3.3.3), PCADG (Section 3.3.4) and SCDG (Section 3.5.1) using both synthetic and real datasets and compare their performance.

4.1 Preliminaries

To evaluate the SEO algorithms three metrics are discussed in the following:

Total Social Welfare: In the context of the SEO problem *total social welfare* (proposed by [24]) is a measure of overall satisfaction acquired through assigning people to the events, considering their innate and social interests. We employ

the function $\omega(M)$ given in Equation 15 to measure the quality of a feasible solution M for a SEO problem instance.

$$\omega(M) = \sum_{a \in A}[(1-\alpha) \sum_{u \in M^{-1}(a)} \sigma_{u,a} + \alpha \sum_{u,v \in M^{-1}(a), u \neq v} w(u,v)] \tag{15}$$

The function $\omega(M)$ is basically the summation over the social welfare of real events in A (events that have received minimum number of users, i.e. $\gamma_a \leq |S_a| \leq \delta_a$). As innate affinity (interest of users in events) and social affinities (interest of users to one another) values are non-negative, the value of $\omega(M)$ is non-negative. The studied and proposed SEO algorithms in this thesis mainly aim to maximize this metric.

Total Personality-Oriented Social Welfare: In order to evaluate our personality-oriented algorithms (discussed in Section 3.5.3) we use the following equation to measure the quality of produced assignments:

$$\omega_{PO}(M) = \sum_{a \in A}(\sum_{u,v \in M^{-1}(a)} f(u,v,a) + \sum_{u \in M^{-1}(a)} (\mathcal{E}(u) - 0.5)|M^{-1}(a)|) \tag{16}$$

It should be noted that Equation 16 is simply a summation of utility gain (Equation 13) over all of the events in A.

Let $\mathcal{Q}$ denote the maximum interest value between users in a graph G =

(U, E, w), i.e. $\mathcal{Q} = Max(w(u, v))$ and let $\beta_{u,v}$ denote the lower extroversion index between users u and v, i.e. $\beta_{u,v} = Min(\mathcal{E}(u), \mathcal{E}(v))$. The function $f(u, v, a)$ (discussed in Section 3.5.3.1) in Equation 16 is defined as follows:

$$\begin{aligned} f(u, v, a) &= (1 - \beta)(\alpha(\sigma_{u,a} + \sigma_{v,a}) + w(u, v)) + \beta(\alpha(\sigma_{u,a} + \sigma_{v,a}) + \mathcal{Q}) \\ &= \alpha(\sigma_{u,a} + \sigma_{v,a}) + (1 - \beta)w(u, v) + \beta\mathcal{Q} \end{aligned} \tag{17}$$

Regret Ratio Another metric aimed to provide a way to evaluate the goodness of a solution to SEO problem (introduced by [24]) is so-called *regret ratio*. *Regret ratio:* is the ratio of actual social welfare that a user has gained to the maximum social welfare that he could have gained. For a user u, the regret ratio function $\rho(u)$ is defined as follows:

$$\rho(u) = 1 - \frac{(1 - \alpha)\sigma_{u,M(u)} + \alpha \sum_{v \in S_{M(u)}} w(u, v)}{max_{a \in A}((1 - \alpha)\sigma_{u,a} + \alpha \sum_{v \in B_u} w(u, v))} \tag{18}$$

The numerator in the equation above is the social welfare gained by user u in the gain assignment M. Assume N_u to be set of users who has interest in u, i.e. $N_u = \{v \in V | w(u, v) > 0\}$. The denominator is the maximum possible social welfare That u could have gained in any assignment. To compute this, we find the maximum, over all events a, the welfare that u could obtain if u were assigned to a while having $\delta_a - 1$ of his best friends. We sort the users

of N_u in descending order of $w(u, v)$, and define B_u to be $min\{|N_u|, \delta_a - 1\}$ elements of N_u.

Note that the regret ratio denominator provides a rough upper bound on the social welfare which is almost unreachable for a user due to violating the one to one constraint of the assignment function M (users cannot be assigned to multiple events at once). The regret ratio value is expected to be between 0 and 1. While smaller values are desirable, for an unassigned user *regret ratio* would be 1. None of the algorithms in this thesis is targeted to minimize the *regret ratio* for individual users, instead the main focus is on maximizing the total social welfare; the overall happiness among all users.

Execution Time For all of our conducted experiments we have measured the execution time of discussed algorithms on a machine with Intel Core i7-2.10GHz CPU and 8 GB of RAM. It should be noted that we have only recorded the assignment process execution time to evaluate the performance of algorithms. Therefore, dataset generation and social welfare calculation execution time is excluded in the provided results.

4.2 Experiments on Synthetic Data

In this section we provide results of experiments that employ synthetic datasets. We use the same parameters as in the experiments on synthetic datasets in [24] which we explain briefly below:

Number of Users $|U|$: The number of users is set to 500 unless otherwise is specified. In Section 4.2.4 we vary the number of users from 500 to 2000.

Number of Events $|A|$: We have set the number of events $|A|$ to 10, 25, 35 and 50.

Social Network Graph Model G:

The SEO problem incorporates a network of users in which the connection between users is given a numerical weight and this social network is modeled as a graph $G = (U, E)$. There are many models in the literature to generate random social network graphs; see for example, [11, 4, 3, 17, 16, 5, 27, 18, 20, 35].

In our experiments, we use the model given in [3] to generate a random power-law graph on the set of users U, attributed with a power-law exponent of 1.5. We used [2] to generate the graphs. Then, for every pair of users namely u and v, we draw a floating-point number from a normal distribution with $\mu = 1.5$ and $\sigma^2 = 3$ to generate the weight $w(u, v)$ of the edge (social affinities) between

the user u and v. In Sections 4.2.2 and 4.2.3 we consider the effect of changing the social network parameters.

In addition to the abovementioned graph model, we conducted experiments using the Erdős –Rényi random graph model [11], Barabási-Albert model [4], and the HavelHakimi algorithm [17, 16] to generate social network graphs. The experiments are not included in this thesis due to the similarity of the results.

Bipartite Network: Another component of the SEO problem is the bipartite graph connecting users to events by weighted edges that indicates interest of users to events. We connect a user to an event with probability 0.05. Then, for each edge we assign the weight by sampling a floating-point number from a normal distribution with $\mu = 1.5$ and $\sigma^2 = 3$.

Coefficient α: As stated in the previous chapter, the objective functions in the SEO problem are a linear combination of innate (user-event) and social (user-user) interests which employ a coefficient to set the relative importance of the two types of affinities. In our experiments, generally we set α to 0.5 to put an equal emphasis on innate and social affinities.

Maximum Cardinality δ: Every event in A, namely a, is associated with a maximum cardinality which is denoted as δ_a and we draw this value from a normal distribution with $\mu = 20$ and $\sigma^2 = 10$.

Minimum Cardinality γ: As discussed in the previous chapter, all the events maintain a minimum cardinality constraint that is a non-negative integer number which states the required number of users for an event to be held (and to be regarded as real). In our experiments the minimum cardinality for event a (denoted by γ_a) is an integer drawn uniformly at random from the interval $[1, \delta_a]$.

4.2.1 Effect of Threshold Parameter

In the algorithms CADG and SCDG we introduced a threshold parameter T, a rational number between 0 and 1, as input to the *User_Substitute* and *Row_Switch* procedures which we use to stop the execution of the event switching process. In this experiment we investigate the effect of the parameter T on total social welfare gain, average regret ratio and execution time of our algorithms.

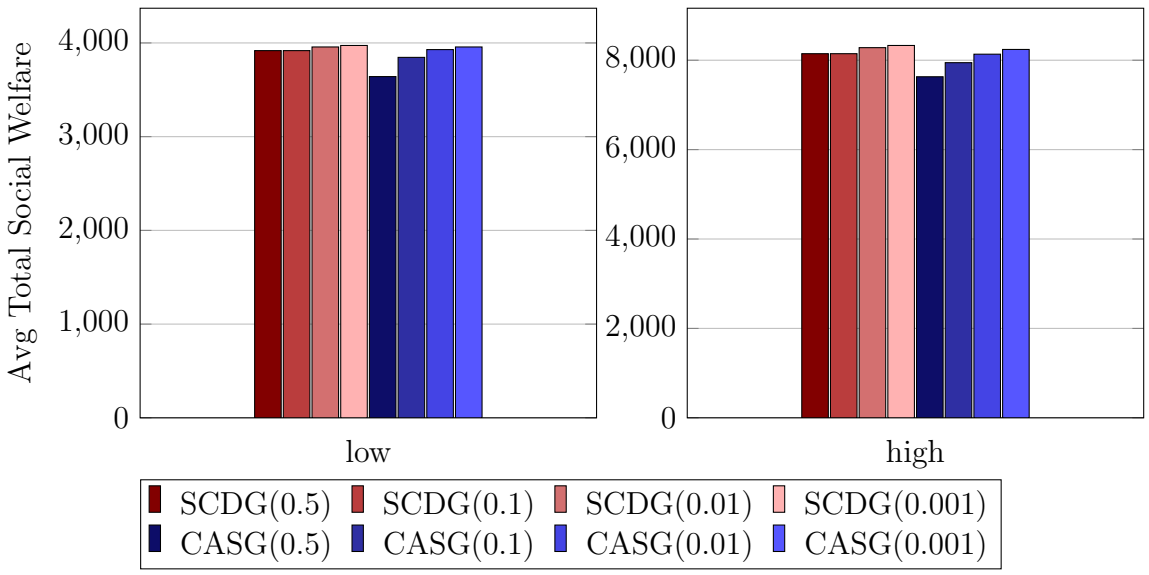
Table 2: Execution time for different T

	T	$d_{min} = 10$	$d_{min} = 100$
SCDG	0.5	00:19.8	00:41.4
SCDG	0.1	00:21.0	00:40.7
SCDG	0.01	00:20.7	00:43.6
SCDG	0.001	00:23.5	00:40.6
CASG	0.5	00:00.7	00:00.8
CASG	0.1	00:01.1	00:01.4
CASG	0.01	00:03.7	00:02.9
CASG	0.001	00:06.7	00:07.7

- **Overall Social Welfare:** Our results from this experiment show that CASG

(a) Avg total welfare for different T (low density)

(b) Avg total welfare for different T (high density)

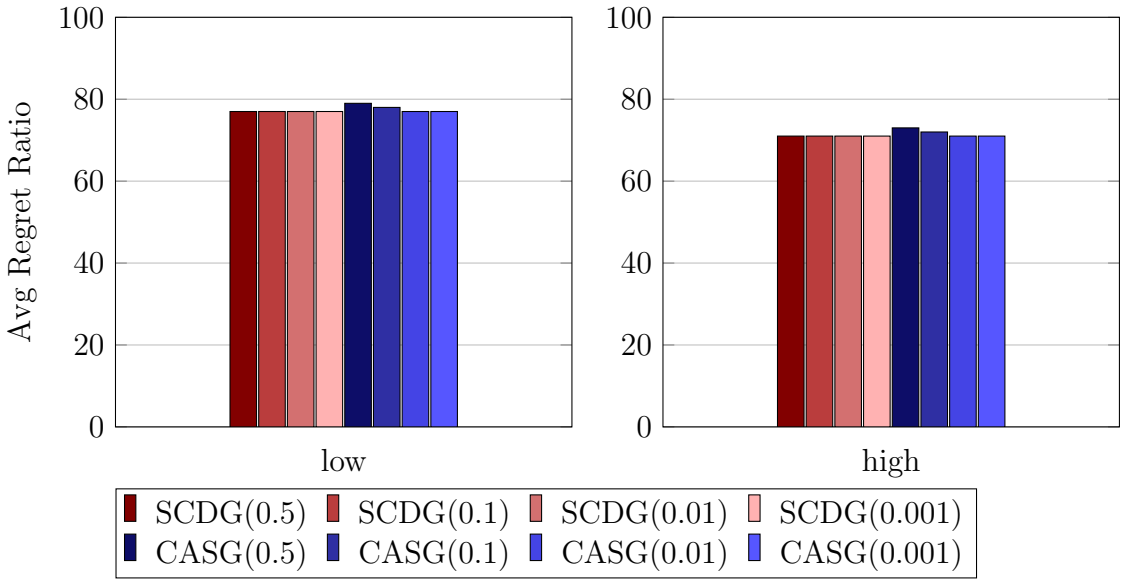


performance is more influenced by the parameter T. By reducing T from 0.5 to 0.001 we observe 8% improvement of total social welfare gain by CASG while less than 2% improvement is obtained by SCDG algorithm. Also, it should be noted that reducing T costs extra execution time, for both CASG and SCDG. As the execution time of CASG is doubled by reducing T from 0.01 to 0.001 while the social welfare improvement is negligible, therefore we have chosen $T = 0.01$ for both CASG and SCDG in the rest of our experiments.

- **Average Regret Ratio:** The average regret ratio is very similar for all versions of the algorithms and both low and high densities, except for CASG, the threshold 0.5 or 0.1 where the regret ratio is slightly higher.

(a) Avg regret ratio for different T (low density)

(b) Avg regret ratio for different T (high density)



- **Execution Time:** Our results (depicted in Table 2) show that varying the threshold does not make difference in the execution time of SCDG algorithm but the smaller values of T increases the execution time of CASG in a great deal. Also, it is notable that regardless of the value of T, the execution time of CASG is shorter than SCDG in all of the cases.

4.2.2 Effect of Power Law Minimum Degree

As discussed in the previous section, we use a power law model [3] to generate the social network graph $G = (U, E, w)$. The input parameters consist of the number of nodes, minimum degree and power law exponent. In this section we investigate the effect of varying the minimum degree on the metrics by which we

measure the quality of produced assignments including social welfare, regret ratio and execution time. We have considered 500 users, 50 events, exponent of 1.5 and minimum degrees of 1, 5, 10, 100 and 200. Figure 3 shows the effect of changing the minimum degree parameter on the number of edges. It can be seen that as the minimum degree increases, the density of the social network increases. We run the rest of our experiments for both low density graphs where $d_{min} = 10$ and high density graphs where $d_{min} = 100$.

