Public Event Scheduling with Busy Agents

Bo Li[1] , **Lijun Li**[2] , **Minming Li**[2] and **Ruilong Zhang**[3]
[1]Department of Computing, The Hong Kong Polytechnic University
[2]Department of Computer Science, City University of Hong Kong
[3]Department of Computer Science and Engineering, University at Buffalo
comp-bo.li@polyu.edu.hk, lijunli3-c@my.cityu.edu.hk, minming.li@cityu.edu.hk,
ruilongzhang.cn@gmail.com

Abstract

We study a public event scheduling problem, where multiple public events are scheduled to coordinate the availability of multiple agents. The availability of each agent is determined by solving a separate flexible interval job scheduling problem, where the jobs are required to be preemptively processed. The agents want to attend as many events as possible, and their agreements are considered to be the total length of time during which they can attend these events. The goal is to find a schedule for events as well as the job schedule for each agent such that the total agreement is maximized.

We first show that the problem is NP-hard, and then prove that a simple greedy algorithm achieves $\frac{1}{2}$-approximation when the whole timeline is polynomially bounded. Our method also implies a $(1-\frac{1}{e})$-approximate algorithm for this case. Subsequently, for the general timeline case, we present an algorithmic framework that extends a $\frac{1}{\alpha}$-approximate algorithm for the one-event instance to the general case that achieves $\frac{1}{\alpha+1}$-approximation. Finally, we give a polynomial time algorithm that solves the one-event instance, and this implies a $\frac{1}{2}$-approximate algorithm for the general case.

1 Introduction

Artificial intelligence algorithms are widely used in many societal settings, such as scheduling public activities and assisting with personal work schedules, in which finding a suitable schedule shall lead to good social welfare. To motivate our study, consider the pubic event scheduling process at a university. Suppose the student/staff development office wants to hold some public activities for students/staffs, e.g., networking events, new student/staff orientation, etc. These events have fixed duration, while the target students/staffs are busy, and they have their own tasks that must be done but with some degree of time flexibility, e.g., homework, lectures, research meetings, submission deadlines, etc. For example, there are two events, both lasting for two hours. Some of the students/staffs must spend two hours on their own tasks between 10:00 and 13:00, and other students/staffs must spend three hours between 14:00 and 18:00. The whole schedulable time span for events is 10:00-18:00. These events are important to students/staffs, and it is crucial to schedule these public events at suitable times so that the target students/staffs can participate as long as possible. The *agreement* of a student/staff to an event schedule is considered as the total time that she can attend the events. Hence, the organizer's task is to make the event schedule admit a maximum agreement. This shall lead to good social welfare, motivating our work.

The above public event scheduling problem aims to find feasible job and event schedules such that social welfare is maximized. Hence, it falls under the umbrella of both job scheduling and computational social choices. The intersection of these two fields has been extensively studied in various literature, such as selfish load balancing [Bilò *et al.*, 2020; Vinci *et al.*, 2022], the Santa clause problem [Bamas *et al.*, 2021; Bansal and Sviridenko, 2006; Springer *et al.*, 2022], fair division with scheduling constraints [Li *et al.*, 2021; Li *et al.*, 2023; Zhou *et al.*, 2023], etc [Elkind *et al.*, 2022; Pascual *et al.*, 2018; Endriss *et al.*, 2022].

From the scheduling perspective, our problem shares some similarities with a set of scheduling problems. Intuitively, the optimal solution to our problem tends to gather the schedule of agents' jobs, which leads to a long consecutive idle time during which events can be scheduled. The optimal solution for a batch of scheduling problems also shares a similar behavior, such as gap scheduling [Antoniadis *et al.*, 2020; Baptiste, 2006], active time slots minimization [Chang *et al.*, 2014; Chau and Li, 2020; Kumar and Khuller, 2018], calibration scheduling [Bender *et al.*, 2013; Chau *et al.*, 2020; Chen *et al.*, 2019], etc. However, our work investigates a distinct objective compared to these classical scheduling problems and thus leads to completely different techniques.

From the perspective of computational social choice, our problem shares some similarities with the recently proposed cake-sharing model [Bei *et al.*, 2022]. We can consider the whole timeline as a cake whose range is from $[0, 1]$, and then normalize all agents' tasks to ensure that jobs' release time and deadline are in $[0, 1]$. Furthermore, we assume the jobs' processing time is equal to the length of its interval, where we call a job *rigid* (otherwise, it is called *flexible*). An agent's task can be interpreted as a preference for the cake, i.e., if an agent has a task $[a, b]$, then the agent does not prefer this part of the cake. Our goal is to pick m pieces of cakes such that the total agreement of the picked cakes is maximized. The above scenario also motivates our work, and it matches the setting of cake-sharing [Bei *et al.*, 2022] in the general sense. Compared to cake sharing, the first main difference is that we allow jobs to be flexible. The second main difference is that we focus on the social welfare maximization problem while [Bei *et al.*, 2022] aims to design truthful mechanisms.

1.1 Our Contributions

We consider the problem of Public Event Scheduling with Busy Agents (PESBA). We distinguish several cases according to the event set or the length of the whole timeline. For each case, we either give an approximate (or exact) algorithm running in polynomial time or show the problem is NP-hard. We remark that, due to space limits, all formal proofs are omitted and can be found in full version [Li *et al.*, 2024].

Main Result 1 (Theorem 1). When the whole timeline is polynomially bounded, a natural greedy algorithm running in polynomial time achieves $\frac{1}{2}$-approximation for PESBA. Moreover, PESBA admits a $(1 - \frac{1}{e})$-approximate algorithm running in polynomial time.

For a better understanding of our method, let us first focus on the case where the whole timeline is polynomially bounded (Section 3). Our main algorithm is an intuitive greedy algorithm which can be viewed as a simple voting process. Namely, in each round of our algorithm, for each unscheduled event, we ask agents to vote on the position for this event, and each agent votes for those time slots that maximize her agreement. The algorithm then picks one position that receives the most votes for this event. Finally, the algorithm schedules the event that maximizes the total agreement among all unscheduled events in this round. By proving (i) a new result for a variant of the submodular maximization (ii) the agreement function is submodular, we show that the greedy algorithm is $\frac{1}{2}$-approximate. This intuitive algorithm enables us to extend it to arbitrary timeline cases. In fact, we show a stronger result that the agreement function is a rank function of some matroid, which is also crucial for the arbitrary timeline case. By using the more involved algorithms for submodular optimization with matroid constraints, our method also implies a better $(1 - \frac{1}{e})$-approximate algorithm, but it is hard to extend to arbitrary timeline cases.

