Fair Scheduling for Time-dependent Resources

Bo Li *
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Hong Kong
comp-bo.li@polyu.edu.hk

Minming Li *
Department of Computer Science
City University of Hong Kong
Kowloon Tang, Hong Kong
minming.li@cityu.edu.hk

Ruilong Zhang * †
Department of Computer Science
City University of Hong Kong
Kowloon Tang, Hong Kong
ruilzhang4-c@my.cityu.edu.hk

Abstract

We study a fair resource scheduling problem,nwhere a set of interval jobs are to be allocated to heterogeneous machines controlled by intellectual agents. Each job is associated with release time, deadline and processing time such that it can be processed if its complete processing period is between its release time and deadline. The machines gain possibly different utilities by processing different jobs, and all jobs assigned to the same machine should be processed without overlap. We consider two widely studied solution concepts, namely, maximin share fairness and envy-freeness. For both criteria, we discuss the extent to which fair allocations exist and present constant approximation algorithms for various settings.

1 Introduction

With the rapid progress of AI technologies, AI algorithms are widely deployed in many societal settings such as the distribution of job and education opportunities where complex social effects may significantly diminish the performance of the algorithms. To motivate our study, let us consider a problem faced by the Students Affairs Office (SAO). An SAO clerk is assigning multiple part-time jobs to the students who submitted job applications. Each part-time job occupies a consecutive time period within a possibly flexible interval. For example, an one-hour math tutorial needs to be given between 8:00am and 11:00am on June 26th. A *feasible* assignment requires that the jobs assigned to an applicant can be scheduled without mutual overlap. The students are heterogeneous, i.e., different students may hold different job preferences. It is important that the students are treated equally in terms of getting job opportunities, and thus the clerk's task is to make the assignment fair.

The SAO problem falls under the umbrella of the research on *job scheduling*, which has been studied in numerous fields, including operations research Gentner *et al.* [2004], machine learning Paleja *et al.* [2020], parallel computing Drozdowski [2009], cloud computing Al-Arasi and Saif [2020], etc. Following the convention of job scheduling research, each part-time job, or *job* for short, is associated with release time, deadline, and processing time. The students are modeled as *machines*, who have different utility gains for completing jobs. Traditionally, the objective of designing scheduling algorithms is solely focused on efficiency or profit. However, motivated by various real-world AI driven deployments where the data points of the algorithms are real human beings who should be

*Equal contribution
†Corresponding author

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

treated unbiasedly, addressing the individual fairness becomes important. Accordingly, the past several years has seen considerable efforts in developing fair AI algorithms Chierichetti *et al.* [2017], where combinatorial structures are incorporated into the design, such as vertex cover Rahmattalabi *et al.* [2019], facility location Chen *et al.* [2019] and knapsack Amanatidis *et al.* [2020].

It is noted that people have different criteria on evaluating fairness, and in this work, we consider two of the most widely accepted definitions. The first is motivated by the max-min objective, i.e., maximizing the worst-case utility, which has received observable attention for various learning scenarios Rahmattalabi *et al.* [2019]. However, for heterogeneous agents, optimizing the worst case is not enough, as different people have different perspectives and may not agree on the output. Accordingly, one popular research agenda is centered around computing an assignment such that everyone believes that it (approximately) maximizes the worst case utility. This criterion is named *maximin* share (MMS) fairness in Budish [2010]. The second one is *envy-freeness* (EF), which has been very widely studied in social sciences and economics but arguably less explored in machine learning. Informally, an assignment is called EF if everyone believes she has obtained the best resource compared with any other agent's assignment. We note that, due to the scheduling-feasible constraint, some jobs may not be allocated. Thus EF alone is not able to satisfy the agents as keeping all resources unallocated does not incur any envy among them, but the agents envy the charity where unallocated/disregarded items are assumed to be donated to a charity. To resolve this issue, in this work, we want to understand how we can compute allocations that are simultaneously EF and Pareto efficient (PO), where an allocation is called PO if there does not exist another allocation that makes nobody worse off but somebody strictly better off.

Recently, Chiarelli *et al.* [2020] and Hummel and Hetland [2021] studied the fair allocation of conflicting items, where the items are connected via graphs. An edge between two items means they are in conflict and should be allocated to different agents. However, in our model, the conflict among items cannot be described as the edges in a graph. For example, two one-hour tutorials between 9:00am and 11:00am can be feasibly scheduled, but three such tutorials are not feasible any more. For a comprehensive introduction to the various constraints, including conflict constraints, studied in fair division, we refer the interested reader to the recent survey of Suksompong [2021].

1.1 Main Results

We study the fair interval scheduling problem (FISP), where fairness is captured by MMS and EF. For each of them, we design approximation algorithms to compute MMS or EF1 schedules.

Maximin Share. Informally, a machine's MMS is defined to be her optimal worst-case utility in an imaginary experiment: she partitions the items into m bundles but was the last to select one, where m is the number of agents. It is noted that as the machines are heterogeneous, they may not have the same MMS value. Our task is to investigate the extent to which everyone agrees on the final allocation. A job assignment is called α-approximate MMS fair if every machine's utility is no less than α fraction of her MMS value. Our main result in this part is an algorithmic framework which ensures a $1/3$-approximate MMS schedule, and thus improves the best known approximation of 1/5 which is proved for a broader class of valuation functions – XOS Ghodsi *et al.* [2018]. Interestingly, in the independent and parallel work Hummel and Hetland [2021], the authors also show the existence of $1/3$-approximate MMS for graphically conflicting items. With XOS valuation oracles, Ghodsi *et al.* [2018] also designed a polynomial-time algorithm to compute a 0.125-approximate MMS allocation. As a comparison, by slightly modifying our algorithm, it returns a 0.24-approximate MMS allocation in polynomial time, without valuation oracles. When all jobs are rigid, i.e., processing time = deadline - release time, our problem degenerates to finding a partition of an interval graph such that the minimum weight of the independent set for each subgraph is maximized. Recently, a pseudo-polynomial-time algorithm is given in Chiarelli *et al.* [2020] for constant number of agents. In this sense, we generalize this problem to flexible jobs and design approximation algorithms for arbitrary number of agents.

Main Result 1. For an arbitrary FISP instance, there exists a $1/3$-approximate MMS schedule, and a $(0.24 - \epsilon)$-approximate MMS schedule can be found in polynomial time, for any constant $\epsilon > 0$.

EF1+PO. EF is actually a demanding fairness notion, in the sense that any approximation of EF is not compatible with PO. Instead, initiated by Lipton *et al.* [2004], most research is focused on its relaxation, *envy-freeness up to one item* (EF1), which means the envy between two agents may exist

but will disappear if some item is removed. Unfortunately, EF1 and PO are still not compatible even if all jobs are rigid and agents have unary valuations. However, the good news is, if all jobs have unit processing time, an EF1 and PO schedule is guaranteed to exist and can be found in polynomial time. This result continues to hold when agent valuations are weighted but identical. It is shown in Biswas and Barman [2018] that under laminar matroid constraint an EF1 and PO allocation exists when agents have identical utilities, but finding it may need exponential time. We improve this result in two perspectives. First, our feasibility constraints, even for unit jobs, are not necessarily laminar matroid. Second, our algorithm runs in polynomial time.

Main Result 2. No algorithm can return an EF1 and PO schedule for all FISP instances, even if all jobs are rigid and valuations are unary. When all jobs have unit processing time and valuations are (weighted) identical, an EF1 and PO schedule can be computed in polynomial time.

Although exact EF1 and PO are not compatible, we prove that for an arbitrary FISP instance, there always exists a $1/4$-approximate EF1 and PO schedule, which coincides with Wu *et al.* [2021]. If all jobs have unit processing time, a $1/2$-approximate EF1 and PO schedule exists. To prove this result, we consider Nash social welfare – the geometric mean of all machines' utilities. We show that a Nash social welfare maximizing schedule satisfies the desired approximation ratio. This result is in contrast to the corresponding one in Caragiannis *et al.* [2016], which shows that without any feasibility constraints, such an allocation is EF1 and PO. We also show that both approximations are tight.

Main Result 3. For any FISP instance, the schedule maximizing Nash social welfare is PO and $1/4$-approximate EF1. If all jobs have unit processing time, it is $1/2$-approximate EF1.

EF1+IO By above results, we observe that PO is too demanding to measure efficiency in our model. One milder requirement is *individual optimality* (IO). Intuitively, an allocation is called IO if every agent gets the best feasible subset of jobs from the union of her current jobs and unscheduled jobs. We show that EF1 is still not compatible with IO in the general case. But for unary valuations, we obtain positive results and design polynomial time algorithms for (1) computing an EF1 and IO schedule for rigid jobs, and (2) computing an EF1 and 1/2-approximate IO schedule for flexible jobs. To prove these results, we utilize two classic algorithms *Earliest Deadline First* and *Round-Robin*. We defer this part completely to the full version Li *et al.* [2021].

1.2 Other Related Works

Since computing feasible job sets to maximize the total weight is NP-hard Garey and Johnson [1979], various approximation algorithms have been proposed Bar-Noy *et al.* [2001]; Berman and DasGupta [2000]; Chuzhoy *et al.* [2006], and the best known approximation ratio is 0.644 Im *et al.* [2020]. For rigid instances, the problem is polynomial-time solvable Schrijver [1999]. Recently, scheduling has been studied from the perspective of machine learning, including developing learning algorithms to empirically solve NP-hard scheduling problem Zhang *et al.* [2020]; Paleja *et al.* [2020], and predicting uncertain data in order to optimize the performance in the online setting Purohit *et al.* [2018]. Fairness has been concerned in the scheduling community in past decades Ajtai *et al.* [1998]; Baruah and Lin [1998]; Baruah [1995]. Most of these works aim at finding a fair schedule for the jobs, such as balancing the waiting and completion time Bilò *et al.* [2016]; Im and Moseley [2020].

MMS allocation for indivisible resources has been widely studied since Budish [2010]. Unfortunately, it is shown in Kurokawa *et al.* [2018]; Ghodsi *et al.* [2018]; Feige *et al.* [2021] that an exact MMS fair allocation may not exist. Thereafter, a string of approximation algorithms for various valuation types are proposed, such as additive Garg and Taki [2020], matroid-rank function Babaioff *et al.* [2021]; Barman and Verma [2021], submodular Barman and Krishnamurthy [2020]; Ghodsi *et al.* [2018], XOS and subadditive Ghodsi *et al.* [2018]. Regarding EF1, in the unconstrained setting, an allocation that is both EF1 and PO is guaranteed to exist Caragiannis *et al.* [2016]; Barman *et al.* [2018]. However, when there are constraints, such as cardinality and knapsack, the general compatibility is still open Biswas and Barman [2018, 2019]; Wu *et al.* [2021]; Gan *et al.* [2021]; Dror *et al.* [2021].

2 Preliminaries

2.1 Fair Interval Scheduling Problem

In a fair interval scheduling problem (FISP), we are given a job-machine system, which is denoted by tuple $(J, A, \mathbf{u}_A)$. $J = \{ j_1, \cdots, j_n \}$ represents a set of n jobs (also called resources or items) and $A = \{ a_1, \cdots, a_m \}$ is a set of $m \geqslant 2$ machines controlled by agents. In this work, machines and agents are used interchangeably. We consider discrete time, and for $t \in \mathbb{N}_+$, let $[t, t+1)$ denote the t-th *time slot*. Each $j_i \in J$ is associated with release time $r_i \in \mathbb{N}_+$, deadline $d_i \in \mathbb{N}_+$, and processing time $p_j \in \mathbb{N}_+$ such that $p_i \leqslant d_i - r_i + 1$. The $[r_i, d_i]$ is called a job interval, which can also be viewed as a set of consecutive time slots, $\{ r_i, r_i + 1, \cdots, d_i \}$. Job j_i can be processed successfully if it is offered p_i consecutive time slots within $[r_i, d_i]$. Each machine can process at most one job at each time slot and a set of jobs $J' \subseteq J$ is called *feasible* if all jobs in J' can be processed without overlap on a single machine. For a job $j_k \in J$, agent $a_i \in A$ gains utility $u_i(\{ j_k \}) \geqslant 0$ if j_k is successfully processed by a_i. We slightly abuse the notation and assume that $u_i(j_k) = u_i(\{ j_k \})$. We use u_i to denote a_i's utility function, and define $\mathbf{u}_A = (u_i)_{i \in A}$. For a feasible set of jobs S, the agent's utility is additive, i.e., $u_i(S) = \sum_{j_k \in S} u_i(j_k)$. For an arbitrary set of jobs that may not be feasible, the agent's utility is the maximum she can obtain by processing a feasible subset, i.e.,

$$u_i(S) = \max_{S' \subseteq S:\ S' \text{ is feasible}} \sum_{j_k \in S'} u_i(j_k).$$

It is noted that $u_i(\cdot)$'s are not additive for infeasible set of jobs and the computation of its value is NP-hard Garey and Johnson [1979]. In the full version Li *et al.* [2021], we show that they are actually XOS, which is a special type of subadditive functions. We call these $u_i(\cdot)$'s *interval scheduling* (IS) functions.

A *schedule* or *allocation* $\mathbf{X} = (X_1, \cdots, X_m)$ is defined as an ordered partial partition of J, where X_i is the jobs assigned to agent a_i, such that $X_i \cap X_j = \emptyset$ for $i \neq j$ and $X_1 \cup \cdots \cup X_m \subseteq J$. Let $X_0 = J \setminus \bigcup_{i \in [m]} X_i$ denote all unscheduled jobs, which is regarded as the donation to a *charity*. A schedule $\mathbf{X}$ is called *feasible* if X_i is feasible for all $a_i \in A$, i.e., all jobs in X_i can be successfully processed by a_i. Note that since jobs in X_0 are not scheduled, X_0 is not necessarily feasible. Observe that any infeasible schedule $\mathbf{X}$ is equivalent to a feasible schedule $\mathbf{X}'$ by setting each X'_i to be the feasible subset of X_i that maximizes a_i's utility and $X'_0 = J \setminus \bigcup_{i \in [m]} X'_i$. We call an instance *rigid* if $p_i = d_i - r_i + 1$, for all $j_i \in J$, i.e., the jobs need to occupy the entire job intervals. For rigid instances, the feasibility constraints can be described via interval graphs and the computation of $u_i(S)$ for any $S \subseteq J$ can be done in polynomial time Kleinberg and Tardos [2006].

2.2 Solution Concepts

We first define the maximin value for any utility function u, item set S and the number of agents k. Let $\mathcal{F}(S, k)$ be the set of all k-partial-partitions of S and

$$\mathsf{MMS}^u(S, k) = \max_{(S_1, \cdots, S_k) \in \mathcal{F}(S,k)} \min_{i \in [k]} u(X_i).$$

For any FISP instance $(J, A, \mathbf{u}_A)$ with $m = |A|$, agent $a_i \in A$'s maximin share (MMS) is given by

$$\mathsf{MMS}_i(J, m) = \mathsf{MMS}^{u_i}(J, m).$$

When the parameters are clear in the context, we write $\mathsf{MMS}_i = \mathsf{MMS}_i(J, m)$ for simplicity. If a schedule $\mathbf{X}$ achieves MMS_i, i.e., $\min_{k \in [m]} u_i(X_k) = \mathsf{MMS}_i$, it is called an MMS schedule for a_i.

Definition 1 (α-MMS Schedule)**.** *For $0 < \alpha \leqslant 1$, a schedule $\mathbf{X} = (X_1, \cdots, X_m)$ is called α-approximate MMS (α-MMS) if $u_i(X_i) \geqslant \alpha \cdot \mathsf{MMS}_i$. When $\alpha = 1$, $\mathbf{X}$ is called an MMS schedule.*

We next introduce envy freeness (EF). An EF schedule $\mathbf{X} = (X_1, \cdots, X_m)$ requires everybody's utility to be no less than her utility for any other agent's bundle, i.e., $u_i(X_i) \geqslant u_i(X_k)$ for any $a_i, a_k \in A$. Since EF is over demanding for indivisible items, following the convention of fair division literature, in this work, we mainly consider EF1.

Definition 2 (α-EF1 Schedule)**.** *For $0 < \alpha \leqslant 1$, a schedule $\mathbf{X} = (X_1, \cdots, X_m)$ is called α-approximate envy-free up to one item (α-EF1) if for any two agents $a_i, a_k \in A$,*

$$u_i(X_i) \geqslant \alpha \cdot u_i(X_k \setminus \{j\}) \text{ for some } j \in X_k.$$

When $\alpha = 1$, $\mathbf{X}$ is called an EF1 schedule.

We observe that an *empty* schedule is trivially EF and EF1, i.e., $X_0 = J$ and $X_i = \emptyset$ for all $a_i \in A$. However, this is a highly inefficient schedule, and thus we also want the schedule to be Pareto optimal.

Definition 3 (PO schedule)**.** *A schedule $\mathbf{X} = (X_1, \cdots, X_m)$ is called Pareto Optimal (PO) if there does not exist an alternative schedule $\mathbf{X}' = (X'_1, \cdots, X'_m)$ such that $u_i(X'_i) \geqslant u_i(X_i)$ for all $a_i \in A$, and $u_k(X'_k) > u_k(X_k)$ for some $a_k \in A$.*

We note that any approximation of EF is not compatible with PO, even in the very simple setting with two machines and a single job. In the full version Li *et al.* [2021], we introduce another efficiency criterion, *individual optimality* (IO), which is weaker than PO and study the compatibility between EF1 and IO.

3 Approximately MMS Scheduling

Before introducing our algorithmic framework, we first recall the best known existential and computation results for MMS scheduling problems.

Observation 1 (Ghodsi *et al.* [2018])**.** *For an arbitrary FISP instance, there exists a 1/5-MMS schedule and a 1/8-MMS schedule can be computed in polynomial time, given XOS function oracle.*

3.1 Algorithmic Framework

In this section, we present our algorithmic framework and prove that it ensures a $1/3$-MMS schedule. The algorithm has two parameters, a threshold vector $(\gamma_1, \cdots, \gamma_m)$ with $\gamma_i \geqslant 0$ and a β-approximation algorithm for IS functions, where $0 \leqslant \beta \leqslant 1$. In this section, we set $\gamma_i = \mathsf{MMS}_i$ for each $a_i \in A$. We can pretend that $\beta = 1$ to understand the existential result easily. Note that the computations of each MMS_i and exact value for IS functions are NP-hard, and in Section 3.2, we show how to gradually adjust the parameters to make it run in polynomial time. The high-level idea of the algorithm is to repeatedly fill a bag with unscheduled jobs (which may not be feasible) until some agent values it for no less than a threshold and takes away the bag. Then this agent reserves her best feasible subset of the bag, and returns the remaining jobs to the algorithm. By carefully designing the thresholds, we show that everybody can obtain at least $\frac{\beta}{\beta+2}$ of her MMS.