Figure 3: Effect of minimum degree on number of edges

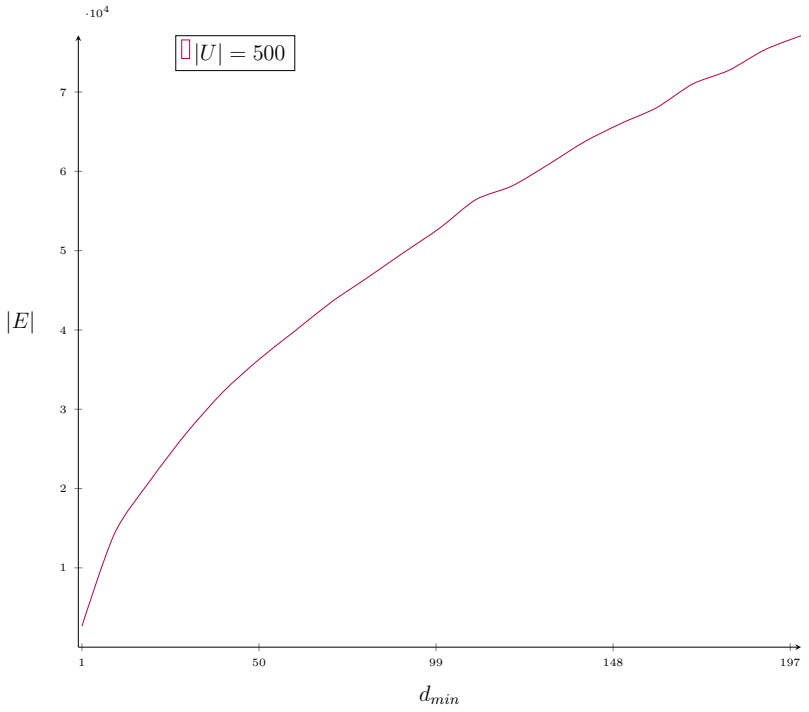


Figure 4: Avg total welfare for different power law minimum degrees

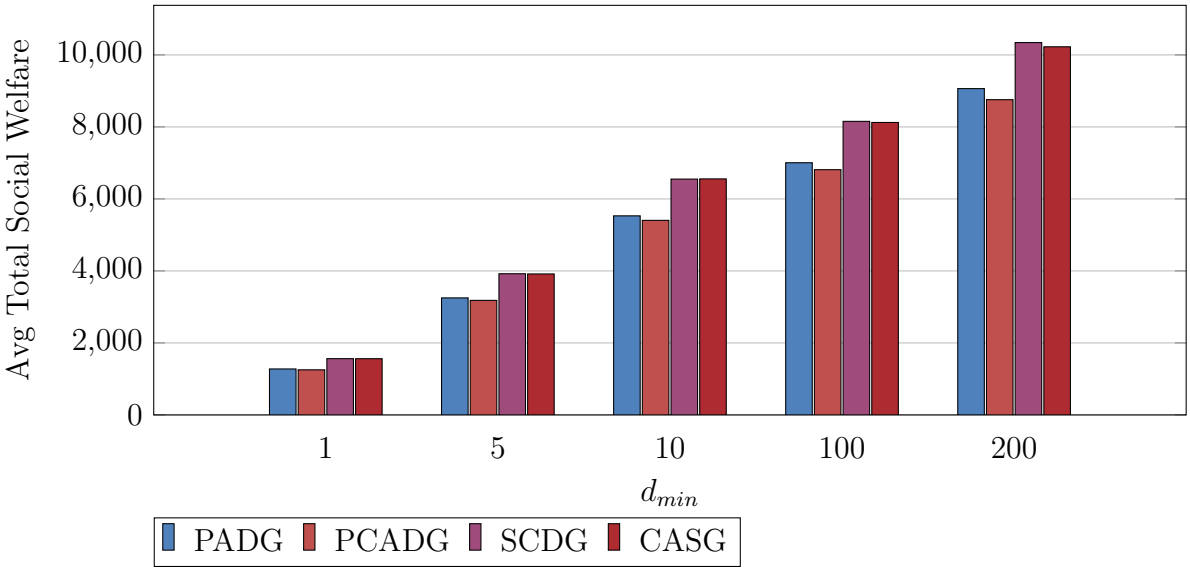


Figure 5: Avg regret ratio for different power law minimum degrees

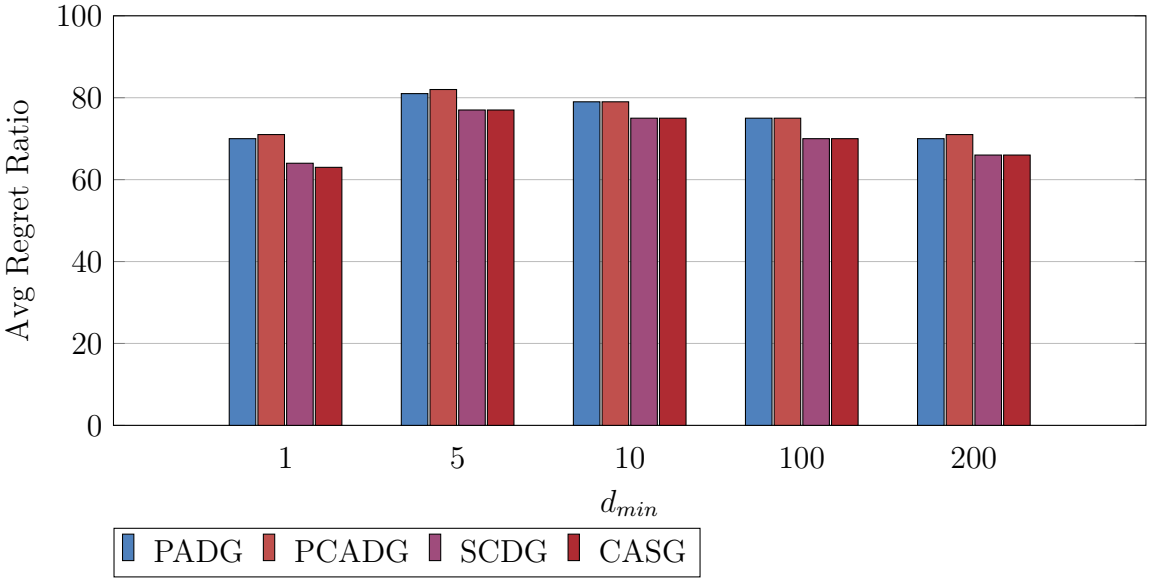


Table 3: Execution time for power different law minimum degrees

	1	5	10	100	200
PADG	00:00.5	00:01.5	00:01.8	00:01.2	00:01.8
PCADG	00:00.5	00:01.5	00:01.9	00:01.4	00:02.5
SCDG	00:22.0	00:31.2	00:37.1	00:29.5	00:43.9
CASG	00:02.5	00:02.8	00:02.9	00:02.0	00:02.3

- **Overall Social Welfare:** As we observe in Figure 4, although the relative ratio of social welfare remains very similar in the cases 5, 10, 100 and 200 min degrees, the difference is clearer for social network graphs with higher densities, where SCDG and CASG outperform PADG and PCADG by a margin of 14% to 16%.

- **Average Regret Ratio:** We observe that in all the cases while PADG and PCADG results in the same average regret ratios, these values are decreased by 9% to 5% by our algorithms, SCDG and CASG.

- **Execution Time:** Although SCDG improvement comes with the price of longer execution times, CASG execution time remains about 2 seconds while it discernibly outperforms PADG and PCADG.

4.2.3 Effect of Power Law Exponent

In this experiment we examine the effect of varying the power law exponent by considering exponents of 1, 1.5, 2, 2.5 and 3.5 while setting the number of users to 500,

the number of events to 50 and the power law minimum degree to 10 and 100.

Figure 6: Avg total welfare for different power law exponents (low density)

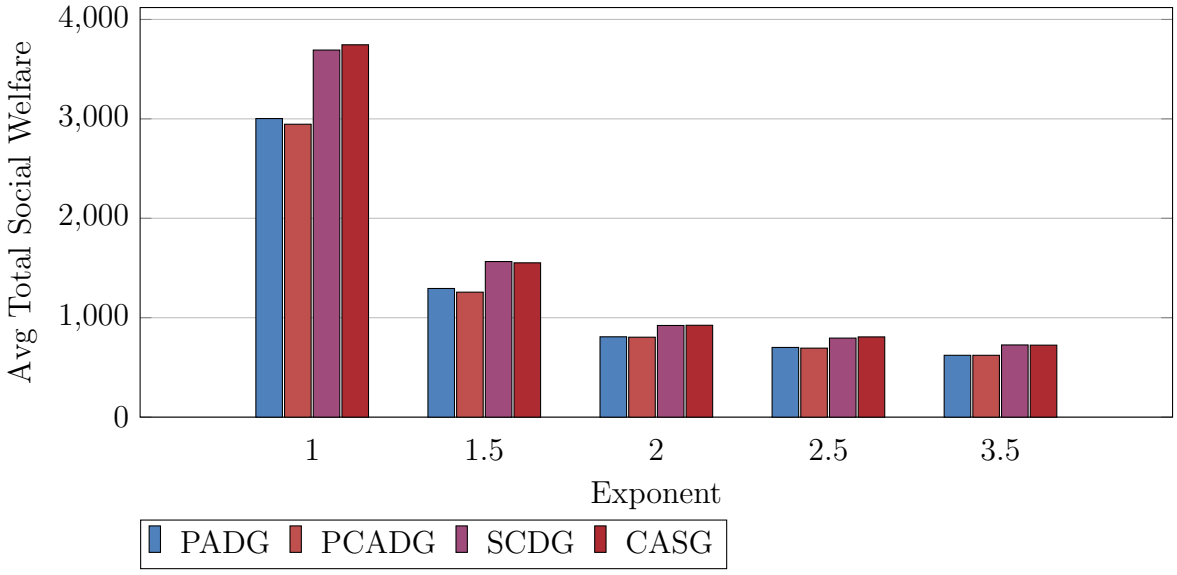


Figure 7: Avg total welfare for different power law exponents (high density)

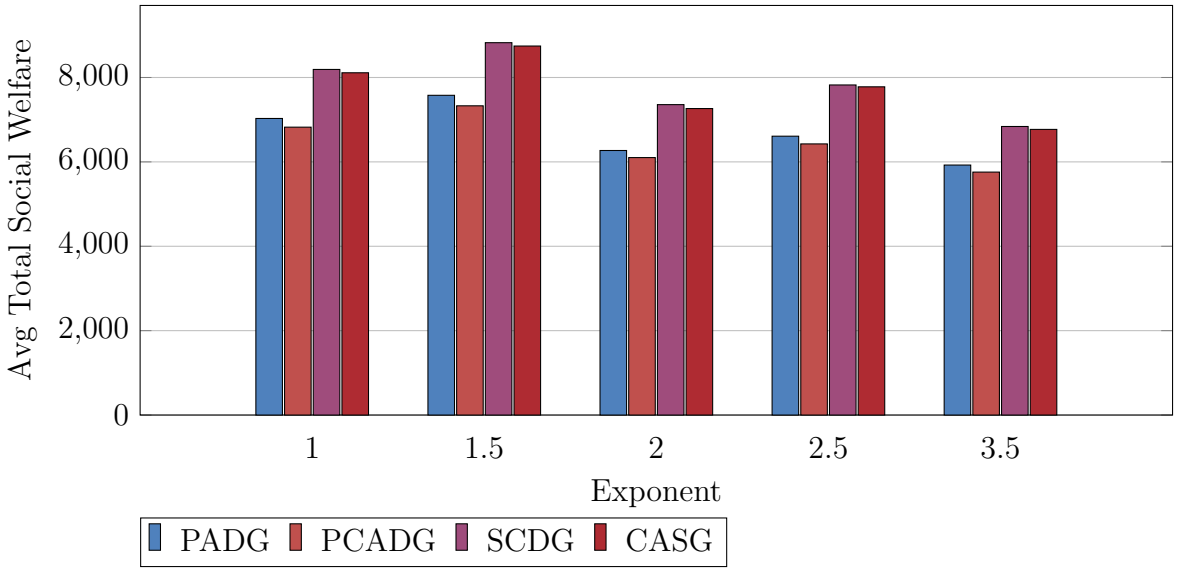


Figure 8: Avg regret ratio for different power law exponents (low density)

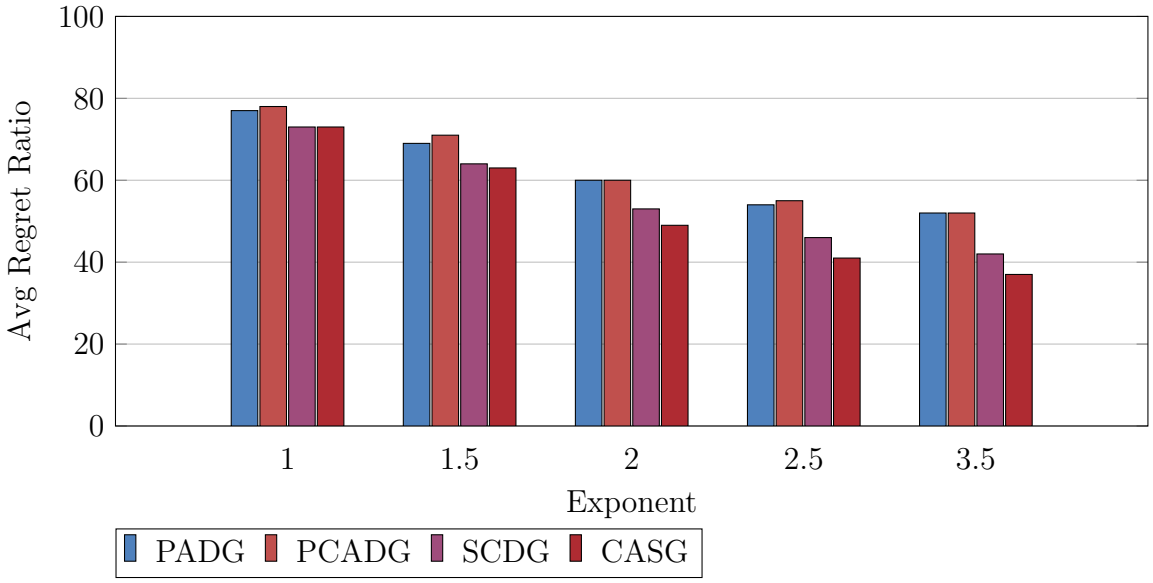


Figure 9: Avg regret ratio for different power law exponents (high density)

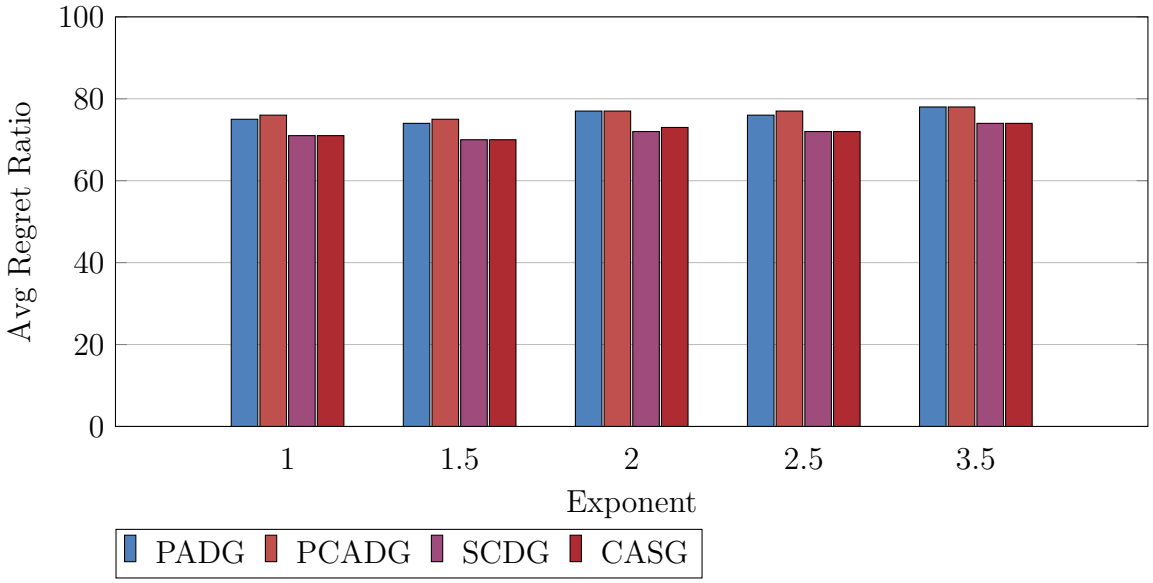


Table 4: Execution time for different power law exponents (low density)

	1	1.5	2	2.5	3.5
PADG	1.21	0.42	0.32	0.24	0.14
PCADG	1.59	0.48	0.25	0.21	0.19
SCDG	25.89	17.51	16.11	11.64	15.22
CASG	2.36	2.19	3.02	2.71	2.86

Table 5: Execution time for different power law exponents (high density)

	1	1.5	2	2.5	3.5
PADG	1.42	1.12	1.22	1.54	1.52
PCADG	1.69	1.62	1.19	1.94	1.49
SCDG	37.72	33.99	35.18	40.12	30.11
CASG	1.94	2.15	2.59	2.67	3.44

- **Overall Social Welfare:** In Figures 6 with the increase of the power law exponent from 1 to 3.5, the average overall social welfare decreases as the social network graph $G = (V, E, w)$ becomes more sparse. In the low density case where exponent is 3.5, the social welfare obtained by different algorithms approaches the same value and thus, it is not a good benchmark. For the rest of our experiments in this chapter we have set the power law exponent to be 1.5 (this is the same value as used in [24]). For higher densities, the change in social welfare with increasing exponent is less clear.