Main Result 2 (Theorem 2). When the whole timeline is arbitrary, we present an algorithmic framework that extends a $\frac{1}{\alpha}$-approximate algorithm for PESBA with one event to the multiple events case achieving $\frac{1}{\alpha+1}$-approximation. We design an optimal algorithm for the one-event instance and hence obtain a $\frac{1}{2}$-approximate algorithm for the arbitrary timeline case.

Our second result focuses on the general PESBA instance (Section 4). We propose an algorithmic framework for the general case. Given any $\frac{1}{\alpha}$-approximate algorithm for the one-event instance, our framework extends the algorithm to multiple events case in polynomial time that achieves $\frac{1}{\alpha+1}$-approximation. Our algorithmic framework is rooted in the greedy algorithm for the polynomially bounded timeline case. In each iteration of the greedy algorithm, it enumerates each time slot for searching the best position of each unscheduled event. This only works for the previous case since the whole timeline is polynomially bounded. We design a new oracle based on the given one-event approximation algorithm that is able to return a "good" position for each unscheduled event without enumerating the whole timeline. How good the position is depends on the approximation ratio of the given one-event algorithm. By using the optimal algorithm for the one-event instance proposed in full version [Li *et al.*, 2024], we directly obtain a $\frac{1}{2}$-approximate algorithm for general PESBA instances, which matches the approximation ratio for restricted timeline case.

Main Result 3. The problem PESBA is NP-hard even in the case where (i) there is only one agent; (ii) this agent has only two rigid jobs. Moreover, the problem PESBA is in P when only one event is required to be scheduled.

Our last result focuses on the computation complexity of the problem. We first show that PESBA is NP-hard even if there is only one agent and the agent has only two rigid jobs. The hardness result is built on the classical partition problem. Then we show PESBA is in P when there is only one event that needs to be scheduled. This result shall be used to get an approximation for general PESBA instances. Our algorithm for one-event instance is involved, and its basic idea is to draw the plot for the agreement function when the event moves over the timeline. To this end, we prove that the agreement function is piecewise linear and has a polynomial number of turning points. These two parts can be found in the full version [Li *et al.*, 2024].

1.2 Other Related Works

Fair Allocation with Public Goods. Our problem shares some similarities with fair allocation of public goods. [Conitzer *et al.*, 2017] first uses "public" to distinguish the previous "private" goods, and subsequently, there is a batch of follow-up works [Fain *et al.*, 2018; Fluschnik *et al.*, 2019; Garg *et al.*, 2021]. In the public goods setting, all agents obtain utilities when some goods are selected, which is similar to the agreement in our problem. However, the techniques are different since we consider a different objective.

Other Related Scheduling Problems. When the preemption is not allowed, it is NP-complete to determine whether the given job set admits a feasible schedule [Garey and Johnson, 1979]. When all jobs are rigid or unit, the problem is equivalent to determining the size of the maximum independent set on interval graphs or a maximum matching. Hence it is in P [Schrijver, 1999]. When preemption is allowed, a simple greedy algorithm (earliest deadline first) can be used to decide instances' feasibility [Lenstra and Shmoys, 2020].

2 Preliminaries

We consider the problem of Public Event Scheduling with Busy Agents (PESBA). An instance of PESBA consists of an agent set $A := \{1, \ldots, n\}$ and a public event set $E := \{e_1, \ldots, e_m\}$. We consider discrete time, and for $t \in \mathbb{N}_{\geq 1}$, let $[t, t+1)$ denote the t-th time slot. T is a time slot set consisting of all time slots where events and jobs can be scheduled. Each agent i has a set of jobs denoted by J_i. Each job j

in J_i has a release time $r_j \in \mathbb{N}_{\geq 1}$, a deadline $d_j \in \mathbb{N}_{\geq 1}$, and a processing time $p_j \in \mathbb{N}_{\geq 1}$ such that $p_j \leq d_j - r_j + 1$. A job j is called a *rigid* job if $p_j = d_j - r_j + 1$; it is called a *unit* job if $p_j = 1$. For each job j, the interval $[r_j, d_j]$ is called a *job interval*, which can also be viewed as a set of consecutive time slots, i.e., $\{ r_j, r_j + 1, \ldots, d_j \}$. The event set E consists of m public events, and each event e has a length $l(e) \in \mathbb{N}_{\geq 1}$. When an event e is scheduled at the t-th time slot, this event occupies $\{ t, t+1, \ldots, t + l(e) - 1 \}$ time slots to be held. We use a tuple (e, t) to denote that event e is scheduled at the t-th time slot. Let $\mathcal{L}$ be a set of all possible tuples, i.e., $\mathcal{L} := \{ (e,t) \}_{e \in E, t \in T}$. Let $\mathcal{L}(e)$ be the set of all possible schedules of event e, i.e., $\mathcal{L}(e) := \{ (e', t) \in \mathcal{L} \mid e' = e \}$.

A solution consists of both the event schedule and the job schedule for each agent. The event schedule $\mathcal{S}$ is considered as a subset of $\mathcal{L}$ such that every event appears in exactly one tuple in $\mathcal{S}$. This ensures that (i) every event $e \in E$ is scheduled at some time slot; (ii) no event is scheduled more than two time slots. An event schedule $\mathcal{S}$ is said to be a partial event schedule if every event appears in at most one tuple; otherwise, it is called a complete event schedule. The job schedule consists of n subschedules, each of which corresponds to a job schedule for each agent. We consider the preemptive schedule, and the job schedule for each $J_i, i \in A$ is required to be feasible, i.e., (i) each job j in J_i is assigned p_j time slots in its job interval; (ii) for each $t \in T$, at most one job from J_i is processed at t. It is ensured that for each $i \in A$, the input job set J_i is feasible.

Each event is considered to be a good activity, and agents are willing to attend the events. A time slot t is said to be an *agreement* time slot for agent i if (i) t is occupied by some scheduled event; (ii) no job from J_i is scheduled at t. Given any event schedule and a job schedule for J_i, agent i's *agreement* is defined as the total number of agreement time slots. We allow event schedules to overlap since we can always eliminate the overlap by adjusting events' schedules without decreasing the total agreement (see Observation 1 in full version [Li *et al.*, 2024]). Our goal is to find a job schedule for each J_i as well as a complete event schedule such that the total agents' agreement is maximized.