3.1.1 Pre-processing

As we will see, the above bag-filling algorithm works well only if the jobs are small, i.e., $u_i(j_k) \leqslant \frac{\beta}{\beta+2} \cdot \gamma_i$ for all $a_i \in A$ and $j_k \in J$. We first introduce the following property, which is used to deal with large jobs. Intuitively, Lemma 1 implies that after allocating an arbitrary job to an arbitrary agent, the remaining agents' MMS values in the reduced sub-instance do not decrease. A similar result for additive valuations is proved in Amanatidis *et al.* [2017].

Lemma 1. *For any instance $\mathcal{I} = (J, A, \mathbf{u}_A)$ with $|A| = m$, the following inequality holds for any $a_i \in A$ and any $j_k \in J$,*

$$\mathsf{MMS}_i(J \setminus \{ j_k \}, m-1) \geqslant \mathsf{MMS}_i(J, m).$$

We use Lemma 1 to design Algorithm 1 which repeatedly allocates a large job to some agent and removes them from the instance until there is no large job.

Algorithm 1. Matching Procedure

Input: Arbitrary FISP instance $\mathcal{I} = (J, A, \mathbf{u}_A)$; Thresholds $(\gamma_1, \cdots, \gamma_m)$.
Output: (1) Sub-instance $\mathcal{I}' = (J', A', \mathbf{u}_{A'})$ such that $u_i(j_k) \leqslant \frac{\beta}{\beta+2} \cdot \gamma_i$ for all $a_i \in A'$ and $j_k \in J'$; (2) Partial Schedule $(X_r)_{a_r \in A \setminus A'}$.

1: Initialize $A' = A$ and $J' = J$.
2: **while** there is an agent $a_i \in A'$ and a job $j_k \in J'$ with $u_i(j_k) > \frac{\beta}{\beta+2} \cdot \gamma_i$ **do**
3: Set $X_i = \{ j_k \}$, $A' = A' \setminus \{ a_i \}$, and $J' = J' \setminus \{ j_k \}$.
4: **end while**

By Lemma 1, it is straightforward to have the following lemma.

Lemma 2. *For any instance $\mathcal{I} = (J, A, \mathbf{u}_A)$ with $(\gamma_1, \cdots, \gamma_m)$, the partial schedule $(X_r)_{a_r \in A \setminus A'}$ and the reduced instance $\mathcal{I}' = (J', A', \mathbf{u}_{A'})$ returned by Algorithm 1 satisfy $u_r(X_r) \geqslant \frac{\beta}{\beta+2} \cdot \gamma_r$ for all $a_r \in A \setminus A'$ and $\mathsf{MMS}_i(J', |A'|) \geqslant \mathsf{MMS}_i(J, |A|)$ for all $a_i \in A'$.*

3.1.2 Bag-Filling Procedure

Let $\mathcal{I} = (J, A, \mathbf{u}_A)$ be an instance such that $|A| = m$ and $u_i(j_k) \leqslant \frac{\beta}{\beta+2} \cdot \gamma_i$ for all $a_i \in A$ and $j_k \in J$. We show the Bag-Filling Procedure in Algorithm 2, with parameters $(\gamma_1, \cdots, \gamma_m)$ and β-approximation algorithm for IS functions. For each $a_i \in A$, we use $u_i' : 2^J \to \mathbb{R}_+$ to denote the approximate utility, and thus $u_i'(S) \geqslant \beta \cdot u_i(S)$ for any $S \subseteq J$. Intuitively, it keeps a bag B and repeatedly adds an unscheduled job into it until some agent a_i first values this bag (under the approximate utility function u_i') for at least $\frac{\beta}{\beta+2} \cdot \gamma_i$. If there are more than one such agents, arbitrarily select one of them. Then a_i gets assigned a feasible subset $X_i \subseteq B$ with $\sum_{j_l \in X_i} u_i(j_l) = u_i'(B)$, and returns $B \setminus X_i$ to the algorithm. This step is crucial, otherwise the other remaining agents may not obtain enough jobs. It is obvious that if agent a_i gets assigned a bag, then her true utility satisfies

$$u_i(X_i) = \sum_{j_l \in X_i} u_i(j_l) = u_i'(X_i) \geqslant \frac{\beta}{\beta+2} \cdot \gamma_i.$$

The major technical difficulty of our algorithm is to prove that everyone can obtain a bag.

Algorithm 2. BagFilling Procedure

Input: An FISP instance $\mathcal{I} = (J, A, \mathbf{u}_A)$ such that $u_i(j_k) \leqslant \frac{\beta}{\beta+2} \cdot \gamma_i$ for all $a_i \in A$ and $j_k \in J$; β-approximation algorithm for IS functions; Thresholds $(\gamma_1, \cdots, \gamma_m)$.

Output: $\frac{\beta}{\beta+2}$-MMS schedule $\mathbf{X} = (X_1, \cdots, X_m)$.

Initialize $A' = A$, $J' = J$, and obtain approximate utility functions u_i' for all $a_i \in A$.
while $A' \neq \emptyset$ and $J' \neq \emptyset$ **do**
  Set $B = \emptyset$.
  while $u_i'(B) < \frac{\beta}{\beta+2} \cdot \gamma_i$ for all $a_i \in A'$ and $J' \neq \emptyset$ **do**
    Let j_k be an arbitrary job in J'. Set $B = B \cup \{ j_k \}$ and $J' = J' \setminus \{ j_k \}$.
  end while
  Let a_i be an arbitrary agent such that $u_i'(B) \geqslant \frac{\beta}{\beta+2} \cdot \gamma_i$.
  Let $X_i \subseteq B$ be a feasible subset such that $\sum_{j_l \in X_i} u_i(j_l) = u_i'(B)$.
  Set $J' = J' \cup (B \setminus X_i)$ and $A' = A' \setminus \{ a_i \}$.
end while

Lemma 3. *Setting $\gamma_i = \mathsf{MMS}_i$ for all $a_i \in A$, Algorithm 2 returns a $\frac{\beta}{\beta+2}$-MMS schedule.*

Proof. As we have discussed, it suffices to prove that at the beginning of any round of the outer while loop, there are sufficiently many remaining jobs in J' for every remaining agent in A', i.e.,

$$u_i'(J') \geqslant \frac{\beta}{\beta+2} \gamma_i, \text{ for any } a_i \in A'.$$

To prove the above inequality, in the following, we actually prove a stronger argument.

Claim 1. *For any $a_i \in A'$, let $\mathbf{X}' = (X_1', \cdots, X_m')$ be a feasible MMS schedule for a_i. Then there exists $k \in [m]$, such that $u_i(X_k' \cap J') \geqslant \frac{1}{\beta+2} \cdot \gamma_i$.*

Given Claim 1 and the β-approximation of u_i', $u_i'(X_k' \cap J') \geqslant \frac{\beta}{\beta+2} \cdot \gamma_i$ and thus the lemma holds. We prove by contradiction and assume Claim 1 does not hold for agent a_i. Since $\mathbf{X}' = (X_1', \cdots, X_m')$ is a feasible MMS schedule for a_i, $u_i(X_k') \geqslant \mathsf{MMS}_i = \gamma_i$ for all $k \in [m]$ and thus

$$\sum_{k \in [m]} u_i(X_k') \geqslant m \cdot \gamma_i. \tag{1}$$

Denote by $(X_r)_{a_r \in A \setminus A'}$ the assignments that are allocated to $A \setminus A'$ in previous rounds by Algorithm 2, and for each a_r, let j_{l_r} be the last item added to the bag B. Note that $j_{l_r} \in X_r$ otherwise a_r will stop the inner while loop (Step 4) before j_{l_r} was added. Moreover, since a_i did not break the while loop either, $u'_i(X_r \setminus \{ j_{l_r} \}) < \frac{\beta}{\beta+2} \cdot \gamma_i$. Thus $u_i(X_r \setminus \{ j_{l_r} \}) \leqslant \frac{1}{\beta+2} \cdot \gamma_i$ as u'_i is β-approximation of u_i. By the assumption that all jobs are small, i.e., $u_i(j_{l_r}) \leqslant \frac{\beta}{\beta+2} \cdot \gamma_i$, we have the following

$$u_i(X_r) = u_i(X_r \setminus \{ j_{l_r} \}) + u_i(j_{l_r}) < \frac{\beta+1}{\beta+2} \cdot \gamma_i. \tag{2}$$

If $u_i(X'_k \cap J') < \frac{1}{\beta+2} \cdot \gamma_i$ for all $k \in [m]$, then

$$\begin{aligned}
\sum_{k \in [m]} u_i(X'_k) &= \sum_{k \in [m]} \left(u_i(X'_k \cap J') + \sum_{a_r \in A \setminus A'} u_i(X'_k \cap X_r) \right) \\
&= \sum_{k \in [m]} u_i(X'_k \cap J') + \sum_{a_r \in A \setminus A'} \sum_{k \in [m]} u_i(X'_k \cap X_r) \\
&\leqslant \sum_{k \in [m]} u_i(X'_k \cap J') + \sum_{a_r \in A \setminus A'} u_i(X_r) \\
&< m \cdot \frac{1}{\beta+2} \cdot \gamma_i + (m - |A'|) \cdot \frac{\beta+1}{\beta+2} \cdot \gamma_i < m \cdot \gamma_i,
\end{aligned}$$

where the first inequality is because the X'_k's are disjoint and the second inequality is because of Equation (2). Thus we obtain a contradiction with Equation (1). □

3.1.3 Main Existential Theorem

Combining Lemma 2 and Lemma 3, it is not hard to prove the main existential result. In the full version Li *et al.* [2021], we will prove that our analysis is asymptotically tight.

Algorithm 3. Main Algorithm: Matching-BagFilling

Input: An arbitrary FISP instance $\mathcal{I} = (J, A, \mathbf{u}_A)$; β-approximation algorithm for IS functions; Thresholds $(\gamma_1, \cdots, \gamma_m)$.

Output: $\frac{\beta}{\beta+2}$-MMS schedule $\mathbf{X} = (X_1, \cdots, X_m)$.

1: Run Algorithm 1 on $\mathcal{I}$ with $(\gamma_1, \cdots, \gamma_m)$. Obtain $\mathcal{I}' = (J', A', \mathbf{u}_{A'})$ and $(X_r)_{a_r \in A \setminus A'}$.
2: Run Algorithm 2 on $\mathcal{I}'$ with $(\gamma_1, \cdots, \gamma_m)$ and the β-approximation algorithm. Obtain $(X_i)_{a_i \in A'}$.

Theorem 1. *Algorithm 3 with the optimal algorithm for IS functions (i.e., $\beta = 1$) and $\gamma_i = \mathsf{MMS}_i$ for all $a_i \in A$ returns a 1/3-MMS schedule for arbitrary FISP instance.*

Interestingly, in the independent and parallel work Hummel and Hetland [2021], via a similar bag-filling algorithm, the authors prove the existence of $1/3$-approximate MMS allocations under the context of graphically conflicting items. However, the two models in our work and theirs are not compatible in general.

3.2 Polynomial-time Implementation

Note that, in general, Algorithm 3 is not efficient, because if P $\neq$ NP, the computation of exact values for IS functions and MMS values cannot be done in polynomial time. For the special case when jobs are rigid or unit, IS functions can be computed in polynomial time. If the number of machines is constant, MMS values for rigid jobs can be computed in pseudo-polynomial time Chiarelli *et al.* [2020]. Thus, in this section, we deal with the general case. Of course, for IS functions, we can directly use the β-approximation algorithms, and the best-known approximation ratio is 0.644 Im *et al.* [2020]. Regarding the MMS barrier, instead of using their approximate values, we utilize a combinatorial trick similar with one used in Barman and Krishnamurthy [2020] such that without knowing their values, we can still execute our algorithm.

First, an important corollary of Lemma 2 and Lemma 3 is that if $\gamma_i \leqslant \mathsf{MMS}_i$ for some a_i, no matter what values are set for γ_j, $j \neq i$, Algorithm 3 always assigns a bag to a_i such that $u_i(X_i) \geqslant \frac{\beta}{\beta+2}\gamma_i$.

Lemma 4. *For any a_i, if $\gamma_i \leqslant \mathsf{MMS}_i$, Algorithm 3 ensures that $u_i(X_i) \geqslant \frac{\beta}{\beta+2}\gamma_i$, regardless of γ_{-i}.*

We prove Lemma 4 in the full version Li *et al.* [2021]. Now, we are ready to introduce the trick. First, we set each γ_i to be sufficiently large such that $\gamma_i \geqslant \mathsf{MMS}_i$ for all a_i. Then we run Algorithm 3. If we found some agent a_i with $u_i(X_i) < \frac{\beta}{\beta+2}\gamma_i$, it means γ_i is higher than MMS_i and we can decrease γ_i by $0 < 1-\epsilon < 1$ fraction and keep the other MMS values unchanged. We repeat the above procedure until everyone is satisfied $u_i(X_i) \geqslant \frac{\beta}{\beta+2}\gamma_i$. By Lemma 4, it must be that $\gamma_i \geqslant (1-\epsilon)\mathsf{MMS}_i$ for all a_i. We summarize this in Algorithm 4, and it is straightforward to have the following theorem.

Algorithm 4. Efficient Implementation: Matching-BagFilling

Input: An arbitrary FISP instance $\mathcal{I} = (J, A, \mathbf{u}_A)$; β-approximation polynomial-time algorithm for IS functions; Thresholds $(\gamma_1 = \frac{u'_1(J)}{\beta}, \cdots, \gamma_m = \frac{u'_m(J)}{\beta})$; $0 < \epsilon < 1$.

Output: $\frac{\beta}{(\beta+2)}(1-\epsilon)$-MMS schedule $\mathbf{X} = (X_1, \cdots, X_m)$.

Run Algorithm 3 on $\mathcal{I}$ with $(\gamma_1, \cdots, \gamma_m)$. Obtain $\mathbf{X} = (X_1, \cdots, X_m)$.
while there exist $a_i \in A$ such that $u'_i(X_i) < \frac{\beta}{\beta+2}\gamma_i$ **do**
    Set $\gamma_i = (1-\epsilon)\gamma_i$.
    Run Algorithm 3 on $\mathcal{I}$ with $(\gamma_1, \cdots, \gamma_m)$ and update $\mathbf{X} = (X_1, \cdots, X_m)$.
end while

Theorem 2. *For any $0 < \epsilon < 1$, Algorithm 4 returns a $\frac{\beta}{\beta+2}(1-\epsilon)$-MMS schedule for arbitrary FISP instance with an β-approximation algorithm for IS functions. The running time is polynomial with $|J|$, $|A|$ and $1/\epsilon$. Particularly, using the 0.64-approximation algorithm in Im* et al. *[2020], we have $0.24(1-\epsilon)$-approximation polynomial-time algorithm.*

4 Approximately EF1 and PO Scheduling

4.1 Compatibility of EF1 and PO

In this section, we investigate the extent to which there is a schedule that is both EF1 and PO. We first show that EF1 and PO are not compatible even if jobs are rigid and valuations are unary, i.e., $u_i(j_k) = 1$ for all $a_i \in A$ and $j_k \in J$. That is no algorithm can return an EF1 and PO schedule for all instances. Fortunately, if the jobs have unit processing time, an EF1 and PO schedule exists and can be computed in polynomial time. This result continues to hold if the agents have weighted but identical utilities, i.e., $u_i(j_k) = u_r(j_k)$ for any job j_k and any two agents a_i and a_r. We sometimes ignore the subscript and use $u(\cdot)$ to denote the identical valuation.

Algorithm 5. m-Matching + Inner-Greedy

Input: An FISP instance $\mathcal{I} = (J, A, \mathbf{u}_A)$, where all jobs have unit processing time and all agents have identical valuation.

Output: EF1 and PO schedule $\mathbf{X} = (X_1, \cdots, X_m)$.

Construct graph $G(J \cup T, E)$, and compute a maximum weighted m-matching $\mathcal{M}^*$.
Define $J_t = \{\, j \in J \mid (j,t) \in \mathcal{M}^* \,\}$ for each $t \in T$.
Set $X_1 = X_2 = \cdots = X_m = \emptyset$.
for $p = 1$ to $|T|$ **do**
    if $J_p \neq \emptyset$ **then**
        Sort A in non-decreasing order of $u_i(X_i)$'s, and J_p in non-increasing order of $u(j_k)$'s.
        for $i = 1$ to $|J_p|$ **do**
            Set $X_i = X_i \cup \{\, j_i \,\}$.
        end for
    end if
end for

Theorem 3. *EF1 and PO are not compatible even if all jobs are rigid and all valuations are unary. If all jobs have unit processing time, Algorithm 5 returns an EF1 and PO schedule in polynomial time, as long as the valuations are identical.*

Here, we briefly discuss the intuition of Algorithm 5 in the following. At the heart of Algorithm 5, there are two tasks: (1) Find jobs with maximum weight to schedule in order to guarantee PO. (2) Assign these jobs to agents such that the assignment is EF1.

To solve the first task, we use bipartite matchings. Let $T_i = [r_i, d_i] = \{ r_i, r_i + 1, \cdots, d_i \}$ be the job interval for each $j_i \in J$, and $T = \bigcup_{1 \leqslant i \leqslant n} T_i$. We first construct a weighted bipartite graph $G(J \cup T, E)$, where each job $j_i \in J$ is a node on the left side, and a time slot $t_l \in T$ is a node on the right side. There is an edge $(j_i, t_l) \in E$ with weight $u(j_i)$ if and only if $t_l \in T_i$. For any set $\mathcal{B} \subseteq E$ and $v \in J \cup T$, let $\mathcal{B}(v) \subseteq E$ be the set of edges in $\mathcal{B}$ that intersects v. Next we define *m-matching* $\mathcal{M} \subseteq E$, which requires that $|\mathcal{M}(j_i)| \leqslant 1$ for all $j_i \in J$ and $|\mathcal{M}(t_l)| \leqslant m$ for all $t_l \in T$, where m is the number of machines. m-matching is used to ensure that at each time slot at most m jobs are processed. Accordingly, by computing a maximum weighted m-matching $\mathcal{M}^*$, we can find the set of jobs by scheduling which we can maximize the social welfare so that the resulting schedule is PO.

However, how shall we assign these jobs to agents? We partition these selected jobs into groups $\{J_1, \cdots, J_{|T|}\}$, and each group J_t contains the jobs that are matched to time slot t by $\mathcal{M}^*$. To maintain feasibility, each agent can get at most one job from each J_t. We process $\{J_1, \cdots, J_{|T|}\}$ one by one, and to satisfy EF1, the agent with low cumulative utility should obtain a better job in the next group. By induction, we can prove that eventually, the assignment is indeed EF1.

A final remark is about the running time. If the graph size is polynomial, by Bernhard and Vygen [2008], finding $\mathcal{M}^*$ can be done in polynomial time. However, $|T|$ can be exponentially large. Therefore, before running Algorithm 5, we first reduce an arbitrary instance to a *condensed* one by discarding some time slots which is essentially equivalent to the original one but only contains polynomial number of time slots. We defer this discussion completely to the full version Li *et al.* [2021].

