Scheduling of Two Agents Task Chains with a Central Selection Mechanism

Alessandro Agnetis
Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche
Università degli Studi di Siena, Italy, agnetis@dii.unisi.it

Gaia Nicosia
Dipartimento di Ingegneria
Università degli Studi Roma Tre, Italy, nicosia@dia.uniroma3.it

Andrea Pacifici
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli Studi di Roma "Tor Vergata", Italy, pacifici@disp.uniroma2.it

Ulrich Pferschy
Department of Statistics and Operations Research
University of Graz, Austria, pferschy@uni-graz.at

Abstract

In this paper we address a deterministic scheduling problem in which two agents compete for the usage of a single machine. Each agent decides on a fixed order to submit its tasks to an external coordination subject, who sequences them according to a known priority rule. We consider the problem from different perspectives. First, we characterize the set of Pareto optimal schedules in terms of size and computational complexity. We then address the problem from the single-agent point-of-view, that is we consider the problem of deciding how to submit one agent's tasks only taking into account its own objective function against the other agent, the opponent. In this regard, we consider two different settings depending on the information available to the agents: In one setting the considered agent knows in advance all information about the submission sequence of the opponent, in the second setting (as in minimax strategies in game theory) the agent has no information on the opponent strategy and wants to devise a strategy that minimizes its solution cost in the worst possible case. Finally, we assess the performance of some classical single-agent sequencing rules in the two-agent setting.

Keywords: scheduling, multi-agent optimization, bicriteria optimization.

1 Introduction

Classical scheduling problems deal with situations in which a set of *tasks* has to be processed on some processing *resource*. In addition, in *two-agent scheduling problems* there are two agents, each task belongs to one agent, and each agent is only interested in optimizing his/her own performance index. Although these problems can be viewed as a special case of bicriteria scheduling problems [22], their specific properties and applications have spurred a considerable amount of research since the seminal work by Agnetis et al. [2] and Baker and Smith [7]. For a detailed and exhaustive view on multiagent scheduling problems, one can refer to the recent book [1].

In the scenarios considered so far in the literature, the analysis mainly focused on the characterization of reasonable compromise solutions. The most basic and relevant concept in this respect is that of a *Pareto optimal solution*, where no better solution exists for one agent without increasing the other agent's cost function. In most previous studies, the viewpoint commonly adopted assumes that complete information concerning the agents' tasks is available, either to the agents themselves, or to an external *coordinator*. In this view, the coordinator role is to compute and actually submit to the agents certain compromise schedules (e.g., chosen among those Pareto optimal schedules that also ensure a certain degree of *fairness*, see [5]).

Here we take a different view, justified by the fact that two additional features may hold in practice. One is that it might be undesirable or technically difficult for the agents to disclose all the information concerning their tasks or strategies to any external subject, either to a coordinator or a competing agent (*Incomplete information*). Another is that the coordinator does not have a facilitating role, rather he/she is specifically involved in the scheduling process, e.g., being the owner of the processing resource. As a consequence, the coordinator may have its own objective, which may be independent from, or even conflicting with, the agents' objectives (*Third-party objective*).

Examples of the above situations can be found in the literature. When information is only partially exchanged, some sort of *mechanism* has to be devised to reach a mutually acceptable resource allocation. In this respect, several authors have analyzed various auction mechanisms [12, 23], in which the auctioneer is the owner of the resource and the agents bid for resource time slots, hence giving explicit monetary evaluations of the various slots (or combinations thereof). Yet, in certain cases the objective of each agent is better expressed by means of traditional logistic measures, such as average completion time, makespan, tardiness etc. An example is the setting described by [21], in which different airlines, each having an earliness/tardiness-related cost function, compete for the usage of runways of an airport, and they are not going to disclose detailed cost information to other companies or to airport managers.

Situations in which a third-party is involved in the scheduling process can be found in some manufacturing contexts. For instance, different orders (agents) share a machine, and there is a due date attached to each task. However, the machine owner also has a global throughput objective which can be expressed in terms of total flow time minimization [20], corresponding to the minimization of average work-in-process. These situations have been addressed as *multi-agent scheduling problems with a global objective function* [13]. Also, scenarios have been analyzed in which multiple organizations, owning multiple tasks, partially share their computing resources. Scheduling algorithms have been proposed to reconcile individual objectives of each organization [8]. Similar circumstances also occur in telecommunication systems, where a *capacity planner* must schedule a channel to two or more users in order to maximize their on-time traffic [6]. Additionally, the design and analysis of coordination mechanisms for machines, which aim to minimize the so-called *price of anarchy*, have been addressed in [24].

