Attendance Maximization for Successful Social Event Planning

Nikos Bikakis
Athens Univ. of Econ. & Bus.
Greece

Vana Kalogeraki
Athens Univ. of Econ. & Bus.
Greece

Dimitrios Gunopulos
University of Athens
Greece

ABSTRACT

Social event planning has received a great deal of attention in recent years where various entities, such as event planners and marketing companies, organizations, venues, or users in Event-based Social Networks, organize numerous social events (e.g., festivals, conferences, promotion parties). Recent studies show that "attendance" is the most common metric used to capture the success of social events, since the number of attendees has great impact on the event's expected gains (e.g., revenue, artist/brand publicity). In this work, we study the *Social Event Scheduling* (SES) problem which aims at identifying and assigning social events to appropriate time slots, so that the number of events attendees is maximized. We show that, even in highly restricted instances, the SES problem is NP-hard to be approximated over a factor. To solve the SES problem, we design three efficient and scalable algorithms. These algorithms exploit several novel schemes that we design. We conduct extensive experiments using several real and synthetic datasets, and demonstrate that the proposed algorithms perform on average half the computations compared to the existing solution and, in several cases, are 3-5 times faster.

1 INTRODUCTION

The event planning industry has grown enormously in the past decade, with large *event planning* and *marketing* companies (e.g., MKG, GPJ), organizing and managing a variety of social events (e.g., multi-themed festivals, promotion parties, conferences). In addition to companies, social events are also organized by *venues* (e.g., theaters, night clubs), *organizations* (e.g., ACM, TED), as well as *users* in *Event-based Social Networks* (e.g., Meetup, Eventbrite, Plancast).

The *Event Marketing Benchmark Report 2017*,[1] where marketing decision-makers from large organizations participate, indicates that "*attendance*" *is the most common metric used to measure the success of social events*, since the number of attendees has a great influence on the event's expected gains (e.g., revenue, artist publicity). Therefore, achieving maximum attendance is the *organizers first challenge*, as also indicated in the *Event Marketing Trends 2018* study.[2]

Examples of events organization include large festivals and conferences where a large number of (multi-themed) events are organized over several stages and sessions attracting several thousands of people. For example, *Summerfest Festival* has performances from over 800 bands, attracting more than 800K people each year. Beyond music concerts, numerous multi-themed events take place, ranging from artmakings and theatrical performances to fitness activities and parties. In such scenarios, successful event planning is extremely challenging, since various factors need to be taken into account, such as the *large number of events* and available *time slots*, the *diversity of events' themes* and *user interests*, the presence of *overlapped events*, the *available resources* (e.g., available stages), etc.

Assume the following scenario. On Monday two events are scheduled to take place during a festival: (1) a *rock concept* from 19:00 to 22:00; and (2) a *fashion show* between 19:00 and 21:00. Additionally, from 18:00 to 20:00 a music concert of a *rock singer* is taking place in a nearby (competing) venue. Consider that Alice enjoys listening to *rock music*, and is a *fashion lover*. Although Alice is interested in all three events, she is only able to attend one of them.

In this work, we study the *Social Event Scheduling* (SES) problem [4]. *Given a set of candidate events, a set of time intervals and a set of users, SES assigns events to time intervals, in order to maximize the overall number of participants.* The assignments are determined by considering several events' and users' factors, such as user preferences and habits, events' spatiotemporal conflicts, etc.

Recently, several studies have been published examining the problem of assigning *users* (i.e., participants) to a set of *pre-scheduled events* in Event-based Social Networks [6, 12, 26–29, 31]. The objective in these works is to find the *user-event assignments that maximizes the satisfaction of the users*. Here, in contrast to existing works, we study a substantially different problem. Briefly, instead of assigning users to events, we assign *events* to *time intervals*. The objective here is to find the *event-time assignments that maximize the number of event's attendees*. More or less, the SES problem studies the "satisfaction" (e.g., revenue, publicity) of the entities involved in event organization (e.g., organizer, artist, sponsors, services' providers). In other words, SES is an "*organizer-oriented*" problem, while the existing works are "*participant-oriented*". Overall, the objective, the solution and the setting of the SES problem are substantially different from the related works.

The SES problem was recently introduced in [4] where a greedy algorithm was proposed. In the proposed solution, in each assignment selection, the algorithm recomputes (i.e., updates) the scores for a large number of assignments. Additionally, in each selection the algorithm has to examine (e.g., check for validity) all the assignments. The aforementioned result to poor performance of this solution. In this work, we design three efficient and scalable algorithms which are implemented on top of the following novel schemes. First, we propose an *incremental updating* scheme in which a reduced number of score computations are performed in an incremental manner. Further, we design an *assignment organization* scheme which significantly reduces the number of assignments that are examined. Finally, an *assignment selection policy* is proposed, minimizing the impact of performing a part of the required score computations, on the quality of the results. In our extensive experiments, we illustrate that the proposed algorithms perform about half the computations and, in several cases, are 3-5× faster compared to the method proposed in [4].

[1] www.certain.com/blog/certain-presents-the-event-marketing-benchmark-report-spring-2017
[2] https://welcome.bizzabo.com/event-marketing-2018

Event	Location
e_1	Stage 1
e_2	Stage 1
e_3	Room A
e_4	Stage 2

(a) Candidate Events

t_1 = ⟨Friday 8–11pm⟩

t_2 = ⟨Sat. 6–9pm⟩

(b) Time Intervals

c_1 ⟨Friday 6–9pm⟩, $t_{c_1} = t_1$

c_2 ⟨Sat. 8–10pm⟩, $t_{c_2} = t_2$

(c) Competing Events

	Event Interest				Comp. Ev. Interest		Activ. Prob.	
User	e_1	e_2	e_3	e_4	c_1	c_2	t_1	t_2
u_1	0.9	0.3	0	0.6	0.8	0.3	0.8	0.5
u_2	0.2	0.6	0.1	0.6	0.4	0.7	0.5	0.7

(d) Users

Figure 1: Running example (4 Candidate events, 2 Intervals, 2 Competing events, 2 Users)

Further, examining the theoretical aspects of the SES problem, we study its approximation, showing that even in highly restricted instances, it is NP-hard to be approximated over a factor larger than $(1 - \epsilon)$.

Contributions. The main contributions of this work are summarized as follows: (1) We show that the SES problem is NP-hard to be approximated over a factor larger than $(1 - \epsilon)$. (2) We design three efficient and scalable approximation algorithms. These algorithms outperform the existing algorithm by exploiting a series of schemes that we develop. (3) We conduct an detailed experimental analysis using several real and synthetic datasets.

2 SOCIAL EVENT SCHEDULING PROBLEM

In this section we first define the *Social Event Scheduling* (SES) problem; and then we study its approximation. In what follows, we present a simple example that introduces the main entities involved in SES problem.

Example 1. [*Running Example*] Figure 1 outlines our running example involving four *candidate events* (e_1–e_4), two *time intervals* (t_1, t_2), two *competing events* (c_1, c_2), and *two users* (u_1, u_2).

The *location* of each *candidate event* is presented in Figure 1a We notice that both e_1 and e_2 are going to be hosted at Stage 1. Hence, these events cannot be scheduled to take place during the same time period. Figure 1c presents the *competing events* along with the time periods during which these are scheduled to take place. For example, c_1 is schedule to take place on Friday between 6:00 and 9:00pm (at a nearby competing venue). Further, in Figure 1b we observe that there is the candidate *time interval* t_1 defined on the same day between 8:00 and 11:00pm. Thus, due to overlapping time periods, a user cannot attend both c_1 and a candidate event that will be possibly scheduled to take place during t_1.

Finally, Figure 1d shows, for each user, the *interest values* (i.e., affinity) for the events, as well as the *social activity probability* (e.g., based on user habits) during the time periods defined by the two intervals. For example, u_1 has high social activity probability (equals to 0.8) at t_1, since Friday night u_1 does not work and usually goes out and participates in social activities.

2.1 Problem Definition

In our problem, we assume an ***organizer*** (e.g., company, venue) managing the events' organization. Each organizer possesses a number of ***available resources*** $\theta \in \mathbb{R}^+$. These are abstractions used to refer to staff, materials, budget or any other means related to event organization.

Further, let $\mathcal{T}$ be a set of ***candidate time intervals***, representing time periods that are available for organizing events.

Assume a set $\mathcal{E}$ of available events to be scheduled, referred as ***candidate events***. Each $e \in \mathcal{E}$ is associated with a ***location*** ℓ_e representing the place (e.g., a stage) that is going to host the event. Further, each event e requires a specific amount of resources $\xi_e \in \mathbb{R}_0^+$ for its organization, referred as ***required resources***.

An ***assignment*** α_e^t denotes that the candidate event $e \in \mathcal{E}$ is scheduled to take place at $t \in \mathcal{T}$. An event ***schedule*** $\mathcal{S}$ is a set of assignments, where there exist no two assignments referring to the same event. Given a schedule $\mathcal{S}$, we denote as $\mathcal{E}(\mathcal{S})$ the set of all candidate events that are scheduled by $\mathcal{S}$, i.e., $\mathcal{E}(\mathcal{S}) = \{e \mid a_e^t \in \mathcal{S}\}$; and $\mathcal{E}_t(\mathcal{S})$ the candidate events that are scheduled by $\mathcal{S}$ to take place at t (i.e., assigned to t). Further, for a candidate event $e \in \mathcal{E}(\mathcal{S})$, we denote as $t_e(\mathcal{S})$ the time interval on which $\mathcal{S}$ assigns e.

A *schedule* $\mathcal{S}$ is said to be ***feasible*** if the following constraints are satisfied: (1) $\forall t \in \mathcal{T}$ holds that $\nexists e_i, e_j \in \mathcal{E}_t(\mathcal{S})$ with $\ell_{e_i} = \ell_{e_j}$ (***location constraint***); and (2) $\forall t \in \mathcal{T}$ holds that $\sum_{\forall e \in \mathcal{E}_t(\mathcal{S})} \xi_e \leq \theta$ (***resources constraint***). In analogy, an *assignment* α_e^t is said to be ***feasible*** if the aforementioned constraints hold for t. Further, we call ***valid assignment***, an assignment α_e^t when the assignment is *feasible* and $e \notin \mathcal{E}(\mathcal{S})$.

Let C be a set of ***competing events***, with $C \cap \mathcal{E} = \varnothing$. As competing events we define events that have already been scheduled by third parties, and will possibly attract potential attendees of the candidate events. Based on its scheduled time, each competing event $c \in C$ is associated with a time interval $t_c \in \mathcal{T}$. Further, as C_t we denote the competing events that are associated with the time interval t.

Consider a set of users $\mathcal{U}$, for each ***user*** $u \in \mathcal{U}$ and event $h \in \mathcal{E} \cup C$, there is a function $\mu\colon \mathcal{U} \times (\mathcal{E} \cup C) \to [0, 1]$, denoted as $\mu_{u,h}$, that models the ***interest*** of user u over h. The interest value (i.e., affinity) can be estimated by considering a large number of factors (e.g., preferences, social connections).

Moreover, we define the ***social activity probability*** σ_u^t, representing the probability of user u participating in a social activity at t. This probability can be estimated by examining the user's past behavior (e.g., number of check-ins).