4.2 Approximate EF1 and PO

Although EF1 and PO are only compatible in special cases, in this section we show that approximate EF1 and PO can be always satisfied. Theorem 4 is proved in the full version Li *et al.* [2021], where we introduce Nash social welfare and show that Nash social welfare maximizing schedule satisfies the desired properties.

Theorem 4. *For arbitrary FISP instance, there is a feasible schedule that is $1/4$-EF1 and PO. If all jobs have unit processing time, there is a feasible schedule that is $1/2$-EF1 and PO.*

5 Experiment

Finally, we evaluate the performance of Matching-BagFilling on randomly generated data sets, and compare it with the extensively adopted heuristic algorithm *Round-Robin*: Each agent takes turns to select a feasible unscheduled job that maximizes her marginal utility gain until no more jobs can be selected. Before conducting the experiments, we note that although Matching-BagFilling has good theoretical guarantee, for many concrete instances, it can be significantly improved. Since each agent only selects a bag with $1/3\mathsf{MMS}_i$, there might be lots of unscheduled jobs that can be feasibly processed. Therefore, we first refine our algorithm as *Matching-BagFilling+*: (1) Run Matching-BagFilling; (2) Run Round-Robin on the remaining unscheduled jobs. We formally describe Round-Robin and Matching-BagFilling+ in the full version Li *et al.* [2021]. Note that Matching-BagFilling+ does not have better theoretical performance than Matching-BagFilling.

In our experiment, as shown in Figure 1, we randomly generate a set of rigid jobs J ($|J| = 100, 500, 1000$) and a set of agents A following some distribution, where for $i = 1, 2, 3$, $U/P/N.i$ means there are $|A| = 5 \times i$ agents whose values are randomly generated from a(n) Uniform, Poisson, or Normal distribution. For each setting, we generate 1000 instances. For each of them we run Matching-BagFilling+ (short for BAG+), Matching-BagFilling (short for BAG), and Round-Robin (short for RR) and record every agent a_i's average utilities $u_i(\text{BAG+}), u_i(\text{BAG}), u_i(\text{RR})$. Then we compute the ratios $u_i(\text{BAG+})/u_i(\text{RR})$ and $u_i(\text{BAG})/u_i(\text{RR})$. Of course, if these ratios are greater than 1, it means our algorithms outperform the Round-Robin. We use two vertical line intervals to represent the range of all agents' ratios. It can be seen that in all experiments, Matching-BagFilling+ clearly outperforms Round-Robin, i.e., every agent gets higher utility in Matching-BagFilling+. Compared with the number of agents, if the number of jobs is small (e.g.

$|A| = 15$ and $|J| = 100, 500, 1000$), Matching-BagFilling has decent performance. However, as the number of jobs gets larger (e.g., $|A| = 5$ and $|J| = 100, 500, 1000$), Matching-BagFilling is clearly worse than Round-Robin, which is because too many jobs are left unscheduled. We defer a detailed description of our experiments and the discussion of results to the full version Li *et al.* [2021].

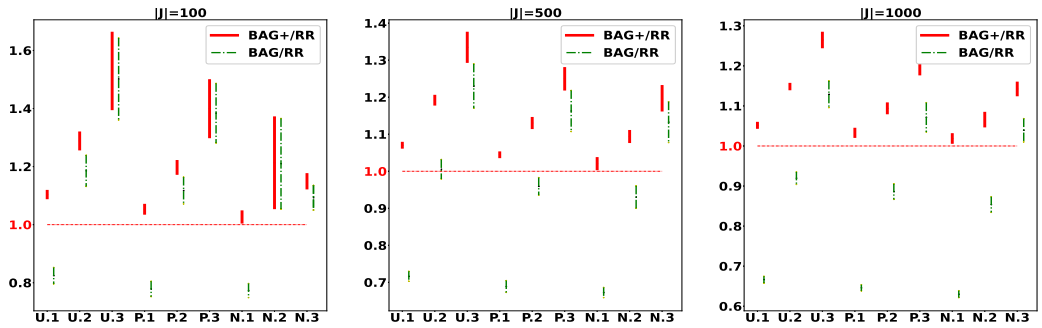


Figure 1: Experiments

6 Conclusion and Future Directions

In this work, we studied the fair scheduling problem for time-dependent resources, and designed constant approximation algorithms for MMS, EF1&PO and EF1&IO schedules. There are many open problems and future directions. An immediate direction is to improve our approximation ratios and investigate the limit of approximation algorithms for different settings. It is also interesting to impose other efficiency criteria on EF1 schedules, such as computing an EF1 schedule that maximizes social welfare. In this work, we have assumed the jobs are resources that bring utility to agents, and leave the case when jobs are chores for future study. Finally, it is of both theoretical interest and practical importance to consider the online setting when jobs arrive dynamically and the strategic setting when agents' valuations are private information.

Acknowledgements and Funding

The authors thanks Warut Suksompong for reading a draft of this paper and for helpful discussions. Bo Li was partially funded by The Hong Kong Polytechnic University under Grant No. P0034420. Minming Li was partially supported by NSFC under Grant No. 11771365, and by Project No. CityU 11200518 from Research Grants Council of HKSAR.

References

Miklós Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J. Schulman, and Orli Waarts. Fairness in scheduling. *J. Algorithms*, 29(2):306–357, 1998.

Rasha A. Al-Arasi and Anwar Saif. Task scheduling in cloud computing based on metaheuristic techniques: A review paper. *EAI Endorsed Trans. Cloud Syst.*, 6(17):e4, 2020.

Georgios Amanatidis, Evangelos Markakis, Afshin Nikzad, and Amin Saberi. Approximation algorithms for computing maximin share allocations. *ACM Trans. Algorithms*, 13(4):52:1–52:28, 2017.

Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, and Rebecca Reiffenhäuser. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. In *NeurIPS*, 2020.

Moshe Babaioff, Tomer Ezra, and Uriel Feige. Fair and truthful mechanisms for dichotomous valuations. In *AAAI*, pages 5119–5126. AAAI Press, 2021.

Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001.

Siddharth Barman and Sanath Kumar Krishnamurthy. Approximation algorithms for maximin fair division. *ACM Trans. Economics and Comput.*, 8(1):5:1–5:28, 2020.

Siddharth Barman and Paritosh Verma. Existence and computation of maximin fair allocations under matroid-rank valuations. In *AAMAS*, pages 169–177. ACM, 2021.

Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding fair and efficient allocations. In *EC*, pages 557–574. ACM, 2018.

Sanjoy K. Baruah and Shun-Shii Lin. Pfair scheduling of generalized pinwheel task systems. *IEEE Trans. Computers*, 47(7):812–816, 1998.

Sanjoy K. Baruah. Fairness in periodic real-time scheduling. In *RTSS*, pages 200–209. IEEE Computer Society, 1995.

Piotr Berman and Bhaskar DasGupta. Multi-phase algorithms for throughput maximization for real-time scheduling. *J. Comb. Optim.*, 4(3):307–323, 2000.

Korte Bernhard and Jens Vygen. Combinatorial optimization: Theory and algorithms. *Springer, Third Edition, 2005.*, 2008.

Vittorio Bilò, Angelo Fanelli, Michele Flammini, Gianpiero Monaco, and Luca Moscardelli. The price of envy-freeness in machine scheduling. *Theor. Comput. Sci.*, 613:65–78, 2016.

Arpita Biswas and Siddharth Barman. Fair division under cardinality constraints. In *IJCAI*, pages 91–97. ijcai.org, 2018.

Arpita Biswas and Siddharth Barman. Matroid constrained fair allocation problem. In *AAAI*, pages 9921–9922. AAAI Press, 2019.

Eric Budish. The combinatorial assignment problem: approximate competitive equilibrium from equal incomes. In *BQGT*, page 74:1. ACM, 2010.

Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. In *EC*, pages 305–322. ACM, 2016.

Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 1032–1041. PMLR, 2019.

Nina Chiarelli, Matjaz Krnc, Martin Milanic, Ulrich Pferschy, Nevena Pivac, and Joachim Schauer. Fair packing of independent sets. In *IWOCA*, volume 12126 of *Lecture Notes in Computer Science*, pages 154–165. Springer, 2020.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *NIPS*, pages 5029–5037, 2017.

Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res.*, 31(4):730–738, 2006.

Amitay Dror, Michal Feldman, and Erel Segal-Halevi. On fair division under heterogeneous matroid constraints. *CoRR*, abs/2010.07280, 2020.

Amitay Dror, Michal Feldman, and Erel Segal-Halevi. On fair division under heterogeneous matroid constraints. In *AAAI*, pages 5312–5320. AAAI Press, 2021.

Maciej Drozdowski. *Scheduling for Parallel Processing*. Computer Communications and Networks. Springer, 2009.

Uriel Feige, Ariel Sapir, and Laliv Tauber. A tight negative example for MMS fair allocations. *CoRR*, abs/2104.04977, 2021.

Jiarui Gan, Bo Li, and Xiaowei Wu. Approximately envy-free budget-feasible allocation. *CoRR*, abs/2106.14446, 2021.

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

Jugal Garg and Setareh Taki. An improved approximation algorithm for maximin shares. In *EC*, pages 379–380. ACM, 2020.

Karsten Gentner, Klaus Neumann, Christoph Schwindt, and Norbert Trautmann. Batch production scheduling in the process industries. In *Handbook of Scheduling*. Chapman and Hall/CRC, 2004.

Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Hadi Yami. Fair allocation of indivisible goods: Improvements and generalizations. In *EC*, pages 539–556. ACM, 2018.

Halvard Hummel and Magnus Lie Hetland. Fair allocation of conflicting items. *CoRR*, abs/2104.06280, 2021.

Sungjin Im and Benjamin Moseley. Fair scheduling via iterative quasi-uniform sampling. *SIAM J. Comput.*, 49(3):658–680, 2020.

Sungjin Im, Shi Li, and Benjamin Moseley. Breaking 1 - 1/e barrier for nonpreemptive throughput maximization. *SIAM J. Discret. Math.*, 34(3):1649–1669, 2020.

Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.

David Kurokawa, Ariel D. Procaccia, and Junxing Wang. Fair enough: Guaranteeing approximate maximin shares. *J. ACM*, 65(2):8:1–8:27, 2018.

Bo Li, Minming Li, and Ruilong Zhang. Fair allocation with interval scheduling constraints. *CoRR*, abs/2107.11648, 2021.

Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *EC*, pages 125–131. ACM, 2004.

Rohan R. Paleja, Andrew Silva, Letian Chen, and Matthew C. Gombolay. Interpretable and personalized apprenticeship scheduling: Learning interpretable scheduling policies from heterogeneous user demonstrations. In *NeurIPS*, 2020.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *NeurIPS*, pages 9684–9693, 2018.

Aida Rahmattalabi, Phebe Vayanos, Anthony Fulginiti, Eric Rice, Bryan Wilder, Amulya Yadav, and Milind Tambe. Exploring algorithmic fairness in robust graph covering problems. In *NeurIPS*, pages 15750–15761, 2019.

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

Warut Suksompong. Constraints in fair division. *SIGecom Exch.*, 19(2):46–61, 2021.

Xiaowei Wu, Bo Li, and Jiarui Gan. Budget-feasible maximum nash social welfare allocation is almost envy-free. In *IJCAI*. ijcai.org, 2021.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *NeurIPS*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Our main results contain three parts (See Section 1.1). All of them are mentioned accurately in abstract and introduction.
   (b) Did you describe the limitations of your work? [N/A]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Due to space constraint, some proofs are deferred to the supplemental material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The data is randomly generated by JAVA programming. The codes and outputs of the algorithms can be found in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The brief summary of our experiment can be found in Section 5. The detailed descriptions can be found in the supplemental material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The brief summary of our experiment can be found in Section 5. The detailed descriptions can be found in the supplemental material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

The appendix is organized as follows:

- Appendix A: In this section, we prove IS functions are XOS.
- Appendix B: In this section, we formally prove our main results in Section 3.
- Appendix C: In this section, we formally prove our main results in Section 4.
- Appendix D: In this section, we introduce a milder efficiency requirement IO and study its compatibility with EF1.
- Appendix E: In this section, we discuss our experiments in more details.

As we will discuss the approximation algorithms and the existences of EF1/PO/IO schedules in different settings, we introduce the following notations to simplify the description of different settings.

Regarding agents' utilities, FISP contains three cases, from the most special to the most general:

- Unweighted: $u_i(j_k) = 1$ for all $a_i \in A, j_k \in J$, i.e., agents have unary utility for jobs.
- Identical: $u_i(j_k) = u_r(j_k)$ for all $a_i, a_r \in A, j_k \in J$, i.e., all agents have the same utility for the same job.
- Non-identical: $u_i(j_k) \geqslant 0$ without any restrictions.

Regarding jobs, there are three cases:

- Unit: $p_i = 1$, for all $j_i \in J$, i.e., all jobs have unit processing time.
- Rigid: $r_i + p_i - 1 = d_i$, for all $j_i \in J$, i.e., the jobs need to occupy the entire time intervals between their release times and deadlines.
- Flexible: $r_i + p_i - 1 \leqslant d_i$, for all $j_i \in J$.

Note that unit jobs may not be rigid and rigid jobs may not be unit either. In the remainder of the appendix, we use notation FISP with $\langle$utility type, job type$\rangle$ to denote a certain case of the general FISP, e.g., FISP with $\langle$unweighted, unit$\rangle$ represents the case where the processing time of each job is 1 and each agent has unweighted utility function.

A Missing Materials in Section 2

A set function $f : 2^V \to \mathbb{R}$ defined on V is called *fractionally subadditive* (XOS) if there is a finite set of additive functions $\{ f_1, \cdots, f_w \}$ such that $f(S) = \max_{1 \leqslant i \leqslant m} f_i(S)$ for any $S \subseteq V$.

Lemma 5. *IS functions are XOS.*

Proof. Let u be an IS function defined on job set $J = \{ j_1, \cdots, j_n \}$ with individual utility $(v_1 = u(j_1), \cdots, v_n = u(j_n))$. To show u is XOS, it suffices to define a finite set of additive functions on J. For each feasible job set $T \subseteq J$, define additive function f_T such that $f_T(j_i) = v_i$ if $j_i \in T$ and $f_T(j_i) = 0$ otherwise. Therefore, for any $S \subseteq T$,

$$u(S) = \max_{T \subseteq S : T \text{ is feasible}} \sum_{j_i \in T} v_i = \max_{T \subseteq S : T \text{ is feasible}} f_T(T) = \max_{T \subseteq S : T \text{ is feasible}} f_T(S) = \max_{T \text{ is feasible}} f_T(S),$$

where the last equality is because any subset of a feasible job set is also feasible. Thus u is XOS. □

B Missing Materials for MMS Scheduling in Section 3

B.1 Proof of Lemma 1

Proof. Let $\mathcal{I} = (J, A, \mathbf{u}_A)$ be an arbitrary instance of FISP with $J = \{ j_1, \cdots, j_m \}$ and $|A| = m$. To show that $\mathsf{MMS}_i(J \setminus \{ j_k \}, m-1) \geqslant \mathsf{MMS}_i(J, m)$ holds for any $j_k \in J, a_i \in A$, we consider an arbitrary agent a_i. Let $\mathbf{X} = (X_1, X_2, \cdots, X_m)$ be a feasible schedule for a_i, i.e.,

$\min_{X_r \in \mathbf{X}} u_i(X_r) = \mathsf{MMS}_i$. Consider an arbitrary job j_k, assume that $j_k \in X_l$. Then remove job set X_l from MMS_i schedule. This generates a new schedule, denoted by $\mathbf{X}' = \{ X'_1, X'_2, \cdots, X'_{m-1} \}$. It is easy to see that $\mathbf{X}'$ is a feasible schedule to the instance with $m-1$ agents and the job set $J \setminus \{ J_k \}$. This implies that $\mathsf{MMS}_i(J \setminus \{ j_k \}, m-1) \geqslant \min_{X'_r \in \mathbf{X}'} u_i(X'_r)$. Note that $\min_{X'_r \in \mathbf{X}'} u_i(X'_r) \geqslant \mathsf{MMS}_i(J, m)$. Therefore, we have

$$\mathsf{MMS}_i(J \setminus \{ j_k \}, m-1) \geqslant \min_{X'_r \in \mathbf{X}'} u_i(X'_r) \geqslant \mathsf{MMS}_i(J, m).$$

In the case where $j_k \notin \bigcup_{r \in [m]} X_r$, we remove an arbitrary job set from $\mathbf{X}$ and the above analysis still works. □

B.2 An Example for Matching-BagFilling and Matching-BagFilling+

It is obvious that our analysis is tight when all jobs have tiny values such that whenever an agent obtains a bag her value is exactly or slightly greater than $\frac{1}{3}\mathsf{MMS}_i$. In the following, we present an instance such that even without the preprocessing procedure and the last agent takes away all remaining jobs, everyone obtains exactly $\frac{1}{3}\mathsf{MMS}_i + \epsilon$. Accordingly, the instance proves that "Matching-BagFilling+ does not have better theoretical performance than Matching-BagFilling" as claimed in Section 5.

Consider the following instance with $|A| = m$ agents where m is a sufficiently large even number.

The job set J can be classified into the following categories:

- $J_1 = \{ j_1^1, j_2^1, \cdots, j_m^1 \}$: There are m rigid jobs in J_1. Every job in J_1 has the same job interval $[1, 2]$. For every job in J_1, a_m has the same utility gain $\frac{1}{3} + \frac{1}{m}$. For every job in J_1, all agents in $A \setminus \{ a_m \}$ have the same utility gain $\frac{2}{3} + \frac{1}{m}$;
- $J_2 = \{ j_1^2, j_2^2, \cdots, j_{m-1}^2 \}$: There are $m-1$ rigid jobs in J_2. Every job in J_2 has the same job interval $[3, \frac{m}{2} + 2]$. For every job in J_2, all agents in A have the same utility gain $\frac{1}{3}$;
- $J_3 = \{ j_1^3, j_2^3, \cdots, j_{m-1}^3 \}$: There are m unit jobs in J_3. Every job in J_3 has the same job interval $[3, m+2]$. For every job in J_3, all agents in A have the same utility gain $\frac{1}{3m}$;
- $J_4 = \bigcup_{r \in [m]} J_4^r$: There are m group rigid jobs in J_4. Each group $J_4^r, r \in [m]$, contains m rigid jobs. Assume that $J_4^r = \{ j_{r1}^4, j_{r2}^4, \cdots, j_{rm}^4 \}, \forall r \in [m-1]$. A job $j_{ri}^4 \in J_4^r, i \in [m]$ has the job interval $[m+3+i, m+4+i]$. Assume that $J_4^m = \{ j_{m1}^4, j_{m2}^4, \cdots, j_{mm}^4 \}$. A job $j_{mi}^4 \in J_3^m$ has the job interval $[m+4+i, m+5+i]$. In total, there are m^2 jobs in J_4. For every job in J_4, a_m has the same utility gain $\frac{1}{3m}$. For every job in J_4, all agents in $A \setminus \{ a_m \}$ have the same utility gain 0.