In this work we propose a different model, introduced in [3], which accounts for both incomplete information exchange and resource owner's objectives. There are two agents, call them A and B, each owning a set of nonpreemptive tasks that require a single machine to be processed and each agent with its own objective. There is also one coordinator (e.g., the machine owner), who is interested in maximizing the machine throughput (a typical goal in many manufacturing settings), which can be defined as the number of processed tasks per time unit. Since the coordinator has no detailed information on the number and length of the tasks which will be submitted by the agents, he/she sensibly gives precedence to shorter tasks, among those that are promoted for execution. More precisely, to regulate access to the machine, the coordinator defines a *selection mechanism*, consisting of the iterative application of the following steps.

1. Each agent submits one of its tasks.

2. The coordinator selects the *shortest* between submitted tasks for processing.

3. The selected task is scheduled at the end of the current schedule, which is initially empty.

Each repetition of the above steps is a *round*, and the selected task is referred to as the *winner* of the round. At the beginning, each agent decides which task to submit for the first round. Then, the agent whose task has been selected submits a new task for the second round, while the losing task remains automatically submitted for the second round, and so on. The above steps are repeated until all tasks of one agent have been processed (the remaining tasks of the other agent are appended thereafter). The mechanism is supposed to be common knowledge for the agents. Some variants of this protocol are discussed in Sections 5 and 6.

Note that, under this mechanism, the coordinator's objective plays an important role and heavily affects the agents' strategies and behavior. This reflects a specific interest of the coordinator. If the coordinator were a neutral subject, other mechanisms would be sensible (e.g. round robin, in which agents simply take turns in submitting the next task to the machine). As a consequence, here the coordinator is not interested in enforcing fair solutions, and the agents have to account for this when deciding their submission sequence. A variant of this mechanism which takes a certain fairness aspect into account will be briefly investigated in Section 5. Actually, in [14] simulation experiments are presented for a computer grid in which a central scheduler selects jobs for processing according to the SPT rule, and this apparently results in enhanced throughput.

In many technical environments it may not be easy (or even possible) for an agent to withdraw a submitted task, i.e., once an agent has submitted a task to the machine, it only submits a new task after the previous task is accepted and hence processed on the machine. In other words, the losing task in each round remains submitted until it wins. Note that this does not imply that the agents must decide beforehand the whole submission sequence, but they can decide on the next submission based on their experience from the previous rounds. This situation can also be represented by two linear conveyor belts, one for each agent, transporting parts (tasks) to the machine. As soon as an agent puts a part on the conveyor, it can not remove it anymore. Hence, at each round, one of the two candidate tasks is the loser of the preceding round.

Looking at the final schedule, the tasks of one agent appear exactly in the order they were submitted by that agent, but interrupted by tasks of the other agent. For this reason, the output of any algorithm (or strategy) that suggests an agent the tasks to submit, may be regarded as a *chain* of tasks. In this context we say that a schedule σ is *feasible* if there exists a pair of chains, i.e., one sequence of tasks for each agent, which allows to obtain σ as a result of the described scheduling mechanism. Note that not all schedules are feasible.

In this scenario, we consider the problem from different perspectives.

- *Centralized perspective.* This analysis, as in a bicriteria optimization problem, aims at characterizing the set of Pareto optimal solutions for the two agents in terms of size and computational complexity. In fact, the two agents may be willing to cooperate and possibly compensate each other in the face of a schedule that favors one of the agents. For this purpose, the agents need to elaborate on the schedules which can actually be attained as a result of the application of the coordinator's selection mechanism.

- *Single agent perspective.* Considering an agent (say, B), its *strategy* consists in deciding which task to submit at each round, taking into account its own objective function. As in

[10, 16, 17, 18] we consider two different settings concerning the information available to agent B.

- *Offline.* Agent B knows in advance the length of each task of the opponent, as well as its submission sequence. In this case, B wants to determine how to sequence its tasks under such an advantageous asymmetry of information.
- *Online.* Agent B knows the length of each task of the opponent (agent A), but B has no information at all on A's strategy. In this case, B may want to select a strategy that minimizes its solution cost in the worst possible case, i.e., for any strategy of A. This corresponds to what is usually called *minimax strategy* in game theory.