- **Average Regret Ratio:** Our results show that the average regret ratio is reduced by 5% to 29% by CASG and 5% to 10% by SCDG in comparison to the results from PADG and PCADG. Also, it is notable that as the density

of social network graphs decreases, our algorithms reduce the regret ratio by a higher rate, as it can be seen in the low density cases of exponents 2, 2.5 and 3.5.

- **Execution Time:** Although SCDG improvement comes with the price of longer execution times, CASG execution time remains about 2 seconds while it discernibly outperforms PADG and PCADG.

4.2.4 Effect of Number of Events

In this experiment we investigate the effect of varying the number of events by considering three cases of 10, 25 and 35 events while setting the number of users to 500.

Figure 10: Avg total Welfare for different numbers of events (low density)

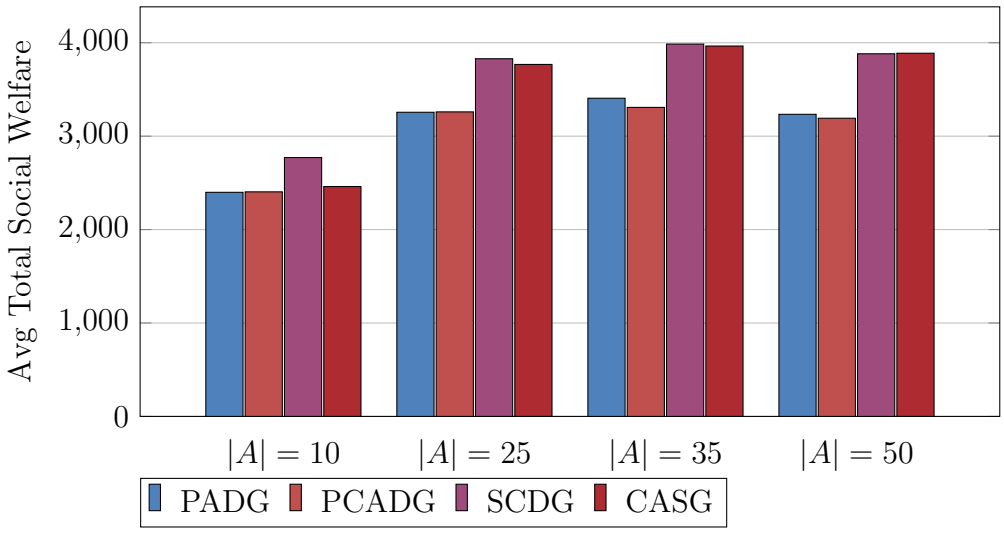


Figure 11: Avg total welfare for different numbers of events (high density)

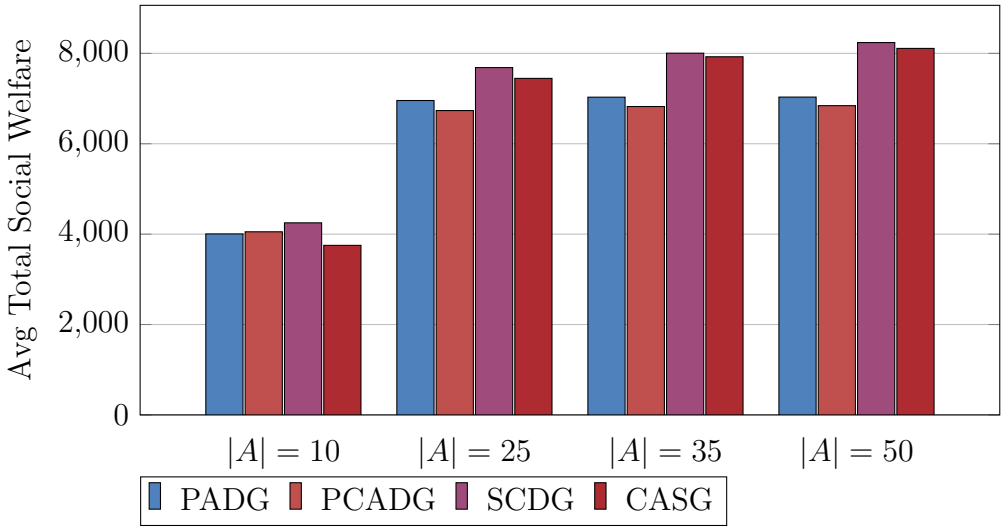


Figure 12: Avg regret ratio for different numbers of events (low density)

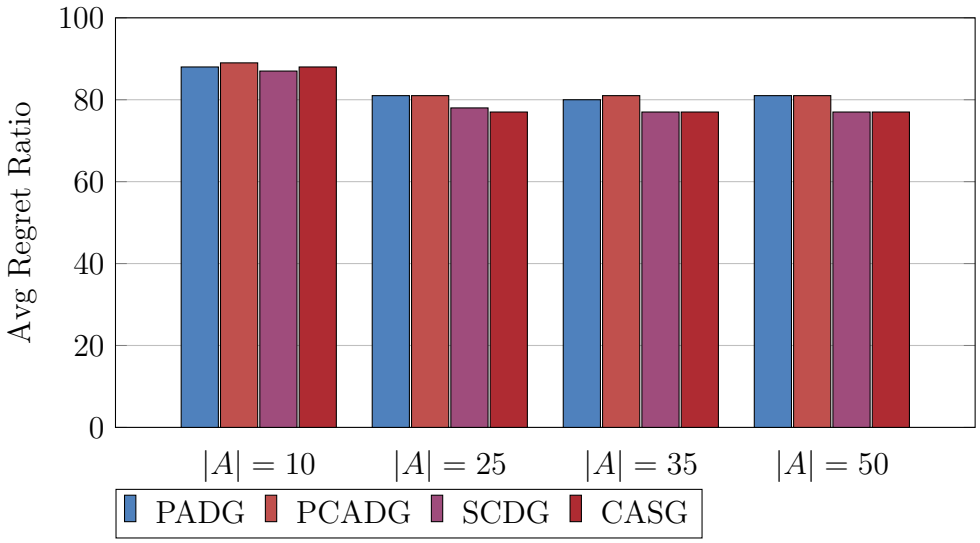


Figure 13: Avg regret ratio for different numbers of events (high density)

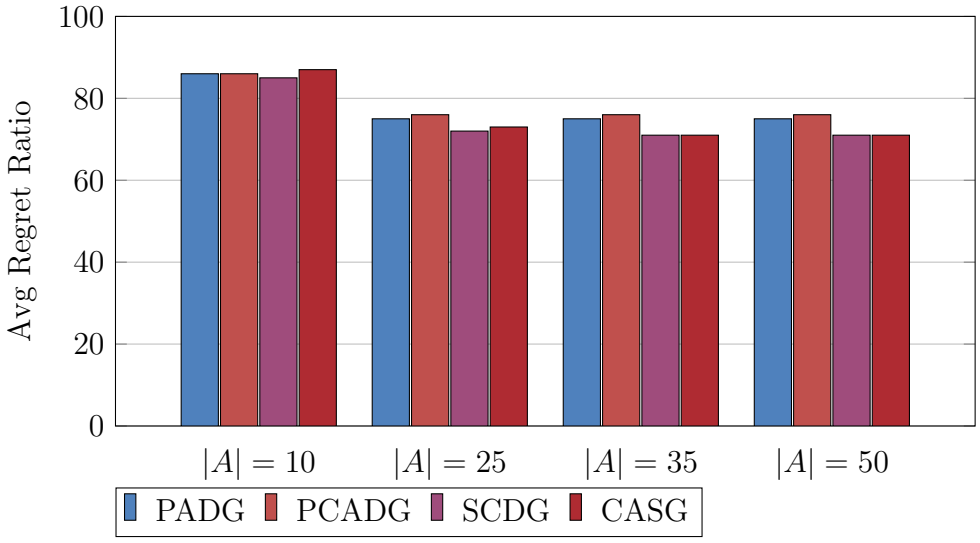


Table 6: Execution time for different numbers of events (low density)

	$\|A\| = 10$	$\|A\| = 25$	$\|A\| = 35$	$\|A\| = 50$
PADG	00:00.3	00:00.5	00:00.9	00:01.3
PCADG	00:00.5	00:00.9	00:01.1	00:01.4
SCDG	00:04.0	00:11.8	00:17.6	00:27.3
CASG	00:01.3	00:03.6	00:03.0	00:02.9

Table 7: Execution time for different numbers of events (high density)

	$\|A\| = 10$	$\|A\| = 25$	$\|A\| = 35$	$\|A\| = 50$
PADG	00:00.6	00:01.1	00:01.3	00:01.9
PCADG	00:01.5	00:02.0	00:02.0	00:02.1
SCDG	00:06.9	00:19.1	00:25.9	00:43.2
CASG	00:01.5	00:02.9	00:02.5	00:03.6

- **Overall Social Welfare:** In the low density case, SCDG's results of overall social welfare improves PADG and PCADG by 16% to 20% and CASG achieves higher social welfare by 3% to 20%. In the higher density case, 6% to 17%

improvement is achieved by SCDG and except for the case $|A| = 10$, 7% to 15% improvement is achieved by CASG.

- **Average Regret Ratio:** Both SCDG and CASG decreases the average regret ratio by 1% to 5% in comparison to results from PADG and PCADG.

- **Execution Time:** Here similar to the previous experiment, PADG is the fastest solution to the problem, CASG does a better job than PADG with 10% improvement of social welfare in average while keeping the execution time under 4 seconds in all of the cases.

4.2.5 Effect of Number of Users

In this experiment we run three instances with 1000, 1500 and 2000 users to be assigned to 50 events. With higher number of users, there will be a competition among users to be assigned to the events. In order to maintain the same social network density for social network graphs with different number of users we have recorded the average density of the social network graph $\frac{|E|}{|V|}$ for 15 graphs generated using power law model [3]. For each graph instance we have set the power law exponent to 1.5 while varying the minimum degree (denoted by d_{min}) from 1 to 110. The results are demonstrated in Figure 14.

As stated in the effect of min degree experiment, for 500 users we have considered d_{min} to be 10 and 100, and for these values, the density of generated graphs $\frac{|E|}{|V|}$ are 28.61 and 105.68 respectively. By inspecting Figure 14 we can see that for 1000, 1500 and 2000 users, the minimum degrees 6,5 and 4 will produce graph with the same density as minimum degree 10 for 500 users. Also, the minimum degrees 57, 42 and 33 produce the same density as minimum degree 100 for 500 users. Therefore, to maintain the same density for graphs with different number of users in this experiment, we consider the parameters represented in Table 8.

Figure 14: Effect of minimum degrees on graph density

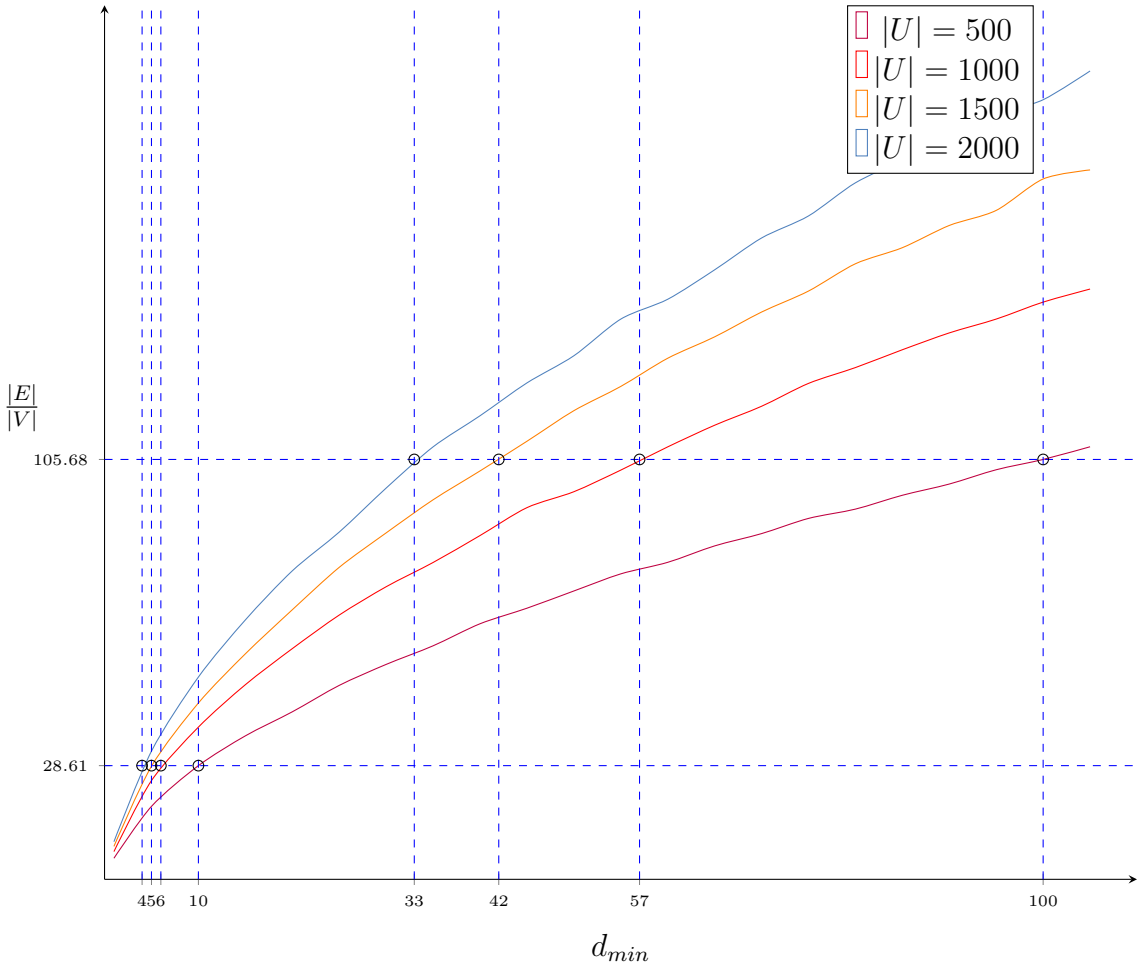


Table 8: Minimum Degrees

	Low Density	High Density
$\|V\| = 500$	10	100
$\|V\| = 1000$	6	57
$\|V\| = 1500$	5	42
$\|V\| = 2000$	4	33

Figure 15, Figure 17 and Figure 6 represents average total social welfare, average regret ratio and execution times (in seconds) respectively, resulting from running

PADG, PCADG, SCDG, SCDG on 15 distinct synthetic datasets generated by the parameters discussed in Section 4.2.