Assume a user u and a candidate event $e \in \mathcal{E}$ that is scheduled by $\mathcal{S}$ to take place at time interval t; $\rho_{u,e}^t$ denotes the ***probability of*** u ***attending*** e ***at*** t. Considering the *Luce's choice theory* [17], the probability $\rho_{u,e}^t$ is influenced by the social activity probability σ of u at t, and the interest μ of u over e, C_t and $\mathcal{E}_t(\mathcal{S})$. We define the *probability of* u *attending* e *at* t as:

$$\rho_{u,e}^t = \sigma_u^t \frac{\mu_{u,e}}{\sum_{\forall c \in C_t} \mu_{u,c} + \sum_{\forall p \in \mathcal{E}_t(\mathcal{S})} \mu_{u,p}} \tag{1}$$

Furthermore, considering all users $\mathcal{U}$, we define the ***expected attendance*** for an event e scheduled to take place at t as:

$$\omega_e^t = \sum_{\forall u \in \mathcal{U}} \rho_{u,e}^t \tag{2}$$

The ***total utility*** for a schedule $\mathcal{S}$, denoted as $\Omega(\mathcal{S})$, is computed by considering the expected attendance over all scheduled events:

$$\Omega(\mathcal{S}) = \sum_{\forall e \in \mathcal{E}(\mathcal{S})} \omega_e^{t_e(\mathcal{S})} \quad (3)$$

The *Social Event Scheduling* (SES) problem is defined as follows: [3]

Social Event Scheduling Problem (SES). Given an positive *integer* k, a set of candidate *time intervals* $\mathcal{T}$; a set of *competing events* C; a set of *candidate events* $\mathcal{E}$; and a set of *users* $\mathcal{U}$; our goal is to find a *feasible schedule* $\mathcal{S}_k$ that determines how to assign k candidate events such that the *total utility* Ω is maximized; i.e., $\mathcal{S}_k = \arg\max \Omega(\mathcal{S})$ and $|\mathcal{S}| = k$.

Note that, by performing trivial modifications to the algorithms proposed here, additional factors and constraints to those defined in SES, can be easily handled. For example, include event's organization cost/fee (to define a "profit-oriented" version of the SES problem), associate events with duration, or define weights over the users (e.g., based on their influence).

2.2 Approximation Hardness

Here, we show that even in highly restricted instances the SES problem is NP-hard to be approximated over a factor. Therefore, SES does not admit a *Polynomial Time Approximation Scheme* (PTAS).

Theorem 1. There exists an $\epsilon > 0$ such that it is NP-hard to approximate the SES problem within a factor larger than $(1 - \epsilon)$. Thus, SES does not admit a PTAS. [4]

PROOF SKETCH. In our proof we reduce the 3-Bounded 3-Dimensional Matching problem (3DM-3) [10] to a restricted instance of SES. The following is an instance of the 3DM-3 problem. Given a set $T \subseteq X \times Y \times Z$, with $|X| = |Y| = |Z| = n$, $|T| = m$ and with each element of $X \cup Y \cup Z$ appearing at most 3 times as a coordinate of an element of T. A matching in T is a subset $M \subseteq T$, such that no elements in M agree in any coordinate. In our proof, we exploit the following result: [10] showed that in 3DM-3 there exists an $\epsilon_0 > 0$ such that it is NP-hard to decide whether an instance has a matching of size n or if every matching has size at most $(1 - \epsilon_0)n$.

Consider the following *associations between* 3DM-3 and SES: edges g in T to time intervals; and elements in X, Y, Z to candidate events, with required resources $\xi = 1$. Let the aforementioned candidate events form a set E_1 (i.e., $|E_1| = 3n$).

Further, in the proof, we consider the following *restricted instance of* SES: (1) The *available resources* are three. (2) There are no *location constraints*. (3) There is only one *competing event* in each time interval. (4) The *social activity probability* is the same for each user and time interval. (5) The *users* are as many as the candidate events. (6) There is a set E_2 that contains $m - n$ additional (w.r.t. E_1) candidate events, with $\xi = 3$. Thus, the candidate events $\mathcal{E}$ in the restricted instance is $\mathcal{E} = E_1 \cup E_2$, with $E_1 \cap E_2 = \varnothing$. (7) Regarding the *interest function* we assume two disjoint sets of users $|U_1| = 3n$ and $|U_2| = m - n$, a well as the following: $(7a)$ Each user $u_1 \in U_1$ likes only one event $e_1 \in E_1$ (as a result, each e_1 is liked only by one user u_1), with $\mu_{u_1,e_1} = 0.25$. $(7b)$ Regarding the *competing events* and the users U_1 we have the following. Fix a positive constant $\delta < \frac{1}{12}$. Let $u_p \in U_1$ the user that likes the event e_p, where e_p corresponds to the element y_p in 3DM-3. Then, if y_p is included in the edge g_t (i.e., $y_p \in g_t$), the interest of u_p in the competing event c that appears in the interval t (which in 3DM-3 corresponds to edge g_t) is $\mu_{u_p,c} = 0.25(0.75 - \delta)/(0.25 + \delta)$ and 0.75, otherwise. $(7c)$ Each user $u_2 \in U_2$ likes only one event $e_2 \in E_2$ (as a result, each e_2 is liked only by one user u_2), with $\mu_{u_2,e_2} = 0.75$. $(7d)$ For each competing event c and user $u_2 \in U_2$, we have $\mu_{u_2,c} = 0$.

We can verify that, for each matching in 3DM-3, we can obtain a schedule in SES with total utility $3(0.25 + \delta)$, by assigning 3 events of E_1 in a same interval. Then, if 3DM-3 has a matching of size n, we can verify that the total utility in SES is $3n(0.25 + \delta) + m - n$. Otherwise, if every matching has size at most $(1 - \epsilon_0)n$, the total utility in SES is $1 - \frac{\epsilon_0 - 12\delta\epsilon_0}{12\delta + 3} < 1 - \frac{1}{3}\epsilon_0$. ■

[3] Several of the problem's involved factors (e.g., user interest, activity and attendance probability) can be computed using event-based mining methods, e.g., [5, 7, 13, 19, 21, 36, 37, 40, 41]. However, this is beyond the scope of this work.

[4] Due to lack of space, we only include proof sketches, while in simple cases, the proof sketch is also omitted.

3 ALGORITHMS

SES is known to be strongly NP-hard, even in highly restricted instances [4]. Due to its hardness, it is computationally prohibitive to find an optimal solution even in small problem sizes. Particularly, in the worst case, we have to enumerate an exponential number of possible assignments, where each assignment requires always $|\mathcal{U}|$ computations. For example, the greedy algorithm proposed in [4], in several cases in our experiments, took more than 5 hours to solve the problem in the default parameters setting, while more than 31 hours in larger settings. To this end, to cope with the hardness of the SES problem we design three efficient and scalable approximation algorithms which perform about half the computations and, in several cases, are 3-5 times faster compared to the method proposed in [4].

3.1 Existing Solution

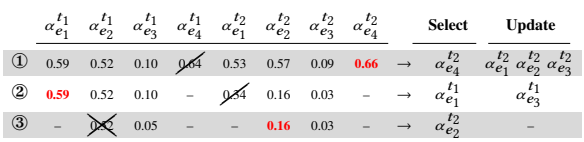
Here, we outline the previously proposed algorithm. First, we define the assignment score. Given a schedule $\mathcal{S}$ and an assignment α_r^t, as ***assignment score*** (also referred as ***score***) of an assignment α_r^t, denoted as $\alpha_r^t.S$, we define the *gain* in the expected attendance by including α_r^t in $\mathcal{S}$. The assignment score (based on Eq. 2) is defined as:

$$\alpha_r^t.S = \sum_{\substack{\forall e_j \in \\ \mathcal{E}_t(\mathcal{S}) \cup \{r\}}} \omega'^t_{e_j} - \sum_{\substack{\forall e_i \in \\ \mathcal{E}_t(\mathcal{S})}} \omega^t_{e_i} \quad (4)$$

Given a set of assignments, the term ***top assignment*** refers to the assignment with the largest score.

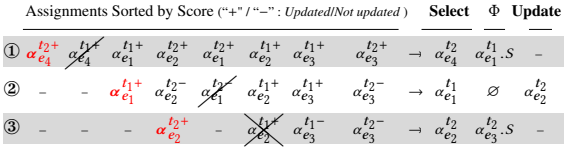
In [4], a simple greedy algorithm is outlined, referred here as ALG. This method starts by initially generating assignments between all pairs of events and intervals. Then, in each iteration, the assignment with the largest score (i.e., top assignment) is selected. After selecting an assignment, a subset of the assignment's scores need to be updated. Recall that, the assignment's score is defined w.r.t. the events assigned in the assignment's interval (Eq. 4). Hence, when an assignment α_e^t is selected, then the scores of the assignments referring to interval t need to be recomputed (updated). The time complexity of ALG is $O(|\mathcal{U}||C| + |\mathcal{E}||\mathcal{T}||\mathcal{U}| + k|\mathcal{E}||\mathcal{T}| + k|\mathcal{E}||\mathcal{U}| - k^2|\mathcal{T}| - k^2|\mathcal{U}|)$; and the space complexity is $O(|\mathcal{E}||\mathcal{T}|)$.

Example 2. [*ALG Algorithm*] Based on our running example, Figure 2 outlines the execution of the ALG algorithm. In this, as well as in the rest of the examples, we assume that $k = 3$. That is, three out of four events have to be scheduled. Each row represents the selection of a single assignment. Rows include the assignment scores (Eq. 4), as well as the selected assignment (presented in bold red



	$\alpha_{e_1}^{t_1}$	$\alpha_{e_2}^{t_1}$	$\alpha_{e_3}^{t_1}$	$\alpha_{e_4}^{t_1}$	$\alpha_{e_1}^{t_2}$	$\alpha_{e_2}^{t_2}$	$\alpha_{e_3}^{t_2}$	$\alpha_{e_4}^{t_2}$		Select	Update
①	0.59	0.52	0.10	~~0.64~~	0.53	0.57	0.09	**0.66**	→	$\alpha_{e_4}^{t_2}$	$\alpha_{e_1}^{t_2}$ $\alpha_{e_2}^{t_2}$ $\alpha_{e_3}^{t_2}$
②	**0.59**	0.52	0.10	–	~~0.64~~	0.16	0.03	–	→	$\alpha_{e_1}^{t_1}$	$\alpha_{e_3}^{t_1}$
③	–	~~0.52~~	0.05	–	–	**0.16**	0.03	–	→	$\alpha_{e_2}^{t_2}$	–

Figure 2: ALG algorithm example



	Assignments Sorted by Score ("+" / "–" : *Updated/Not updated*)									Select	Φ	Update
①	$\alpha_{e_4}^{t_2+}$	~~$\alpha_{e_4}^{t_1+}$~~	$\alpha_{e_1}^{t_1+}$	$\alpha_{e_2}^{t_2+}$	$\alpha_{e_1}^{t_2+}$	$\alpha_{e_2}^{t_1+}$	$\alpha_{e_3}^{t_1+}$	$\alpha_{e_3}^{t_2+}$	→	$\alpha_{e_4}^{t_2}$	$\alpha_{e_1}^{t_1}.S$	–
②	–	–	$\alpha_{e_1}^{t_1+}$	$\alpha_{e_2}^{t_2-}$	~~$\alpha_{e_1}^{t_2+}$~~	$\alpha_{e_2}^{t_1+}$	$\alpha_{e_3}^{t_1+}$	$\alpha_{e_3}^{t_2-}$	→	$\alpha_{e_1}^{t_1}$	∅	$\alpha_{e_2}^{t_2}$
③	–	–	–	$\alpha_{e_2}^{t_2+}$	–	~~$\alpha_{e_2}^{t_1+}$~~	$\alpha_{e_3}^{t_1-}$	$\alpha_{e_3}^{t_2-}$	→	$\alpha_{e_2}^{t_2}$	$\alpha_{e_3}^{t_2}.S$	–

Figure 3: Incremental updating scheme example

font) and the assignments that have to be updated after the selection. Initially (i.e., ① selection), the algorithm selects the assignment with the largest score (i.e., $\alpha_{e_4}^{t_2}$). Thus, after this selection the assignments referring to e_4 have to be omitted (marked with /), and the assignments referring to t_2 have to be updated. After the second selection, the algorithm has to update only $\alpha_{e_3}^{t_1}$ since $\alpha_{e_2}^{t_1}$ is no longer feasible (marked with ×) due to location constraint. Note that, for the sake of simplicity, the resources constraint has been omitted from the running example. Finally, the schedule contains $\alpha_{e_4}^{t_2}$, $\alpha_{e_1}^{t_1}$ and $\alpha_{e_2}^{t_2}$.