We use function $\mathsf{agr}_i : 2^{\mathcal{L}} \to \mathbb{N}_{\geq 0}$ to represent the maximum agreement that is produced by an event schedule $\mathcal{S}$ for agent i. Given any partial event schedule $\mathcal{S}$ and an agent i, in Section 3.2, we show that the value of $\mathsf{agr}_i(\mathcal{S})$ can be computed in polynomial time. Such an algorithm is used to compute the optimal job schedule, and this allows us to focus only on finding an event schedule. Based on the agreement function, our objective is equivalent to finding a complete event schedule $\mathcal{S}$ such that $\sum_{i \in A} \mathsf{agr}_i(\mathcal{S})$ is maximized. An example can be found in Figure 1. In the remainder of this paper, we refer to $\mathcal{S}$ as a *partial* solution if $\mathcal{S}$ is a partial event schedule; otherwise, it is a *complete* solution.

3 Algorithms for Polynomial T

In this section, for a better understanding of our method, we focus on the case where the whole timeline is polynomially bounded; the assumption shall be removed in Section 4. We show that a natural greedy algorithm achieves

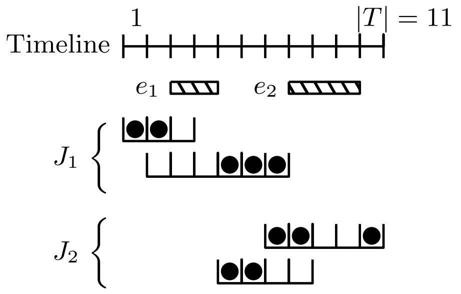

Figure 1: Illustration of the model of PESBA problem. There are two public events $E = \{ e_1, e_2 \}$ with length $l(e_1) = 2$ and $l(e_2) = 3$ represented by shaded rectangles, and two agents $A = \{ 1, 2 \}$. Agent 1's job set is J_1 containing two jobs $[1,3]$ with processing time 2 and $[2,7]$ with processing time 3. Agent 2's job set is J_2 containing two jobs $[7,11]$ with processing time 3 and $[5,8]$ with processing time 2. The time span is $T = [1,11]$ containing 11 time slots. The figure shows one optimal event schedule $\mathcal{S} = \{ (e_1, 3), (e_2, 8) \}$ maximizing $\sum_{i \in A} \mathsf{agr}_i(\mathcal{S})$ and a corresponding optimal job schedule where the black disk indicates job's processing. Agent 1 can attend the whole course of e_1 and e_2 under schedule $\mathcal{S}$ and thus get agreement 5; agent 2 can attend whole e_1 but can only attend e_2 for 2 times and thus get agreement 4 from $\mathcal{S}$.

$\frac{1}{2}$-approximation. In Section 4, we shall extend the $\frac{1}{2}$-approximation algorithm to the arbitrary $|T|$ case with the approximation ratio being preserved. Our method also implies a slightly stronger algorithm that achieves $(1 - \frac{1}{e})$-approximation in the polynomial bounded $|T|$ case. Such a stronger result uses the submodular maximization subject to a matroid constraint [Călinescu *et al.*, 2011] as a black box, and thus, it is hard to be extended to the arbitrary $|T|$ case. Thus, we focus on the greedy algorithm, and we state how to get a $(1 - \frac{1}{e})$-approximation in the proof of Theorem 1.

Theorem 1. *Given any instance of* PESBA *with polynomially bounded $|T|$, there exists a greedy algorithm running in polynomial time that achieves $\frac{1}{2}$-approximation. The approximation ratio can be further improved to $(1 - \frac{1}{e})$.*

Algorithmic Framework. For each event $e \in E$ and a time slot $t \in T$, the tuple (e, t) represents that the event e is scheduled at the t-th time slot. Since $|T|$ is assumed to be polynomially bounded, we can find the optimal position for event e by enumerating all possible positions. Our algorithm runs in rounds and schedules exactly one event in each round until all events are assigned to a starting position. In each round, the algorithm checks the unscheduled events one by one. For each unscheduled event e', every agent assigns some "votes" to every possible position of e'. Then, our algorithm chooses the schedule (e', t') that receives the most votes (break ties arbitrarily), which gives a scheduled position t' for event e'. For each unscheduled event, we have a position in this round, and then, the algorithm just picks an event and its position (e^*, t^*) with the maximum votes among all unscheduled events. Intuitively, the votes of some schedule (e, t) stands for the agent's preference for this schedule. Our algorithm shall select the schedule that is liked by the most agents and schedule the corresponding event at the corresponding time slot.

The first step of the algorithm is to set up an appropriate preference rule such that each agent assigns suitable votes to each schedule. Our idea is intuitive, i.e., agent i's preference for some schedule (e, t) is just the maximum incremental agreement that agent i can gain by scheduling e at t, i.e., $\mathsf{agr}_i(\mathcal{S} \cup \{ (e,t) \}) - \mathsf{agr}_i(\mathcal{S})$ where $\mathcal{S}$ is the current solution.

3.1 The Complete Algorithm

This section presents the formal description of the algorithm. Given the partial solution $\mathcal{S}$, we define F to be the total agreement produced by $\mathcal{S}$ for notation convenience, i.e., $F(\mathcal{S}) := \sum_{i \in A} \mathsf{agr}_i(\mathcal{S})$. For an unscheduled event e and any schedule (e, t), the total preference for the schedule (e, t) is $F(\mathcal{S} \cup \{ (e,t) \}) - F(\mathcal{S})$. Following the basic idea stated above, the algorithm picks a schedule with the maximum total preference in each round until all events are scheduled.

Algorithm 1 The Complete Algorithm for PESBA

```
Input: A PESBA instance with polynomially bounded $|T|$.
Output: A schedule $\mathcal{S}$ of all events in $E$.
 1: $\mathcal{S} \leftarrow \emptyset$.
 2: while there exists an unscheduled event do
 3:     $\mathcal{H} \leftarrow \emptyset$.
 4:     for each unscheduled event $e'$ do
 5:         $t' \leftarrow \arg\max_{t \in T} F(\mathcal{S} \cup \{ (e',t) \}) - F(\mathcal{S})$.
 6:         $\mathcal{H} \leftarrow \mathcal{H} \cup \{ (e',t') \}$.
 7:     end for
 8:     $(e^*, t^*) \leftarrow \arg\max_{(e,t) \in \mathcal{H}} F(\mathcal{S} \cup \{ (e,t) \}) - F(\mathcal{S})$.
 9:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{ (e^*, t^*) \}$.
10: end while
11: return $\mathcal{S}$.
```

The analysis of Algorithm 1 is built on a variant of a classical submodular maximization problem, i.e., the selected elements are additionally required to satisfy the group constraints, and an imperfect oracle exists. The formal definition is shown in Definition 1.