Let us focus on a_m first. The upper bound of MMS_m is:

$$\frac{1}{m} \cdot \left((\frac{1}{3} + \frac{1}{m}) \cdot m + \frac{m-1}{3} + \frac{1}{3m} \cdot m + \frac{1}{3m} \cdot m^2 \right) = 1 + \frac{1}{m}.$$

We consider the schedule $\mathbf{X} = (X_1, \cdots, X_m)$, where $X_i = \{ j_i^1, j_i^2 \} \cup J_4^i, \forall i \in [m-1]$ and $X_m = \{ j_m^1 \} \cup J_3 \cup J_4^m$ (See Figure 2). It is not hard to see that $\mathbf{X}$ is a feasible schedule and $\min_{i \in [m]} u_m(X_i) = u_m(X_m) = 1 + \frac{1}{m}$. Therefore, $\mathbf{X}$ is a feasible schedule that obtains the value $1 + \frac{1}{m}$ which is also the upper bound of MMS_m. Thus, $\mathsf{MMS}_m = 1 + \frac{1}{m}$. Hence, once a_m values the bag greater than or equal to $\frac{1}{3} + \frac{1}{3m}$, a_m will take the bag away.

Now, we consider an arbitrary agent $a_i \in A \setminus \{ a_m \}$. Since all agents in $A \setminus \{ a_m \}$ have utility gain 0 for all jobs in J_4, we can ignore the job set J_4. Therefore, the upper bound of $\mathsf{MMS}_i, \forall i \in [m-1]$ is:

$$\frac{1}{m} \left((\frac{2}{3} + \frac{1}{m}) \cdot m + (\frac{m-1}{3}) + (\frac{1}{3m}) \cdot m \right) = 1 + \frac{1}{m}.$$

We consider the schedule $\mathbf{X}' = (X'_1, \cdots, X'_m)$, where $X'_i = \{ j_i^1, j_i^2 \}, \forall i \in [m-1]$ and $X'_m = \{ j_m^1 \} \cup J_3$. It is not hard to see that $\mathbf{X}'$ is a feasible schedule and $u_i(X'_k) = u_i(X'_r) = 1 + \frac{1}{m}, \forall k, r \in [m], \forall i \in [m-1]$. Therefore, $\mathbf{X}'$ is a feasible schedule that obtains the value $1 + \frac{1}{m}$ which is also an upper bound of $\mathsf{MMS}_i, \forall i \in [m-1]$. Thus, $\mathsf{MMS}_i = 1 + \frac{1}{m}, \forall i \in [m-1]$. Hence, once agent $a_i, \forall i \in [m-1]$, values the bag greater than or equal to $\frac{1}{3} + \frac{1}{3m}$, a_i will take the bag away.

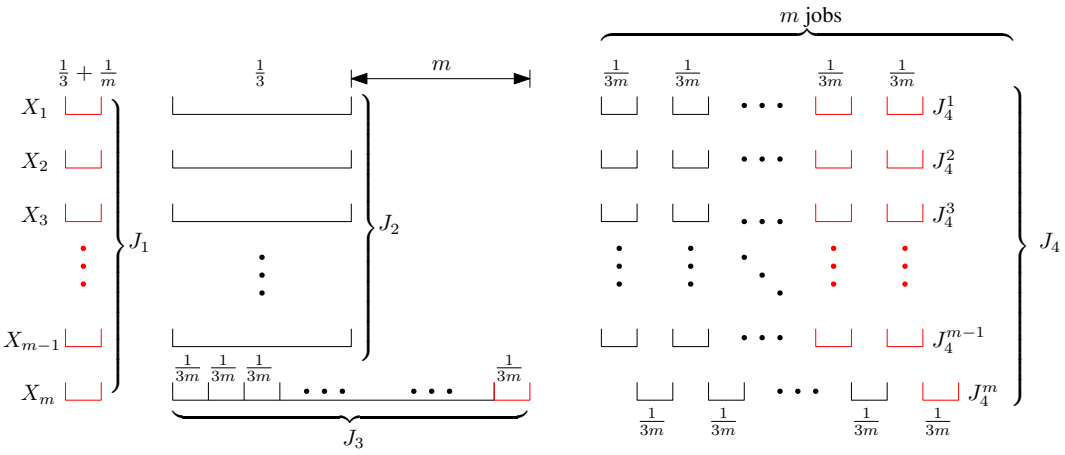


Figure 2: Illustration for the tight instance for Algorithm 3. The above schedule is $\mathbf{X}$ which is also the MMS schedule for agent a_m. The red jobs are the remaining jobs at the end of the $(m-1)$-th round of Algorithm 3 with the specified job sequence described in the "The specified job sequence" paragraph.

The specified job sequence Now, we consider the following job sequence. In the first round, Algorithm 3 adds $J_1^4 \setminus \{ j_{1(m-1)}^4, j_{1m}^4 \}$ to the bag, and then adds j_1^3, j_{m1}^4 to the bag, and then adds j_1^2 to the bag, i.e., $\mathsf{BAG} = \{ j_{11}^4, j_{12}^4, \cdots, j_{1(m-2)}^4 \} \cup \{ j_1^3, j_{m1}^4 \} \cup \{ j_1^2 \}$. It is not hard to see that BAG is a feasible job set and all agents in $A \setminus \{ a_m \}$ value the bag exactly $\frac{1}{3}+\frac{1}{3m}$. Without loss of generality, we assume that a_1 takes the bag away at the end of the first round. In the l-th round, $2 \leqslant l \leqslant m-1$, Algorithm 3 first adds $J_l^4 \setminus \{ j_{l(m-1)}^4, j_{lm}^4 \}$, and then adds j_l^3, j_{ml}^4, and then adds j_l^2 to the bag. Without loss of generality, we assume that a_l takes the bag away at the end of the l-th round, where $2 \leqslant l \leqslant m-1$. Note that, at the end of the $(m-1)$-th round, all agents in $A \setminus \{ a_m \}$ obtain the utility gain exactly $\frac{1}{3}+\frac{1}{3m}$.

It is not hard to see that, at the end of the $(m-1)$-th round,

$$J' = J_1 \cup \{ j_m^3 \} \cup \{ j_{mm}^4 \} \cup \{ j_{1(m-1)}^4, j_{1m}^4 \} \cup \{ j_{2(m-1)}^4, j_{2m}^4 \} \cup \cdots \{ j_{(m-1)(m-1)}^4, j_{(m-1)m}^4 \}.$$

See the red jobs in Figure 2. Thus, $u_m(J') = (\frac{1}{3}+\frac{1}{m}) + \frac{1}{3m} + \frac{3}{3m} = \frac{1}{3} + \frac{7}{3m}$.

Therefore, everyone obtains exactly $\frac{1}{3}\mathsf{MMS}_i + \epsilon$ at the end of Algorithm 3. Moreover, it is not hard to see that if we run round-robin procedure at the end of Algorithm 3, the utility gains of all agents in $A \setminus \{ a_m \}$ will be increased but the utility gain of a_m is not able to be further improved. Thus, the above instance implies that "Matching-BagFilling+ does not have better theoretical performance than Matching-BagFilling".

B.3 Proof of Lemma 4

Note that the algorithm only ensures that agent a_i with $\gamma_i \leqslant \mathsf{MMS}_i$ can obtain a bag but not everyone. This is natural as if for some $a_j \neq a_i$ and γ_j is super large compared with MMS_j, a_j will never stop the algorithm and get a bag.

Recall that we can assume that there is no large job in the instance, i.e., $u_i(j_k) \leqslant \frac{\beta}{\beta+2} \cdot \gamma_i$, where $0 \leqslant \beta \leqslant 1$. Observe that if agent a_i gets assigned a bag, then her true utility satisfies:

$$u_i(X_i) = \sum_{j_l \in X_i} u_i(j_l) = u_i'(X_i) \geqslant \frac{\beta}{\beta+2}\gamma_i.$$

The above inequality also holds no matter whether $\gamma_i \leqslant \mathsf{MMS}_i$ or not. Similar as the proof of Lemma 3, the core is to prove that a_i can be guaranteed to obtain a bag as long as $\gamma_i \leqslant \mathsf{MMS}_i$. We consider the R-th round of the outer while loop of Algorithm 4 (line 2-5) in which the value of γ_i is decreased below MMS_i. In the R-th round of Algorithm 4 (line 2-5), we assume that the order of the agents that break the while loop of Algorithm 2 (line 4-10) is $\{ a_1, \cdots, a_{i-1}, a_i, \cdots \}$. It suffices to

prove that at the beginning of the i-th while loop of Algorithm 2 (line 4-10), there are sufficiently many remaining jobs in J' for the agent a_i, i.e.,

$$u'_i(J') \geqslant \frac{\beta}{\beta+2} \cdot \gamma_i, \forall a_i \in A'.$$

Similar as the proof of Lemma 3, we prove the following stronger claim. Given the following claim and the β-approximation of u'_i, we have $u'_i(X'_k \cap J') \geqslant \frac{\beta}{\beta+2} \cdot \gamma_i$ Therefore Lemma 4 holds.

Claim 2. *For any $a_i \in A'$ with $\gamma_i \leqslant \mathsf{MMS}_i$, let $\mathbf{X}' = \{X'_1, \cdots, X'_m\}$ be a feasible MMS schedule for a_i. Then, there exists $k \in [m]$ such that $u_i(X'_k \cap J') \geqslant \frac{1}{\beta+2} \cdot \gamma_i$, where $\gamma_i \leqslant \mathsf{MMS}_i$.*

Proof. We consider an arbitrary agent a_i. Since $\mathbf{X}' = (X'_1, X'_2, \cdots, X'_m)$ is a feasible MMS schedule for a_i, we have $u_i(X'_k) \geqslant \mathsf{MMS}_i \geqslant \gamma_i, \forall k \in [m]$ and therefore

$$\sum_{k=1}^{m} u_i(X'_k) \geqslant m \cdot \mathsf{MMS}_i \geqslant m \cdot \gamma_i. \tag{3}$$

Same as the proof of Lemma 3, the key idea of the proof is to show that agent a_i values the bundles that are taken by the agents before a_i less than $\frac{\beta+1}{\beta+2} \cdot \gamma_i$, i.e.,

$$u_i(X_r) < \frac{\beta+1}{\beta+2} \cdot \gamma_i, \forall r \in [i-1]. \tag{4}$$

We consider an arbitrary bundle that is taken by agent $a_r, r \in [i-1]$ and assume that job j_r is the last job added to the Bag. Since a_i did not break the while loop, we have $u'_i(X_r \setminus \{j_r\}) < \frac{\beta}{\beta+2} \cdot \gamma_i$. This implies that $u_i(X_r \setminus \{j_r\}) \leqslant \frac{1}{\beta+2} \cdot \gamma_i$. Since all jobs are small, i.e., $u_i(j_r) \leqslant \frac{\beta}{\beta+2} \cdot \gamma_i$, we have

$$u_i(X_r) = u_i(X_r \setminus \{j_r\}) + u_i(j_r) < \frac{\beta+1}{\beta+2} \cdot \gamma_i.$$

Therefore, Equation (4) holds. To help understand the following proof, an example is shown in Figure 3. Every rectangle in Figure 3 represents a job in J. The area of every rectangle j_l in Figure 3 represents the value of $u_i(j_l)$. The non-white rectangles represent the jobs that are assigned to some agents in $\{a_1, \cdots, a_{i-1}\}$. According to Equation (4), the total area of non-white rectangles in Figure 3 is at most $\frac{(\beta+1)(i-1)}{\beta+2}\gamma_i$, i.e., $\sum_{r=1}^{i-1} u_i(X_r) < \frac{(\beta+1)(i-1)}{\beta+2}\gamma_i$. According to Equation (3), the total area of rectangles in Figure 3 is at least $m\gamma_i$. Therefore, the total area of white rectangles in $\{X'_1, \cdots, X'_m\}$ is at least $m\gamma_i - \frac{(\beta+1)(i-1)}{\beta+2}\gamma_i$, i.e.,

$$\sum_{r=1}^{m} u_i(X'_r \setminus \bigcup_{l \in [i-1]} X_l) > m\gamma_i - \frac{(\beta+1)(i-1)}{\beta+2}\gamma_i \geqslant \frac{m+\beta+1}{\beta+2}\gamma_i, \tag{5}$$

where the last inequality is due to $i \leqslant m$.

According to Equation (5), the total area of white rectangles is at least $\frac{m+\beta+1}{\beta+2}\gamma_i$. There must exist an $r \in [m]$ such that $u_i(X'_r \cap J') \geqslant \frac{m+\beta+1}{m(\beta+2)}\gamma_i$. Therefore, Claim 2 holds. □

C Missing Materials for EF1 and PO Scheduling in Section 4

C.1 The Impossible Result for Theorem 3

In this subsection, we prove the first part of Theorem 3 (See Lemma 6).

Lemma 6. *EF1 and PO are not compatible for FISP with ⟨unweighted, rigid⟩, i.e., no algorithm can return a feasible schedule that is simultaneously EF1 and PO for all FISP with ⟨unweighted, rigid⟩ instances.*

Proof. To prove Lemma 6, we show that any PO schedule must not be an EF1 schedule for the instance in Figure 4. We consider an arbitrary PO schedule, denoted by $\mathbf{X} = (X_1, \cdots, X_m)$ and let $X_0 = J \setminus \bigcup_{i \in [m]} X_i$. We claim that $\mathbf{X}$ must satisfy the following two properties:

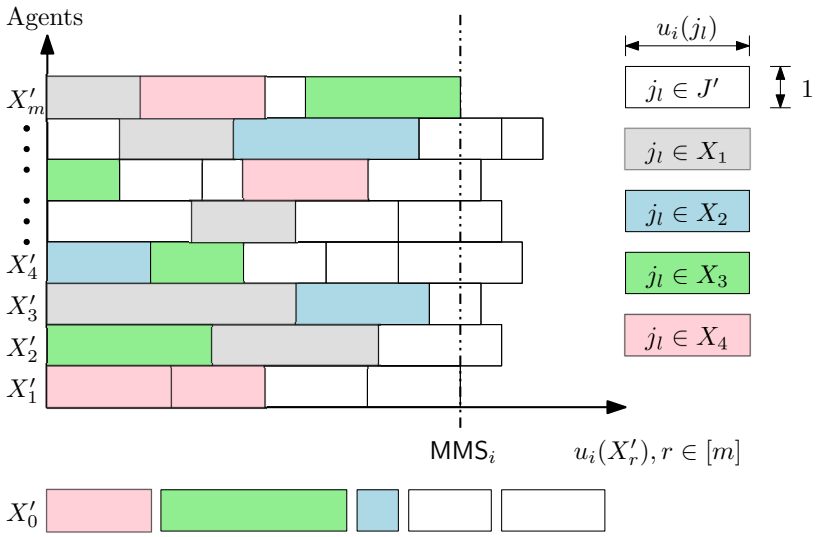


Figure 3: Illustration of Claim 2. The schedule is the feasible schedule $\mathbf{X}'$ which implies that job set X'_r is a feasible for all $r \in [m]$. Every rectangle represents a job. The width of rectangle j_l is the value of $u_i(j_l)$ while the height is 1. The area of rectangle j_l is also the value of $u_i(j_l)$. The four agents $a_1, a_2, a_3, a_4 \in \{ a_1, \cdots, a_{i-1} \}$. The non-white rectangles represent the jobs that are assigned in some agents in $\{ a_1, \cdots, a_{i-1} \}$ in schedule $\mathbf{X}$, e.g., the gray, blue, green, pink rectangles are the jobs that are assigned to a_1, a_2, a_3, a_4, respectively. Recall that $\mathbf{X}$ is the schedule returned by Algorithm 2. The white rectangles are the jobs in J'. In Claim 2, we show that there exist a $r \in [m]$ such that total area of white rectangles in X'_r is at least $\frac{1}{\beta+2}\gamma_i$.

1. $\exists i \in [m]$ such that $X_i = J_1$;
2. $X_0 = \emptyset$.

We first prove that there exists an $i \in [m]$ such that $X_i = J_1$. Suppose, towards to the contradiction, that there is no $i \in [m]$ such that $X_i = J_1$. Note that there must exist $i \in [m]$ such that $X_i \cap J_1 \neq \emptyset$ otherwise $\mathbf{X}$ is not a PO schedule. Now we consider the job set X_l such that $X_l \cap J_1 \neq \emptyset$. In the case where no job set except X_l in $\mathbf{X}$ contains jobs in J_1, we can construct another feasible schedule $\mathbf{X}' = \mathbf{X} \cup \{ J_1 \} \setminus X_l$. It is easy to see that $u_i(X'_i) \geqslant u_i(X_i)$ for all $a_i \in A$ and $u_l(J_1) > u_l(X_l)$ for agent a_l. This implies that $\mathbf{X}$ is not a PO schedule. In the case where there exist another one or two subsets $X_r, X_p \in \mathbf{X}$ such that $X_r, X_p \cap J_1 \neq \emptyset$. Since there are only three jobs in J_1, there are at most three job sets in $\mathbf{X}$ that contains some job in J_1. Without loss of generality, we assume that both X_r and X_p exist. Since every job in J_2 overlaps with every job in J_1, we have $X_l, X_r, X_p \cap J_2 = \emptyset$. Therefore, $|X_l| = |X_r| = |X_p| = 1$. Since there are $m-1$ long jobs and every job set in $\mathbf{X} \setminus (X_l \cup X_r \cup X_p)$ contains only one job in J_2, X_0 contains two jobs from J_2. Now we can construct another feasible schedule $\mathbf{X}' = (X'_1, \cdots, X'_m)$ in following way: move all jobs in $X_r \cup X_p$ to X_l; assign one of two jobs in X_0 to X_r and another one to X_p; keep the remaining job sets same as the corresponding one in $\mathbf{X}$. It is easy to see that $u_i(X'_i) \geqslant u_i(X_i)$ for all $a_i \in A$ and $u_l(X'_l) = u_l(J_1) > u_l(X_l)$. This implies that $\mathbf{X}$ is not a PO schedule.