3.2 Incremental Updating Algorithm (INC)

The ALG algorithm proposed in [4], has the following shortcoming: (1) each time ALG selects an assignment, it has to recompute (i.e., update) from scratch all the scores for all the assignments associated with the selected assignment's interval. This process is referred to assignment updating or simply as updates; and (2) in each step, ALG has to examine (and traverse) all the available assignments in order to perform its tasks (e.g., select assignment, perform updates).

Considering the aforementioned issues, we design the *Incremental Updating algorithm* (INC). Regarding the first issue, INC exploits an *incremental updating scheme*, performing incremental assignment updates. Incremental updating allows INC, to provide the same solution as ALG, while, in each step, INC performs only a part of the updates (i.e., score computations). Regarding the second issue, INC attempts to reduce the number of assignments that should be examined in each step, i.e., search space. To this end, we devise an *assignment organization* that takes into account the incremental updating scheme. In several cases in our experiments, INC *is more than three times faster* than the existing algorithm.

Essentially, INC follows a similar assignment selection process to ALG, selecting the top assignment in each step, in a greedy fashion. However, in INC the assignments' update process has been designed based on the introduced incremental updating scheme.

3.2.1 Incremental Updating

In the proposed scheme, the updates are computed in an incremental manner, where after each assignment selection only a part of the updates are performed. As a result, during the algorithm execution, some of the assignments may not be up-to-date.

An assignment is denoted as ***updated***, if its score has been computed by considering all the (previously) selected assignments, and ***not updated*** otherwise. In analogy, a set of assignment is referred as *updated*, when all its assignments are updated, and ***partially updated***, otherwise.

The basic idea of our scheme is that we can determine a subset of the not updated assignments that have to be updated before each selection. First we show that, from the available assignments $\mathcal{A}$, we can find a set $\mathcal{B} \subseteq \mathcal{A}$ which includes the next algorithm selection χ. Then, we also show that the not updated assignments included in $\mathcal{B}$ are the only not updated assignments that have to be updated in order to find χ.

In order to specify $\mathcal{B}$, we use a numeric *bound* Φ. As shown next, the value of Φ is the score of the *top*, *updated* and *valid* assignment of $\mathcal{A}$.

Proposition 1. Let Φ be the score of the *top*, *updated* and *valid* assignment of the available assignments $\mathcal{A}$. Then, the next selected assignment χ is one of the assignments that in $\mathcal{A}$ have score larger or equal to Φ; i.e., $\chi \in \mathcal{B}$, where $\mathcal{B} = \{\alpha_e^t \in \mathcal{A} \mid a_e^t.S \geq \Phi\}$.

PROOF SKETCH. First, we show that the score of a not updated assignment is always larger or equal to the score of the assignment resulted by its update. Note that the proof for this is not trivial for arbitrary numbers of candidate and competing events. Based on the aforementioned, the not updated assignments of $\mathcal{A}$, having score lower than Φ, also have score lower than Φ in $\mathcal{A}'$, where $\mathcal{A}'$ be the set of assignments resulting from $\mathcal{A}$ by updating its not updated assignments. Further, the score of each updated assignment of $\mathcal{A}$ remains the same in $\mathcal{A}'$. So, both the updated and the not updated assignments of $\mathcal{A}$ have scores lower than Φ; their scores in $\mathcal{A}'$ also remain lower than Φ. Thus, the Proposition 1 holds. ■

Based on Proposition 1, since χ is included in $\mathcal{B}$, we can easily verify that, χ is the *top* assignment of $\mathcal{B}'$, where $\mathcal{B}'$ results from $\mathcal{B}$ by updating its not updated assignments. Thus, in order to find χ, we have to update the not updated assignments of $\mathcal{B}$. Based on the aforementioned, the following corollary describes the incremental updating process.

Corollary 1. In each step, in order to select the next assignment, only the not updated assignments having score larger or equal to Φ have to be updated.

Example 3. [*Incremental Updating Scheme*] Figure 3 illustrates the utilization of the incremental updating scheme. For clarity of presentation, we omit the assignment scores since these are the same as in Example 2. To better understand the procedure, in each row the assignments are presented in descending order, based on their score. The +/- notation is used to denote that the assignment is *updated*, or *not updated*, respectively. After the first selection, Φ is equal to $\alpha_{e_1}^{t_1}.S$ (i.e., top, updated and valid assignment), and all the assignments referring to t_2 change to *not updated*. Further, since all the not updated assignments have score lower than Φ, none of the assignments have to be updated. Then (② selection), after selecting $\alpha_{e_1}^{t_1}$, all the assignments become not updated; so Φ becomes unavailable. Next, the algorithm updates $\alpha_{e_2}^{t_2}$ and sets Φ equal to its score (0.16). In the last selection, since the current Φ is larger than the scores (0.10 and 0.9) of the not updated assignments $\alpha_{e_3}^{t_1}$ and $\alpha_{e_3}^{t_2}$, the algorithm does not have to update it. Compared to the ALG algorithm (Example 2) which performs four updates, our scheme performs only one.

3.2.2 Assignments Organization over Incremental Updating

In each step, the algorithm needs to examine and traverse all the available assignments, in order to perform the following main tasks: (1) select the top assignment; (2) perform updates; and (3) maintain

the bound. In order to accomplish these tasks, for each of the available assignments, the algorithm has to perform numerous computations. Indicatively, it has to check validity constraints, compare scores, consider bounds and possibly compute the new assignment score, etc.

Given the above, we introduce an *interval-based assignment organization* that incorporates with our incremental updating scheme. This organization attempts to reduce the number of assignments that are accessed and examined, i.e., *search space*. Using our organization, in most cases in our experiments, INC *examines slightly more than half assignments* compared to the existing algorithm.

Search Space Reduction in Assignment Updates. Here, we describe how we reduce the assignments that should be examined in order to perform the updates. An interval-based assignment organization allows to access at the interval-level, only the assignments that should be examined for update. Adopting this organization in a simple (not incremental) updating process, like in ALG, in each step, the algorithm needs only to access the assignments of one interval in order to performs the updates. On the other hand, in the incremental updating setting, several intervals become partially updated during the execution. In this scenario, in order to identify the assignments that need to be updated, we have to examine all the assignments included in partially updated intervals. As a result, in our setting, a simple interval-based organization will not be effective, since it will allow to skip accessing only the updated intervals.

Beyond ignoring only the updated intervals, in order to further reduce the search space, we have to be able to identify (and skip) the partially updated intervals whose assignments are not going to be updated. In our organization scheme, this is addressed by defining a score over each interval. Particularly, for each interval $t \in \mathcal{T}$, a value M_t is defined, where M_t is equal to the score of the *top*, *updated* and *valid* assignment of interval t. Exploiting M_t, we can directly identify the partially updated intervals that have to be accessed through the updating process. Particularly, it is easy to verify that we have to access all the partially updated intervals $t \in \mathcal{T}$ for which $M_t \leq \Phi$.

Search Space Reduction in Assignment Selection and Bound Maintenance. The organization described so far allows to reduce the search space during the assignment updating process. However, after performing the updates, the algorithm has to accomplish also the tasks of selecting the next assignment and maintaining the bound Φ, which in turn enforce the examination of all the available assignments. In what follows, we show how to perform all the tasks without examining any further assignments beyond the ones examined during the updating process.

The intuition is that, in each step, only a subset of the assignments is updated, while the rest remain the same as in the previous step, referred here as *static*. Therefore, it is reasonable to assume that the algorithm is able to accomplish all of its tasks by utilizing "information" previously captured from the static assignments $\mathcal{W}$. So, if this "information" is known, then, after performing the updates, we can ignore $\mathcal{W}$ (i.e., avoid access). As shown next, this "information" can be captured by two numeric values I_χ and I_Φ determined from $\mathcal{W}$. Briefly, I_χ is exploited to specify the next selected assignment χ, and I_Φ to compute the new Φ.

Proposition 2. Given a set of static assignments $\mathcal{W}$. Let I_χ and I_Φ be the scores of the *first* and the *second larger top, updated* and *valid* assignment of $\mathcal{W}$, respectively. Then, if I_χ and I_Φ are know, the algorithm can ignore $\mathcal{W}$.

Algorithm 1. INC (k, $\mathcal{T}$, $\mathcal{E}$, $\mathcal{C}$, $\mathcal{U}$)

Input: k: number of scheduled events; $\mathcal{T}$: time intervals; $\mathcal{E}$: candidate events; $\mathcal{C}$: competing events; $\mathcal{U}$: users;
Output: $\mathcal{S}$: feasible schedule containing k assignments
Variables: $\mathcal{L}_i$: assignment list for interval i; Φ: bound; $\mathcal{M}$: top, valid and updated assign list

1 $\mathcal{S} \leftarrow \varnothing$; $\mathcal{M} \leftarrow \varnothing$; $\mathcal{L}_i \leftarrow \varnothing$, $\mathcal{L}_i.U \leftarrow$ `updated` $1 \leq i \leq |\mathcal{T}|$
2 **foreach** $(e, t) \in \mathcal{E} \times \mathcal{T}$ **do** //generate assignments
3 compute $\alpha_e^t.S$; $\alpha_e^t.U \leftarrow$ `updated`; //by Eq. 4
4 insert α_e^t into $\mathcal{L}_t$ //initialize assignment lists
5 $\mathcal{M}_{[t]} \leftarrow$ getBetterAssgn($\mathcal{M}_{[t]}$, α_e^t) //initialize $\mathcal{M}$ with the top assignment from each interval
6 **while** $|\mathcal{S}| < k$ **do**
7 $\alpha_{e_p}^{t_p} \leftarrow$ getTopAssgn($\mathcal{M}$) //select the top, valid & updated assignment
8 insert $\alpha_{e_p}^{t_p}$ into $\mathcal{S}$ //insert into schedule
9 remove $\alpha_{e_p}^{t_p}$ from $\mathcal{L}_{t_p}$; $\mathcal{L}_{t_p}.U \leftarrow$ `prtl updated`; //update $\mathcal{L}_{t_p}$ status
10 $\alpha_e^t.U \leftarrow$ `prtl updated`, $\forall \alpha_e^t \in \mathcal{L}_{t_p}$
11 **foreach** $\alpha_e^t \in \mathcal{M}$ **do** //update $\mathcal{M}$ list based on selected assignment
12 **if** $t = t_p$ **then**
13 $\alpha_e^t \leftarrow \varnothing$ //i.e., $\alpha_e^t.S \leftarrow -\infty$
14 **else if** $e = e_p$ **then**
15 $\alpha_e^t \leftarrow$ getTopAssgn($\mathcal{L}_i$)
16 $\Phi \leftarrow$ score of top $\mathcal{M}$ //set bound
17 **for** $i = 1$ **to** $|\mathcal{T}|$ **do** //update assignments
18 **if** $\mathcal{L}_i.U =$ `prtl updated` **and** $\mathcal{M}_{[i]}.S \leq \Phi$ **then** //check for updates
19 **foreach** $\alpha_e^t \in \mathcal{L}_i$ **do**
20 **if** α_e^t *is not* `valid` **then**
21 remove α_e^t from $\mathcal{L}_i$
22 **else if** $\alpha_e^t.U =$ `prtl updated` **and** $\alpha_e^t.S \geq \Phi$ **then**
23 compute new $\alpha_e^t.S$; $\alpha_e^t.U \leftarrow$ `updated`; //by Eq. 4
24 $\mathcal{M}_{[i]} \leftarrow$ getBetterAssgn($\mathcal{M}_{[i]}$, α_e^t) //update top assignment
25 $\Phi \leftarrow$ getBetterAssgn(Φ, $\alpha_e^t.S$) //update bound
26 **if** *all* $\alpha_e^t \in \mathcal{L}_i$ *is* `updated` **then** $\mathcal{L}_i.U =$ `updated`;
27 **return** $\mathcal{S}$