Definition 1 (Submodular Maximization with Group Constraints and Imperfect Oracle (SMGC))**.** *Given some parameter $n \in \mathbb{N}_{\geq 1}$ and there is a ground element set U with $|U| = 2^{\mathrm{poly}(n)}$. There is a monotone and submodular function $f : 2^U \rightarrow \mathbb{R}_{\geq 0}$ defined over U. The ground element U is partitioned into ℓ groups: $G_1, \dots, G_\ell$ with $\ell = \mathrm{poly}(n)$. There is a polynomial α-approximate oracle $g(\cdot, \cdot)$ that takes two parameters as the input: a subset $S \subseteq U$ and a group G_i, and it returns an element $e^* \in G_i$ in $\mathrm{poly}(n)$ time such that $f(S \cup \{ e^* \}) - f(S) \geq \frac{1}{\alpha} \max_{e \in G_i} f(S \cup \{ e \}) - f(S)$, where $\alpha \geq 1$ is some given parameter. The goal is to pick a subset S of the ground element set such that $f(S)$ is maximized, where S is required to satisfy two conditions: (i) $|S| \leq k$ (ℓ is guaranteed to be larger than k); (ii) S contains at most one element from G_i for all $i \in [\ell]$. The running time of the algorithm is required to be $\mathrm{poly}(n)$.*

SMGC v.s. PESBA. To see the connection, consider U as all possible schedules $\mathcal{L}$, n as the input size of PESBA, $f(\cdot)$ as the total preference function $F(\cdot)$. The additional group constraint corresponds to the requirement that each event can be assigned at most one starting point in our problem. The imperfect oracle states that to get an element with a good marginal value in polynomial time, the algorithm has to ask an imperfect oracle, which may only be able to return some non-optimal elements. The α-approximate oracle is used to deal with general $|T|$ cases, and its role will be clear in Section 4. In this section, we can assume that we have a 1-approximate oracle (i.e., $\alpha = 1$) since we consider the polynomially bounded $|T|$ case. This implies that the size of $\mathcal{L}$ (i.e., the ground element set) is polynomially bounded, so finding an element with the maximum marginal value takes polynomial time.

When $\alpha = 1$, SMGC is a special case of submodular maximization subject to a matroid constraint [Călinescu *et al.*, 2011], which admits a $(1 - \frac{1}{e})$-approximation. However, the algorithm is involved, and it is based on the multilinear extension of a submodular function. For the matroid constraint, the approximate ratio of the natural greedy algorithm is proved to be $\frac{1}{2}$ [Horel, 2015] whose proof is based on both the exchange property of a matroid and the submodularity. In Section 3.4, we show that the ratio only depends on the submodularity for our problem and thus simplifies the proof for the $\alpha = 1$ case.

Formally, the analysis mainly consists of three steps. In the first step, we show that the agreement function can be computed in polynomial time. In the second step, we prove that the agreement function is submodular. In the last step, we demonstrate that the natural greedy algorithm achieves $\frac{1}{\alpha+1}$-approximation to SMGC with an α-approximate oracle; when $\alpha = 1$, it is a $\frac{1}{2}$-approximate algorithm. The detailed description is as follows.

1. We show that given any partial solution $\mathcal{S}$, the value of $\mathsf{agr}_i(\mathcal{S})$ can be computed in polynomial time for any $i \in A$. This implies that the cumulative function F in Algorithm 1 can also be computed in polynomial time. This part is deferred to Section 3.2.

   Lemma 1. *Given any partial solution $\mathcal{S} \subseteq \mathcal{L}$, there is an algorithm running in polynomial time that computes the value of $\mathsf{agr}_i(\mathcal{S})$.*

2. We demonstrate that the agreement function $\mathsf{agr}_i(\cdot)$ is a submodular function. We actually show a stronger result, i.e., the agreement function is a rank function of some matroid which also plays an important role in the arbitrary T case. This part is shown in Section 3.3.

   Lemma 2. *Consider any agent $i \in A$, the agreement function $\mathsf{agr}_i : 2^{\mathcal{L}} \rightarrow \mathbb{N}_{\geq 0}$ is a submodular function.*

3. We prove that greedily picking the element with a good marginal value from unselected groups achieves $\frac{1}{\alpha+1}$-approximation for SMGC. This part is in Section 3.4.

   Lemma 3. *The natural greedy algorithm (Algorithm 2) achieves $\frac{1}{\alpha+1}$-approximation to the problem of submodular maximization with group constraints and an α-approximate oracle stated in Definition 1.*

The first part of Theorem 1 can be proved by combining the above lemmas and setting $\alpha = 1$, and the other part is based on another result for submodular maximization.

3.2 Agreement Function Computation

In this section, we show that the maximum agreement of any partial solution can be computed in polynomial time. We fix an agent $i \in A$ and analyze her agreement function $\mathsf{agr}_i(\cdot)$. For notation convenience, we drop the subscript i for the time.

For an event or job interval, we use $s(\cdot)$ and $f(\cdot)$ to denote the first and the last time slot occupied by the interval. Given a partial solution $\mathcal{S} \subseteq \mathcal{L}$, let E' be the scheduled event set. Define Ψ to be the set of $s(j), f(j), s(e), f(e)$ for all $j \in J$ and $e \in E'$ (remove duplicate values). Note that the size of Ψ is polynomially bounded regardless of the size of the time span T. We sort all time slots in Ψ in increasing order obtaining $\Psi = \{ t_1, \dots, t_\kappa \}$. Then, the time slots in Ψ split the whole time span into several segments, i.e., every adjacent two-time slot in Ψ defines a segment ϕ. See the left part of Figure 2 for an example. Let $\Phi := \{ \phi_1, \dots, \phi_\ell \}$ be the set of segments. Note that a segment ϕ may contain more than one time slot. Let $c(\phi)$ be the capacity of the segment ϕ, i.e., $c(\phi)$ is the number of time slots that are included in ϕ. Note that it must be the case that for any segment $\phi \in \Phi$ and any job or event interval I, ϕ is either completely included in I or ϕ has no overlap with I.