Therefore, we can assume that there must exist an $i \in [m]$ such that $X_i = J_1$. Without loss of generality, we assume that $X_1 = J_1$. Now we show that the second property holds. Since $|J_2| = m-1$, $X_0 \neq \emptyset$ implies that there must exist a job set $X_l \in \mathbf{X}$ such that $X_l = \emptyset$. This would imply that $\mathbf{X}$ is not a PO schedule. Since $\mathbf{X}$ holds the above two properties, we assume that every remaining agent in $A \setminus \{ a_1 \}$ will receive exactly one job in J_2. Without loss of generality, we assume that $X_i = \{ j_{i+2} \}$. Therefore, we have $\mathbf{X} = (X_1, \cdots, X_m)$, where $X_1 = J_1, X_2 = \{ j_4 \}, \cdots, X_m = \{ j_{m+2} \}$. Since $u_i(X_1 \setminus \{ j \}) = 2 > u_i(X_i) = 1, \forall a_i \in A \setminus \{ a_1 \}, \forall j \in X_1$, $\mathbf{X}$ is not an EF1 schedule. □

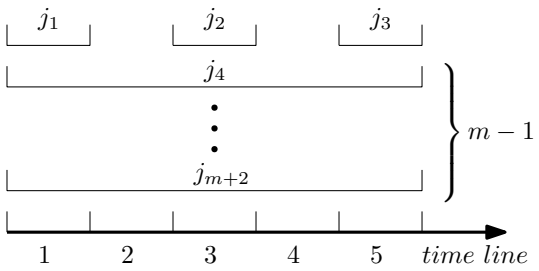


Figure 4: Instance for Lemma 6. There are $|A| = m$ agents and $|J| = m + 2$ jobs with $m \geqslant 2$. Job set J can be partitioned as $J_1 \cup J_2$, where $J_1 = \{ j_1, j_2, j_3 \}$ and $J_2 = \{ j_4, j_5, \cdots, j_{m+2} \}$. Each job J_1 has unit processing time and each job J_2 has processing time 5. All jobs are rigid such that $j_i \in J_1$ needs to occupy the entire time slot $2i - 1$, where $i \in \{ 1, 2, 3 \}$. And $j \in J_2$ occupies the entire time period from 1 to 5.

C.2 The Algorithm for Theorem 3

In this subsection, we mainly show Theorem 5, which is the second part of Theorem 3. Before give the proof of Theorem 5, we first give the definition of the *condensed instance* which is used to improve the running time.

Given an arbitrary instance of FISP with $\langle$identical, unit$\rangle$, denoted by I, for each job $j_i \in J$, let $\mathcal{T}_i$ be the set of time slots included in the job interval of j_i, i.e., $\mathcal{T}_i = \{ r_i, r_i + 1, \cdots, d_i \}$. Let $\mathcal{T}$ be the set of *condensed time slots* (Definition 4). We construct another instance, denoted by I', by condensing T_i, i.e., for every job in J, $T_i = T_i \cap T$. We show that these two instances are equivalent (Lemma 7). Let J' be the set of jobs in the instance I'.

Definition 4. *Let T be the condensed time slots set.*

$$T = \bigcup_{1 \leqslant l \leqslant n} \{ d_l - n + 1, d_l - n + 2, \cdots, d_l \}$$

where d_l is the deadline of job j_l.

To prove Lemma 8, it suffices to prove the following lemma.

Lemma 7. *Let $\hat{J} \subseteq J$ be an arbitrary subset of jobs in the instance I. Let $\hat{J}' \subseteq J'$ be the corresponding jobs in the instance I'. Then, $\hat{J}$ is a feasible job set if and only if $\hat{J}'$ is a feasible job set.*

Proof. ($\Leftarrow$) This direction is straightforward.

($\Rightarrow$) To prove this direction, we define a *job block* as a maximal set of consecutive jobs such that they are scheduled after each other. Since $\hat{J}$ is a feasible job set, there is a feasible schedule for all jobs in $\hat{J}$. We start from the first job $j_l \in \hat{J}$ which is scheduled in time slot t_l such that $t_l \notin T$, we show that we can always shift this job block to the right. Time slot $t_l \notin T$ implies that t_r is in a distance more that n from any elements in deadline set $D = \bigcup_{j_q \in J} \{ d_q \}$. We can shift the job block j_l to the right. An example is shown in Figure 5. We show that we can always shift j_l to the right until:

- Either job j_l is scheduled in a time slot in T.
- Or the job block starting from j_l reaches another scheduled job and form a bigger job block.

If job j_l is scheduled in a time slot in T, then the lemma follows. If the job block starting from j_l reaches another scheduled job and form a bigger job block, we keep shifting the bigger job block to the right unit j_l is scheduled in a time slot $t'_l \in T$. Note that no job would miss its deadline, since the distance between t'_l and any deadline in D exceeds n which implies that there is enough time slots to schedule all jobs in the current job block. □

Lemma 8. *For an arbitrary instance of FISP with $\langle$identical, unit$\rangle$, if there is a polynomial-time algorithm that returns an EF1 and PO schedule for all condensed instances, there also exists one for non-condensed instances.*

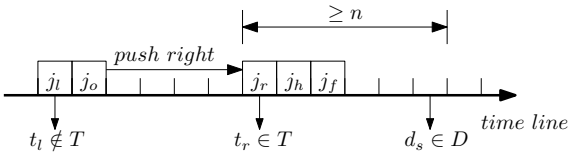


Figure 5: Illustration of Lemma 7. Initially, job j_l is scheduled in the time slot t_r which is not in T. The job block starting from j_l only includes two jobs: j_l and j_o. We can always shift job j_l to the right until j_l is scheduled in a time slot in T. Or the leftmost time slot in T is occupied by a certain job j_r. The job block starting from j_r contains three jobs: j_r, j_h and j_f. We can still shift the merged job block, which contains j_l, j_o, j_r, j_h, j_f, to the right, since the distance between time slot t_r and any deadline in D exceeds n.

Theorem 5. *Given an arbitrary instance of FISP with ⟨identical, unit⟩, Algorithm 5 returns a schedule that is simultaneously EF1 and PO in polynomial time.*

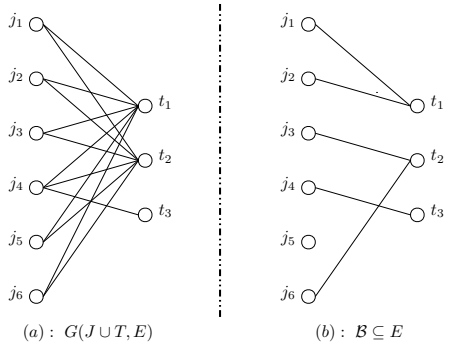


Figure 6: An example of the bipartite graph $G(J \cup T, E)$ and a corresponding maximum m-matching, where $J = \{ j_1, \cdots, j_6 \}, A = \{ a_1, a_2 \}, T = \{ t_1, t_2, t_3 \}$. Assume all jobs have identical utility to the agents. The job intervals are $T_1 = T_2 = T_3 = T_5 = T_6 = \{ t_1, t_2 \}$, and $T_4 = \{ t_1, t_2, t_3 \}$. A possible maximum weighted m-matching $\mathcal{M}^*$ is shown one the right, according to which the jobs are partitioned as $J_1 = \{ j_1, j_2 \}, J_2 = \{ j_3, j_6 \}, J_3 = \{ j_4 \}$. Then, $X_0 = J \setminus (J_1 \cup J_2 \cup J_3) = \{ j_5 \}$.

Arbitrarily fix a maximum weighted m-matching $\mathcal{M}^*$. For any $t \in T$, let J_t be the set of jobs which are matched with time slot t, i.e., $J_t = \{ j \in J \mid (j, t) \in \mathcal{M}^* \}$. Note that the J_t's are mutually disjoint. Therefore we can refer t as the *type* of jobs in J_t. An example can be found in Figure 6 (a). The key idea of Algorithm 5 is to first use the above procedure to find the job set with the maximum total weight that can be processed and classify jobs into different types, and then greedily assign each type of jobs to the agents. The jobs that are not matched by $\mathcal{M}^*$, i.e., $J \setminus (\bigcup_{t \in T} J_t)$, are kept unallocated and will be assigned to charity.

Let $\mathbf{X} = (X_1, \cdots, X_m)$ be the schedule returned by Algorithm 5 and let $X_0 = J \setminus \bigcup_{i \in [m]} X_i$. Note that although the agents have identical utilities, we sometimes use u_i for agent $a_i \in A$ to make the comparison clear.

Lemma 9. *For any $a_i, a_k \in A$, $u_i(X_i) \geqslant u_i(X_k \setminus \{ j_l \})$ for some $j_l \in X_k$.*

Proof. We prove the lemma by induction. Let X_i^p be the set of jobs assigned to agent a_i after the p-th round of Algorithm 5, and $\mathbf{X}^p = (X_1^p, \cdots, X_m^p)$.

Base Case. When $p = 1$, each agent gets at most one job as $|J_t| \leqslant m$ for all $t \in T$, and thus $\mathbf{X}^1$ is EF1.

Induction Hypothesis. For any $p > 1$, after the p-th round of Algorithm 5, suppose Lemma 9 holds, i.e., for any $a_i, a_k \in A$, there exists a job, denoted by j_h, in X_i^p such that $u_k(X_k^p) \geqslant u_k(X_i^p \setminus \{ j_h \})$.

Now we consider the $(p+1)$-th round. Arbitrarily fix two agents $a_i, a_k \in A$ and without loss of generality assume $u_i(X_i^p) \geqslant u_k(X_k^p)$. In the following we prove that after this round, a_i and a_k continue not to envy each other for more than one item. Note that, in the $(p+1)$-th round, a_k chooses a job from J_{p+1} before a_i.

Suppose that $j_{\hat{k}}$ is assigned to a_k while $j_{\hat{i}}$ is assigned to a_i in the $(p+1)$-th round. Therefore, we have $X_k^{p+1} = X_k^p \cup \{ j_{\hat{k}} \}$ and $X_i^{p+1} = X_i^p \cup \{ j_{\hat{i}} \}$. Since $|J_{p+1}| \leqslant m$, $\{ j_{\hat{k}} \}$ and $\{ j_{\hat{i}} \}$ may be empty, in which case, we assume that $u(j_{\hat{k}}) = u(j_{\hat{i}}) = 0$. Since a_k chooses the job before a_i and all jobs in J_{p+1} are sorted in non-increasing order, $u(j_{\hat{k}}) \geqslant u(j_{\hat{i}})$ always holds no matter whether $\{ j_{\hat{k}} \}$ is empty or not.

Regarding agent a_i, as $u_i(X_i^p) \geqslant u_k(X_k^p)$, we have

$$u_i(X_i^{p+1}) \geqslant u_i(X_i^p) \geqslant u_i(X_k^p) = u_i(X_k^{p+1} \setminus \{ j_{\hat{k}} \}).$$

Regarding agent a_k, because $u_k(j_{\hat{k}}) \geqslant u_k(j_{\hat{i}})$ and $u_i(X_i^p) \geqslant u_i(X_k^p \setminus \{ j_h \})$ (induction hypothesis),

$$u_k(X_k^{p+1}) = u_k(X_k^p) + u_k(j_{\hat{k}}) \geqslant u_k(X_i^p \setminus \{ j_h \}) + u_k(j_{\hat{i}}) = u_k(X_i^{p+1} \setminus \{ j_h \}).$$

Thus, after the $(p+1)$-th round, a_i and a_k continue not to envy each other for more than one item. By induction, Lemma 9 holds. □

Proof of Theorem 5. Since schedule $\mathbf{X}$ returned by Algorithm 5 maximizes social welfare $\sum_{a_i \in A} u_i(X_i)$, $\mathbf{X}$ must be PO. According to Lemma 9, $\mathbf{X}$ is EF1. For time complexity, we have already discussed that computing a maximum m-matching can be done in polynomial time. Further, as allocating jobs by types only needs to sort jobs or agents, which can also be done in polynomial time, we finished the proof. □

We note that Algorithm 5 fails to return an EF1 and PO schedule if the agents' utilities are not identical. Actually, the existence of EF1 and PO schedule for this case is left open in Biswas and Barman [2018]; Dror *et al.* [2020]; Wu *et al.* [2021] even when the scheduling constraints degenerate to cardinality constraints.

Remark 1. *We noted that the proof of Lemma 9 only uses the ranking of jobs' weight. Therefore, Algorithm 5 is able to return to a feasible schedule that is simultaneously EF1 and PO in the setting where agents value jobs in the same order but the concrete jobs' weight are not known by the algorithm.*

C.3 1/4-EF1 and PO for general FISP instances

Before giving the proof, we first introduce the formal definition of MaxNSW-schedule which will be used to prove Theorem 4.

Definition 5 (MaxNSW Schedule)**.** *A feasible schedule $\mathbf{X} = (X_1, \cdots, X_m)$ is called MaxNSW schedule if and only if*

$$\mathbf{X} \in \arg\max_{\mathbf{X}' \in \mathcal{F}} \prod_{i=1}^{m} u_i(X_i')$$

where $\mathcal{F}$ is the set of all feasible schedules and $\mathbf{X}' = (X_1', \cdots, X_m')$.

Note that in the standard definition of Nash social welfare maximizing schedule, $\mathbf{X}$ was supposed to be a member of $\arg\max_{\mathbf{X}' \in \mathcal{F}} \left(\prod_{i=1}^{m} u_i(X_i') \right)^{\frac{1}{m}}$. Here, we ignore the power of $\frac{1}{m}$ to simplify the formula.

In this section, we mainly prove Theorem 6 which is the first part of Theorem 4. The proof of Theorem 6 is essentially the same with corresponding one in Wu *et al.* [2021], and we include the proof for completeness.

Theorem 6. *Given an arbitrary instance of general FISP, any schedule that maximizes the Nash social welfare is a 1/4-EF1 and PO schedule.*

Proof. Given an arbitrary instance of general FISP, let $\mathbf{X} = (X_1, \cdots, X_m)$ be the MaxNSW schedule and let $X_0 = J \setminus \bigcup_{i \in [m]} X_i$. Since any MaxNSW schedule must be a PO schedule, we only prove that $\mathbf{X}$ is a 1/4-EF1 schedule i.e., $\forall i, k \in [m], u_i(X_i) \geqslant \frac{1}{4} u_i(X_k \setminus \{ j_p \}), \exists j_p \in X_k$. Suppose, on the contrary, that there exists $i, k \in [m]$ such that $u_i(X_i) < \frac{1}{4} u_i(X_k \setminus \{ j_p \}), \forall j_p \in X_k$.

Now, we sort all jobs in X_k in non-increasing order according to the value of $u_k(j_p), j_p \in X_k$. Assume that $X_k = \{ j_1, j_2, \cdots \}$ after sorting. Without loss of generality, we assume that $|X_k|$ is an odd number; otherwise, we add a dummy job j_o to X_k such that $u_i(j_o), \forall i \in [m]$. Now we partition $X_k \setminus \{ j_1 \}$ into two subsets X_k^1, X_k^2, where $X_k^1 = \{ j_2, j_4, j_6, \cdots \}$ and $X_k^2 = \{ j_3, j_5, j_7, \cdots \}$. Note that $X_k = \{ j_1 \} \cup \{ X_k^1 \} \cup \{ X_k^2 \}$. Note that $u_k(X_k^1) \geqslant u_k(X_k^2)$ and $u_k(X_k^2 \cup \{ j_1 \}) \geqslant u_k(X_k^1)$ since all jobs in X_k are sorted in non-increasing order. Since $u_k(X_k^1) \geqslant u_k(X_k^2)$, we have $u_k(j_1) + u_k(X_k^1) \geqslant u_k(X_k^2)$. Therefore, we have

$$u_k(X_k^d \cup \{ j_1 \}) \geqslant \frac{1}{2} u_k(X_k), \forall d \in \{ 1, 2 \}. \tag{6}$$

Since $u_i(X_i) < \frac{1}{4} u_i(X_k \setminus \{ j_p \}), \forall j_p \in X_k$, we have $u_i(X_i) < \frac{1}{4} u_i(X_k^1 \cup X_k^2)$. Since X_k is a feasible job set, we have $u_i(X_k^1 \cup X_k^2) = u_i(X_k^1) + u_i(X_k^2)$ which implies that either $u_i(X_k^1) \geqslant \frac{1}{2} u_i(X_k^1 \cup X_k^2)$ or $u_i(X_k^1) \geqslant \frac{1}{2} u_i(X_k^1 \cup X_k^2)$. Therefore, we have

$$u_i(X_i) < \frac{1}{4} u_i(X_k^1 \cup X_k^2) \leqslant \frac{1}{8} u_i(X_k^d), \exists d \in \{ 1, 2 \}. \tag{7}$$

Now we construct a new schedule, denoted by $\mathbf{X}' = (X_1', \cdots, X_m')$, where $X_r' = X_r, \forall r \in [m], r \neq i, k$. Let $X_0' = J \setminus \bigcup_{i \in [m]} X_i'$. We discard all jobs in X_i, i.e., $X_0' = X_0 \cup X_i$. If $u_i(X_k^1) \geqslant \frac{1}{2} u_i(X_k^1 \cup X_k^2)$, let $X_i' = X_k^1$ and $X_k' = X_k^2 \cup \{ j_1 \}$; otherwise, let $X_i' = X_k^2$ and $X_k' = X_k^1 \cup \{ j_1 \}$. It is easy to see that $\mathbf{X}'$ is a feasible schedule. Note thar all job sets in $\mathbf{X}'$ except X_0', X_i', X_k' are the same as the corresponding job sets in $\mathbf{X}$. Observe that if we can prove that $u_i(X_i') u_k(X_k') > u_i(X_i) u_k(X_k)$, then $\mathbf{X}$ is not a MaxNSW schedule which will contradict our assumption. In the case where $u_i(X_k^1) \geqslant \frac{1}{2} u_i(X_k^1 \cup X_k^2)$, we have $X_i' = X_k^1$. By Equation (6), we have $u_k(X_k') = u_k(X_k^2 \cup \{ j_1 \}) \geqslant \frac{1}{2} u_k(X_k)$. By Equation (7), we have $u_i(X_i') = u_i(X_k^1) > 8 u_i(X_i)$. In the case where $u_i(X_k^1) < \frac{1}{2} u_i(X_k^1 \cup X_k^2)$, we have $X_i' = X_k^2$. By Equation (6), we have $u_k(X_k') = u_k(X_k^1 \cup \{ j_1 \}) \geqslant \frac{1}{2} u_k(X_k)$. By Equation (7), we have $u_i(X_i') = u_i(X_k^2) > 8 u_i(X_i)$. By combining above two cases, we have $4 u_i(X_i) u_i(X_k) < u_i(X_i') u_k(X_k')$. Hence, we have $u_i(X_i) u_i(X_k) < u_i(X_i') u_k(X_k')$. □

In the following, we show that our proof in Theorem 6 is tight.

Lemma 10. *Given an arbitrary instance of general FISP, a MaxNSW schedule can only guarantee 1/4-EF1 and PO.*

Proof. To prove Lemma 10, it is sufficient to give an instance such that MaxNSW schedule is exactly 1/4-EF1 schedule and PO. In this instance, all jobs in job set J are rigid and J can be partitioned into two sets J_L and J_S. There is only one job in J_L which is very long and has weight 1. There are $\frac{4}{\epsilon}$ jobs in J_S each of which has unit length and weight ϵ. Note that $\frac{4}{\epsilon}$ is assumed to be an even integer number. All jobs in J_S are disjoint and the job in J_L intersects with all jobs in J_S. The agent set A contains only two agents, i.e., $|A| = 2$. The instance can be found in Figure 7.