In our interval-based scheme implementation, the static assignments $\mathcal{W}$ correspond to a set of static intervals $\mathcal{T}_\mathcal{W} \subseteq \mathcal{T}$. Both I_χ and I_Φ can be be directly computed based on the values of M_t of the static intervals $\mathcal{T}_\mathcal{W}$. Particularly, $I_\chi = \max_{\forall t \in \mathcal{T}_\mathcal{W}} M_t$ and $I_\phi = \max_{\forall t \in \{\mathcal{T}_\mathcal{W} \setminus t_\chi\}} M_t$, where t_χ is the interval of I_χ. Therefore, based on Proposition 2, in each step, the algorithm needs to access only intervals that have been updated (i.e., subset of partially updated intervals).

Assignment Organization Summary. To sum up, the presented organization allows: (1) the reduction of the assignments that are examined during the updating process; and (2) skipping the examination of any further assignments beyond the updated ones.

3.2.3 INC Algorithm Description & Analysis

Algorithm Description. Algorithm 1 describes the INC algorithm; INC receives the same inputs as ALG. Additionally, INC employs $|\mathcal{T}|$ lists, with each list $\mathcal{L}_i$ filled with the assignments of interval i. Further, each assignment α_e^t and list $\mathcal{L}_i$, use a flag U (denoted as $\alpha_e^t.U$) to define its update status. Finally, the algorithm uses a list $\mathcal{M}$ that holds the top, valid and updated assignments of each interval. Initially, like ALG, INC calculates the scores for all possible assignments (*loop in line* 2). At the same time, the assignments are inserted into the corresponding list $\mathcal{L}_i$ (*line* 3-4). Note that, the *getBetterAssgn* function returns the assignment with the larger score.

Then, at the beginning of each iteration (*line* 6), the algorithm selects the top assignment from $\mathcal{M}$, and inserts it into schedule (*lines* 7-8). Also, the algorithm has to revise the information related to update status (*lines* 9-10). After the assignment's selection phase, the algorithm performs score updates. Initially, the bound Φ is defined by the top updated assignment (*line* 16). Then, the algorithm traverses the lists $\mathcal{L}_i$, using as upper bound the $\mathcal{M}_{[i]}.S$, to identify the lists that have to be checked for updates (*line* 18). From the verified lists, the

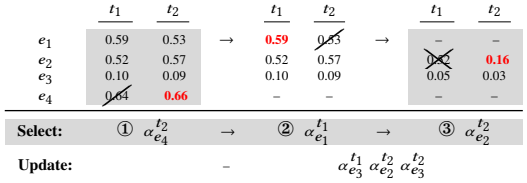


Figure 4: HOR algorithm example

algorithm performs incremental updates (*lines* 22-23), updating also $\mathcal{M}$ and Φ (*lines* 24-25).

INC vs. ALG Solution. The following proposition states that both INC and ALG, return the same solutions.

Proposition 3. INC and ALG always return the same solution.

Complexity Analysis. The cost in the first loop (*line* 2) is $O(|\mathcal{E}||\mathcal{T}||\mathcal{U}|)$. Note that, each assignment score (Eq. 4) is computed in $O(|\mathcal{U}|)$. Then, the second loop (*line* 6) performs k iterations. The overall cost for the getTopAssgn operation (*line* 7) is $O(\sum_{i=0}^{k-1}|\mathcal{T}|)$. Further, the loop in *line* 11 performs $|\mathcal{T}|$ iterations, whereas in the worst case, in $|\mathcal{T}|-1$ of these iterations, INC performs a getTopAssgn operation which costs $O(|\mathcal{E}|-(i+1))$, with $0 \le i \le k-1$. Thus, the overall cost for this getTopAssgn operation is $O(\sum_{i=0}^{k-1}(|\mathcal{T}|-1)(|\mathcal{E}|-i-1))$. Next (*line* 17), in the worst case, all the not updated assignments are updated (same as ALG). Note that, in the *Best case*, INC does not perform any computations for assignment updates, while in ALG, in every case, the cost for updates is $O(|\mathcal{U}|\sum_{i=0}^{k-2}|\mathcal{E}|-i-1)$.

Hence, in the worst case, the overall cost of INC is same as ALG; i.e., $O(|\mathcal{U}||\mathcal{C}|+|\mathcal{E}||\mathcal{T}||\mathcal{U}|+k|\mathcal{E}||\mathcal{T}|+k|\mathcal{E}||\mathcal{U}|-k^2|\mathcal{T}|-k^2|\mathcal{U}|)$. Finally, the space complexity is $O(|\mathcal{E}||\mathcal{T}|+|\mathcal{T}|)$.

3.3 Horizontal Assignment Algorithm (HOR)

In this section we propose the *Horizontal Assignment algorithm* (HOR), which in general, is more efficient than the ALG and INC algorithms and in most cases in practice provides same solutions. The goal of HOR is twofold. First, to reduce the number of updates by performing only a part of the required updates; and, at the same time, minimize the impact of not regular updates, in the solution quality. In HOR both of these issues are realized by the policy that are employed to select the assignments. In our experiments, in several cases HOR *is around 5 and 3 times faster* than ALG and INC, respectively. Also, in more than 70% of our experiments, HOR reports the same solution as ALG, while in the rest cases the difference is marginal.

Horizontal Selection Policy. The key idea of HOR is that it adopts a selection policy, named *horizontal selection policy*, that selects assignments in a "horizontal" fashion. In this policy, in each iteration the algorithm selects a set of assignments consisting of one assignment from each interval. Particularly, the top assignment from each interval is selected. This way, essentially, a layer of assignments is generated in each iteration. For example, consider the scenario where $k > |\mathcal{T}|$ (and the assignments are feasible in all cases). In the first iteration, HOR will assign one event in each interval; equally, in the n^{th} iteration, n events will have been assigned in each interval.[5]

This policy allows HOR to avoid performing updates after each assignment selection. This holds, since, in each iteration at most one assignment per interval is selected. Thus, during an iteration, when an assignment is selected for an interval, the algorithm stops examining the selection of further assignments for this interval. As a result, there is no need to perform any updates until the end of each iteration, where the scores for all the assignments have to recomputed.

In what follows we outline the intuition behind the horizontal selection policy. Considering that the users' attendance is shared between the events that take place during the same or overlapping time intervals. The horizontal policy assigns the same number of events to each interval, ignoring the possibility that it may be preferable to assign a different number of events to some intervals.

[5]With an exception in the last iteration l, in which if $k \bmod |\mathcal{T}| \neq 0$, then, $|\mathcal{T}| - (k \bmod |\mathcal{T}|)$ intervals will have $l-1$ events.

Algorithm 2. HOR (k, $\mathcal{T}$, $\mathcal{E}$, $\mathcal{C}$, $\mathcal{U}$)

Input: k: number of scheduled events; $\mathcal{T}$: time intervals; $\mathcal{E}$: candidate events; $\mathcal{C}$: competing events; $\mathcal{U}$: users;
Output: $\mathcal{S}$: feasible schedule containing k assignments
Variables: $\mathcal{L}_i$: assignment lists for interval i; $\mathcal{M}$: top assignments list

1 $\mathcal{S} \leftarrow \varnothing$; $\mathcal{M} \leftarrow \varnothing$;
2 **while** $|\mathcal{S}| < k$ **do**
3 $\mathcal{L}_i \leftarrow \varnothing$; $1 \le i \le |\mathcal{T}|$
4 **foreach** $(e,t) \in \{\mathcal{E} \setminus \mathcal{E}(\mathcal{S})\} \times \mathcal{T}$ **do** //generate assignments
5 **if** α_e^t *is* `valid` **then**
6 compute $\alpha_e^t.S$; //by Eq. 4
7 insert α_e^t into $\mathcal{L}_t$ //initialize assignment lists
8 $\mathcal{M}_{[t]} \leftarrow$ getBetterAssgn($\mathcal{M}_{[t]}, \alpha_e^t$) //insert into $\mathcal{M}$ the top assignment from each interval
9 **while** $\mathcal{M} \neq \varnothing$ **and** $|\mathcal{S}| < k$ **do** //select assignments from $\mathcal{M}$
10 $\alpha_{e_p}^{t_p} \leftarrow$ popTopAssgn($\mathcal{M}$)
11 **if** $e_p \notin \mathcal{S}$ **then**
12 insert $\alpha_{e_p}^{t_p}$ into $\mathcal{S}$ //insert into schedule
13 **else** //assignment is invalid; select the top valid from $\mathcal{L}_{t_p}$ and insert it into $\mathcal{M}$
14 insert the top assignment $\alpha_{e_i}^{t_p}$ from $\mathcal{L}_{t_p}$ into $\mathcal{M}$, s.t. $e_i \notin \mathcal{S}$
15 **return** $\mathcal{S}$

Example 4. [*HOR Algorithm*] Figure 4 outlines the execution of the HOR algorithm, presenting assignments following an interval-based organization. Initially, HOR selects the assignment with the largest score. Since the first selected assignment refers to t_2, in the next selection, HOR will select the top assignment from t_1. After selecting assignments from both intervals, HOR has to update all the available assignments in order to perform the third selection. Therefore, HOR performs three updates, whereas it finds the same schedule as ALG/ INC.

Algorithm Description. Algorithm 2 presents the pseudocode of HOR. Note that, since the horizontal selection policy performs selections at the interval-level, we implement interval-based assignment organization. Finally, similarly to INC, HOR uses the $|\mathcal{T}|$ lists $\mathcal{L}_i$ and the list $\mathcal{M}$. At the beginning of each iteration the algorithm calculates the scores for all possible assignments (*loop in line* 4) and initializes $\mathcal{M}$ (*line* 8). In the next phase (*line* 9), the algorithm selects the assignments based on $\mathcal{M}$. Particularly, in each step, the top valid assignment from $\mathcal{M}$ is selected.

3.3.1 HOR Algorithm Analysis

ALG vs. HOR Score Computations Analysis. Here we study the number of score computations, comparing the ALG and the HOR algorithms. The following proposition specifies the cases where HOR performs less score computations than ALG.