Our algorithm for computing $\mathsf{agr}(\cdot)$ is built based on the min-cost max-flow algorithm. In the following, we first introduce the construction of the flow network and then prove the correctness. Given any partial solution $\mathcal{S}$ and a job set J, we first construct the starting point set Ψ and the segment set Φ. Then, we construct a network $G := (s, t, A \cup B, E)$ according to Ψ and J. (I) For each job $j \in J$, we create a vertex a in A. (II) For each segment $\phi \in \Phi$, we have one vertex b in B. (III) There is a directed edge e from a to b if and only if a's corresponding job can be scheduled at the time slots included in b's corresponding segment (let E_2 be these edges). (IV) Add a source s and create a directed edge from s to each vertex in A (let E_1 be these edges). (V) Add a sink t and create a directed edge from each vertex in B to t (let E_3 be these edges). (VI) Every edge in E_1 has the same cost 0 and only connects one vertex in A. Let p_j be $e := (s, a)$'s capacity where a's corresponding job is j. (VII) Every edge in E_2 has the same cost 0, and edges that are connected with the same job vertex j have the same capacity p_j. (VIII) Each edge in E_3 only connects one vertex from B; let $c(\phi)$ be $e := (b, t)$'s capacity where ϕ is the corresponding segment of b; the cost of e is 1 if there is an event occupying ϕ otherwise the cost of e is 0. The goal is to find the cheapest way to send $\sum_{j \in J} p_j$ amount of flows from s to t. An example is shown in Figure 2.

It is well-known that the min-cost max-flow problem can be solved in polynomial time. The solution to min-cost max-flow has integral property as long as all capacities are integral, i.e., there exists an optimal solution to min-cost max-flow such that all flow values are integral. Lemma 4 captures the equivalence between the instance of our problem and the constructed min-cost max-flow instance.

Lemma 4. *Given any instance $\mathcal{I}$ of our problem with job set J and partial solution $\mathcal{S}$. Let $\mathcal{I}'$ be the constructed min-cost max-flow instance. Then, the constructed min-cost max-flow instance has a feasible integral flow assignment S' with cost $\mathsf{ct}(S')$ if and only if the instance $\mathcal{I}$ has a job schedule S with the agreement $|\mathsf{st}(\mathcal{S})| - \mathsf{ct}(S')$, where $\mathsf{st}(\mathcal{S})$ is a set of time slots that are occupied by some scheduled event in $\mathcal{S}$.*

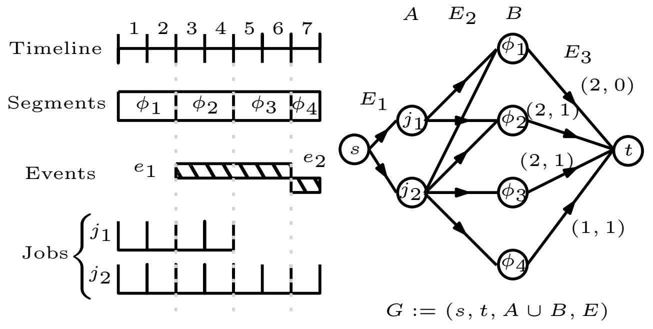

Figure 2: An example for min-cost max-flow construction. The left part of the figure shows the original job set and event schedule with $\Psi = \{ [1,2), [3,4), [4,5), [6,7), [7,8) \}$. We cut the whole timeline into a set of segments $\Phi = \{ [1,3), [3,5), [5,7), [7,8) \}$ with polynomial size according to Ψ. The constructed flow network is shown in the right part of the figure. For each job, we have a vertex in A, and for each segment, we have a vertex in B. The capacity and the cost of each edge are designed to ensure that a flow assignment corresponds to a job schedule, e.g., $(2, 0)$ of the edge in E_3 connecting ϕ_1 and t indicates the capacity of this edge is 2 and cost is 0.

Lemma 1 can be proved by Lemma 4.

3.3 Submodularity of the Agreement Function

In this section, we show that the agreement function $\mathsf{agr}_i(\cdot)$ is submodular for any agent $i \in A$. To this end, we fix an agent $i \in A$ and analyze her agreement function. For notation convenience, we drop the subscript i.

For convenience, each unit of a job is considered as a new job, i.e., we create p_j new jobs with unit length for each job j. In this way, we can assume that all jobs in J have unit length. Given any partial solution $\mathcal{S} \subseteq \mathcal{L}$, let $\mathsf{st}(\mathcal{S})$ be a set of time slots that are occupied by scheduled events in $\mathcal{S}$, so $\mathsf{st}(\mathcal{S}) \subseteq T$. Given any schedule of jobs in J, the agreement produced by the partial solution $\mathcal{S}$ and the job schedule is equal to the difference between the size of $\mathsf{st}(\mathcal{S})$ and the number of time slots that are occupied by some jobs in the job schedule. So, finding the maximum agreement produced by the partial solution $\mathcal{S}$ is equivalent to finding a schedule of jobs in J that uses the minimum number of time slots in $\mathsf{st}(\mathcal{S})$.

This motivates us to define the following set system, which is called *scheduling matroid* (Definition 2). We shall connect the size of an independent set in the defined set system with the value of the maximum agreement later.

Definition 2 (Scheduling Matroid)**.** *The scheduling matroid $\mathcal{M}(J) := (T, \mathcal{I})$ is defined over the time slot set T for a job set J, where T is the ground element set and $\mathcal{I}$ is a collection of all independent sets. Each job in J has a release time, unit processing time, and deadline. A set $I \subseteq T$ is an independent set if there exists a feasible schedule of J only using time slots $T \setminus I$.*

Consider any partial solution $\mathcal{S}$, let $I \subseteq \mathsf{st}(\mathcal{S})$ be an independent set with the maximum size that is included in $\mathsf{st}(\mathcal{S})$.

By definition, $|I|$ is the maximum number of time slots in $\mathsf{st}(\mathcal{S})$ such that all jobs in J can be scheduled in $T \setminus I$. Thus, the size of I is equal to the maximum agreement produced by the partial solution $\mathcal{S}$ since all other time slots in $\mathsf{st}(\mathcal{S}) \setminus I$ have to be used to ensure a feasible job schedule. The above connection is captured by Observation 1.