Figure 15: Avg total welfare for different numbers of users (low density)

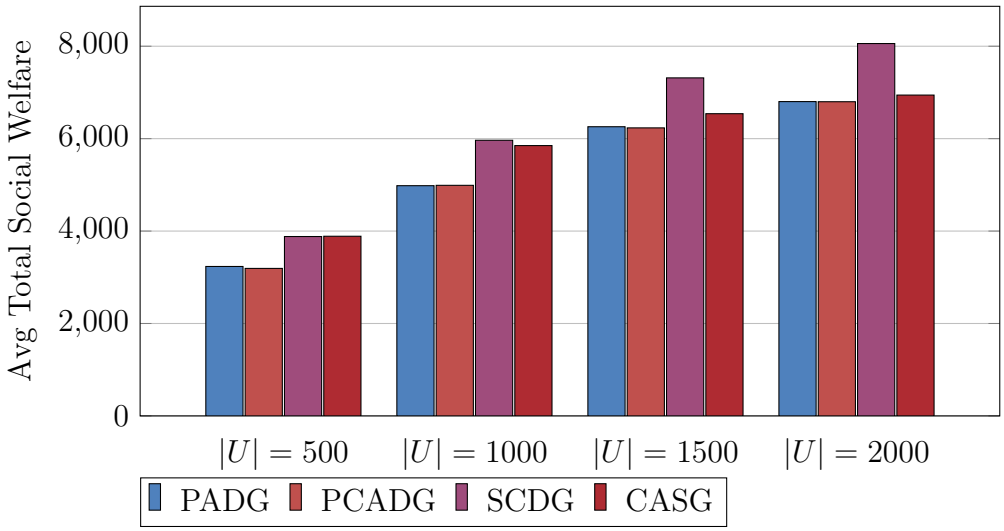


Figure 16: Avg total welfare for different numbers of users (high density)

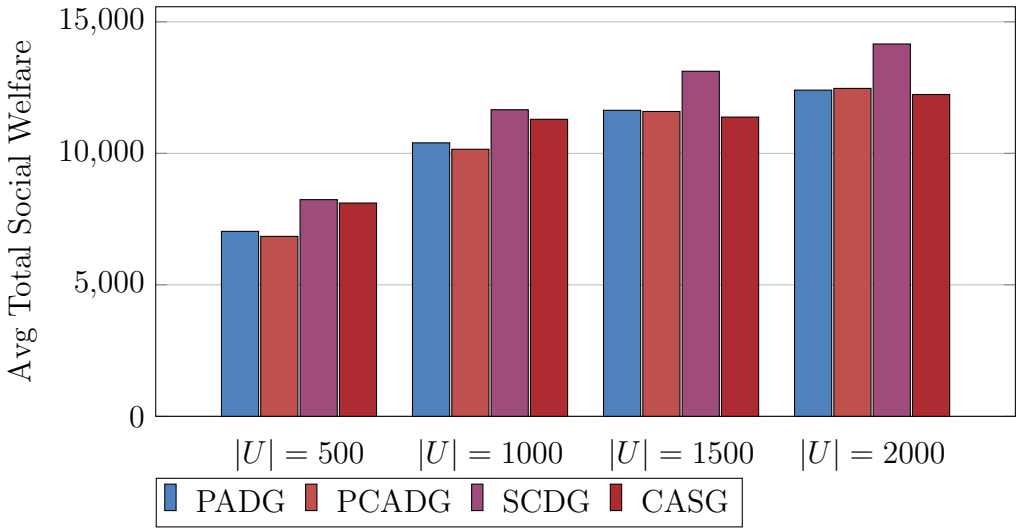


Figure 17: Avg regret ratio for different numbers of users (low density)

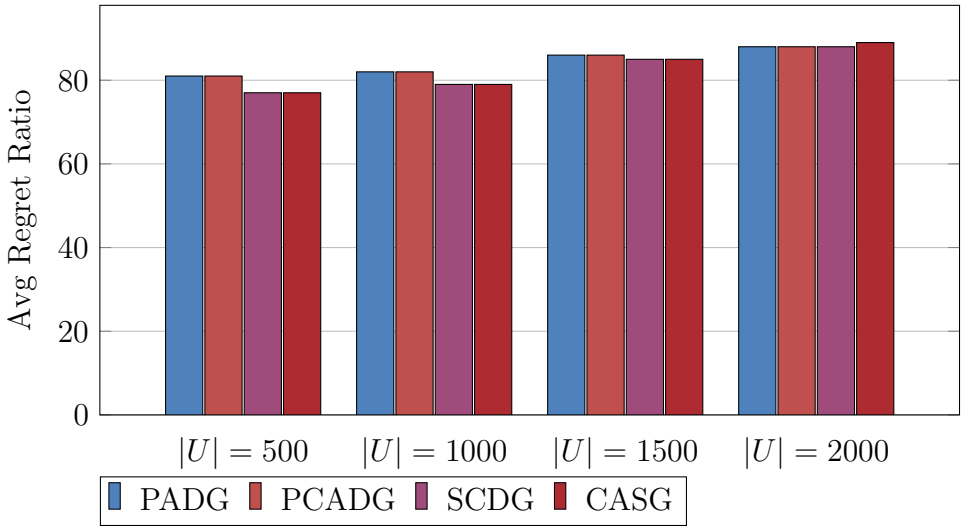


Figure 18: Avg regret ratio for different numbers of users (high density)

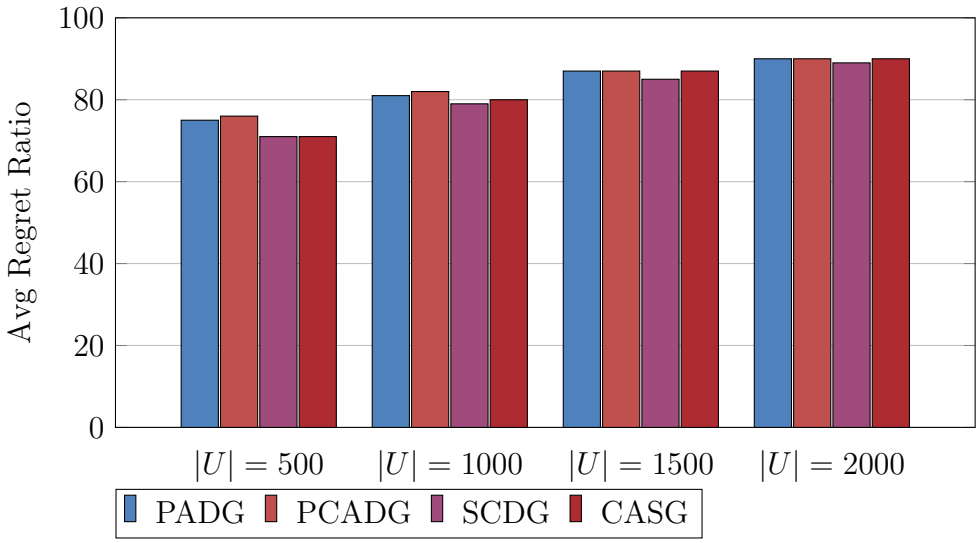


Table 9: Execution time for different numbers of users (low density)

	$\|U\| = 500$	$\|U\| = 1000$	$\|U\| = 1500$	$\|U\| = 2000$
PADG	00:01.3	00:06.3	00:10.9	00:17.9
PCADG	00:01.4	00:06.0	00:12.3	00:18.5
SCDG	00:27.3	02:06.8	03:46.0	06:24.1
CASG	00:02.9	00:13.0	00:15.9	00:24.8

Table 10: Execution time for different numbers of users (high density)

	$\|U\| = 500$	$\|U\| = 1000$	$\|U\| = 1500$	$\|U\| = 2000$
PADG	00:01.9	00:07.8	00:35.2	00:44.5
PCADG	00:02.1	00:08.9	00:33.9	00:40.9
SCDG	00:43.2	02:30.1	05:34.6	08:17.9
CASG	00:03.6	00:11.2	00:17.0	00:23.5

- **Overall Social Welfare:** As we observe SCDG achieves the highest improvement of overall social welfare among all of the discussed algorithms with 12% to 20%. Also, CASG improve PADG and PCADG results by higher margins in the cases with lower number of users.

- **Average Regret Ratio:** In this experiment SCDG reduces the average regret ratio in comparison to results of both PADG and PCADG by 1% to 5%.

- **Execution Time:** While SCDG takes the longest time to execute (and also obtains the highest social welfare gain), CASG executes as fast as PADG and PCADG while producing higher social welfare than both of them by 10% in average.

4.2.6 Meetup

In this experiment we employed the datasets from [25] to evaluate PADG, PCADG, SCDG and CASG. The Meetup data[1] includes the datasets and their descriptions as follows:

1. "user_tag": This file contains the tags that are selected by users.

2. "group_tag": This file contains the tags selected by social groups.

3. "tag_text": This file provides a mapping from tag_id to real tag text.

4. "event_group": This file represents the event and its hosting group. In meetup.com, group may host an social event.

5. "user_event": In meetup.com, users create social events. This file contains information about offline social event participations.

6. "user_group": This file represents the online social connections. Meetup allows users to form online social groups. Thus this file stores information for online group membership between user and group.

7. "event_lon_lat": This file represents the geographic location for each user's home location (inferred from user's own home address by meetup.com).

[1] http://www.largenetwork.org/ebsn

8. "user_lon_lat": This file represents the geographic location for offline social events. Similarly, the file has three columns, which are event_id, longitude and latitude, respectively.

   The process of preparing the data to conduct three different datasets involved the following steps:

1. First, we calculate the distances between all pairs of users and events from "user_lon_lat" and "event_lon_lat" datasets. Then, we create the dataset of "user event distance" by finding user-event pairs such that the distance between them is less than 50 miles. After identifying the cities of users and events according to their geolocation, we split the pairs into user-event sets of "stlouis" with 1365 users and 10 events, "louisville" with 3834 users and 14 events and "knoxville" with 2588 users and 23 events. For each dataset we use $dist(u, a)$ to denote the distance between u and a. Also, we denote the maximum distance by max_dist.

2. For each user-event pair $<u, a>$ from the sets of previous step, we use the datasets "user_group" and "event_group" from the original datasets to find out the Jaccard similarity coefficient, denoted by $\lambda_{u,a}$, by calculating the ratio of number of common groups to the number of overall groups which user u and event a have subscribed to. Let G_u and G_a denote the set of groups that users

u and event a has subscribed to respectively, then $\lambda_{u,a}$ is defined as follows:

$$\lambda_{u,a} = \frac{|G_u \cap G_v|}{|G_u \cup G_a|} \tag{19}$$

3. The dataset *user_event* provides the function $\diamond(u, e)$ that returns one if user u has been assigned to e and otherwise zero. Finally, for each user-event pair $<u, a>$, we estimate the innate interest of user u in event a according to the following formula:

$$\delta_{u,a} = \lambda_{u,a} + \diamond(u, e) + (1 - dist(u, a)/max\_dist) \tag{20}$$

4. To estimate the social interest between users of each dataset, we use Jaccard similarity coefficient, similar to [24]. Let T_u denote the set of tags assigned to u. By using the dataset "user_tag" we calculate the social interest of every pair u, v as follows:

$$w(u, v) = \frac{|T_u \cap T_v|}{|T_u \cup T_v|} \tag{21}$$

5. As these datasets do not indicate the capacities for events, we generate the maximum capacity of each event, denoted by δ_a, to be a random integer drawn from the range of $[1, 3r]$ such that $r = \frac{|U|}{|A|}$. Also, the minimum cardinality has been randomly drawn from the range of $[1, \delta_a]$.

Figure 19: Total social welfare for Meetup

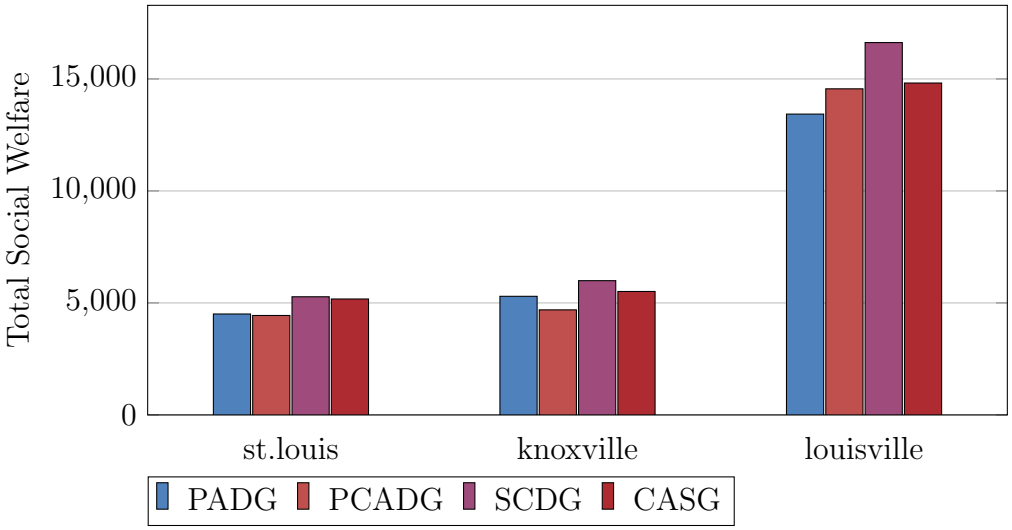


Figure 20: Average regret ratio for Meetup

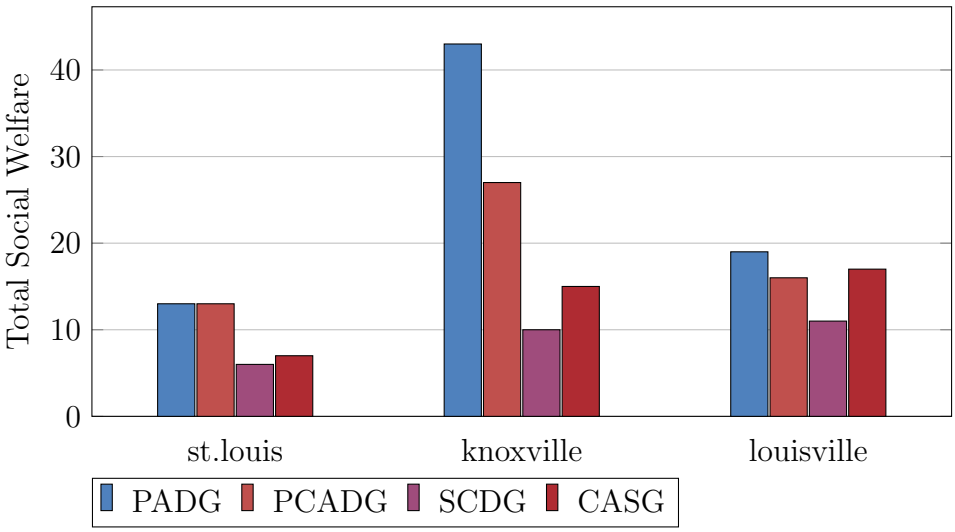


Table 11: Execution time for Meetup

	st.louis	knoxville	louisville
PADG	00:38.9	00:04.0	00:51.6
PCADG	00:39.1	00:06.9	00:55.8
SCDG	13:20.9	04:23.6	55:48.3
CASG	08:07.6	04:15.2	07:05.3

- **Overall Social Welfare:** As we can observe in our results, demonstrated in Table 19, although in the case of "louisville" PCADG performs better than PADG, in both other cases it falls behind PADG, similar to the previous experiments. Also, SCDG outperforms PADG by 17%, 13% and 24% in the cases of "st.louis", "knoxville" and "louisville" respectively. Also, CASG improves PADG results by 15%, 4% and 10% in the cases of "st.louis", "knoxville" and "louisville" respectively.