Proposition 4. HOR performs less score computations than ALG when $k \le |\mathcal{T}|$ or $|\mathcal{E}| < \frac{k}{2}(3|\mathcal{T}|+1)$.

Proof Sketch. In case that $k \le |\mathcal{T}|$, HOR computes only the scores for the initial assignments (i.e., $|\mathcal{T}||\mathcal{E}|$) without performing any updates. In case that $k > |\mathcal{T}|$, HOR computes the same initial assignments, as well as the scores for $\sum_{i=0}^{(k/|\mathcal{T}|)-1}|\mathcal{T}|(|\mathcal{E}|-i|\mathcal{T}|-|\mathcal{T}|)$ updates. On the other hand, in the ALG algorithm, in any case, we

have to compute the same number of initial assignments as in HOR, as well as $k|\mathcal{E}| + \frac{k}{2} - \frac{k^2}{2} - |\mathcal{E}|$ updates. ■

From Proposition 4 we can infer that in "rational/typical" (i.e., real-world) scenarios, HOR perform fewer computations than ALG. Particularly, even in cases that $k > |\mathcal{T}|$, it should also hold that $|\mathcal{E}| \geq \frac{k}{2}(3|\mathcal{T}| + 1)$ in order for HOR to perform more computations. Considering the setting of our problem, the second relation is difficult to hold in practice. For example, consider the scenario where $|\mathcal{T}| = 10$ and $k = 20$. Then, in order for HOR to perform more computations, it should hold that $|\mathcal{E}| \geq 301$, which seems unrealistic due to the noticeable difference between the number of scheduled ($k = 10$) and candidate events ($|\mathcal{E}| \geq 301$).

Worst Case w.r.t. k and $|\mathcal{T}|$. Considering the horizontal selection policy, beyond the size of the input (e.g., $|\mathcal{E}|$, $|\mathcal{U}|$, k), the number of computations in HOR is also influenced by the ratio between parameters k and $|\mathcal{T}|$. During the execution, HOR performs $\lceil k/|\mathcal{T}| \rceil$ iterations. At the beginning of each iteration, it computes the scores according to which $|\mathcal{T}|$ assignments are selected. In the last iteration, if $k \bmod |\mathcal{T}| \neq 0$, then only $k \bmod |\mathcal{T}|$ assignments need to be selected, while the algorithm has already performed the computations that are required to select $|\mathcal{T}|$ assignments. Thus, in this case, HOR has performed more computations than the ones required for selecting its assignments. For example, assume that we have $|\mathcal{T}| = 10$ and $k = 11$. In this case, the score computations performed by HOR are the same as in the case that we have $k = 20$.

The case in which the difference between the number of computed assignment selections and the number of the selections that need to be performed is maximized, is referred as *worst case w.r.t. k and T*. Note that, even in the worst case, in our experiments, HOR outperforms INC in several cases.

Proposition 5. In HOR, the worst case w.r.t. k and $|\mathcal{T}|$ occurs when $k > |T|$ and $k \bmod |\mathcal{T}| = 1$.

Complexity Analysis. In the first while loop (*line* 2), HOR performs $\lceil k/|\mathcal{T}| \rceil$ iterations. In each iteration, in the worst case, there are $|\mathcal{E}| - i|\mathcal{T}|$ available events, where $0 \leq i \leq \frac{k}{|\mathcal{T}|}$. Thus, in each iteration, it computes $|\mathcal{T}|(|\mathcal{E}| - i|\mathcal{T}|)$ assignments (*line* 6). The overall cost for computing the assignments is $O(|\mathcal{U}| \sum_{i=0}^{k/|\mathcal{T}|} |\mathcal{T}|(|\mathcal{E}| - i|\mathcal{T}|))$. Further, in each iteration of the first loop (*line* 2), the nested loop (*line* 9) performs $|\mathcal{T}|$ iterations (referred as nested iterations). In each nested iteration, in the worst case, HOR performs two popTopAssgn operations (*lines* 10 & 14). The cost for the first and the second popTopAssgn operations is $O(|\mathcal{T}|)$ and $O(|\mathcal{E}| - i|\mathcal{T}|)$, respectively. Hence, in sum, the cost for popTopAssgn operations is $O(\sum_{i=0}^{k/|\mathcal{T}|} |\mathcal{T}|(|\mathcal{T}| + |\mathcal{E}| - i|\mathcal{T}|))$. The worst case can occur when all the assignments contained in $\mathcal{M}$ refer to the same event.

Thus, the overall cost of HOR is

$O(|\mathcal{U}||C|+|\mathcal{E}||\mathcal{T}||\mathcal{U}|+k|\mathcal{E}||\mathcal{U}|+|\mathcal{T}|^2-k|\mathcal{T}||\mathcal{U}|-k^2|\mathcal{U}|)$. Finally, the space complexity is $O(|\mathcal{E}||\mathcal{T}| + |\mathcal{T}|)$.

3.4 Horizontal Assignment with Incremental Updating Algorithm (HOR-I)

This section introduces the *Horizontal Assignment with Incremental Updating algorithm* (HOR-I). HOR-I combines the basic concepts from the INC and HOR algorithms, in order to further reduce the computations performed by HOR. Particularly, HOR-I adopts an incremental updating scheme, similar to INC (Sect. 3.2.1), as well as the horizontal selection policy employed by HOR (Sect. 3.3). Note that, in several cases, in our experiment HOR-I *performs about half computations and is up to two times faster compared to* HOR.

Algorithm 3. HOR-I (k, $\mathcal{T}$, $\mathcal{E}$, C, $\mathcal{U}$)

Input: k: number of scheduled events; $\mathcal{T}$: time intervals; $\mathcal{E}$: candidate events; C: competing events; $\mathcal{U}$: users;
Output: $\mathcal{S}$: feasible schedule containing k assignments
Variables: $\mathcal{L}_i$: assignment lists for interval i; Φ: bound; $\mathcal{M}$: top assignments list

```
1  S ← ∅;  M ← ∅;  L_i ← ∅  1 ≤ i ≤ |T|
2  while |S| < k do
3      if S = ∅ then                                              //first iteration
4          foreach (e, t) ∈ {E\E(S)} × T do                       //generate assignments
5              compute α_e^t.S;   α_e^t.U ← updated;              //by Eq. 4
6              insert α_e^t into L_t                              //initialize assignment lists
7              M_[t] ← getBetterAssgn(M_[t], α_e^t)
8      else                                                       //incremental assignments updating
9          for i = 1 to |T| do
10             Φ ← 0                                              //initialize bound
11             foreach α_e^t ∈ L_i do
12                 if α_e^t is valid then
13                     if α_e^t.S ≥ Φ then
14                         compute new α_e^t.S;   α_e^t.U ← updated;
15                         Φ ← getBetterAssgn(Φ, α_e^t.S)         //update bound
16                     else
17                         α_e^t.U ← prtl updated                 //partially updated
18                 else
19                     remove α_e^t from L_i
20             M_[i] ← Φ                                          //update top assignment
21     while M ≠ ∅ and |S| < k do                                 //select assignments from M
22         α_ep^tp ← popTopAssgn(M)
23         if e_p ∉ S then
24             insert α_ep^tp into S                              //insert into schedule
25         else                  //select the top, valid & updated from L_tp and insert it into M
26             α_ep^tp ← top & updated assignment from L_tp, s.t. e_p ∉ S
27             if α_ep^tp = ∅ and ∃α_e^t ∈ L_tp s.t. α_e^t is valid then
28                 .                                              //incremental updates in interval p
29                 .  Same as lines 10 to 20, with i = p
30                 .
31 return S
```

Recall that, at the beginning of each iteration, HOR calculates the scores for all available assignments. Particularly, in the first iteration, the algorithm generates the assignments and calculates their (initial) scores, while in each of the following iterations the scores for all the assignments are updated. On the other hand, after the first iteration, HOR-I instead of updating all the assignments, uses an incremental updating scheme. This way, in each iteration, a reduced number of updates are performed.

Note that since the updates are performed after the first iteration, it is obvious that HOR-I is identical to HOR in cases where only one iteration is required (i.e., $k \leq |\mathcal{T}|$).

Example 5. [*HOR-I Algorithm*] The difference between HOR-I and HOR example (Example 4), appears at the third selection, where from t_2 only the $\alpha_{e_2}^{t_2}$ is updated. This happens because after updating $\alpha_{e_2}^{t_2}$, its score (0.16) is the current bound for this interval. Then, when checking $\alpha_{e_3}^{t_2}$ for update, its score (0.09) is lower than the bound, so there is no need to update it. Hence, HOR-I performs two of the three updates performed by HOR.

Algorithm Description. Algorithm 3 presents HOR-I; HOR-I uses the same structures as HOR, as well as a bound Φ. At the first iteration (*loop in line* 4), as is the case with HOR, HOR-I generates the assignments and initializes $\mathcal{M}$. In the next iterations (*loop in line* 9), it performs incremental updates for each interval, determining a different bound Φ for each interval. Then, similarly to HOR, HOR-I performs the assignment selection based on $\mathcal{M}$ (*loop in line* 21). In contrast to HOR, HOR-I has also to examine the update status of the assignments. In case that there is not a valid and updated assignment

left on an interval (*lines* 27-30), HOR-I has to perform incremental updates in this interval.

3.4.1 HOR-I Algorithm Analysis

HOR-I Solution & Worst Case w.r.t. k and $|\mathcal{T}|$. The following propositions state that both HOR-I and HOR return the same solutions and also have the same worst case w.r.t. k and $|\mathcal{T}|$.

Proposition 6. HOR-I and HOR always return the same solution.

Proposition 7. In HOR-I, the worst case w.r.t. k and $|\mathcal{T}|$ occurs when $k > |T|$ and $k \bmod |\mathcal{T}| = 1$.

Complexity Analysis. In the worst case, the computation cost in HOR-I, is the same as HOR. Particularity, in the worst case, the bound employed by HOR-I cannot prevent any of the assignment updates (*line* 9). This case arises, when the assignments in each interval list $\mathcal{L}_i$ are sorted in ascending order by its score, and there are no assignments having the same score. Thus, the computation cost for HOR-I is $O(|\mathcal{U}||\mathcal{C}| + |\mathcal{E}||\mathcal{T}||\mathcal{U}| + k|\mathcal{E}||\mathcal{U}| + |\mathcal{T}|^2 - k|\mathcal{T}||\mathcal{U}| - k^2|\mathcal{U}|)$. Note that, in the *Best case* HOR-I does not perform any computations for updates, while in HOR, in any case (where $k > |\mathcal{T}|$), the cost for updates is $O(\sum_{i=0}^{(k/|\mathcal{T}|)-1} |\mathcal{U}||\mathcal{T}|(|\mathcal{E}| - i|\mathcal{T}| - |\mathcal{T}|))$. Finally, the space complexity is $O(|\mathcal{E}||\mathcal{T}| + |\mathcal{T}|)$.

4 EXPERIMENTAL ANALYSIS

4.1 Setup

Datasets. In our experimental evaluation we present the results from four datasets, *two real* and *two synthetic*. The *first real* is the *Meetup dataset* (Meetup) from [21], which contains data from California, and is the dataset used in [4]. We follow the same approach as in [4, 26–28, 31], in order to define the interest of a user to an event. After preprocessing, we have the Meetup dataset containing 42,444 users and about 16K events.