Observation 1. *Given any partial solution $\mathcal{S}$, let I be a maximum independent set defined in Definition 2 that are included in $\mathsf{st}(\mathcal{S})$. Then, $|I|$ is equal to the maximum agreement produced by the partial solution $\mathcal{S}$.*

As one may observe, the maximum independent set that is included in some given element set aligns with the concept of the *rank* in matroid theory. The rank function of a matroid is one of the famous submodular functions. Hence, in the following, we aim to prove the scheduling matroid defined in Definition 2 is a matroid. In fact, besides the submodularity, the polynomial implementation of Algorithm 1 in Section 4 also depends on the property that the scheduling matroid is a matroid. We start with restating the definition of a matroid and its rank function in Definition 3.

Definition 3 (Matroid). *A set system $\mathcal{M} := (U, \mathcal{I})$ is a matroid if the collection $\mathcal{I}$ of subsets of U has the following properties: (i) $\emptyset \in \mathcal{I}$; (ii) if $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$; (iii) If $A, B \in \mathcal{I}$ and $|B| > |A|$, then there exists $x \in (B \setminus A)$ such that $A \cup \{ x \} \in \mathcal{I}$. Given any set $S \subseteq U$, the rank of S is defined as the size of the maximum independent set included in S, i.e., $\mathsf{rank}(S) := \max_{I \in \mathcal{I} : I \subseteq S} |I|$.*

Lemma 5. *The scheduling matroid is a matroid.*

Lemma 2 can be proved by Lemma 5 and Observation 1.

3.4 Submodular Maximization with Group Constraints and Imperfect Oracle

In this section, we show greedily picking an element with a "good" marginal value from the unselected groups is a $\frac{1}{\alpha+1}$-approximate algorithm for SMGC under an α-approximate oracle $g(\cdot,\cdot)$ (see Algorithm 2). We cannot find the element with the exact maximum marginal value because the oracle is α-approximate. When $\alpha = 1$, Algorithm 2 is $\frac{1}{2}$-approximate, which aligns with the result for submodular maximization subject to matroid constraint stated in [Horel, 2015]. But our proof only uses the monotone and submodular properties and thus simplifies the proof in [Horel, 2015]. For $\alpha > 1$, our result generalizes the approximation ratio stated in [Chekuri and Kumar, 2004] where they also consider the α-approximate oracle but only focus on coverage functions.

4 An Algorithmic Framework for Arbitrary T

In this section, we consider the case where $|T|$ is an arbitrary value and present an algorithmic framework for this case. Given any approximation algorithm for the one-event instance, the proposed algorithmic framework rooted in Algorithm 1 is able to extend the given approximation algorithm to the general case by only losing a small constant factor on the approximation ratio. The formal description can be found in Theorem 2. The one-event instance is a special case of PESBA where there is only one event that needs to be scheduled; we use PESBA with $|E| = 1$ to denote this case.

Algorithm 2 Natural Greedy for SMGC

```
Input: The ground element set $U$ and its group collection
    $G_1, \dots, G_\kappa$; A monotone submodular function $f : 2^U \to \mathbb{R}_{\geq 0}$; The capacity $k \in \mathbb{N}_{\geq 0}$.
Output: A subset $S \subseteq U$ with $|S| \leq k$.
 1: $S \leftarrow \emptyset$.
 2: while $|S| \leq k$ do
 3:     $H \leftarrow \emptyset$.
 4:     for each unselected group $G_i$ do
 5:         $a_i \leftarrow g(S, G_i)$.
 6:         $H \leftarrow H \cup \{ a_i \}$.
 7:     end for
 8:     $a^* \leftarrow \max_{b \in H} f(S \cup \{ b \}) - f(S)$.
 9:     $S \leftarrow S \cup \{ a^* \}$.
10: end while
11: return $S$.
```

Theorem 2. *Given any $\frac{1}{\alpha}$-approximate algorithm ($\alpha \geq 1$) running in polynomial time for* PESBA *with $|E| = 1$, there is a polynomial time algorithm that achieves $\frac{1}{\alpha+1}$-approximation for general instances of* PESBA.

We remark that, for our problem, it is hard to use the standard technique to make Algorithm 1 run in polynomial time for the arbitrary $|T|$ case; see full version [Li *et al.*, 2024] for discussions.

Algorithmic Ideas. To design an algorithmic framework that achieves Theorem 2, our idea is to release the full power of Algorithm 1 and Lemma 3. Recall that in Section 3, we consider the polynomially bounded $|T|$, which is equivalent to assuming that we have a 1-approximate oracle for SMGC (Definition 1). By Lemma 3, as long as we have an α-approximate oracle, Algorithm 1 with line 5 replaced (which is Algorithm 4 in full version [Li *et al.*, 2024]) is a $\frac{1}{\alpha+1}$-approximate algorithm. Thus, in this section, we aim to design such an α-approximate oracle to replace line 5 of Algorithm 1. To this end, we need to build an algorithm ALG (Algorithm 3) such that ALG satisfies the following two properties: (i) ALG runs in polynomial time (Observation 2); (ii) given any partial solution $\mathcal{S}$ and an event e, ALG returns a position for the event e such that the resulting schedule is $\frac{1}{\alpha}$-approximate (Lemma 7).

We shall build the polynomial time oracle based on the given $\frac{1}{\alpha}$-approximate algorithm for PESBA with $|E| = 1$. To achieve this, the main obstacle is that the given one-event instance algorithm only works for the case where there is no partial solution; this disagrees with the requirement of the oracle. Our idea to fix this issue is to construct an equivalent pure one-event instance so that when we run $\frac{1}{\alpha}$-approximate algorithm on the constructed one-event instance, it is able to return a good position for the instance with the partial solution. This method works mainly due to the exchange property of the scheduling matroid.