- **Regret Ratio:** In all of the three cases, the average regret ratio resulting from PADG and PCADG are reduced by 42% to 77% by SCDG and 11% to 65% by CASG. One interesting point demonstrated in Figure 20, is that in the case "knoxville" SCDG obtains the highest total social welfare and lowest regret ratio at the same time in all three cases, showing a reliable performance.

- **Execution Time Analysis:** While the longest execution time is required by SCDG in this experiment, both PADG and CADG executes in distinctly shorter times. Also, it is notable that while CASG produces the assignments with

higher overall social welfare and lower regret ratios than PADG and PCADG, its execution time is shorter than SCDG.

4.2.7 Plancast

In our last experiment in this section, we have conducted an experiment on the Plancast data published by [25], containing the follwoing files and descriptions:

1. "plancast_user_event": This file contains all the plancast social event participation information.

2. "plancast_user_subscription": This file contains the online directed social network among Plancast users.

3. "plancast_event_lon_lat": This file contains the geographic locations for Plancast social events (if specified).

We have extracted a subset of data with 6440 users and 78 events in which the innate interest between every pair of user-event has been set to 1 if the corresponding pair is found in the dataset "plancast_user_event" and otherwise it is set to 0. Also, for every pair of user1-user2, the social interest is assumed as 1 if the corresponding pair is found in the dataset "plancast_user_subscription" and otherwise 0. Finally, as there are no capacity constraints available in this dataset, for each event a we have generated the maximum capacity δ_a by sampling a normal distribution with

parameters $\mu = r$ and $\sigma = \sqrt{r}$ such that $r = \frac{|U|}{|A|}$, and minimum capacity is generated by drawing a random number from the range of $[1, \delta_a]$.

Table 12: Plancast Experiment Results

	Total Social Welfare	Avg Regret Ratio	Execution Time
PADG	13982	90%	02:32.7
PCADG	14439	90%	01:24.8
PSCDG	22981	83%	35:04.1
CASG	22765	83%	17:39.4

Figure 21: Total social welfare for Plancast

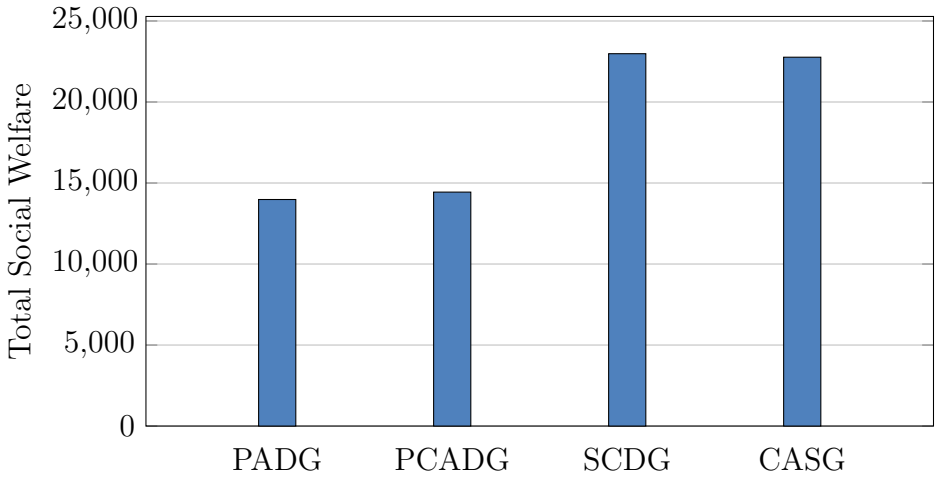
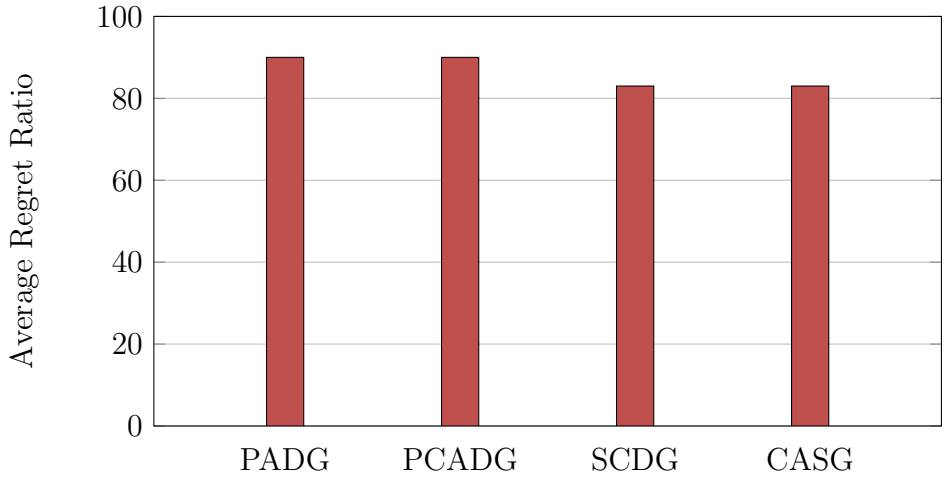


Figure 22: Average regret ratio for Plancast



- **Overall Social Welfare:** In this experiment a 64% and 58% improvement of total social welfare is obtained by SCDG and CASG respectively, proving the reliability and excellence of SCDG, CASG over PADG and PCADG.

- **Regret Ratio:** Both of our algorithms enhance PADG and PCADG results in terms of reducing the average regret ratio by 8%.

- **Execution Time Analysis:** Similar to previous experiments, PADG and PCADG execute in shorter time than SCDG and CASG. Moreover, CASG reduces the execution time of SCDG by half while providing better results than PADG and PCADG.

4.2.8 Analysis of SCDG Execution Time

As shown in the previous sections, SCDG achieves very good social welfare, but has significantly higher execution time. SCDG has three additional components, compared to PCADG: community-aware initialization, second-chance strategy, and post-processing procedures of user substitution and event switch. Which of these contributes the most to social welfare? Which of these contributes most to the execution time? In this experiment we investigate the effect of community-aware initialization, second-chance strategy, and post-processing, on the social welfare achieved as well as the overall execution time of SCDG. We consider a synthetic dataset with 1000 users and 50 events for both low density and high density social network graphs, and also, the real datasets from Meetup and Plancast, as described in Sections 4.2.4 and 4.2.4 respectively.

Figure 23: Low density (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

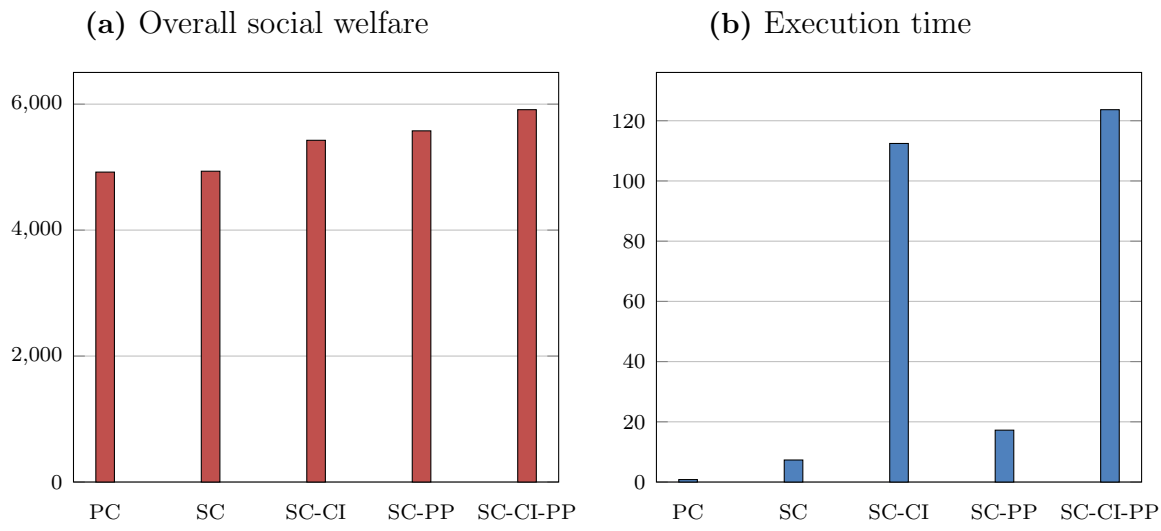


Figure 24: Regret ratio (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

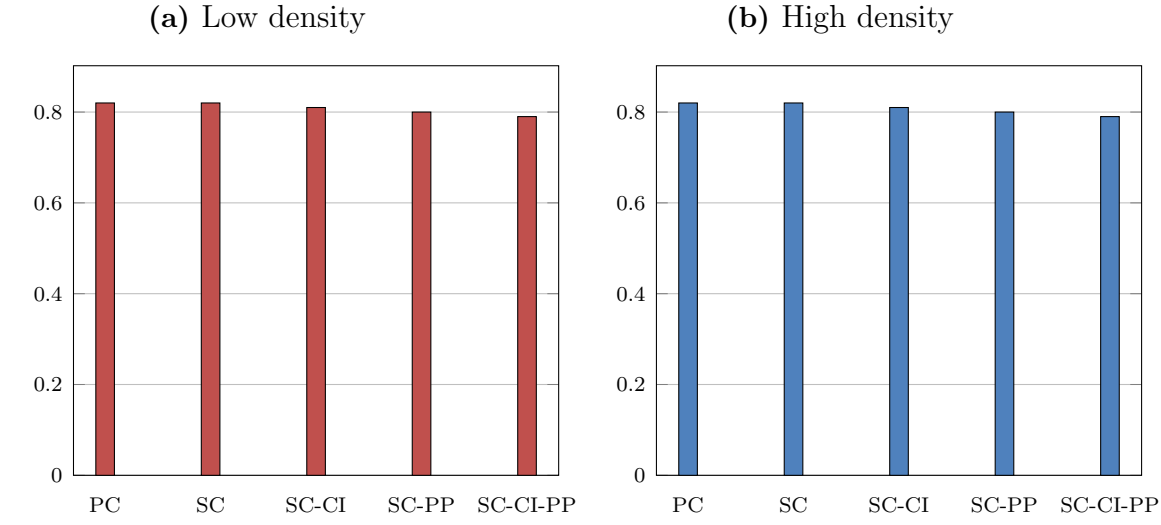
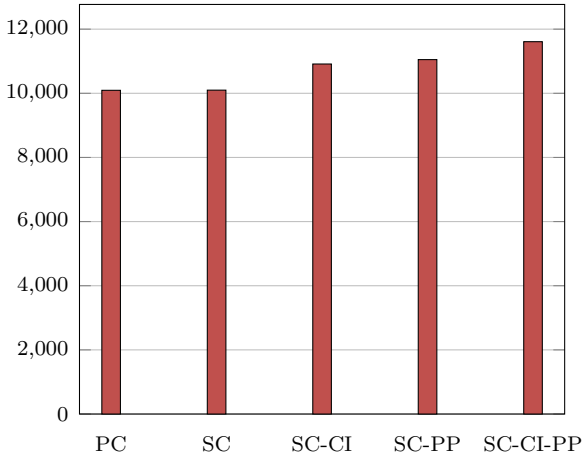


Figure 25: High density (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

(a) Overall social welfare



(b) Execution time

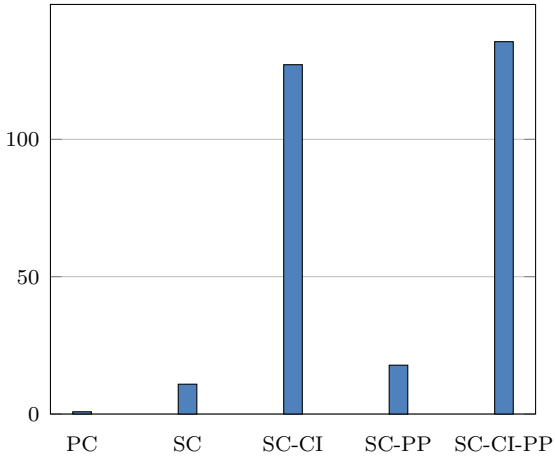
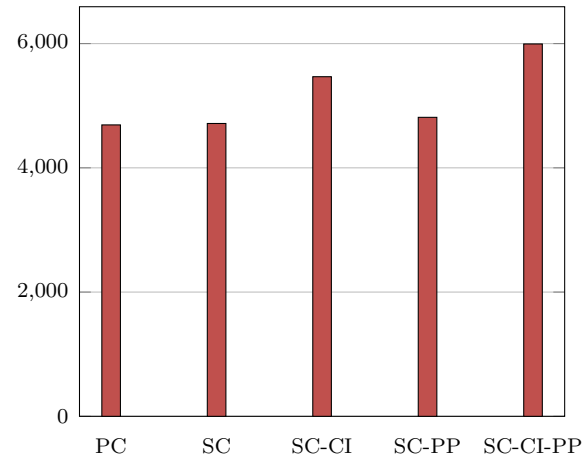


Figure 26: Meetup - knoxville (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

(a) Overall social welfare



(b) Execution time

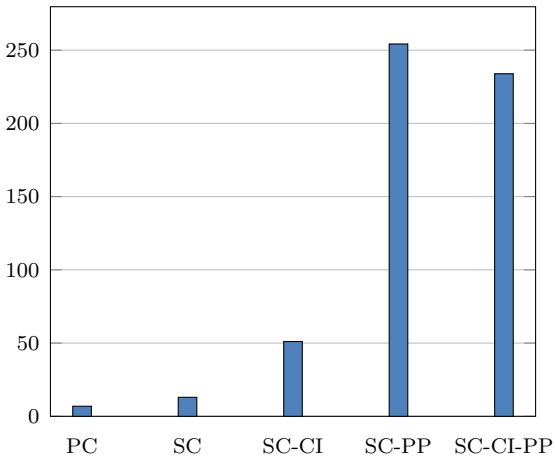


Figure 27: Meetup - stlouis (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