The next *real dataset* (Concerts) which is the largest, is related to music and provided from Yahoo! ("Music user ratings of musical tracks, albums, artists and genres dataset"). Concerts is used to demonstrate the scenario of music festival organization. Particularly, Concerts contains data for several music entities (i.e., tracks, albums, artists, genres), as well as ratings of users over these entities. In this dataset, we consider albums to represent the events (i.e., music concerts). We select the albums that are associated with at least one genre, which results to 89K albums. Further, as users we select the users that have rated at least 10 genres, which result to 379,391 users.

In order to compute user interest over the albums, we consider the users ratings over the music genres, as well as the genres that are associated with the music albums. Let a user u, R_u denote the set of ratings r_i over genres, where $r_i \in [0, 1]$ is the rating over the i genre. Also, let G_a be a set of genres associated with a music album a. Here, we define the interest of a user u over the album a as $(\sum_{\forall g \in G_a} r_g)/|G_a|$, where $r_g = 1$ if the genre g is not specified in R_u. Note that, similar results are reported using alternative methods, such as setting $r_g = 0$ for genres not specified in R_u, or considering only the common user-album genres.

Finally, regarding *synthetic datasets* (Table 1), we generate the users' *interest values* for the events, following the *three distribution types* examined in the related literature [12, 26–28, 31]: *Uniform* (Unf), *Normal* (Nrm) and *Zipfian* (Zip). Note that, for brevity, the results for the Normal distribution are not presented here since they are similar to Uniform. Further for Zipfian, we present only the results with parameter equal to 2 which are similar to those of 1 and 3.

Table 1: Parameters

Description (Parameter)	Values
Synthetic & Real Datasets	
Num of scheduled events (k)	50, 70, **100**, 200, 500
Num of candidate events ($\|\mathcal{E}\|$)	k, $\mathbf{2k}$, $3k$, $5k$, $10k$
Num of time intervals ($\|\mathcal{T}\|$)	$\frac{k}{5}$, $\frac{k}{2}$, k, $\frac{3k}{2}$, $2k$, $3k$
Competing events per interval	Uniform: [1, 4], [1, 8], **[1, 16]**, [1, 32], [1, 64]
Num of available locations	5, 10, **25**, 50, 70
Num of available resources (θ)	10, **20**, 30, 50, 100
Num of required resources per event (ξ_e)	Uniform: [1, $\frac{\theta}{4}$], **[1, $\frac{\theta}{3}$]**, [1, $\frac{\theta}{2}$], [1, $\frac{3\theta}{4}$], [1, θ]
Distribution of social activity probability (σ_u^t)	**Uniform**, Normal (0.5, 0.25)
Synthetic Datasets	
Num of users ($\|\mathcal{U}\|$)	10K, **50K**, 100K, 500K, 1M
Distribution of interest ($\mu_{u,e}$)	Uniform, Normal (0.5, 0.25), Zipfian: 1, 2, 3

Parameters. Table 1 summarizes the parameters that we vary and the range of values examined; default values are presented in bold.

Adopting the same setting as in the related works [4, 12, 26–28, 31], we set the the default and maximum value of the of *scheduled events* k, to 100 and 500, respectively. In order to select the values for the number of *competing events per interval*, we analyze the two Meetup datasets used in our evaluation [21]. Particularly, we are interested in the number of events taking place during overlapped time intervals. As event interval we consider the period spanning from one hour before to two hours after the event's scheduled time. From the analysis, we found that, on average, 8.1 events are taking place during overlapping intervals. Therefore, in the default setting the number of competing events per interval is selected by a uniform distribution having 8.1 as mean value. Further, we vary the mean value from 2 to 32 (Table 1). In our experiments, the reported results are similar to the default setting, with the utility score being slightly lower for larger numbers of competing events, as expected (results are omitted due to lack of space).

In order to select the default and the examined values for the *number of available events' locations*, we consider the percentage of pairs of events that are spatio-temporally conflicting, as specified in [26]. Also, we vary the *number of required resources* for each event, as well as the number of *available resources* (Table 1). Here, as resources we consider agents (i.e., organizer's staff). In the aforementioned experiment, the methods are marginally affected by the examined parameters. Thus, due to lack of space, the results are omitted. Finally, regarding the *social activity probability*, we use Uniform and Normal distribution. Note that, the results for Normal distribution are not presented here, since they are the same as in Uniform.

Methods. In our evaluation we study the three proposed algorithms (INC, HOR, HOR-I), as well as the ALG algorithm proposed in [4]. Further, we include the baselines used in [4]. The first, denoted as TOP, computes the assignment scores for all the events and selects the events with top-k score values. Since TOP computes the scores only once, TOP is always performing the minimum number of computations. The second, RAND assigns events to intervals, randomly. Note that, since the objective, the solution and the setting of our problem are substantially different (see Sect. 1) from the related works [6, 12, 26–29, 31], the existing methods cannot be used to solve the SES problem.

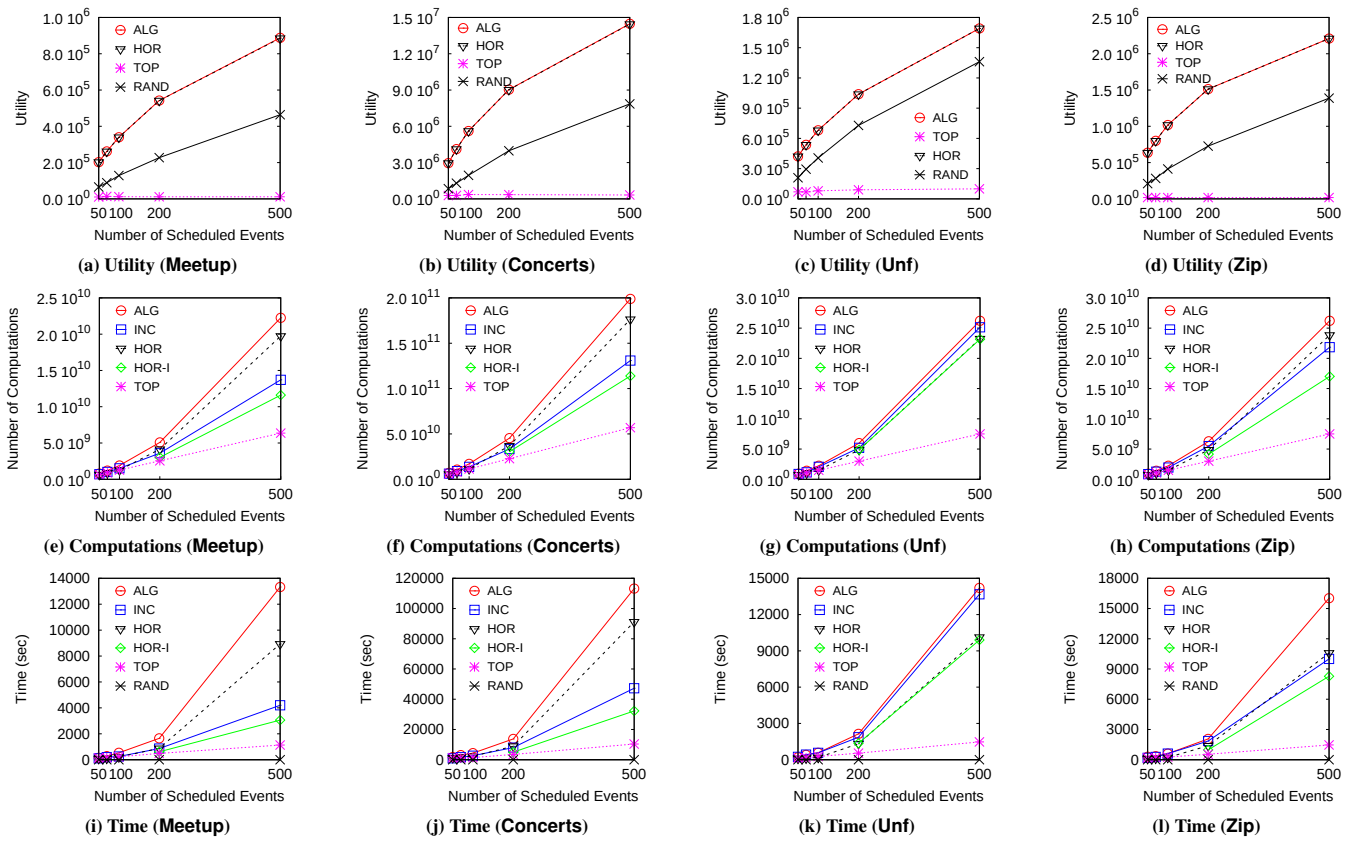


Figure 5: Varying the number of scheduled events k

Metrics & Implementation. In each experiment, we measure: (1) the *total utility score*; (2) the *execution time*; and (3) the *number of computations* for assignment scores ($|\mathcal{U}|$ per assignment score). All algorithms were written in C++ and the experiments were performed on an 2.67GHz Intel Xeon E5640 with 32GB of RAM.

4.2 Results

Recall that, the HOR and HOR-I algorithms return the same solutions (i.e., equal utilities); the same also holds for the ALG and INC algorithms. Hence, only the former utility plots are presented. Further, in cases where $k < |\mathcal{T}|$, the HOR-I algorithm is identical to HOR (Sect. 3.4); thus, in these cases only the HOR results are included in the plots.

4.2.1 Effect of the Number of Scheduled Events

In the first experiment, we study the effect of varying the number of scheduled events k.

Utility. In terms of utility (Fig. 5a–5d), we observe that, in all cases, our HOR method has the same utility score as the ALG (details are presented in Sect. 4.2.8). Further, the difference between RAND and the other methods increases, as k increases. This is reasonable considering the fact that the larger the k, the larger the number of "better", compared to random, selected assignments.

Regarding the Unf dataset (Fig. 5c), we observe the following. First, the difference between the random and the other methods is the smallest one, compared to the other datasets. Second, the difference between the methods is roughly the same for all k values. The reason for the aforementioned is that the uniform distribution results to utility values being very close, for all assignments. Thus, an effective assignment selection cannot significantly improve the overall utility.

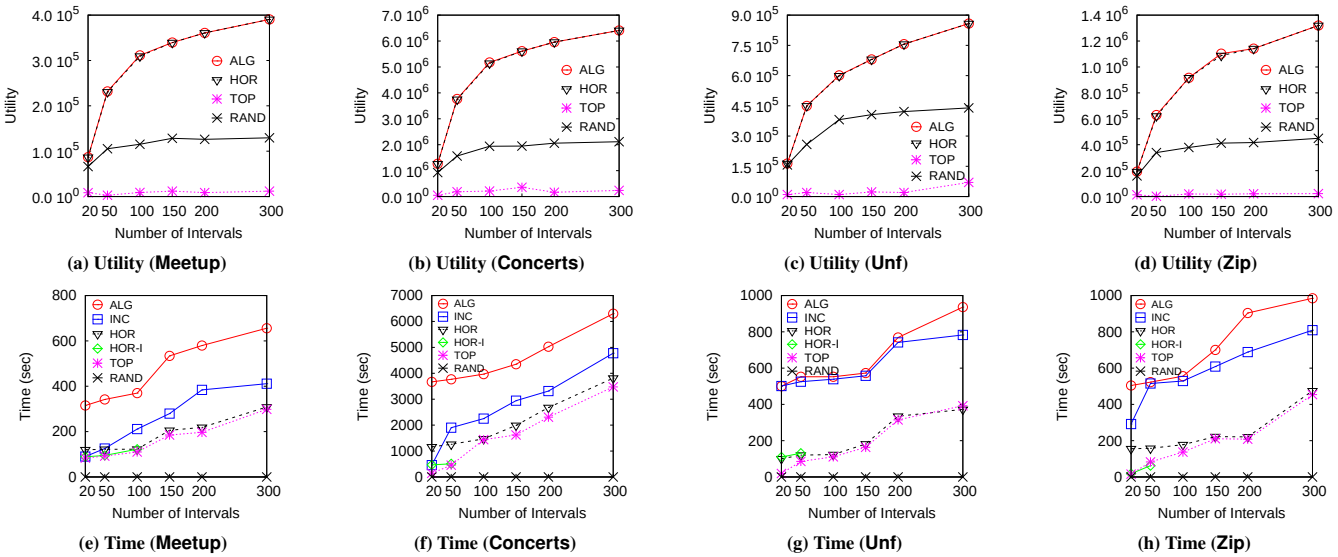
Finally, we can observe that TOP reports considerably low utility scores in all cases (which is also observed in the following experiments). The reason is that TOP assigns the events to a small number of intervals. This results to a large number of parallel events which "share" assigned interval's utility.