Formally, we use `OneEvent(·,·)` to represent the given $\frac{1}{\alpha}$-approximate algorithm for PESBA with $|E| = 1$. The algorithm `OneEvent(A,e*)` takes the agent set A (including their job sets) and an event e^* as the input; its output is a time slot t^* that indicates the scheduled position for the given event. By our assumption, the returned time slot t^* satisfies the following property: $\sum_{i\in A} \mathsf{agr}_i(\{ (e^*, t^*) \}) \geq \frac{1}{\alpha} \cdot \max_{t\in T} \sum_{i\in A} \mathsf{agr}_i(\{ (e^*, t) \})$. Let $\texttt{GoodPosn}(\cdot,\cdot)$ be the desired approximate oracle, and $\texttt{GoodPosn}(\mathcal{S}, e^*)$ shall use the given $\frac{1}{\alpha}$-approximate one-event algorithm as a subroutine. It takes two parameters as the input: the given partial solution $\mathcal{S}$ and an event e^*, and will return a time slot that indicates the scheduled time of the given event. Recall that, given a partial solution $\mathcal{S}$, we need to construct an equivalent pure one-event instance A' such that the time slot t^* returned by $\texttt{OneEvent}(A', e^*)$ is a good position for e^*, i.e., $\sum_{i\in A} \mathsf{agr}_i(\mathcal{S} \cup \{ (e^*, t^*) \}) \geq \frac{1}{\alpha} \cdot \max_{t\in T} \sum_{i\in A} \mathsf{agr}_i(\mathcal{S} \cup \{ (e^*, t) \})$. The above inequality shall imply that $\texttt{GoodPosn}(\cdot,\cdot)$ is exactly the same as the α-approximate oracle g in Definition 1, which leads to a $\frac{1}{\alpha+1}$-approximate algorithm by Lemma 3.

In the following, we focus on how to construct such an equivalent instance A'. We start with an important property of the scheduling matroid.

Lemma 6. *For a scheduling matroid $\mathcal{M}(J) := (T, \mathcal{I})$ defined in Definition 2, consider a time slot set $S \subseteq T$ and let $F \in \mathcal{I}$ be a maximum independent set that is included in S. For each time slot in F, create a rigid job and let J' be all these rigid jobs. The job set $J \cup J'$ defines a new scheduling matroid, denoted by $\mathcal{M}(J \cup J') := (T, \mathcal{I}')$. Consider a time slot set $H \subseteq T$ and let $B \in \mathcal{I}'$ be a maximum independent set that is included in H. Then, $F \cup B \in \mathcal{I}$ is a maximum independent set in $\mathcal{M}(J)$ that is included in $S \cup H$.*

Given any partial solution $\mathcal{S}$, recall that a maximum independent set included in $\mathsf{st}(\mathcal{S})$ is a set of time slots with the maximum size that produces the agreement (Observation 1). We call these time slots *agreement time slots*. Thus, Lemma 6 actually suggests a "stability" property of these agreement time slots. In other words, consider two partial solutions $\mathcal{S}$ and $\mathcal{H}$. The agreement time slot of the partial solution $\mathcal{S}$ is a subset of the agreement time slot of the partial solution $\mathcal{S}\cup\mathcal{H}$. This crucial stable property motivates the pure one-event instance construction method, where we shall replace all agreement time slots of partial solution $\mathcal{S}$ with many rigid jobs. In this way, we eliminate the impact of the partial solution $\mathcal{S}$ and focus on finding the maximum agreement of the partial solution $\mathcal{H}$. Then, by Lemma 6, the union of $\mathcal{S}$'s and $\mathcal{H}$'s agreement time slots shall be the maximum agreement time slots of the whole solution $\mathcal{S} \cup \mathcal{H}$. The only issue is that we need to be careful when creating rigid jobs since the number of constructed rigid jobs is required to be polynomial. This requirement can be satisfied by aggregating all agreement time slots in each segment when computing the min-cost flow (Algorithm 3). An example is shown in Figure 3.

Observation 2. *Algorithm 3 runs in polynomial time, i.e., Algorithm 3 creates a polynomial number of rigid jobs.*

Lemma 7 states that Algorithm 3 $\texttt{GoodPosn}(\cdot,\cdot)$ is a valid α-approximate oracle for the submodular maximization problem stated in Definition 1. The correctness of Lemma 7 mainly relies on Lemma 6.

Lemma 7. *Given any partial solution $\mathcal{S}$ and an unscheduled event e^*, let t^* be the time slot returned by Algorithm 3. Then, we have $\sum_{i\in A} \mathsf{agr}_i(\mathcal{S} \cup \{ (e^*, t^*) \}) \geq \frac{1}{\alpha} \cdot \max_{t\in T} \sum_{i\in A} \mathsf{agr}_i(\mathcal{S} \cup \{ (e^*, t) \})$.*

Algorithm 3 $\texttt{GoodPosn}(\cdot,\cdot)$

```
Input: A partial solution $\mathcal{S}$; An unscheduled event $e^*$.
Output: A good position $t^*$ for event $e^*$.
1: for each agent $i \in A$ do
2:     Compute a schedule for $J_i$ by min-cost max-flow.
3:     for each segment in $\{ \phi \in \Phi \mid \phi \subseteq \mathsf{st}(\mathcal{S}) \}$ do
4:         Aggregate all agreement time slots forming an agreement segment $\phi'$.
5:         Create a rigid job $j'$ for $\phi'$.
6:     end for
7:     $J_i \leftarrow J_i \cup \{$ all created rigid jobs $\}$.
8: end for
9: return $t^* \leftarrow \texttt{OneEvent}(A, e^*)$.
```

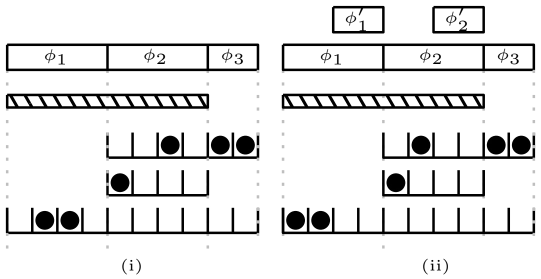

Figure 3: Illustration of time slots aggregation (line 4 of Algorithm 3). The subfigure (i) presents an optimal job schedule computed by the min-cost max-flow algorithm. If we do not gather the agreement time slots, it is possible that the agreement time slots are separated by the job's processing. In subfigure (ii), such time slots producing agreements are aggregated to form agreement segments. Each segment can have at most one agreement segment.

The complete algorithm for arbitrary $|T|$ (see Algorithm 4 in full version [Li *et al.*, 2024]) just replaces line 5 of Algorithm 1 with Algorithm 3. Theorem 2 is implied by Lemma 3 and Lemma 7. Combing Theorem 4 in full version [Li *et al.*, 2024] with Theorem 2 implies a $\frac{1}{2}$-approximate algorithm for the general $|T|$ case.

5 Conclusion

In this paper, we study the problem of public event scheduling with busy agents, in which a set of public events are required to be scheduled to let agents attend for a period of time as long as possible. We present a general algorithmic framework and give a polynomial time algorithm with a constant approximation ratio for such an NP-hard problem.