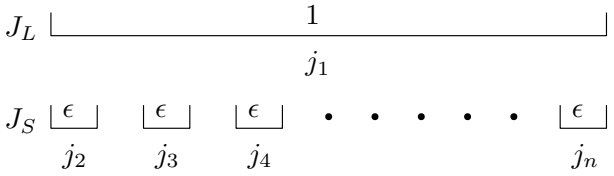


Figure 7: Tight example of MaxNSW schedule for general FISP.

Note that the total weight of jobs in J_S is 4. Let $\mathbf{X} = (X_1, X_2)$ be the schedule, where $X_1 = J_L, X_2 = J_S$. let $\mathbf{X}' = (X_1', X_2')$, where $X_1' = \{ j_2, \cdots, j_{\frac{2}{\epsilon}+1} \}, X_2' = J_S \setminus X_1'$, i.e., J_S is partitioned into two subsets with equal size. Note that $X_0' = J_L$. It is not hard to see that $\mathbf{X}'$ is a MaxNSW schedule. And we have $u_1(X_1) u_2(X_2) = 4$, $u_1(X_1') u_2(X_2') = 4$. Therefore, $\mathbf{X}$ is a MaxNSW schedule. Note that $u_1(X_2 \setminus \{ j_p \}) = \frac{4-\epsilon}{\epsilon} \cdot \epsilon = 4 - \epsilon, \forall j_p \in X_2$. Therefore, we have

$$\lim_{\epsilon \to 0} \frac{1}{4} u_1(X_2 \setminus \{ j_p \}) = 1 = u_1(X_1), \forall j_p \in X_1.$$

This implies that $\mathbf{X}$ is a 1/4-EF1 schedule. □

C.4 1/2-EF1 and PO for FISP with ⟨non-identical, unit⟩ instances

In this section, we mainly prove Theorem 7 which is the second part of Theorem 4.

Theorem 7. *Given an arbitrary instance of FISP with ⟨non-identical, unit⟩, a MaxNSW schedule is a 1/2-EF1 and PO schedule.*

Proof. We show that a feasible schedule $\mathbf{X} = (X_1, \cdots, X_m)$ that maximizes Nash social welfare is simultaneously 1/2-EF1 and PO. Since any MaxNSW schedule must be a PO schedule, we only prove that $\mathbf{X}$ is a 1/2-EF1 schedule Hence, we only show that $\mathbf{X}$ is an 1/2-EF1 schedule, i.e., $\forall i, k \in [m], u_i(X_i) \geqslant \frac{1}{2} \cdot u_i(X_k \setminus \{ j \}), \exists j \in X_k$.

We prove by contradiction and assume that there exists $i, k \in [m]$ such that $u_i(X_i) < \frac{1}{2} \cdot u_i(X_k \setminus \{ j \}), \forall j \in X_k$. Then, we have

$$u_i(X_i) + u_i(j) < u_i(X_k) - u_i(X_i), \forall j \in X_k. \tag{8}$$

Since X_i, X_k are feasible job set, there is a maximum weighted matching in $G(X_i \cup T, E_i), G(X_k \cup T, E_k)$ with size $|X_i|, |X_k|$, respectively. Let M_i, M_k be the maximum weighted matching in $G(X_i \cup T, E_i), G(X_k \cup T, E_k)$, respectively. An example can be found in Figure 8.

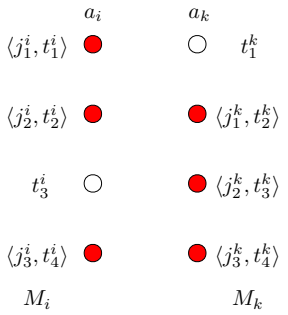


Figure 8: Illustration for M_i, M_k. In the above example, we have $X_i = \{ j_1^i, j_2^i, j_3^i \}$, $X_k = \{ j_1^k, j_2^k, j_3^k \}$ and $T = \{ t_1, t_2, t_3, t_4 \}$. We have matching $M_i = \{ \langle j_1^i, t_1^i \rangle, \langle j_2^i, t_2^i \rangle, \langle j_3^i, t_4^i \rangle \}$ and $M_k = \{ \langle j_1^k, t_2^k \rangle, \langle j_2^k, t_3^k \rangle, \langle j_3^k, t_4^k \rangle \}$. Moreover, we have $M_i(J) = X_i, M_i(T) = \{ t_1, t_2, t_4 \}$ and $M_k(J) = X_k, M_k(T) = \{ t_2, t_3, t_4 \}$.

For every time slot $t_l \in M_i(T) \cup M_k(T)$, we find the pair $\langle j^i, t_l^i \rangle \in M_i, \langle j^k, t_l^k \rangle \in M_k$. Note that there may exist some time slot t_l such that t_l is only matched in M_i or M_k, e.g., time slot t_1, t_3 in the example shown in Figure 8. In this case, we add a dummy pair to M_k or M_i, e.g., in the example shown in Figure 8, $M_i = M_i \cup \{ \langle j_o, t_3^i \rangle \}, M_k = M_k \cup \{ \langle j_o, t_1^k \rangle \}$ and let $u_i(j_o) = u_k(j_o) = 0, \forall i \in [m]$. For every time slot $t_l \in M_i(T) \cup M_k(T)$, we find the pair $\langle j^i, t_l^i \rangle \in M_i, (j^k, t_l^k) \in M_k$ and define the big pair $[\langle j^i, t_l^i \rangle, \langle j^k, t_l^k \rangle]$ as (j^i, j^k) for convenience. For each pair (j^i, j^k), we define $|(j^i, j^k)|$ as its value, where

$$|(j^i, j^k)| = \frac{u_i(j^k) - u_i(j^i)}{u_k(j^k) - u_k(j^i)}.$$

Note that there may exist two pairs: $\langle j^i, t_l^i \rangle \in M_i, \langle j^k, t_l^k \rangle \in M_k$ such that $u_i(j^k) - u_i(j^i) = 0$ and $u_k(j^k) - u_k(j^i) = 0$. In this case, we have

$$|(j^i, j^k)| = \begin{cases} 0, \text{ if } & u_i(j^k) - u_i(j^i) = 0, u_k(j^k) - u_k(j^i) \neq 0; \\ \infty, \text{ if } & u_i(j^k) - u_i(j^i) \neq 0, u_k(j^k) - u_k(j^i) = 0. \end{cases}$$

Let $\mathcal{P}_+, \mathcal{P}_-$ be the set of all (j^i, j^k) such that $u_i(j^k) - u_i(j^i) > 0$ and $u_i(j^k) - u_i(j^i) \leqslant 0$, respectively. We consider an arbitrary pair (j_+^i, j_+^k) in $\mathcal{P}_+$, i.e., $u_i(j_+^k) - u_i(j_+^i) > 0$. Note that $u_k(j_+^k) - u_k(j_{+0}^i)$ holds; otherwise, we can construct a new feasible schedule by swapping job j_+^i and j_+^k will have larger Nash Social Welfare. This would imply that $\mathbf{X}$ does not maximize the Nash Social Welfare. Let X_i^+, X_k^+ be the set of jobs in X_i, X_k that are covered by some pair in

$\mathcal{P}_+$, respectively, i.e., $X_i^+ = \{\, j^i \in X_i \mid \exists (j^i, j^k) \in \mathcal{P}_+ \,\}$ and $X_k^+ = \{\, j^k \in X_k \mid \exists (j^i, j^k) \in \mathcal{P}_+ \,\}$. Notations X_i^-, X_k^- are defined in similar ways. Note that

$$u_i(X_k) - u_i(X_i) = \Big(u_i(X_k^+) - u_i(X_i^+)\Big) + \Big(u_i(X_k^-) - u_i(X_i^-)\Big).$$

Since $u_i(X_k^-) - u_i(X_i^-) \leqslant 0$, we have

$$u_i(X_k) - u_i(X_i) \leqslant u_i(X_k^+) - u_i(X_i^+). \tag{9}$$

Then, we have:

$$\frac{u_i(X_k^+) - u_i(X_i^+)}{u_k(X_k^+)} \geqslant \frac{u_i(X_k^+) - u_i(X_i^+)}{u_k(X_k)} \geqslant \frac{u_i(X_k) - u_i(X_i)}{u_k(X_k)}, \tag{10}$$

where the first inequality is due to $u_i(X_k^+) - u_i(X_i^+) > 0$ and $u_k(X_k) \geqslant u_k(X_k^+)$, the last inequality is due to Equation (9). Now, we define (g^i, g^k) as:

$$(g^i, g^k) = \underset{(j^i, j^k) \in \mathcal{P}_+}{\arg\max} \Big\{ |(j^i, j^k)| \Big\}.$$

Note that $\mathcal{P}_+ \neq \emptyset$, i.e., there must exist a pair (j_+^i, j_+^k) such that $u_i(j_+^k) - u_i(j_+^i) > 0$ because of $u_i(X_k) > u_i(X_i)$. Since every pair (j^i, j^k) in $\mathcal{P}_+$ has property $u_i(j^k) - u_i(j^i) > 0$ and $u_k(j^k) - u_i(j^i) > 0$, we have:

$$\frac{u_i(g^k) - u_i(g^i)}{u_k(g^k) - u_k(g^i)} \geqslant \frac{u_i(X_k^+) - u_i(X_i^+)}{u_k(X_k^+) - u_k(X_i^+)} \geqslant \frac{u_i(X_k^+) - u_i(X_i^+)}{u_k(X_k^+)}, \tag{11}$$

where the last inequality is due to $u_i(X_k^+) - u_i(X_i^+) > 0$ and $u_k(X_i^+) \geqslant 0$. By combining Equation (10) and Equation (11), we have

$$\frac{u_i(g^k) - u_i(g^i)}{u_k(g^k) - u_k(g^i)} \geqslant \frac{u_i(X_k) - u_i(X_i)}{u_k(X_k)} > \frac{u_i(X_i) + u_i(g^k)}{u_k(X_k)}, \tag{12}$$

where the last inequality is due to Equation (8).

Since $u_i(g^k) - u_i(g^i) > 0$ and $(u_k(g^k) - u_k(g^i) > 0$, we have:

$$\Big(u_i(g^k) - u_i(g^i)\Big) \cdot u_k(X_k) > \Big(u_k(g^k) - u_k(g^i)\Big) \cdot \Big(u_i(X_i) + u_i(g^k)\Big). \tag{13}$$

Holding Equation (13) on our hand, we are ready to prove that $\mathbf{X}$ does not maximize the Nash social welfare. Now, we construct another feasible schedule, denoted by $\mathbf{X}' = (X_1', \cdots, X_m')$. We construct $\mathbf{X}'$ by swapping the job g^i with g^k, i.e., $X_o' = X_o, \forall o \in [m]$ and $o \neq i, k$, $X_i' = X_i \cup \{\, g^k \,\} \setminus \{\, g^i \,\}$ and $X_k' = X_k \cup \{\, g^i \,\} \setminus \{\, g^k \,\}$. Note that all job sets in $\mathbf{X}'$ except X_i', X_k' are the same as the corresponding job sets in $\mathbf{X}$. Observe that if we can show that $u_i(X_i')u_k(X_k') > u_i(X_i)u_k(X_k)$, then it implies that $\mathbf{X}$ does not maximize the Nash social welfare. Note that

$$u_i(X_i') = u_i(X_i) + u_i(g^k) - u_i(g^i);$$
$$u_k(X_k') = u_k(X_k) + u_k(g^i) - u_k(g^k).$$

We define Γ as follows for convenience:

$$\Gamma = \Big(u_i(g^k) - u_i(g^i)\Big) \cdot u_k(X_k) - \Big(u_k(g^k) - u_k(g^i)\Big) \cdot \Big(u_i(X_i) + u_i(g^k)\Big),$$

where $\Gamma > 0$ because of Equation (13). Then, we have

$$u_i(X_i')u_k(X_k') - u_i(X_i)u_k(X_k) = \Gamma + \Big(u_k(g^k) - u_k(g^i)\Big) \cdot u_i(g^i).$$

Since $u_k(g^k) - u_k(g^i) > 0$ and $\Gamma > 0$, we have $u_i(X_i')u_k(X_k') - u_i(X_i)u_k(X_k) > 0$. Hence $\mathbf{X}$ does not maximize the Nash social welfare which contradicts our assumption. Therefore, $\forall i, k \in [m], u_i(X_i) \geqslant \frac{1}{2} \cdot (X_k \setminus \{\, j \,\}), \exists j \in X_k$. $\square$

In the following, we show that our proof of Theorem 7 is tight.

Lemma 11. *The schedule which maximizes the Nash social welfare can only guarantee 1/2-EF1 and PO for FISP with ⟨non-identical,unit⟩.*

Proof. To prove Lemma 11, we give an instance for which a schedule that maximizes the Nash social welfare is an 1/2-EF1 schedule.

We consider the job set $J = \{ j_1, \cdots, j_n, j_{n+1}, \cdots, j_{2n} \}$ which contains $2n$ jobs. All jobs have the same release time 1 and deadline n. Moreover, all jobs have unit processing time. The agent set $A = \{ a_1, a_2 \}$ contains two agents. The utilities matrix is as follows:

	j_1	j_2	$\cdots$	j_n	j_{n+1}	$\cdots$	j_{2n}
a_1	2	2	$\cdots$	2	1	$\cdots$	1
a_2	1	1	$\cdots$	1	0	$\cdots$	0

To find the schedule that maximizes the Nash social welfare, we consider an arbitrary schedule $\mathbf{X} = (X_1, X_2)$ and assume that $X_0 = J \setminus (X_1 \cup X_2)$. We define $J_1 = \{ j_1, \cdots, j_n \}$ and $J_2 = \{ j_{n+1}, \cdots, j_{2n} \}$. Observe that $X_0 = \emptyset$ otherwise $\mathbf{X}$ does not maximize the value of $u_1(X_1) \cdot u_2(X_2)$. We assume that x jobs in J_1 are assigned to a_1 and y jobs in J_2 are assigned to a_2, where $0 \leqslant x, y \leqslant n$. Then, we have

$$f(x, y) = u_1(X_1) \cdot u_2(X_2) = (2x + y) \cdot (n - x).$$

To find the maximum value of $f(x, y)$ under the constraints $0 \leqslant x, y \leqslant n$, we compute partial derivative.

$$\begin{cases} \frac{\partial f(x,y)}{\partial x} = 2n - 4x - y = 0 \\ \frac{\partial f(x,y)}{\partial y} = n - x = 0 \end{cases}$$

The solution to the above two equations is $(n, -2n)$. Since the point $(n, -2n) \notin \{ (x, y) \mid 0 \leqslant x, y \leqslant n \}$, the maximum value will be taken at a certain vertex. We can find that the maximum value will be taken at the point $(x, y) = (0, n)$ by computing the value of $f(0, 0), f(0, n), f(n, 0), f(n, n)$.

Hence, we found the schedule $\mathbf{X} = (X_1, X_2)$ maximizes the Nash social welfare, where $X_1 = J_2, X_2 = J_1$. Then, we have $u_1(X_1) = 2n$ and $u_1(X_1 \setminus \{ j \}) = 2(n-1), \forall j \in X_1$. Then, we have

$$\lim_{n \to +\infty} \frac{u_1(X_1)}{u_1(X_1 \setminus \{ j \})} = \frac{n}{2(n-1)} = \frac{1}{2}.$$

□

D EF1 and IO Scheduling

Lemma 6 shows that PO is very demanding since even if agents have unweighted utilities, EF1 and PO are not compatible. Accordingly, in this section, we will consider the following weaker efficiency criterion – Individual Optimality.

Definition 6 (α-IO schedule)**.** *A feasible schedule $\mathbf{X} = (X_1, \cdots, X_m)$ with $X_0 = J \setminus \bigcup_{i \in [m]} X_i$ is called α-approximate individual optimal (α-IO) if $u_i(X_i) \geqslant \alpha \cdot u_i(X_0 \cup X_i)$ for all $a_i \in A$, where $\alpha \in (0, 1]$ and when $\alpha = 1$, $\mathbf{X}$ is called IO schedule.*

As we will see, although EF1 and IO are still not compatible for weighted utilities, they are when agents have unweighted utilities.

D.1 An Impossibility Result

We first show that EF1 and IO are not compatible even for FISP with ⟨identical, rigid⟩, i.e., given an arbitrary instance of FISP with ⟨identical, rigid⟩, there is no algorithm can always find a feasible schedule that is simultaneously EF1 and IO (Lemma 12).

Lemma 12. *EF1 and IO are not compatible even for FISP with ⟨identical, rigid⟩.*

Proof. To prove Lemma 12, it suffices to consider the instance in Figure 9, and prove the following two claims.

Claim 3. *For any IO schedule* $\mathbf{X} = (X_1, \cdots, X_m)$, $X_0 = J \setminus \bigcup_{i\in[m]} X_i = \emptyset$.

We prove this claim by contradiction. If $X_0 \cap J_1 \neq \emptyset$, as $|J_2| = m-1$, there will be at least one agent, without loss of generality say a_1, for whom $X_1 \cap J_2 = \emptyset$. Note that by the design of the instance, $X_1 \cup (X_0 \cap J_1)$ is feasible, and thus by allocating $X_0 \cap J_1$ to a_1, a_1's utility strictly increases.

If $X_0 \cap J_2 \neq \emptyset$, as $|J_2| = m-1$, there will be at least two agents, without loss of generality say a_1 and a_2, for whom $X_1 \cap J_2 = \emptyset$ and $X_2 \cap J_2 = \emptyset$. Furthermore, as $|J_1| = 4$ one of them gets at most two jobs in J_1. Again without loss of generality assume this is agent a_1. Accordingly, $u_1(X_1) \leqslant 4$ and by exchanging X_1 with one job in $X_0 \cap J_2$, a_1's utility strictly increases.

Claim 4. *For any EF1 schedule* $\mathbf{X} = (X_1, \cdots, X_m)$, $X_0 = J \setminus \bigcup_{i\in[m]} X_i \neq \emptyset$.

We note that the only possible and feasible schedule $\mathbf{X}$ such that $X_0 = \emptyset$ is that some agent, say a_1, gets entire J_1 and every other agent gets one job in J_2. Then to prove this claim, it suffices to prove $\mathbf{X}$ cannot be EF1. It is not hard to check that under $\mathbf{X}$, for any agent a_i with $i \geqslant 2$ and any job $j \in X_1$,

$$u_i(X_i) = 6 - \epsilon < u_i(X_1 \setminus \{j\}) = 6.$$

That is all a_i envies a_1 for more than one item.