Computations. Regarding the number of computations (Fig. 5e–5h), we should mention that, the computations that are performed due to updates increases with k, while the number of initially computed scores is the same for all k. Thus, the difference between the ALG and our methods increases with k. Overall, we can observe that, in all cases, ALG reports the larger number of computations, while HOR-I the lower (excluding the TOP baseline).

Regarding our methods, comparing HOR with HOR-I, we can observe that the difference between our HOR versions increases with k, with HOR-I performing noticeably less computations compared to HOR for large k. An exception is reported in Unf dataset (Fig. 5g), in which all bound-based methods (INC, HOR-I) report poor performance (as explained later).

Further, comparing the HOR with INC, for $k < 200$, HOR performs less computations than INC. However, in the remaining cases where $k \geq 200$, INC outperforms HOR (with an exception in Unf). The reason why INC performs better than HOR for $k \geq 200$ is that, in these cases HOR performs update computations; while, for the cases where $k \leq |\mathcal{T}|$ only the initial computations are performed.

In the Unf dataset (Fig. 5g), we can observe that the bound-based methods (i.e., INC, HOR-I) demonstrate poor performance, with HOR-I performing same as HOR, and INC performing worse than both of them. The reason lies to the uniform distribution, where, as previously stated, the scores are very close for all assignments. As



(a) Utility (**Meetup**) (b) Utility (**Concerts**) (c) Utility (**Unf**) (d) Utility (**Zip**)

(e) Time (**Meetup**) (f) Time (**Concerts**) (g) Time (**Unf**) (h) Time (**Zip**)

Figure 6: Varying the number of time intervals $|\mathcal{T}|$

a result, the values of the bounds are larger than a small number of assignments' scores. Hence, a small number of score updates can be avoided by exploiting bounds.

Time. In terms of execution time (Fig. 5i–5l), we can observe that time is determined by the number of computations performed. HOR-I outperforms the other methods in all cases with HOR-I being around 4 times faster than ALG (for large k in real datasets).

4.2.2 *Effect of the Number of Time Intervals*

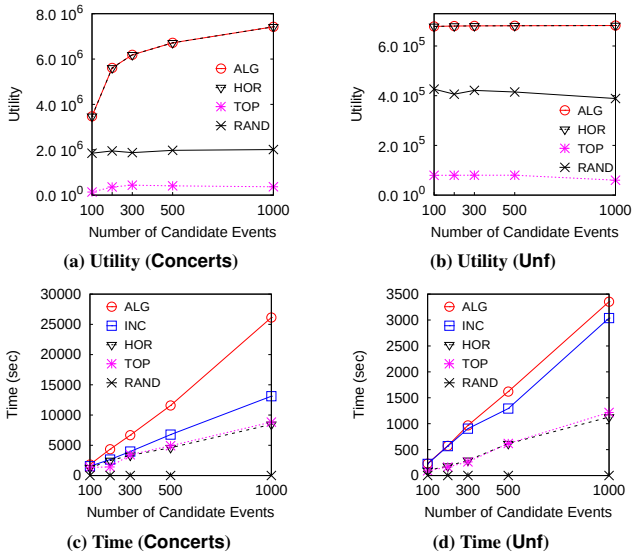
In this experiment (Fig. 6), we vary the number of time intervals $|\mathcal{T}|$. Due to lack of space, for this and the following experiments, the plots presenting the number of computations are omitted.

Utility. Regarding utility (Fig. 6a –6d), similarly to the previous experiment, our HOR algorithm performs the same as the ALG. We observe that, as the number of intervals increases, the utility of all methods increases too. This happens since the increase of available intervals results to a smaller number of events assigned in the same interval, as well as to a larger number of candidate assignments. The former results to the assignment scores (in general) being larger in cases where fewer parallel events take place. The latter offers more options, which possibly result to better assignments.

Time. As for execution time (Fig. 6e–6h), excluding TOP, the HOR-I is the most efficient in the cases which differs from HOR (i.e., $|\mathcal{T}| < 100$); while in the rest cases, HOR is the most efficient. Notice that, in general, HOR performs very close to TOP. Overall, HOR and HOR-I are about 2 to 4 times faster than the ALG, and around 5 times faster for a small number of intervals. Finally, as explained in the previous experiment, we can observe that, also in this experiment, the bound-based methods (i.e., INC, HOR-I) are less effective in Unf.

4.2.3 *Effect of the Number of Candidate Events*

We next study the effect of varying the number of candidate events $|\mathcal{E}|$. Note that, in this experiment, since $k < |\mathcal{T}|$, HOR-I is identical to HOR. Due to lack of space, in this experiment, the plots for the Meetup and Zip are not presented, since they are similar to Concerts.



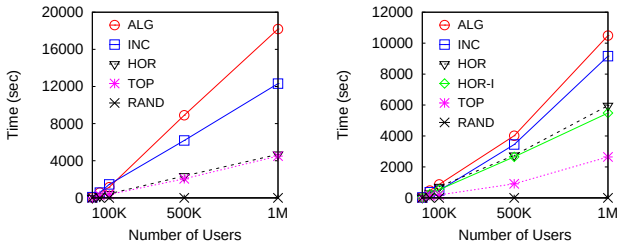
(a) Utility (**Concerts**) (b) Utility (**Unf**)

(c) Time (**Concerts**) (d) Time (**Unf**)

Figure 7: Varying the number of candidate events $|\mathcal{E}|$

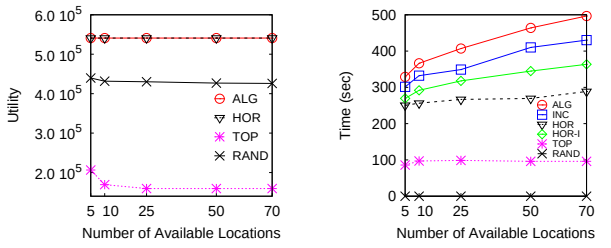
Utility. Also in this experiment (Fig. 7a–7b) our HOR method has the same utility score as the ALG in all cases. We observe that the utility of ALG and HOR increases with $|\mathcal{E}|$ (with an exception in Unf). On the other hand, for RAND it is either stable or is decreasing. This happens since the increase of $|\mathcal{E}|$ results to more candidate assignments. So, there are more options for the ALG and HOR methods, while for RAND it is less possible to select "good" assignments. Notice that, in the Unf case (Fig. 7b), the utility for the non-random methods remains stable. The reason is that increasing the number of "similar" events (as previously explained) cannot result to better assignments.

Time. Also in this experiment (Fig. 7c–7d), our HOR method outperforms the other, with INC having noticeably bad performance in Unf, compared to HOR (as in the previous experiments). Further, the difference between ALG and our methods increases with $|\mathcal{E}|$, due to the increasing number of update computations. Overall, in general HOR is around 3 to 4 times faster than ALG, and up to 5 times faster in Zip dataset specifically.



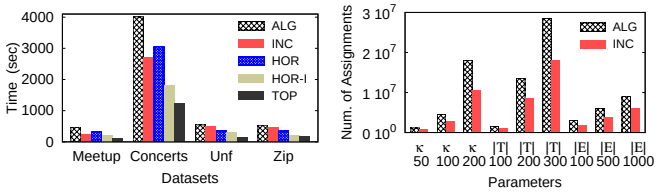
(a) Time ($|\mathcal{T}| = 150$) (b) Time ($|\mathcal{T}| = 65$)

Figure 8: Varying the number of users $|\mathcal{U}|$ (Unf Dataset)



(a) Utility vs. Locations (b) Time vs. Locations

Figure 9: Varying the number of locations (Unf, $|\mathcal{T}| = 65$)



(a) HOR & HOR-I worst case (b) ALG & INC search space

Figure 10: HOR/-I worst case and ALG/INC search space

4.2.4 Effect of the Number of Users

We then study the effect of varying the number of users (Fig. 8). Results for the Zip dataset are omitted since they are similar to the ones reported for Unf. Here, the HOR-I algorithm cannot be defined with the default parameters setting ($k = 100, \mathcal{T} = 150$). Hence, in order to also study HOR-I, we examine a supplementary experiment (Fig. 8b), where $\mathcal{T} = 65$. Note that, this setting ($k = 100, \mathcal{T} = 65$) corresponds to the average case for the HOR and HOR-I algorithms (w.r.t. the relation between k and $|\mathcal{T}|$; see Sect. 3.3.1 & 3.4.1).

In terms of utility (the plot is omitted due to lack of space), as expected, the utility increases with the number of users. The HOR and ALG methods have the same utility scores in all cases. Regarding performance, in the first experiment (Fig. 8a), HOR performs increasingly better than INC and ALG, as the number of users increases. In the second experiment (Fig. 8b), for larger numbers of users (i.e., $|\mathcal{U}| > 100$K), INC performs close to ALG. On the other hand, HOR and HOR-I outperforms INC, with the difference increases with $|\mathcal{U}|$. Overall, in the first experiment, HOR is around 3 to 4 times faster than ALG; in the second one, HOR and HOR-I are around 2 times faster than ALG.

4.2.5 Effect of the Number of Available Locations

In this experiment we vary the number of available locations of each candidate event (Fig. 9). The results correspond to the Unf dataset; though, similar results are reported in all datasets. We can observe, that the utility score (Fig. 9a) remains almost unaffected for the ALG and HOR methods, while TOP and RAND perform slightly better in 5 locations. This is expected, since, as the number of locations decreases, the number of feasible assignments decreases too. Regarding the execution time (Fig. 9b), in all methods, increases with number of locations. This is due to the fact that the number of feasible assignments (as well as the computations) increases too.

4.2.6 HOR & HOR-I Worst Case w.r.t. k and $|\mathcal{T}|$

Here, we consider the setting that corresponds to the worst case w.r.t. k and $|\mathcal{T}|$ for the HOR and HOR-I algorithms (Sect. 3.3.1 & 3.4.1). Thus, for $k = 100$, the worst case corresponds to $|\mathcal{T}| = 99$. Fig. 10a presents the execution time for all datasets. We can observe that even in the worst case, HOR-I outperforms all methods in all cases (excluding the TOP). Also, in synthetic datasets, where the INC demonstrates poor performance, HOR is more efficient.

4.2.7 Search Space

In this experiment (Fig. 10b) we study the effectiveness of the proposed assignment organization (Sect. 3.2.2). We measure the number of assignments examined by the ALG and our INC algorithm, varying the main parameters (k, $|\mathcal{T}|$, $|\mathcal{E}|$). In all cases, INC accesses noticeable less assignments. Also, in each parameter, the differences between INC and ALG increases in large parameter values. Overall, in most cases, INC examines slightly more than half assignments that ALG accesses.