This work points out many interesting future directions. Firstly, it would be interesting to see a better approximation algorithm or a lower bound. Secondly, our $\frac{1}{2}$-approximate algorithm for general cases uses an involved optimal algorithm for one-event instances. So, a simplified algorithm for one-event instances would be beneficial, even if some approximation ratio needs to be sacrificed. Lastly, the weighted setting of our problem is also interesting, i.e., different agents hold a distinct preference for the same event.

Contribution Statement

All authors (ordered alphabetically) have equal contributions and are corresponding authors.

Acknowledgements

We thank the anonymous reviewers for their many insightful suggestions. Bo Li is funded by HKSAR RGC (No. PolyU 15224823) and GDSTC (No. 2023A1515010592). Ruilong Zhang was supported by NSF grant CCF-1844890.

References

[Antoniadis *et al.*, 2020] Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. Parallel machine scheduling to minimize energy consumption. In *SODA*, pages 2758–2769. SIAM, 2020.

[Bamas *et al.*, 2021] Étienne Bamas, Paritosh Garg, and Lars Rohwedder. The submodular santa claus problem in the restricted assignment case. In *ICALP*, volume 198 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Bansal and Sviridenko, 2006] Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *STOC*, pages 31–40. ACM, 2006.

[Baptiste, 2006] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA*, pages 364–367. ACM Press, 2006.

[Bei *et al.*, 2022] Xiaohui Bei, Xinhang Lu, and Warut Suksompong. Truthful cake sharing. In *AAAI*, pages 4809–4817. AAAI Press, 2022.

[Bender *et al.*, 2013] Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, and Cynthia A. Phillips. Efficient scheduling to minimize calibrations. In *SPAA*, pages 280–287. ACM, 2013.

[Bilò *et al.*, 2020] Vittorio Bilò, Gianpiero Monaco, Luca Moscardelli, and Cosimo Vinci. Nash social welfare in selfish and online load balancing. In *WINE*, volume 12495 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2020.

[Călinescu *et al.*, 2011] Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

[Chang *et al.*, 2014] Jessica Chang, Harold N. Gabow, and Samir Khuller. A model for minimizing active processor time. *Algorithmica*, 70(3):368–405, 2014.

[Chau and Li, 2020] Vincent Chau and Minming Li. Active and busy time scheduling problem: A survey. In *Complexity and Approximation*, volume 12000 of *Lecture Notes in Computer Science*, pages 219–229. Springer, 2020.

[Chau *et al.*, 2020] Vincent Chau, Minming Li, Elaine Yinling Wang, Ruilong Zhang, and Yingchao Zhao. Minimizing the cost of batch calibrations. *Theor. Comput. Sci.*, 828-829:55–64, 2020.

[Chekuri and Kumar, 2004] Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM*, volume 3122 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2004.

[Chen *et al.*, 2019] Lin Chen, Minming Li, Guohui Lin, and Kai Wang. Approximation of scheduling with calibrations on multiple machines (brief announcement). In *SPAA*, pages 237–239. ACM, 2019.

[Conitzer *et al.*, 2017] Vincent Conitzer, Rupert Freeman, and Nisarg Shah. Fair public decision making. In *EC*, pages 629–646. ACM, 2017.

[Elkind *et al.*, 2022] Edith Elkind, Sonja Kraiczy, and Nicholas Teh. Fairness in temporal slot assignment. In *International Symposium on Algorithmic Game Theory*, pages 490–507. Springer, 2022.

[Endriss *et al.*, 2022] Ulle Endriss, Arianna Novaro, and Zoi Terzopoulou. Representation matters: Characterisation and impossibility results for interval aggregation. In *Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)*, pages 286–292. International Joint Conferences on Artificial Intelligence Organization, 2022.

[Fain *et al.*, 2018] Brandon Fain, Kamesh Munagala, and Nisarg Shah. Fair allocation of indivisible public goods. In *EC*, pages 575–592. ACM, 2018.

[Fluschnik *et al.*, 2019] Till Fluschnik, Piotr Skowron, Mervin Triphaus, and Kai Wilker. Fair knapsack. In *AAAI*, pages 1941–1948. AAAI Press, 2019.

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

[Garg *et al.*, 2021] Jugal Garg, Pooja Kulkarni, and Aniket Murhekar. On fair and efficient allocations of indivisible public goods. In *FSTTCS*, volume 213 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Horel, 2015] Thibaut Horel. Notes on greedy algorithms for submodular maximization. *CORR*, 2015.

[Kumar and Khuller, 2018] Saurabh Kumar and Samir Khuller. Brief announcement: A greedy 2 approximation for the active time problem. In *SPAA*, pages 347–349. ACM, 2018.

[Lenstra and Shmoys, 2020] Jan Karel Lenstra and David B. Shmoys. Elements of scheduling. *CoRR*, abs/2001.06005, 2020.

[Li *et al.*, 2021] Bo Li, Minming Li, and Ruilong Zhang. Fair scheduling for time-dependent resources. In *NeurIPS*, pages 21744–21756, 2021.

[Li *et al.*, 2023] Bo Li, Fangxiao Wang, and Yu Zhou. Fair allocation of indivisible chores: Beyond additive costs. In *NeurIPS*, page to appear, 2023.

[Li *et al.*, 2024] Bo Li, Lijun Li, Minming Li, and Ruilong Zhang. Public event scheduling with busy agents. *CoRR*, abs/2404.11879, 2024.

[Pascual *et al.*, 2018] Fanny Pascual, Krzysztof Rzadca, and Piotr Skowron. Collective schedules: Scheduling meets computational social choice. In *Seventeenth International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[Schrijver, 1999] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[Springer *et al.*, 2022] Max Springer, MohammadTaghi Hajiaghayi, Debmalya Panigrahi, and Mohammad Reza Khani. Online algorithms for the santa claus problem. In *NeurIPS*, 2022.

[Vinci *et al.*, 2022] Cosimo Vinci, Vittorio Bilò, Gianpiero Monaco, and Luca Moscardelli. Nash social welfare in selfish and online load balancing. *ACM Trans. Economics and Comput.*, 10(2):8:1–8:41, 2022.

[Zhou *et al.*, 2023] Shengwei Zhou, Rufan Bai, and Xiaowei Wu. Multi-agent online scheduling: MMS allocations for indivisible items. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 42506–42516. PMLR, 2023.