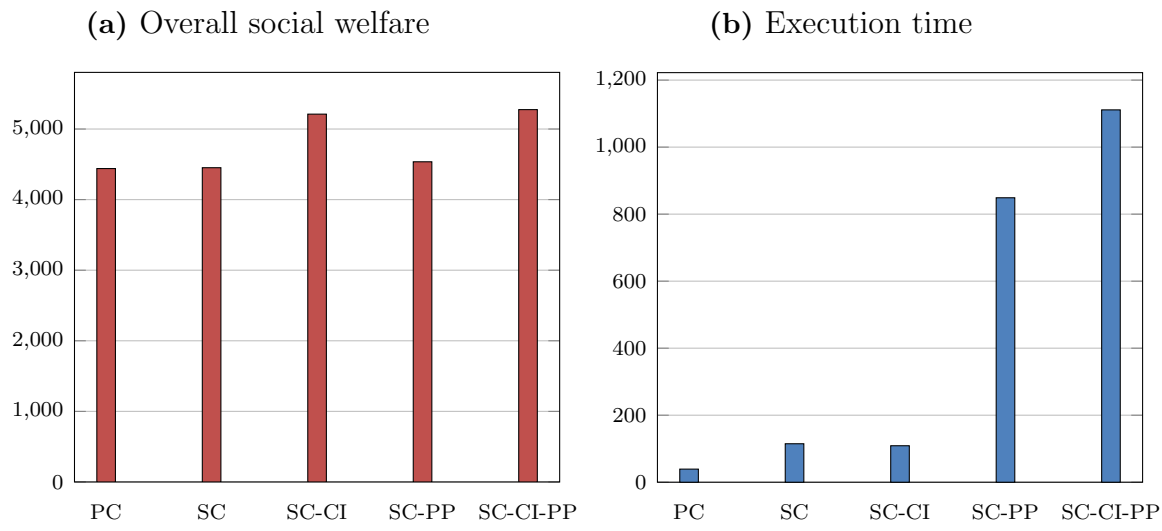


Figure 28: Meetup - louisville (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

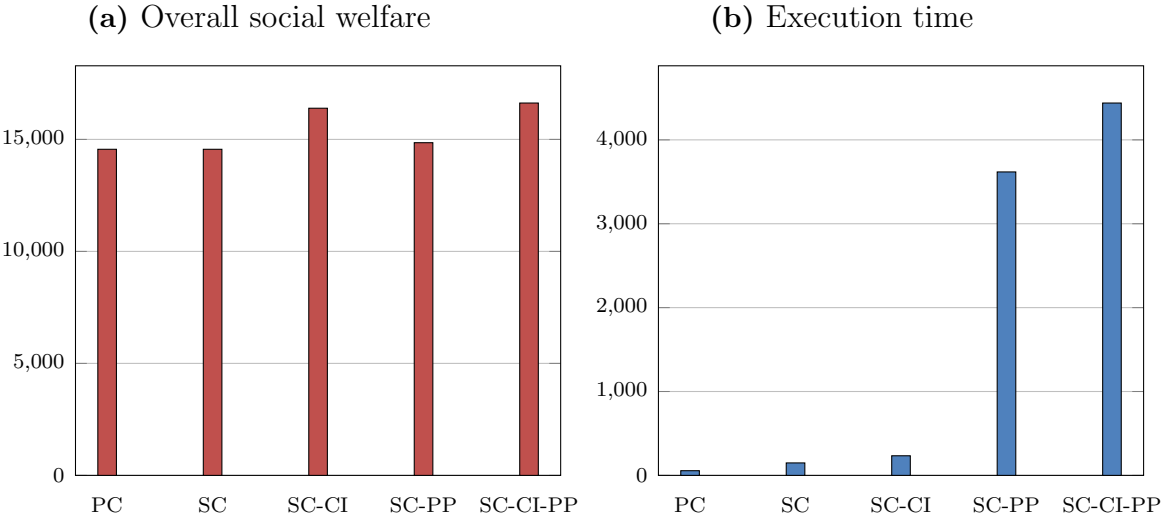


Figure 29: Average regret ratio for Meetup (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

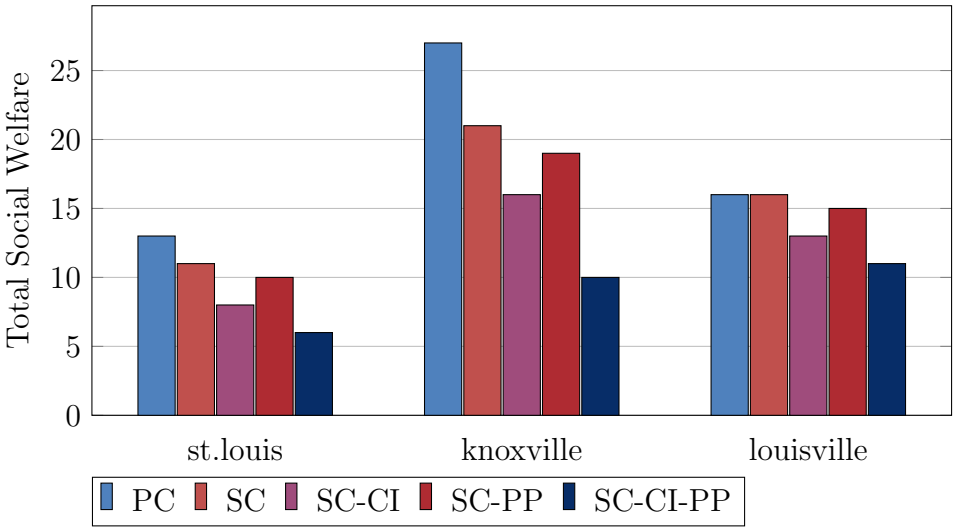
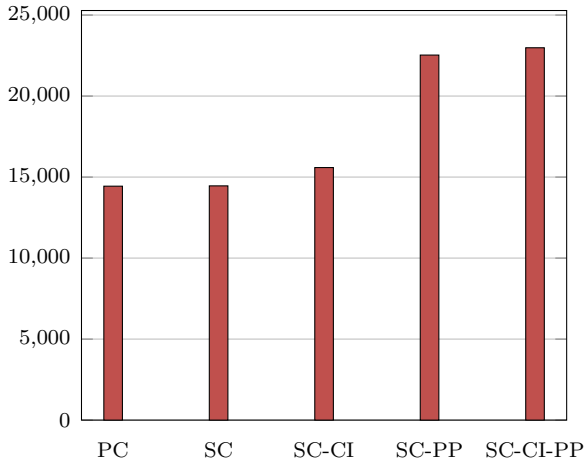


Figure 30: Plancast (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)

(a) Overall social welfare



(b) Execution time

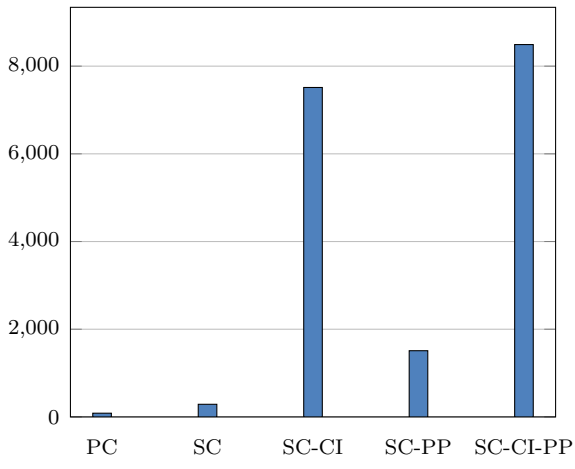
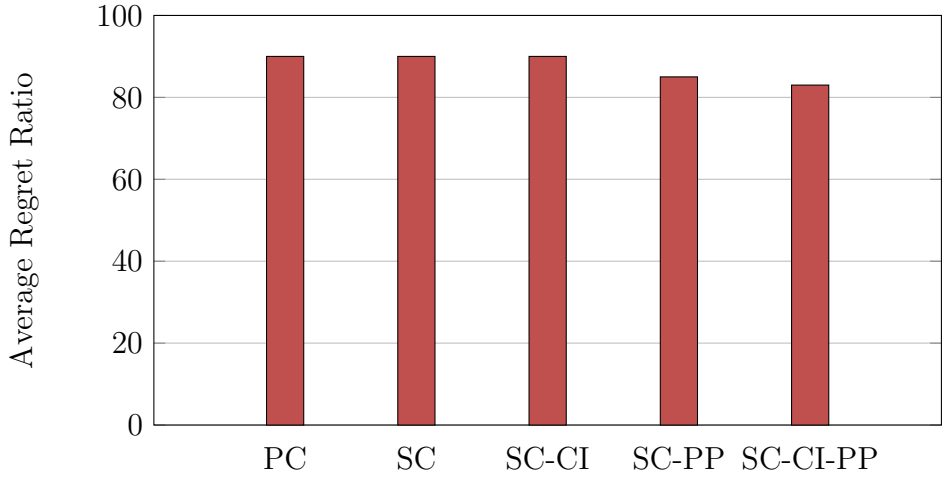


Figure 31: Average regret ratio for Plancast (PC denotes PCADG, SC denotes SCDG, CI denotes community-aware initialization, PP denotes post-processing)



- **Overall Social Welfare:** In the low density case, we observe that using the second-chance strategy using the community-aware initialization gives a very small improvement to the social welfare. However, using the community-aware initialization improves the overall social welfare by 10%. Also, using the post-processing improves the results by 13%. Moreover, using both of them together gives us overall 20% improvement in comparison to the version in which none of them are employed.

  In the high density case, once again, the second-chance strategy gives a negligible improvement to the social welfare, but we observe 8% improvement by employing community-aware initialization, 9% improvement by employing post-processing and finally, by using both of them we obtain a 14% improvement.

The Plancast data set has similar results to the above synthetic data sets, except that PP alone gives a 45% improvement compared to community-aware initialization alone; as before, the combination gives the best results. The Meetup data sets are different in the sense that the community-aware initialization strategy gives a better social welfare than using just post-processing, but as before the combination gives the best results.

- **Average Regret Ratio:** It can be observed that by using community-aware initialization and post-processing the average regret ratio decreases. The best results are achieved through employing both of the techniques in all of the cases.

- **Execution Time:** In the synthetic and Plancast datasets the community-aware initialization increases the execution time the most while the second-chance strategy and the post-processing procedure contributes the least to the execution time. However in the case of Meetup the post-processing takes the longest time to execute.

4.3 Personality-Oriented Algorithm Experiments

In this section we conduct experiments on POCASG and POSCDG using synthetic datasets. For each experiment we have considered two approaches to generate the extrovert indices:

1. **Random (R):** A rational number is drawn uniformly at random from interval $[0, 1]$.

2. **Proportional To Node Degree (ND):** Based on the assumption that the user with highest number of users is the most extroverted and the user with the least number of friends is the most introverted user in the set of users U, for each user u we calculate $\frac{|F_u|}{Max_{v \in U}(|F_v|)}$, where F_u is the set of users who user u has social interest in, i.e. $F_u = \{v \in U | w(u, v) > 0\}$.

Finally, we use the parameters from Section 4.2 where the social network graph is generated by the power law model [3] with power law exponent = 1.5.

4.3.1 Effect of Number of Users

In this experiment we consider 1000, 1500 and 2000 users to be assigned to 50 events.

Figure 32: Avg total welfare for different numbers of users (R, low density)

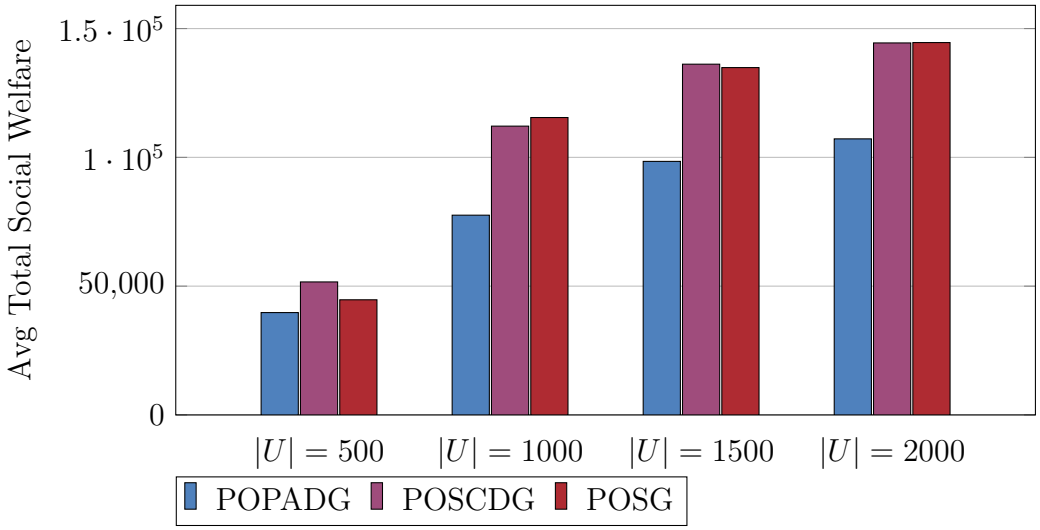


Figure 33: Avg total welfare for different numbers of users (R, high density)

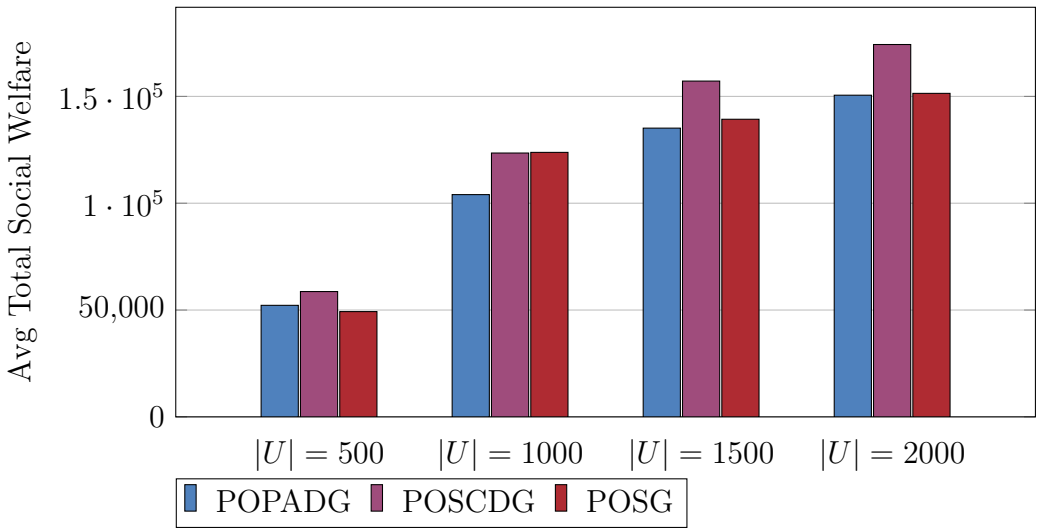


Figure 34: Avg total welfare for different numbers of users (ND, low density)

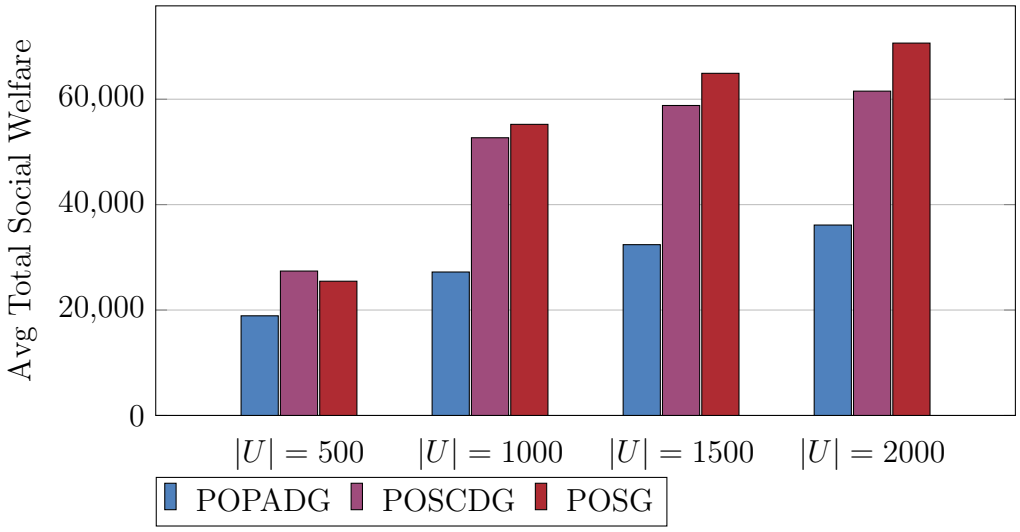


Figure 35: Avg total welfare for different numbers of users (ND, high density)