4.2.8 Summary

In what follows, we summarize our findings: (1) *Between the datasets used*, with the exception of Unf, all the methods report similar results. In the Unf dataset, the bound-based methods (INC & HOR-I) demonstrate lower performance than in other datasets.

(2) *Regarding utility score*, in all cases, our HOR (and HOR-I) algorithm achieves almost the same utility score as ALG. Particularly, in more than 70% of the performed experiments (including the experiments omitted from the manuscript), HOR and ALG report the same utility scores, while in the rest of the cases, the difference in utility is on average 0.008%; with the largest difference being 1.3%. Recall that, the INC algorithm always return the same solution as ALG.

(3) *Comparing* ALG [4] *with our methods*, in several cases: (*i*) our HOR and HOR-I methods are around 5× faster and perform less than half of the computations. (*ii*) INC is more than 3× faster, performs less than half of the computations.

(4) *Comparing our methods*: (*i*) HOR-I is always faster than the other methods. In several cases, HOR-I is around 3 and 2 times faster than INC and HOR, respectively. (*ii*) HOR outperforms INC in terms of time and computations, with some exceptions, in cases where $k > |\mathcal{T}|$. Overall, in several cases, HOR is around 3× faster than INC.

5 RELATED WORK

Event Management & Mining. The SES problem studied in this work was recently introduced in [4], where a simple greedy algorithm was proposed. Compared to [4], here we show that SES is hard to be approximated over a factor and we design three efficient and scalable algorithms which perform on average half the computations compared the method presented in [4] and, in most cases, are 3 to 5 times faster (more details in Sect. 1).

Recently, a number of studies have been proposed in the context of event-participant planing. These works examine the problem of finding assignments between a set of users and a set of pre-scheduled events. The determined assignments aim to maximize the satisfaction of the users while satisfying several constraints. Particularly, [12] assigns one event to each user, based on her interests and social relations. [27] finds an user-event arrangement by assigning users to events. The latter work is extended in [28], where the online setting of the problem is examined. A similar user-event arrangement problem

is defined in a more advanced setting [26], where more factors are considered (e.g., complex spatio-temporal factors, travel cost). This work is extended in [6], in which participation lower bounds on event and potential changes induced either by event organizer or by users (e.g., changes on event location) are considered. In the same context, [31] tries to maximize the satisfaction of the least satisfied user. In an online scenario, [29] exploits the user feedback (i.e., accept or reject the assigned events) in order to adaptively learn user interests. This work tries to maximize the number of accepted assigned events. Compared to our work, as discussed in Sect. 1, the objective, the solution and the setting of our problem substantially differ from the aforementioned approaches.

In a different context, [8, 9] attempt to find influential event organizers and promoters from online social networks. [25] studies the influence of early respondents in online event scheduling process. Further, a number of works [5, 7, 36, 41] analyze several factors form (Event-based) Social Networks data in order to study user attendance and provide event recommendations. Our work studies a different problem compared to the aforementioned approaches. However, some of the aforementioned methods can be exploited in our problem to estimate the user attendance probability.

Assignment & Matching Problems. The problem studied shares common characteristics with the Generalized assignment (GAP) and Multiple knapsack (MKP) problems. Particularly, our problem is a generalized case of the GAP and MKP problems with identical bin capacities [18]. A major difference of SES compared to GAP and MKP is that in SES the expected attendance (resp. profit) of assigning an event (resp. item) to an interval (resp. bin) is determined based on the other events assigned to this interval. Also, beyond the event and interval entities which are also considered in the aforementioned problems, in our problem further core entities are involved (e.g., users, organizer, competing events). Additionally, assignment/matching problems (similar to bipartite matching) have been studied in spatial context [16, 30, 32, 35]. In general, the main differences of these works compared to SES, are the same as the ones that hold in GAP and MKP problems (see above).

Recommender Systems. Numerous approaches have been proposed in the context of location and event recommendations. Particularly, several works recommend events to users [13, 19, 21, 37, 40], while others offer location-based recommendations [2, 11, 14, 15, 34, 38, 39]. Further, in a more general setting, approaches have been proposed for recommending locations or items to groups of people (i.e., group recommendations) [1, 3, 20, 22–24, 33]. Compared to our work, the aforementioned approaches study a different problem, that is, recommending objects (e.g., venues, events) to users.

6 CONCLUSIONS

This paper studied the *Social Event Scheduling* (SES) problem, which assigns a set of events to time intervals, so that the number of attendees is maximized. We showed that SES is NP-hard to be approximated over a factor, and we proposed three efficient and scalable algorithms. The proposed algorithms are evaluated over several real and synthetic datasets, outperforming the existing solution three to five times in several of cases.

Acknowledgment. This research has been financed by the European Union through the FP7 ERC IDEAS 308019 NGHCS project, the Horizon2020 688380 VaVeL project, and a Google Faculty award.

REFERENCES

[1] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu. Group Recommendation: Semantics and Efficiency. *PVLDB*, 2(1), 2009.
[2] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel. Recommendations in Location-based Social Networks: A Survey. *GeoInformatica*, 19(3), 2015.
[3] N. Bikakis, K. Benouaret, and D. Sacharidis. Finding Desirable Objects under Group Categorical Preferences. *KAIS*, 49(1), 2016.
[4] N. Bikakis, V. Kalogeraki, and D. Gunopulos. Social Event Scheduling. In *ICDE*, 2018.
[5] I. Boutsis, S. Karanikolaou, and V. Kalogeraki. Personalized Event Recommendations Using Social Networks. In *MDM*, 2015.
[6] Y. Cheng, Y. Yuan, L. Chen, C. G. Giraud-Carrier, and G. Wang. Complex Event-participant Planning and Its Incremental Variant. In *ICDE*, 2017.
[7] R. Du, Z. Yu, T. Mei, Z. Wang, Z. Wang, B. Guo Predicting activity attendance in event-based social networks: Content, context and social influence. *UbiComp* 2014
[8] K. Feng, G. Cong, S. S. Bhowmick, and S. Ma. In Search of Influential Event Organizers in Online Social Networks. In *SIGMOD*, 2014.
[9] K. Han, Y. He, X. Xiao, S. Tang, F. Gui, C. Xu, and J. Luo. Budget-Constrained Organization of Influential Social Events. In *ICDE*, 2018.
[10] V. Kann. Maximum Bounded 3-Dimensional Matching Is Max Snp-complete. *Inf. Process. Lett.*, 37(1), 1991.
[11] T. K. Lee, S. Kim, M. Balduini, D. Dell'Aglio, I. Celino, Y. Huang, V. Tresp, and E. D. Valle. Location-based Mobile Recommendations by Hybrid Reasoning on Social Media Streams. In *JIST*, 2013.
[12] K. Li, W. Lu, S. Bhagat, L. V. Lakshmanan, and C. Yu. On Social Event Organization. In *KDD*, 2014.
[13] X. Liu, Q. He, Y. Tian, W. Lee, J. McPherson, and J. Han. Event-based Social Networks: Linking the Online and Offline Social Worlds. In *KDD*, 2012.
[14] X. Liu, Y. Liu, K. Aberer, and C. Miao. Personalized Point-of-interest Recommendation by Mining Users' Preference Transition. In *CIKM*, 2013.
[15] Y. Liu, T. Pham, G. Cong, Q. Yuan. An Experimental Evaluation of Point-of-interest Recommendation in Location-based Social Networks. *PVLDB*, 2017.
[16] C. Long, R. C. Wong, P. S. Yu, and M. Jiang. On Optimal Worst-case Matching. In *SIGMOD*, 2013.
[17] R. D. Luce. *Individual Choice Behavior: A theoretical analysis*. Wiley, 1959.
[18] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.
[19] E. Minkov, B. Charrow, J. Ledlie, S. J. Teller, and T. S. Jaakkola. Collaborative Future Event Recommendation. In *CIKM*, 2010.
[20] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H. Kriegel. Fast Group Recommendations by Applying User Clustering. In *ER*, 2012.
[21] T. N. Pham, X. Li, G. Cong, and Z. Zhang. A General Graph-based Model for Recommendation in Event-based Social Networks. In *ICDE*, 2015.
[22] S. Qi, N. Mamoulis, E. Pitoura, and P. Tsaparas. Recommending Packages to Groups. In *ICDM*, 2016.
[23] Y. Qian, Z. Lu, N. Mamoulis, and D. W. Cheung. P-LAG: Location-aware Group Recommendation for Passive Users. In *SSTD*, 2017.
[24] E. Quintarelli, E. Rabosio, and L. Tanca. Recommending New Items to Ephemeral Groups Using Contextual User Influence. In *ACM Recsys*, 2016.
[25] D. Romero, K. Reinecke, and L. Robert. The Influence of Early Respondents: Information Cascade Effects in Online Event Scheduling. In *WSDM*, 2017.
[26] J. She, Y. Tong, and L. Chen. Utility-aware Social Event-participant Planning. In *SIGMOD*, 2015.
[27] J. She, Y. Tong, L. Chen, and C. C. Cao. Conflict-aware Event-participant Arrangement. In *ICDE*, 2015.
[28] J. She, Y. Tong, L. Chen, and C. C. Cao. Conflict-aware Event-participant Arrangement and Its Variant for Online Setting. *TKDE*, 28(9), 2016.
[29] J. She, Y. Tong, L. Chen, and T. Song. Feedback-aware Social Event-participant Arrangement. In *SIGMOD*, 2017.
[30] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu. Online Minimum Matching in Real-time Spatial Data: Experiments and Analysis. *PVLDB*, 2016.
[31] Y. Tong, J. She, and R. Meng. Bottleneck-aware Arrangement Over Event-based Social Networks: The Max-min Approach. *WWWJ*, 19(6), 2015.
[32] L. H. U, M. L. Yiu, K. Mouratidis, and N. Mamoulis. Capacity Constrained Assignment in Spatial Databases. In *SIGMOD*, 2008.
[33] H. Wang, G. Li, and J. Feng. Group-based Personalized Location Recommendation on Social Networks. In *APWeb*, 2014.
[34] W. Wang, H. Yin, S. W. Sadiq, L. Chen, M. Xie, and X. Zhou. SPORE: a Sequential Personalized Spatial Item Recommender System. In *ICDE*, 2016.
[35] R. C. Wong, Y. Tao, A. W. Fu, and X. Xiao. On Efficient Spatial Matching. In *VLDB*, 2007.
[36] T. Xu, H. Zhong, H. Zhu, H. Xiong, E. Chen, G. Liu. Exploring the Impact of Dynamic Mutual Influence on Social Event Participation. In *SDM*, 2015.
[37] H. Yin, Q. V. H. Nguyen, Z. Huang, and X. Zhou. Joint Event-partner Recommendation in Event-based Social Networks. In *ICDE*, 2018.
[38] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen. LCARS: a Location-content-aware Recommender System. In *KDD*, 2013.
[39] H. Yin, X. Zhou, B. Cui, H. Wang, K. Zheng, and N. Q. V. Hung. Adapting to User Interest Drift for Poi Recommendation. *TKDE*, 28(10), 2016.
[40] W. Zhang, J. Wang, and W. Feng. Combining Latent Factor Model with Location Features for Event-based Group Recommendation. In *KDD*, 2013.
[41] X. Zhang, J. Zhao, and G. Cao. Who Will Attend? - Predicting Event Attendance in Event-based Social Network. In *MDM*, 2015.