Combing the above two claims, we complete the proof of Lemma 12. □

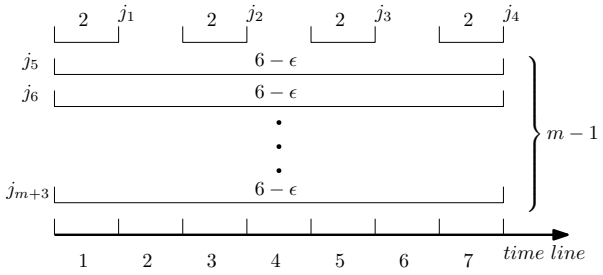


Figure 9: Instance for Lemma 12. There are $|A| = m$ agents and $|J| = m+3$ jobs with $m \geqslant 2$. Job set J can be partitioned as $J_1 \cup J_2$ with $J_1 = \{j_1, j_2, j_3, j_4\}$ and $J_2 = \{j_5, \cdots, j_{m+3}\}$. Each job in J_1 has unit processing time with weight 2 and each job in J_2 has processing time 7 with weight $6-\epsilon$. All jobs are rigid such that $j_i \in J_1$ needs to occupy the entire time slot $2i-1$, and $j \in J_2$ occupies the entire time period from 1 to 7.

D.2 A Polynomial-time Algorithm for FISP with ⟨unweighted, rigid⟩

In the following, we design a polynomial-time algorithm to compute a schedule that is EF1 and IO for any instance of FISP with ⟨unweighted, rigid⟩.

Theorem 8. *Given an arbitrary instance of FISP with ⟨unweighted, rigid⟩, Algorithm 6 returns a feasible schedule that is simultaneously EF1 and IO in polynomial time.*

Let $\mathbf{X} = (X_1, \cdots, X_m)$ be the schedule returned by Algorithm 6 and let $X_0 = J \setminus \bigcup_{i\in[m]} X_i$. Suppose that all agents receive a job at every round of first L rounds of Algorithm 6, i.e., in the $(L+1)$-th round, $\exists i \in [m]$ such that a_i receives nothing. Note that $L \leqslant n$, where n is the number of jobs. Let $j_i^l, 1 \leqslant i \leqslant m, 1 \leqslant l \leqslant L$, be the job assigned to agent a_i in the l-th round of Algorithm 6. Let r_i^l, d_i^l be the release time and deadline of job j_i^l. Let X_i^l be the job set that is assigned to agent a_i after the l-th round of Algorithm 6.

Lemma 13. $d_1^l \leqslant d_2^l \leqslant \cdots \leqslant d_m^l, \forall l \in [L]$.

Algorithm 6. Earliest Deadline First + Round-Robin

Input: Agent set A and job set J.
Output: EF1 schedule $\mathbf{X} = (X_1, \cdots, X_m)$

Sort all jobs by their deadline in non-decreasing order.
$X_1 = X_2 = \cdots = X_m = \emptyset$.
$i = 1, k = 1$. // The index.
for all $j_k \in J$ **do**
  if $X_i \cup \{ j_k \}$ is a feasible job set **then**
    $X_i = X_i \cup \{ j_k \}$.
    $J = J \setminus \{ j_k \}$.
    $i = (i + 1) \mod m$.
  else
    $i = i \mod m$.
  end if
end for
$X_0 = J \setminus \bigcup_{i \in [m]} X_i$.

Proof. We consider two agents a_i, a_k such that $1 \leqslant i < k \leqslant m$. Note that j_i^l, j_k^l must exist since, in the first L rounds, all agents receive a job. We prove by induction.

Base case and induction hypothesis. In the base case where $l = 1$, it is strainghtforward to see that $d_i^1 \leqslant d_k^1$, otherwise j_k^1 will be assigned to agent a_i in the first round of Algorithm 6. Now, we have induction hypothesis $d_i^l \leqslant d_k^l$.

We need to prove $d_i^{l+1} \leqslant d_k^{l+1}$. We prove by contradiction and assume that $d_i^{l+1} > d_k^{l+1}$. Since agent a_i chooses j_i^{l+1} instead of j_k^{l+1} in the $(l+1)$-th round, we know that $X_i^l \cup \{ j_k^{l+1} \}$ is a not feasible job set which implies that $r_k^{l+1} \leqslant d_i^l$. By induction hypothesis, we have $r_k^{l+1} \leqslant d_i^l \leqslant d_k^l$. This implies that $X_k^l \cup \{ j_k^{l+1} \}$ is not a feasible job set. This contradicts our assumption. Thus, $d_i^{l+1} \leqslant d_k^{l+1}$. □

Lemma 14. $d_m^l \leqslant d_1^{l+1}, \forall l \in [L-1]$. *Moreover,* $d_m^L \leqslant d_1^{L+1}$ *if* j_1^{L+1} *exists.*

Proof. Let a_i, a_k be two agents such that $1 \leqslant i < k \leqslant m$. We assume that j_1^{L+1} exists and prove that the lemma holds for all $l \in [L]$. We prove by contradiction.

In the base case where $l = 1$, it is not hard to see that $d_m^l \leqslant d_1^2$; otherwise a_m will choose j_1^2 in the first round. Now, we have induction hypothesis $d_m^{l-1} \leqslant d_1^l$.

Suppose, towards to a contradiction, that there exist $l \in [L]$ such that $d_m^l > d_1^{l+1}$. In the l-th round, agent a_m selects j_m^l instead of j_1^{l+1} because $X_m^{l-1} \cup \{ j_1^{l+1} \}$ is not a feasible job set; otherwise a_m will select j_1^{l+1}. Since $X_m^{l-1} \cup \{ j_1^{l+1} \}$ is not a feasible job set, we have $r_1^{l+1} \leqslant d_m^{l-1}$. By induction hypothesis, we have $d_m^{l-1} \leqslant d_1^l$. Therefore, we have $r_1^{l+1} \leqslant d_1^l$ which implies that $X_1^l \cup \{ j_1^{l+1} \}$ is not a feasible job set. This contradicts our assumption. □

Lemma 15. $|X_i| - |X_k| \in \{ -1, 0, 1 \}, \forall i, k \in [m]$.

Proof. We consider the $(L+1)$-th round of Algorithm 6 in which $\exists f \in [m]$ such that a_f receives nothing in this round. Let J_f^L be the set of remaining jobs in J after a_f chooses in the $(L+1)$-th round. We consider the agent a_k such that $1 \leqslant f \leqslant k < m$. Since a_f receives nothing, we have $r_j \leqslant d_f^L, \forall j \in J_f^L$. According to Lemma 13, we have $d_f^L \leqslant d_k^L$. Then, we have $r_j \leqslant d_f^L \leqslant d_k^L, \forall j \in J_f^L$ which implies that agent a_k also receives nothing in this round. Therefore, a_m must receive nothing in the $(L+1)$-th round because there exist an agent that does not receive job in the $(L+1)$-th round. Let J_m^L be the remaining jobs in J before a_m chooses in the $(L+1)$-th round. Since a_m receives nothing in the $(L+1)$-th round, we have $r_j \leqslant d_m^L, \forall j \in J_m^L$. According to Lemma 14, we have $d_m^L \leqslant d_1^{L+1}$ if j_1^{L+1} exists. Therefore, we have $r_j \leqslant d_1^{L+1}, \forall j \in J_m^L$. Thus, a_1 will receive nothing in the $(L+2)$-th round. Now, we consider an arbitrary agent $a_h, 1 \leqslant h \leqslant m$, it is strainghtforward

to see that if a_h receives nothing in $(L+1)$-th round, then a_h will receive nothing in any L'-th round, where $L+1 < L'$. Note that, in the $(L+1)$-th round, there may exist many agents that receive nothing. Without loss of generality, we assume that a_f is the agent with the smallest index who receives nothing in the $(L+1)$-th round. Therefore, we have

$$|X_i| = \begin{cases} L, & \forall f \leqslant i \leqslant m; \\ L+1, & \forall 1 \leqslant i < f. \end{cases}$$

Thus, we have $|X_i| - |X_k| \in \{-1, 0, 1\}, \forall i, k \in [m]$. □

Lemma 16. $u_i(X_i) \geqslant u_i(X_0 \cup X_i), \forall i \in [m]$.

We will use the optimal argument for classical interval scheduling to prove Lemma 16. We restate the problem and optimal argument for completeness.

In classical interval scheduling, we are given a set of intervals $\mathcal{I} = \{I_1, I_2, \cdots, I_n\}$. Each interval is associated with a release time and a deadline. A set of intervals $\mathcal{I}'$ is called a compatible set if and only if, for every two intervals $I_k, I_h \in \mathcal{I}'$, I_k, I_h do not intersect. The goad is to find the compatible set with the maximum size. This problem can be easily solved by *Earlier Deadline First* (EDF) Kleinberg and Tardos [2006].

Proof. We consider an arbitrary agent a_i. We prove by constructing an instance of classical interval scheduling problem. Let $\mathcal{I} = X_0 \cup X_i$. Let ALGE be the interval set selected by EDF algorithm. Observe that if we can prove that $\mathsf{ALGE} = X_i$, then it implies that $u_i(X_i) \geqslant u_i(X_0 \cup X_i)$ since ALGE is the optimal solution. Suppose that $\mathsf{ALGE} = \{j'_1, j'_2, \cdots, j'_h\}$ and assume that the interval is added to ALGE by EDF algorithm in this order. Suppose that $X_i = \{j_1, j_2, \cdots, j_k\}$ and assume that the job is added to X_i by Algorithm 6 in this order. Note that $|X_i| \leqslant |\mathsf{ALGE}|$ since ALGE is the compatible set with the maximum size.

We prove by comparison. Assume that ALGE and X_i become different from the R-th element, i.e., $j_l = j'_l, \forall l \in [R-1]$ and $j_R \neq j'_R$. This implies that j'_R instead of j_R is the job with the smallest deadline in $X_0 \cup X_i \setminus \{j_1, \cdots, j_{R-1}\}$ to make $\{j_1, \cdots, j_{R-1}\} \cup \{j'_R\}$ be compatible. Note that both $\{j_1, \cdots, j_{R-1}\} \cup \{j'_R\}$ and $\{j_1, \cdots, j_{R-1}\} \cup \{j_R\}$ are feasible. Since j'_R is left to charity, there is no agent takes it away. Therefore, Algorithm 6 will assign j'_R instead of j_R to a_i. Hence, we proved $X_i \subseteq \mathsf{ALGE}$. It is easy to see that there is no interval $j'_u \in \mathsf{ALGE}$ such that $j'_u \notin X_i$ which implies that $X_i = \mathsf{ALGE}$. □

Now, we are ready to prove Theorem 8.

Proof of Theorem 8. According to Lemma 15, we know that the feasible schedule $\mathbf{X}$ returned by Algorithm 6 is an EF1 schedule. According to Lemma 16, $\mathbf{X}$ is also an IO schedule. Hence, Algorithm 6 returns a feasible schedule that is simultaneously EF1 and IO.

Now, we prove the running time. Line 1 requires running time $O(n \log n)$, where n is the number of jobs. Line 4-13 requires running time $O(n)$. Hence, the running time of Algorithm 6 can be bounded by $O(n \log n)$. □

Now, we show an instance that Algorithm 6 returns a schedule that is not PO schedule. See Figure 10. By applying Algorithm 6 to the instance in Figure 10, let $\mathbf{X}$ be the returned schedule. Then, we have $\mathbf{X} = (X_1, X_2)$, where $X_1 = \{j_1, j_4\}, X_2 = \{j_2, j_5\}$ and $X_0 = \{j_3, j_6\}$. But a possible PO schedule is $\mathbf{X}' = (X'_1, X'_2)$, where $X'_1 = \{j_1, j_4, j_5\}, X'_2 = \{j_2, j_6\}$ and $X'_0 = \{j_3\}$.

D.3 A polynomial time algorithm for FISP with ⟨unweighted, flexible⟩

Note that Algorithm 6 can be modified to run on instances of FISP with ⟨unweighted, flexible⟩. But this modified algorithm fails to return an IO schedule. This is not surprising as it has been proved in Garey and Johnson [1979] that even with a single machine, finding an IO schedule is NP-hard. Fortunately, the modified algorithm still runs in polynomial time and always returns a schedule that is EF1 and 1/2-IO.

Before giving the round-robin algorithm, we first re-state the following classical scheduling problem.

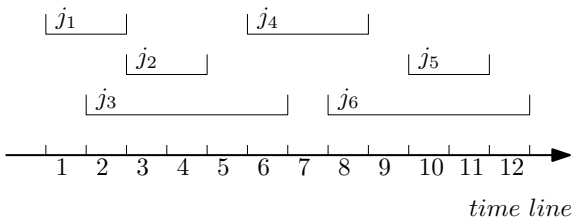


Figure 10: Instance for which Algorithm 6 fails to return a PO schedule. In the above instance, we have $J = \{ j_1, j_2, j_3, j_4, j_5, j_6 \}, A = \{ a_1, a_2 \}$. The job windows are $T_1 = \{ 1, 2 \}, T_2 = \{ 3, 4 \}, T_3 = \{ 2, 3, 4, 5, 6 \}, T_4 = \{ 6, 7, 8 \}, T_5 = \{ 10, 11 \}, T_6 = \{ 8, 9, 10, 11, 12 \}$, respectively.

Scheduling to find the maximum compatible job set We are given a job set J which contains n jobs, i.e., $J = \{ j_1, j_2, \cdots, j_n \}$, with each job regraded as a tuple, i.e., $j_i = (r_i, p_i, d_i), i \in [n], 1 \leqslant p_i \leqslant d_i - r_i + 1$, where r_i, p_i, d_i are the release time, processing time and deadline, respectively. There is one machine which is used to process jobs. A subset J' of jobs is called *compatible job set* if and only if all jobs in J' can be finished without preemption before their deadlines. The objective is to find a compatible job set with the maximum size.

The above scheduling problem is the optimization version of the scheduling problem SEQUENCING WITH RELEASE TIMES AND DEADLINES, which is strongly NP-complete Garey and Johnson [1979]. In Bar-Noy *et al.* [2001], they give an $\frac{(m+1)^m}{(m+1)^m - m^m}$-approximation algorithm for m identical machines case. In particular, the approximation ratio is 2 when $m = 1$. We restate the greedy algorithm for completeness (Algorithm 7).

Algorithm 7. 2-approximation for scheduling problem on single machine.

$\mathsf{ALGC} = \emptyset$.
$J^* = J$.
$D = 0$.
while $J^* \neq \emptyset$ **do**
  $J^* = \emptyset$. // reset J^*.
  for every job $j \in J$ **do**
    if $d_j \leqslant \max\{D, r_j\} + p_j$ **then**
      $J^* = J^* \cup \{ j \}$.
    end if
  end for
  $j^* = \arg\min_{j \in J^*} \{\max\{D, r_j\} + p_j\}$.
  Schedule job j^* at time slot $\max\{D, r_j\}$.
  $D = \max\{D, r_{j^*}\} + p_{j^*}$.
  $\mathsf{ALGC} = \mathsf{ALGC} \cup \{ j^* \}$.
end while

Theorem 9. *A schedule that is simultaneously EF1 and 1/2-IO exists and can be found in polynomial time for all instance of FISP* $\langle$*unweighted, flexible*$\rangle$.

Now, we are ready to give the algorithm (Algorithm 8) for instance of FISP $\langle$unweighted, flexible$\rangle$.

Let $\mathbf{X} = (X_1, \cdots, X_m)$ be the schedule returned by Algorithm 8 and $X_0 = J \setminus \bigcup_{i \in [m]} X_i$. We first show that $\mathbf{X}$ is an 1/2-IO schedule and then prove that $\mathbf{X}$ is an EF1 schedule.

Lemma 17. $u_i(X_i) \geqslant \frac{1}{2} \cdot u_i(X_i \cup X_0), \forall a_i \in A$.

Proof. We consider an arbitrary agent $a_i \in A$ and the job set $X_i \cup X_0$. Let ALGC be the set of jobs selected from $X_0 \cup X_i$ by Algorithm 7. Let OPTC be the set of jobs selected by the optimal algorithm. Since Algorithm 7 is a 2-approximation algorithm, we have $|\mathsf{ALGC}| \geqslant \frac{1}{2} \cdot |\mathsf{OPTC}|$. Observe that if we can prove that $\mathsf{ALGC} = X_i$, then we have $u_i(X_i) \geqslant \frac{1}{2} \cdot u_i(X_i \cup X_0)$ since ALGC has the size at least half of the optimal solution. Let $\mathsf{ALGC} = \{ j_1, j_2, \cdots, j_k \}$ and assume that the jobs are added to the solution by Algorithm 7 in this order. Let $X_i = \{ j'_1, j'_2, \cdots, j'_r \}$ and assume that the jobs

Algorithm 8. Round-Robin for FISP ⟨unweighted, flexible⟩

Input: Agent set A and job set J.
Output: EF1 schedule $\mathbf{X} = (X_1, \cdots, X_m)$.

$X_1 = \cdots = X_m = \emptyset$.
$J_1^* = \cdots = J_m^* = J$.
$D_1 = \cdots = D_m = 0$.
$i = 1$. // The index.
while there is a $J_i^* \neq \emptyset$ **do**
  for all $a_i \in A$ **do**
    $J_i^* = \emptyset$. // reset J_i^*.
    for every job $j \in J$ **do**
      if $d_j \leqslant \max\{D_i, r_j\} + p_j$ **then**
        $J_i^* = J_i^* \cup \{ j \}$.
      end if
    end for
    $j_i^* = \arg\min_{j \in J_i^*} \{\max\{D_i, r_j\} + p_j\}$.
    $X_i = X_i \cup \{ j_i^* \}$.
    Schedule job j_i^* at $\max\{D_i, r_j\}$.
    $D_i = \max\{D_i, r_{j_i^*}\} + p_{j_i^*}$.
    $J = J \setminus \{ j_i^* \}$.
  end for
end while
$X_0 = J \setminus \bigcup_{i \in [m]} X_i$.

are added to X_i by Algorithm 8 in this order. We prove by comparison. Assume that ALGC and X_i become different from the R-th element, i.e., $j_l = j'_l, \forall l \in [R-1]$ and $j_R \neq j'_R$. Assume that the completion time of j_{R-1} is D_{R-1}. Then, we have

$$j'_R \neq j_R = \arg\min_{j \in J_R^*} \{\max\{D_{R-1}, r_j\} + p_j\},$$

where $J_R^* \subseteq \mathcal{J}_r = X_0 \cup X_i \setminus \{ j_1, \cdots, j_R \}$ is a set of jobs which can be feasibly scheduled after j_R, i.e.,

$$J_R^* = \{ j \in \mathcal{J}_r \mid \max\{D_{R-1}, r_j\} + p_j \leqslant d_j \}.$$

Note that $j'_R \in J_R^*$. Since j'_R instead of j_R is assigned to agent a_i in a certain round, we know that j_R must be assigned to a certain agent before agent a_i chooses, i.e., $j_R \in X_k, \exists k \in [m]$. This contradicts our assumption since $j_R \in X_0$.