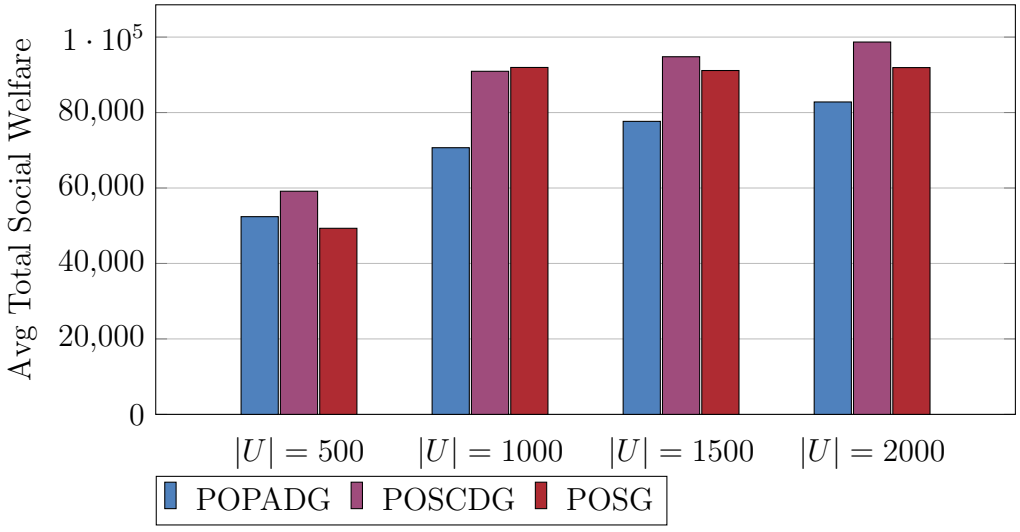


Table 13: Execution time for different numbers of users (R, low density)

	$\|U\| = 500$	$\|U\| = 1000$	$\|U\| = 1500$	$\|U\| = 2000$
POPADG	00:00.7	00:02.8	00:03.9	00:09.0
POSCDG	01:19.7	05:07.0	04:30.9	05:04.8
POSG	000:33.3	04:30.5	02:45.9	03:18.7

Table 14: Execution time for different numbers of users (R, high density)

	$\|U\| = 500$	$\|U\| = 1000$	$\|U\| = 1500$	$\|U\| = 2000$
POPADG	00:02.0	00:07.3	00:14.1	00:21.4
POSCDG	01:26.8	06:04.4	05:31.9	05:39.4
POSG	00:40.6	03:34.2	03:15.4	03:29.7

Table 15: Execution time for different numbers of users (ND, low density)

	$\|U\| = 500$	$\|U\| = 1000$	$\|U\| = 1500$	$\|U\| = 2000$
POPADG	00:00.8	00:02.0	00:05.7	00:07.8
POSCDG	01:41.3	05:40.9	06:13.3	05:05.7
POSG	00:44.0	03:40.6	04:39.6	02:12.4

Table 16: Execution time for different numbers of users (ND, high density)

	$\|U\| = 500$	$\|U\| = 1000$	$\|U\| = 1500$	$\|U\| = 2000$
POPADG	00:02.0	00:07.5	00:14.2	00:22.1
POSCDG	01:32.5	06:40.4	06:19.5	06:13.1
POSG	00:39.7	03:44.3	03:40.0	03:56.6

- **Overall Social Welfare:** In the case of random extroversion indices and low density network (Figure 32), our algorithms improve the overall social welfare in comparison to POPADG results by the margins of 17% to 45% improvement resulting from running POSCDG and 15% to 49% improvement resulting from running POSG. Moreover, in the case of random extroversion indices and high

density network (Figure 33), POSCDG gives the best results in all of the cases with a margin of 12% to 19% improvement in comparison to POPADG.

In the case where extroversion indices are generated proportional to the node degrees and the density social network graph is low (Figure 34), the results from POPADG are improved with the margins of 41% to 94% by POSCDG and 28% to 103% by POSG. Also, in the case of high density social network graph (Figure 35), the POPADG results are improved by the margins of 13% to 28% by POSCDG and 11% to 30% by POSG (except in the case of 500 users where POPADG outperforms POSG by 6%.

- **Execution Time:** While POPADG executes the fastest and POSCDG executes in a discernibly longer time, POSG executes faster than POSCDG while improving the results from POPADG in most of the cases.

4.3.2 Effect of Number of Events

In this experiment we investigate the effect of varying the number of events by considering three cases of 10, 25 and 35 events while setting the number of users to 500.

Figure 36: Avg total Welfare for different numbers of events (R, low density)

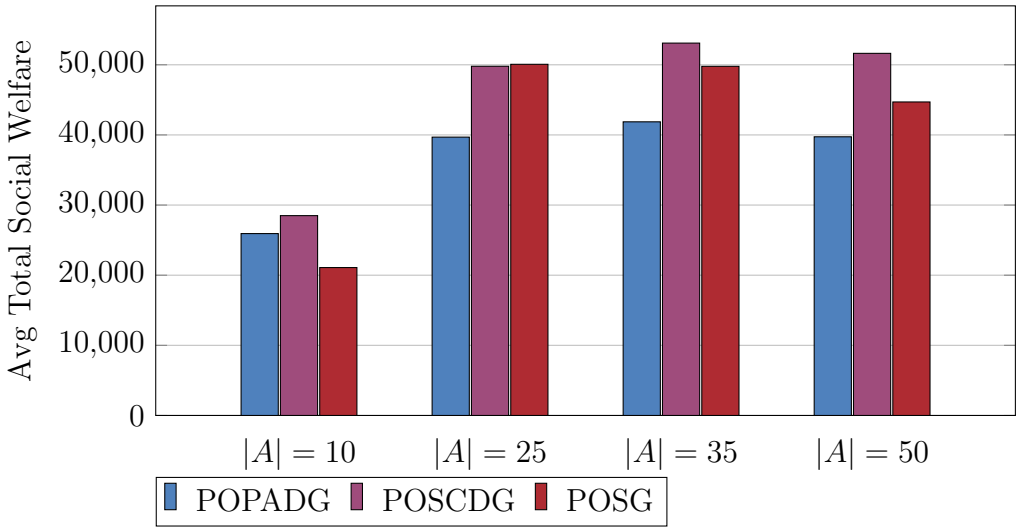


Figure 37: Avg total Welfare for different numbers of events (R, high density)

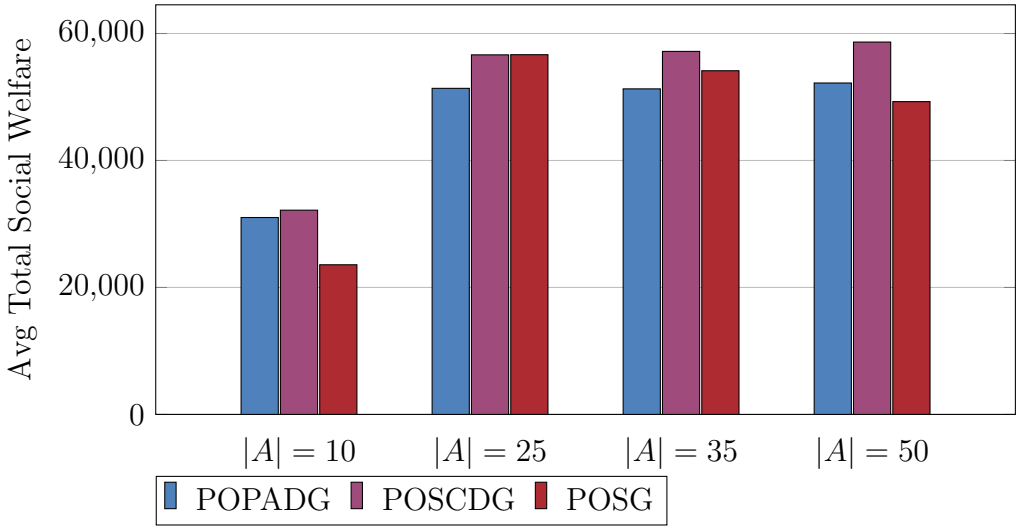


Figure 38: Avg total welfare for different numbers of events (ND, low density)

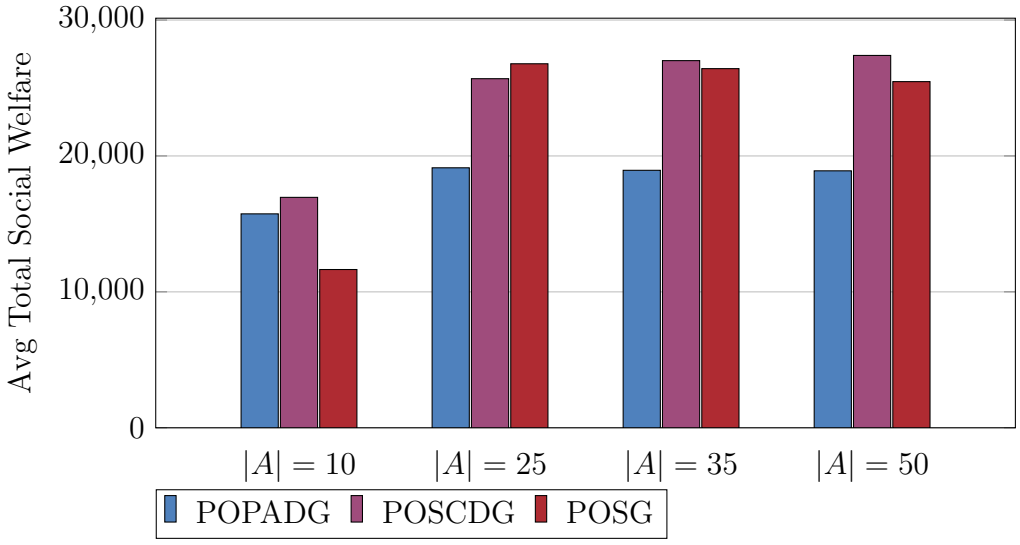


Figure 39: Avg total welfare for different numbers of events (ND, high density)

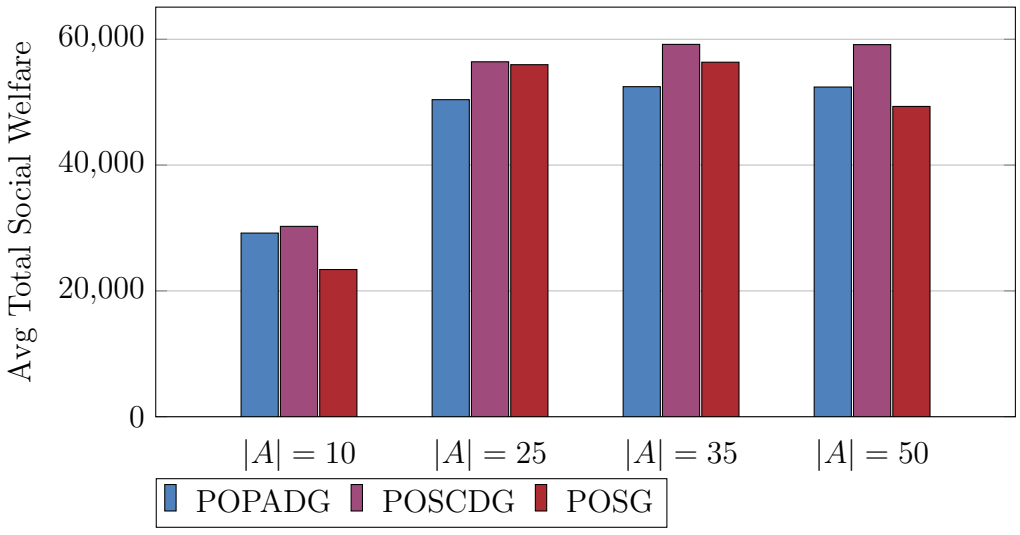


Table 17: Execution time for different numbers of events (R, low density)

	$\|A\| = 10$	$\|A\| = 25$	$\|A\| = 35$	$\|A\| = 50$
POPADG	00:00.3	00:00.9	00:00.9	00:00.7
POSCDG	00:12.9	01:20.1	01:38.6	01:19.7
POSG	00:09.8	01:01.9	01:00.1	00:33.3

Table 18: Execution time for different numbers of events (R, high density)

	$\|A\| = 10$	$\|A\| = 25$	$\|A\| = 35$	$\|A\| = 50$
POPADG	00:01.1	00:01.9	00:02.0	00:02.0
POSCDG	00:21.1	01:25.5	01:30.4	01:26.8
POSG	00:14.9	00:51.1	00:52.3	00:40.6

Table 19: Execution time for different numbers of events (ND, low density)

	$\|A\| = 10$	$\|A\| = 25$	$\|A\| = 35$	$\|A\| = 50$
POPADG	00:00.5	00:00.6	00:00.8	00:00.8
POSCDG	00:25.0	01:58.0	01:52.6	01:41.3
POSG	00:24.9	01:14.0	01:07.0	00:44.0

Table 20: Execution time for different numbers of events (ND, high density)

	$\|A\| = 10$	$\|A\| = 25$	$\|A\| = 35$	$\|A\| = 50$
POPADG	00:01.3	00:01.9	00:01.9	00:02.0
POSCDG	00:19.7	01:29.3	01:32.8	01:32.5
POSG	00:13.1	00:53.5	00:49.5	00:39.7

- **Overall Social Welfare:** In the case of random extroversion indices and the low density social network graph, POSCDG improves the results from POPADG by the margin of 10% to 30%. Moreover, POSG improves POPADG by 12% to 26%, except for the case where $|A| = 10$ that POPADG outperforms POSG.

In the case of high density social network graph, we observe 4% to 13% improvement by POSCDG and 7% to 11% improvement by POSG in comparison to POPADG results (except for the cases of $|A| = 10$ and $|A| = 50$ where POPADG gives better results than POSG).

In the case where extroversion indices are generated proportional to the node degrees, POSCDG outperforms POPADG by a margin of 8% to 45% improvement, and except in the case $|A| = 10$ in which POPADG outperform POSG by 26%, in the rest of cases POSG outperforms POPADG by 28% to 40%. Moreover, in the case of high density social network graph, POSCDG outperforms POPADG by the margin of 12%.

Similar to the previous experiment, POSCDG gives the best results in terms of the overall social welfare.

- **Execution Time:** Similar to Section 4.3.1, POPADG executes under a second and while POSCDG promises the best results in terms of overall social welfare, it executes in a longer time than POPADG and POSG.

4.3.3 Effect of Minimum Cardinality

In this experiment, for 500 users and 50 events, we have drawn minimum cardinality from three intervals which are defined as functions of maximum cardinality bound.