□

Let J_i^l be the job set J_i^* for agent $a_i \in A$ in the l-th round, where $1 \leqslant i \leqslant m$ and $1 \leqslant l \leqslant L$. Suppose that in first L-th rounds of Algorithm 8, $J_i^* \neq \emptyset, \forall i \in [m]$, i.e., in the $(L+1)$-th round, $\exists i \in [m]$ such that $J_i^* = \emptyset$. Let D_i^l be the parameter D_i in Algorithm 8 for agent $a_i \in A$ at the end of the l-th round, where $1 \leqslant i \leqslant m$ and $1 \leqslant l \leqslant L$.

Lemma 18. *$D_1^l \leqslant D_2^l \leqslant \cdots \leqslant D_m^l, \forall l \in [L]$. Moreover, we have $D_m^l \leqslant D_1^{l+1}, \forall l \in [L-1]$, and $D_m^L \leqslant D_1^{L+1}$ if $J_1^{L+1} \neq \emptyset$.*

Proof. We first prove $D_1^l \leqslant D_2^l \leqslant \cdots \leqslant D_m^l, \forall l \in [L]$. Let $a_k, a_h \in A$ be two agents, where $1 \leqslant k < h \leqslant n$. We only need to prove $D_k^l \leqslant D_h^l, \forall l \in [L]$. We prove by induction. In the base case where $l = 1$, $D_k^1 \leqslant D_h^1$ obviously holds since agent a_k chooses the job before a_h. Now, we have induction hypothesis $D_k^l \leqslant D_h^l$ and we need to prove $D_k^{l+1} \leqslant D_h^{l+1}$. We prove by contradiction and assume that $D_k^{l+1} > D_h^{l+1}$. Let j_k^{l+1}, j_h^{l+1} be the jobs that are selected by agent a_k, a_h in the $(l+1)$-round of Algorithm 8, respectively. Hence, we have

$$D_k^{l+1} = \max\{D_k^l, r_{j_k^{l+1}}\} + p_{j_k^{l+1}},$$
$$D_h^{l+1} = \max\{D_h^l, r_{j_h^{l+1}}\} + p_{j_h^{l+1}}.$$

By induction hypothesis $D_k^l \leqslant D_h^l$, we have

$$\max\{D_k^l, r_{j_h^{l+1}}\} + p_{j_h^{l+1}} \leqslant \max\{D_h^l, r_{j_h^{l+1}}\} + p_{j_h^{l+1}} \leqslant d_{j_h^{l+1}}.$$

This implies that $j_h^{l+1} \in J_k^{l+1}$. Since

$$\max\{D_k^l, r_{j_h^{l+1}}\} + p_{j_h^{l+1}} \leqslant D_h^{l+1} < D_k^{l+1},$$

we have

$$\max\{D_k^l, r_{j_h^{l+1}}\} + p_{j_h^{l+1}} < \max\{D_k^l, r_{j_k^{l+1}}\} + p_{j_k^{l+1}}.$$

This implies that j_h^{l+1} instead of j_k^{l+1} will be chosen by agent a_k in the $(l+1)$-th round of Algorithm 8. This contradicts our assumption.

We assume that $J_1^{L+1} \neq \emptyset$ and prove that $D_m^l \leqslant D_1^{l+1}, \forall l \in [L]$. We prove $D_m^l \leqslant D_1^{l+1}$ holds for any $1 \leqslant l \leqslant L$. Note that $D_r^0 = 0, \forall r \in [m]$. We prove by induction. In the base case where $l = 1$, if $D_m^1 > D_1^2$, a_m will choose j_1^2 instead of j_m^1 in the first round. Now, we have induction hypothesis $D_m^l \leqslant D_1^{l+1}$ and we need to prove that $D_m^{l+1} \leqslant D_1^{l+2}$ holds. We prove by contradiction and assume that $D_m^{l+1} > D_1^{l+2}$. Let j_m^{l+1}, j_1^{l+2} be the jobs that are selected by agent a_m, a_1 in the $(l+1), (l+2)$-th round of Algorithm 8, respectively. Note that in the case where $l = L-1$, there always exists a job j_m^{l+2} since $\{ j_m^{L+1} \} \neq \emptyset$. Hence, we have

$$D_m^{l+1} = \max\{D_m^l, r_{j_m^{l+1}}\} + p_{j_m^{l+1}},$$
$$D_1^{l+2} = \max\{D_1^{l+1}, r_{j_1^{l+2}}\} + p_{j_1^{l+2}}.$$

By induction hypothesis $D_m^l \leqslant D_1^{l+1}$, we have

$$\max\{D_m^l, r_{j_1^{l+2}}\} + p_{j_1^{l+2}} \leqslant \max\{D_1^{l+1}, r_{j_1^{l+2}}\} + p_{j_1^{l+2}} \leqslant d_{j_1^{l+2}}.$$

This implies that $j_1^{l+2} \in J_m^{l+1}$. Since

$$\max\{D_m^l, r_{j_1^{l+2}}\} + p_{j_1^{l+2}} \leqslant D_1^{l+2} < D_m^{l+1},$$

we have

$$\max\{D_m^l, r_{j_1^{l+2}}\} + p_{j_1^{l+2}} < \max\{D_m^l, r_{j_m^{l+1}}\} + p_{j_m^{l+1}}.$$

This implies that j_1^{l+2} instead of j_m^{l+1} will be chosen by agent a_m in the $(l+1)$-th round of Algorithm 8. This contradicts our assumption. □

Lemma 19. *$J_1^l \supseteq J_2^l \supseteq \cdots \supseteq J_m^l, \forall l \in [L+1]$. Moreover, we have $J_m^l \supseteq J_1^{l+1}, l \in [L]$.*

Proof. We first prove that $J_1^l \supseteq J_2^l \supseteq \cdots \supseteq J_m^l, \forall l \in [L]$. Let $a_k, a_h \in A$ be two agents, where $1 \leqslant k < h \leqslant n$. We only need to prove $J_k^l \supseteq J_h^l$. To prove $J_h^l \subseteq J_k^l$, we consider an arbitrary job $j \in J_h^l$ and show that $j \in J_k^l$. Note that $J_k^l, J_h^l \neq \emptyset, \forall l \in [L]$. Let $\mathcal{J}_s^k, \mathcal{J}_s^h$ be the set of jobs that are already assigned to the agents before agent a_k and a_h select, respectively. Note that $\mathcal{J}_s^k \subseteq \mathcal{J}_s^h$. According to Algorithm 8, we have

$$J_k^l = \{ j \in (J \setminus \mathcal{J}_s^k) \mid d_j \leqslant \max\{D_k^{l-1}, r_j\} + p_j \},$$
$$J_h^l = \{ j \in (J \setminus \mathcal{J}_s^h) \mid d_j \leqslant \max\{D_h^{l-1}, r_j\} + p_j \}.$$

Since $\mathcal{J}_s^k \subseteq \mathcal{J}_s^h$, we have $J \setminus \mathcal{J}_s^k \supseteq J \setminus \mathcal{J}_s^h$. Now we consider an arbitrary job $j \in J_h^l$ and show that j is also a member of J_k^l. Since $j \in (J \setminus \mathcal{J}_s^h)$ and $J \setminus \mathcal{J}_s^k \supseteq J \setminus \mathcal{J}_s^h$, we have $j \in (J \setminus \mathcal{J}_s^k)$. Since $j \in J_h^l$, we have

$$d_j \leqslant \max\{D_h^{l-1}, r_j\} + p_j.$$

According to Lemma 18, $D_h^{l-1} \geqslant D_k^{l-1}$, we have

$$d_j \leqslant \max\{D_k^{l-1}, r_j\} + p_j,$$

which implies that $j \in J_k^l$. Now we consider the case where $l = L+1$. Note that in the $(L+1)$-th round of Algorithm 8, $\exists i \in [m]$ such that $J_i^{L+1} = \emptyset$. Now we prove that $J_k^{L+1} \supseteq J_h^{L+1}$. If

$J_h^{L+1} = \emptyset$, then we are done. Hence, we assume that $J_h^{L+1} \neq \emptyset$. By a similar argument, a job $j \in J_h^{L+1}$ has the property $d_j \leqslant \max\{D_h^L, r_j\} + p_j$. Then we have $d_j \leqslant \max\{D_k^L, r_j\} + p_j$ holds since $D_k^L \leqslant D_h^L$. Then we have $j \in J_k^{L+1}$.

Now, we prove that $J_m^l \supseteq J_1^{l+1}, \forall l \in [L]$. Note that it is possible that $J_1^{L+1} = \emptyset$. In this case $J_m^L \supseteq J_1^{L+1}$ trivially holds. Hence, we assume that $J_1^{L+1} \neq \emptyset$. To prove $J_m^l \supseteq J_1^{l+1}$, we consider an arbitrary job $j \in J_1^{l+1}$ and show that $j \in J_m^l$. Let $\mathcal{J}_s^m, \mathcal{J}_s^1$ be the set of jobs that are already assigned to the agents before agent a_m and a_1 select in the $l, (l+1)$-th round of Algorithm 8, respectively. Note that $\mathcal{J}_s^m \subseteq \mathcal{J}_s^1$. According to Algorithm 8, we have

$$J_m^l = \{\, j \in (J \setminus \mathcal{J}_s^m) \mid d_j \leqslant \max\{D_m^{l-1}, r_j\} + p_j \,\},$$
$$J_1^{l+1} = \{\, j \in (J \setminus \mathcal{J}_s^1) \mid d_j \leqslant \max\{D_1^l, r_j\} + p_j \,\}.$$

Since $\mathcal{J}_s^m \subseteq \mathcal{J}_s^1$, we have $J \setminus \mathcal{J}_s^m \supseteq J \setminus \mathcal{J}_s^1$. Now we consider an arbitrary job $j \in J_1^{l+1}$ and show that $j \in J_m^l$. Since $j \in J_1^{l+1}$, we have

$$d_j \leqslant \max\{D_1^l, r_j\} + p_j.$$

According to Lemma 18, we have $D_m^{l-1} \leqslant D_1^l$. Then, we have

$$\max\{D_1^l, r_j\} + p_j \geqslant \max\{D_m^{l-1}, r_j\} + p_j.$$

Hence, we have $d_j \leqslant \max\{D_m^{l-1}, r_j\} + p_j$ which implies that $j \in J_m^l$. □

Lemma 20. $|X_i| - |X_k| \in \{-1, 0, 1\}, \forall i, k \in [m]$.

Proof. We consider the $(L+1)$-th round of Algorithm 8 in which there exists an agent $a_i \in A$ such that a_i does not choose any jobs for the first time. Note that there may exist many agents that do not choose any job for the first time in $(L+1)$-th round. We assume that a_f is the first agent that chooses nothing in the $(L+1)$-th round. Since a_f chooses nothing, we have $J_f^{L+1} = \emptyset$. According to Lemma 19, we have $J_i^{L+1} = \emptyset, \forall f \leqslant i \leqslant n$. Moreover, we have $J_i^{L'} = \emptyset, \forall i \in [m]$ and $L+1 < L'$. Therefore, we have

$$|X_i| = \begin{cases} L, & \forall f \leqslant i \leqslant m; \\ L+1, & \forall 1 \leqslant i < f. \end{cases}$$

This implies that Lemma 20 holds. □

Now we are ready to prove Theorem 9.

Proof of Theorem 9. According to Lemma 20 and Lemma 17, we know that Algorithm 8 will return a feasible schedule that is simultaneously EF1 and 1/2-IO.

Now we bound the running time. According to Lemma 20 and Lemma 17, we know that line 5-20 will be run at most $\lceil \frac{n}{m} \rceil$ times, where n is the number of jobs and m is the number of agents. In each while loop, line 6-18 will be run at most m times. In each for loop, line 8-12 will be run at most n times and the running time of line 13 can be bounded by $O(n)$. Hence, we have the running time of Algorithm 8 $O(\lceil \frac{n}{m} \rceil \cdot m \cdot (n^2 + n)) = O(mn^3)$. □

E Missing Discussions in Experiments in Section 5

We now empirically test the performance of Algorithm 4 when jobs are rigid, comparing it against a simple Round-Robin algorithm. In this simple Round-Robin algorithm, all jobs are sorted by their deadlines in non-decreasing order. Then every agent picks a job in round-robin manner. Finally, every agent computes the compatible intervals with the maximum weight and all the remaining jobs will be assigned to charity. The formal description can be found in Algorithm 9 with $J' = J$ and $A' = A$. For the experiments, we have implemented both Algorithm 4 and the above round-robin algorithm.

We run our experiments on three job sets with different sizes: 100 (Figure 1 (a)), 500 (Figure 1 (b)) and 1000 (Figure 1 (c)). The release time and deadline of each job is uniformly randomly sampled from the interval [0,50]. For each job set, we further set up three subgroups according to the agents'

Algorithm 9. Round-Robin (RR)

Input: Agent set A' and job set J'.
Output: EF1 schedule $\mathbf{X} = (X_1, \cdots, X_{|A'|})$

Sort all jobs by their deadline in non-decreasing order.
$X_1 = X_2 = \cdots = X_{|A'|} = \emptyset$.
$i = 1, k = 1$. // The index.
for all $j_k \in J'$ **do**
  for all $a_i \in A'$ **do**
    if $k \mod |A'| = i$ **then**
      $X_i = X_i \cup \{ j_k \}$.
    end if
  end for
end for
$i = 1$. // Reset the index
for all X_i **do**
  Let $X_i' \subseteq X_i$ be the compatible job set with the maximum weight for agent a_i.
  $X_i = X_i'$.
end for
$X_0 = J \setminus \bigcup_{i \in [|A'|]} X_i$.

utility of every job: (i) the utility gain is sampled uniformly randomly from [1,20]; (ii) the utility gain follows Poisson Distribution with means 50; (iii) the utility gain follows Normal Distribution with means 25 and variance 10. For each subgroup, we further set up three subsubgroups according to the size of agent set: 5, 10 and 15.

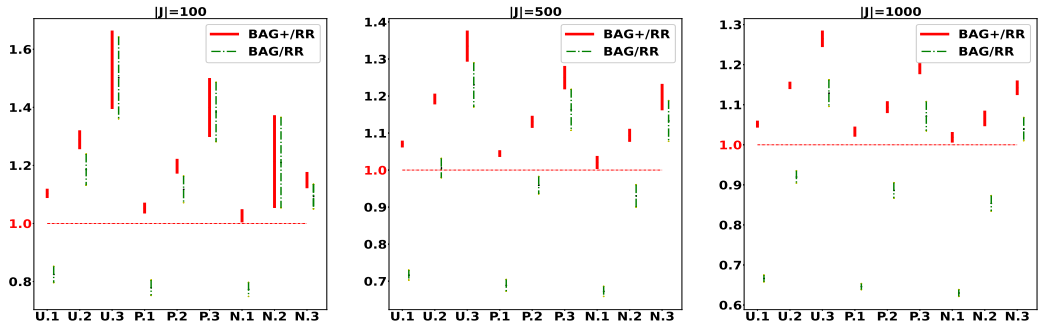
In total, our experiment contains $3 \times 3 \times 3$ groups. For each group, we run Algorithm 4 and Round-Robin algorithm on 1000 different instances. Noted that Algorithm 4 does not have a good performance when the number of jobs is much larger than the number of agents, e.g., the groups with 5 agents (U.1, P.1, N.1) in Figure 1. The reason Algorithm 4 performances unsatisfactorily is that Algorithm 4 stops at the threshold while there are a lot of remaining jobs. To fix this problem, we add the Round-Robin procedure at the end of Algorithm 4, i.e., if there exist some unallocated jobs at the end of Algorithm 4, we run Round-Robin algorithm on the remaining job set. Finally, every agent computes the maximum compatible job set from the union of the job set returned by Algorithm 4 and Round-Robin algorithm. The formal description can be found in Algorithm 10. Let BAG+ be the updated version of Algorithm 4 and BAG be the original one. With the help of the Round-Robin procedure, the performance of Algorithm 4 is better than the Round-Robin algorithm in all groups.

Algorithm 10. Matching-BagFilling + Round-Robin (BAG+)

Input: Agent set A and job set J.
Output: EF1 schedule $\mathbf{X} = (X_1, \cdots, X_m)$

Run Algorithm 4.
Let $\mathbf{X} = (X_1, \cdots, X_m)$ be the schedule returned by Algorithm 4.
Let $X_0 = J \setminus \bigcup_{i \in [m]} X_i$.
if $X_0 \neq \emptyset$ **then**
  Run Algorithm 9 with job set X_0 and agent set A.
  Let $\mathbf{X}' = (X_1', \cdots, X_m')$ be the schedule returned by Algorithm 9.
end if
$i = 1$. // The index.
for all X_i **do**
  Let $X_i'' \subseteq (X_i \cup X_i')$ be the compatible job set with the maximum weight for agent a_i.
  $X_i = X_i''$.
end for
$X_0 = J \setminus \bigcup_{i \in [m]} X_i$.

According to Figure 1, it is not hard to see that Algorithm 4 is not able to achieve a good performance when the number of jobs is much larger than the number of agents. When the size of the job set is



(a) Groups with $|J| = 100$. (b) Groups with $|J| = 500$. (c) Groups with $|J| = 1000$.

Figure 1: The results of the evaluation of Algorithm 4, Algorithm 10 and Algorithm 9 on different settings. Every subfigure represents the groups with same job set size. Every notation in x-axis represents a setting. Notations "U.", "P.", "N.", represent the utility gain follows the Uniform, Poisson, Normal Distribution, respectively. Notations "1", "2", "3", represent the number of agents is 5, 10, 15, respectively. The top and bottom point of every solid red interval represent the maximum and minimum value of BAG+/RR among all the agents, where BAG+/RR is the ratio of total gain that the agents receive when we run BAG+ and RR algorithm. The top and bottom point of every dot-dashed green interval represent the maximum and minimum value of BAG/RR among all the agents, where BAG/RR is the ratio of total gain that the agents receive when we run BAG and RR algorithm.

100, Algorithm 4 performs worse than Round-Robin only in the setting where the agent set is 5 (see Figure 1 (a), only U.1, P.1, N.1's green interval is behind 1.0). When we increase the number of jobs to 500, the situation that Algorithm 4 is worse than Round-Robin begins to appear at $|A| = 10$ (see Figure 1 (b), part of green interval of U.2 begins to appear behind 1.0). When we further increase the number of jobs to 1000, Algorithm 4 performs better than Round-Robin only in the setting where there are 15 agents (see Figure 1 (c), only U.3, P.3, N.3's green interval is above 1.0).

The reason is that Algorithm 4 stops at the case where every agent gets the threshold but there are a lot of remaining jobs. We can fix this issue by adding an extra round-robin procedure to allocate the remaining jobs, and thus yield BAG+ algorithm. According to Figure 1, we can find that the performance of BAG+ is better than Round-Robin in all settings as all red intervals are above 1.0. Thus, BAG+ algorithm can achieve a good performance in practices and guarantee the approximation in the worst case.