For each event, namely a, We use the generated maximum value from 4.2 and we draw the minimum cardinality integer value randomly from three intervals of $[\delta_a/2, \delta_a]$ (low mean), $[\delta_a - \delta_a/4, \delta_a]$ (medium mean) and $[\delta_a - \delta_a/8, \delta_a]$ (high mean).

Figure 40: Total social welfare for different minimum cardinalities (R, low density)

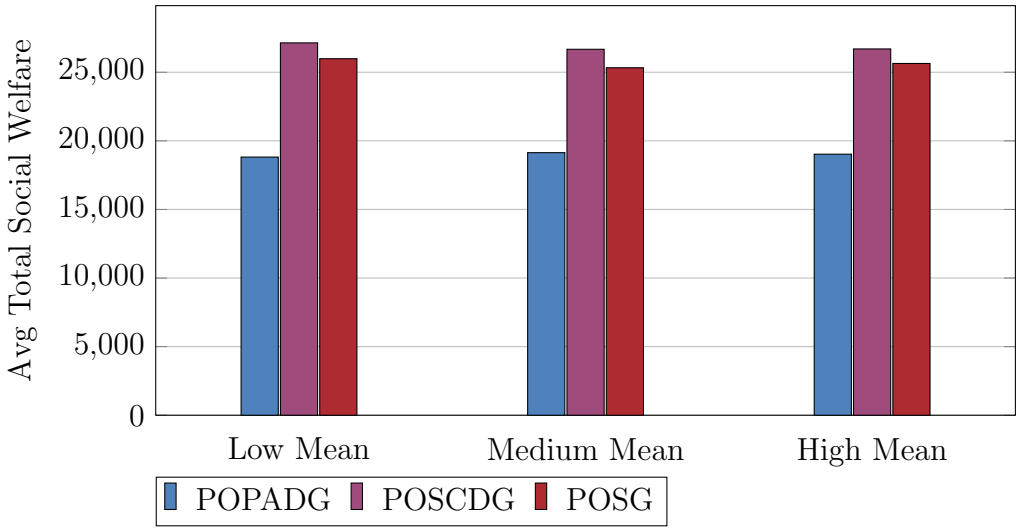


Figure 41: Total social welfare for different minimum cardinalities (R, high density)

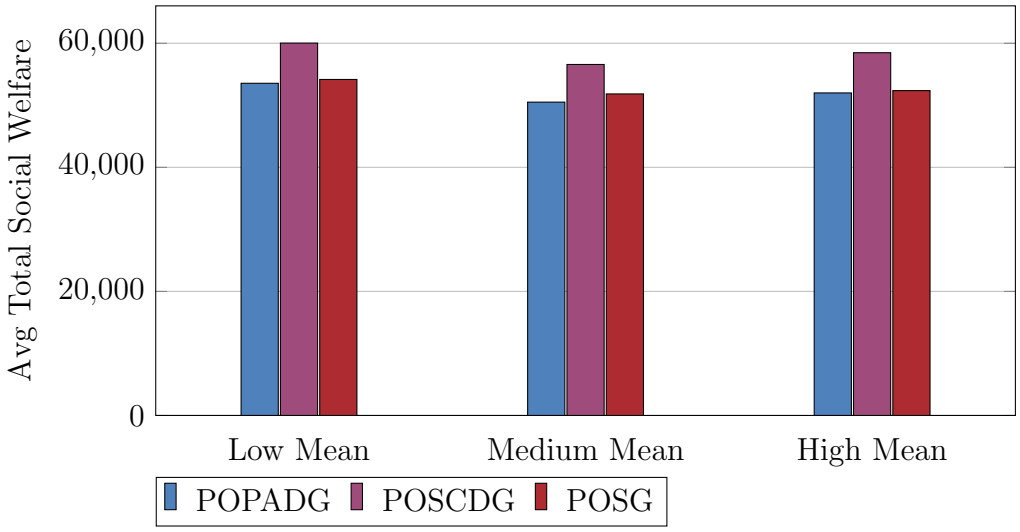


Figure 42: Total social welfare for different minimum cardinalities (ND, low density)

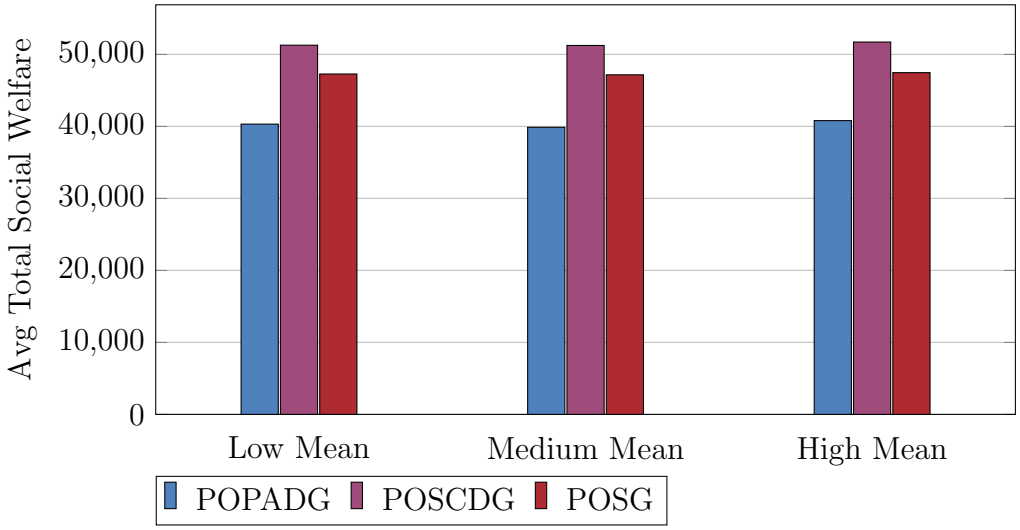


Figure 43: Total social welfare for different minimum cardinalities (ND, high density)

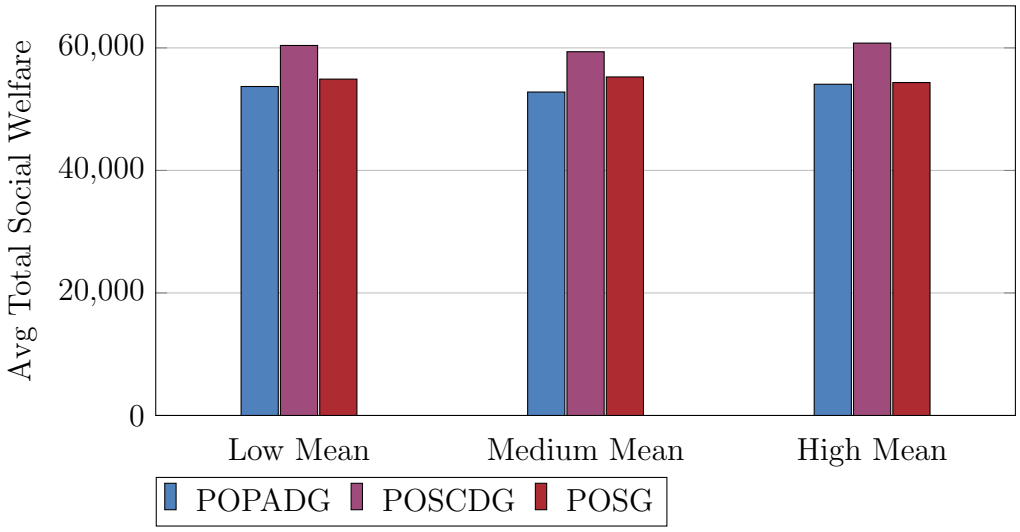


Table 21: Execution time for different numbers of events (R, low density)

	Low Mean	Medium Mean	High Mean
POPADG	00:01.0	00:01.1	00:01.0
POSCDG	01:39.5	01:36.6	01:29.8
POSG	00:43.1	01:02.6	00:53.3

Table 22: Execution time for different numbers of events (R, high density)

	Low Mean	Medium Mean	High Mean
POPADG	00:02.2	00:02.2	00:02.3
POSCDG	01:35.5	01:33.7	01:40.9
POSG	00:46.2	00:48.3	00:51.8

Table 23: Execution time for different numbers of events (ND, low density)

	Low Mean	Medium Mean	High Mean
POPADG	00:00.8	00:01.0	00:01.1
POSCDG	01:35.7	01:24.9	01:49.3
POSG	00:52.0	01:01.2	00:46.7

Table 24: Execution time for different numbers of events (ND, high density)

	Low Mean	Medium Mean	High Mean
POPADG	00:02.1	00:02.2	00:02.5
POSCDG	01:28.6	01:32.5	01:42.7
POSG	00:40.5	00:46.5	00:48.4

- **Overall Social Welfare:** In the case of random extroversion indices and the low density social network graph, POSCDG outperforms POPADG by the margins of 39% to 44% and POSCDG outperforms POPADG by the margins of 32% to 38%. Also, in the case of high density social network graph, we achieve 12% improvement by POSCDG.

  In the case where extroversion indices are generated proportional to the node degrees, POSCDG outperforms POPADG by a margin of 27% to 29% improvement, and POSG outperforms POPADG by 16% to 18%. Moreover, in the case of high density social network graph, POSCDG outperforms POPADG by the margin of 12%.

- **Execution Time:** While POPADG executes under 2 seconds, POSG and POSCDG execute in a longer time than POPADG. Moreover, POSG is two times faster than POSCDG.

4.3.4 Conclusion

While PADG scores the fastest execution time while achieving higher social welfare than PCADG in most of the experiments, SCDG with a discernible overload of execution, outperforms PADG in terms of total social welfare and average regret ratio in all the experiment cases provided in this chapter, providing the most reliable and dependable solution to the SEO problem among all the studied and proposed algorithms. Also, CASG performance stands between PADG and SCDG, both in terms of the execution time and the overall social welfare, proposing a more scalable solution to the problem.

In the case of the personality-oriented algorithms, the main conclusion is that POSCDG delivers the best social welfare in all experiments, for all parameter settings. In general, POSG also outperforms POPADG, however, in some experiments, it achieves a smaller social welfare than POPADG (when the number of events is very small, or when the number of users is very small and the network has high density).

Finally, as observed throughout our experiments, choosing one of the studied and proposed algorithms is a matter of requirements. In the cases in which the short execution time is crucial then PADG seems to be the most rewarding choice while in the cases in which the longer execution time is acceptable, then SCDG is the more promising answer. Moreover, CASG gives a good trade-off between execution time and social welfare of the solution.

Chapter 5

Conclusions and Future Work

We studied the social event organization problem defined in [24]: given innate interests of users in events, and a social network between the users, and minimum and maximum number of users that can be assigned to an event, find an assignment of users to events that respects the event cardinality bounds, and maximizes the overall social welfare. The problem is shown to be NP-complete, and in fact, hard to approximate [24].

First, we reviewed the previous solutions proposed in [24] and outlined some problems in these algorithms. We then proposed two new algorithms for the SEO problem: SCDG and CASG. Our experiments show that both our algorithms outperform the solutions proposed by [24] across a wide range of network parameters. In particular, they obtain a 60% increase in social welfare on real-world network data sets.

Second, we proposed a personality-oriented objective function that considers for every user, an extroversion index, that affects the willingness of the user to participate in an event as a function of the size of the event and the presence of strangers at the event. Moreover, we propose three algorithms, POPADG, POSCDG, and POSG, which are adaptations of the abovementioned solutions, for the personality-oriented objective function. We did an extensive experimental analysis of these algorithms. Considering execution time POSG is the best algorithm over a wide range of factors such as the social network density and the number of events, while POSCDG produces the most promising results in terms of the overall social welfare at the cost of longer execution time.

In terms of future research, we propose a number of directions to pursue. As discussed in Chapter 4 our main focus in this thesis was to improve the social welfare obtained by the SEO algorithms, however another possible research can be conducted to improve and optimize the performance of the SEO algorithms. For example, it would be useful to devise a solution that can be exerted for big data, e.g. employing models such as MapReduce to run the algorithm on multiple machines in parallel. To improve the performance of our personality-oriented algorithms, it would be interesting to find more realistic ways of modeling the extroversion index. Another lead is to categorize users into leader and follower types and apply different assignment strategies for different types of users.

References

[1] Meetup, 2017. URL https://www.meetup.com/about/.

[2] Networkit, 2017. URL https://networkit.iti.kit.edu/.

[3] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC 2000)*, pages 171–180. ACM, 2000.

[4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, 2005.

[6] Vincent Boyer, Didier El Baz, and Moussa Elkihel. Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound. *European Journal of Industrial Engineering*, 4(4):434–449, 2010.

[7] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment problems*. SIAM Press, 2012.

[8] Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. pages 213–222, 2000.

[9] Donald Cohen and James P. Schmidt. Ambiversion: characteristics of midrange responders on the introversion-extraversion continuum. *Journal of Personality Assessment*, 43(5):514–516, 1979.

[10] George B Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[11] P. Erdös and A. Rényi. On random graphs I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[12] Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 611–620, 2006.

[13] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[14] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[15] P. C. Gilmore and Ralph E. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14(6):1045–1074, 1966.

[16] S Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, 1962.

[17] Václav Havel. A remark on the existence of finite graphs. *Casopis Pest. Mat.*, 80:477–480, 1955.

[18] Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(2):026107, 2002.

[19] Jianbin Huang, Yu Zhou, Xiaolin Jia, and Heli Sun. A novel social event organization approach for diverse user choices. *The Computer Journal*, 2016.

[20] Iaroslav Ispolatov, PL Krapivsky, and A Yuryev. Duplication-divergence model of protein interaction network. *Physical Review E*, 71(6):061911, 2005.

[21] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on*

Foundations of Computer Science, 1982. SFCS'08. (FOCS 82), pages 312–320, 1982.

[22] Peter J Kolesar. A branch and bound algorithm for the knapsack problem. *Management Science*, 13(9):723–735, 1967.

[23] Sven O Krumke and Clemens Thielen. The generalized assignment problem with minimum quantities. *European Journal of Operational Research*, 228(1):46–55, 2013.

[24] Keqian Li, Wei Lu, Smriti Bhagat, Laks V. S. Lakshmanan, and Cong Yu. On social event organization. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (KDD '14)*, pages 1206–1215, 2014.

[25] Xingjie Liu, Qi He, Yuanyuan Tian, Wang-Chien Lee, John McPherson, and Jiawei Han. Event-based social networks: Linking the online and offline social worlds. In *Proceedings of the 18th ACM SIGKDD conference on Knowledge Discovery and Data Mining*, (KDD'12), 2012.

[26] Isabel Briggs Myers. *The Myers-Briggs type indicator*. Consulting Psychologists Press, 1962.

[27] Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4):341–346, 1999.

[28] Jakob Puchinger, Günther R Raidl, and Ulrich Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.

[29] Alvin E. Roth and Elliott Peranson. The redesign of the matching market for American physicians: Some engineering aspects of economic design. Technical report, 1999.

[30] Jieying She, Yongxin Tong, and Lei Chen. Utility-aware social event-participant planning. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1629–1643, 2015.

[31] Wei Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, pages 369–378, 1979.

[32] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1-3):461–474, 1993.

[33] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2009.

[34] Yongxin Tong, Jieying She, and Rui Meng. Bottleneck-aware arrangement over event-based social networks: the max-min approach. *In Proceedings of the 2016 World Wide Web Conference*, 19(6):1151–1177, 2016.

[35] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *Nature*, 393(6684):440–442, 1998.

[36] H. Martin Weingartner and David N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103, 1967.