  Also, in this context one is interested in assessing the performance of some well-known, classical single-agent sequencing rules in the two-agent setting.

In this paper we consider that the agents pursue the minimization of the most commonly used objective functions in scheduling problems, i.e., makespan, total flow time and total weighted flow time.

The paper is organized as follows. In Section 2 we formally introduce the problem and the notation, and summarize all the results of the paper in a table. Section 3 deals with the centralized perspective. We show that there can be an exponential number of Pareto optimal schedules and finding one of them is an $\mathcal{NP}$-complete problem. Section 4 considers the algorithmic perspectives for one agent: Minimization of makespan and total completion time are easy to handle as shown in Section 4.1. Far more involved is the case of weighted total completion time which is considered in Sections 4.2 - 4.4. Section 5 reports results concerning a variation of the coordinator selection rule in which some fairness issues are taken into account. Finally, in Section 6 some conclusions are drawn.

2 Problem formulation and notation

2.1 Notation and problem setting

Let A and B denote the two agents. Each agent owns a set of n nonpreemptive tasks[1] to be performed on a single machine. Tasks have nonnegative integer deterministic processing times $a_1 \leq a_2 \leq \ldots \leq a_n$ for agent A and $b_1 \leq b_2 \leq \ldots \leq b_n$ for agent B. All tasks are available at time 0. All processing times are known to both agents. To keep notation simple, we will frequently identify tasks by their processing times and we denote the task sets as A and B whenever this does not create confusion. Sometimes each task also has a *weight* indicating its importance. We will only need explicit weight values for agent B and thus define a weight w_j for each task b_j, $j = 1, \ldots, n$.

Each agent chooses a strategy, i.e., a submission sequence (chain) of its tasks denoted by σ^A and σ^B. The application of the coordinator selection mechanism to σ^A and σ^B results in a schedule $\sigma = \mathcal{M}(\sigma^A, \sigma^B)$. Each agent wants to optimize its own objective function, which only depends on the completion times of its tasks: $f^A(\sigma) = f^A(\sigma^A, \sigma^B) = f^A(C_1^A(\sigma), \ldots, C_n^A(\sigma))$ and, similarly, $f^B(\sigma) = f^B(\sigma^A, \sigma^B) = f^B(C_1^B(\sigma), \ldots, C_n^B(\sigma))$, where $C_i^X(\sigma)$ is the completion time of task i of agent X ($i = 1, \ldots, n$, $X = A, B$) in σ. In this paper, we will consider various objective function pairs (f^A, f^B), namely we consider the minimization of

1. makespan $f^X = \max\{C_1^X, \ldots, C_n^X\}$.

[1]Note that this assumption is without loss of generality: The case in which the two agents tasks sets have different cardinality can be easily addressed by considering dummy tasks with 0 processing times and weights.

2. total completion time $f^X = \sum_{i=1}^{n} C_i^X$

3. total weighted completion time, e.g. $f^B = \sum_{i=1}^{n} w_i C_i^B$

The machine can process only one task at a time and each agent wants to optimize its own cost function. The decision process is divided into $2n$ *rounds* in which each agent submits one task — if available — for possible processing. The *shortest* between the two submitted tasks is selected and scheduled at the end of the current schedule, which is initially empty. Ties are broken by giving preference to one of the agents, e.g. agent A. In the following, we say that the agent owning the selected task *wins* against the adversary agent that, therefore, *loses*. Note that in the last rounds, possibly only one agent has available tasks to submit, because all the n tasks of the other agent have been already processed.

At the beginning, each agent decides which task to submit for the first round. Then, the winning agent submits a new task for the second round, while the losing task remains automatically submitted for the second round, and so on. Recalling the definition of Section 1, any schedule obtained through this mechanism is *feasible*.

A feasible solution can be viewed as a sequence of A-*blocks* and B-*blocks* alternately:

$$\langle A_1, B_1, A_2, B_2, \ldots A_q, B_q \rangle \tag{1}$$

where an A-block (resp., B-block) is a maximal subsequence of consecutive A-tasks (resp., B-tasks). Note that the first and the last blocks A_1 and B_q may be empty. Each A-block starts with some task a' which wins against some task b', i.e. $a' \leq b'$. Since b' remains submitted until it wins, the A-block may contain a number of additional tasks also winning against b'. The block ends when A submits an item losing against b', which now starts a new B-block. Obviously, the first task of each block is longer than all the tasks in the preceding block.

2.2 Problem definitions and summary of results

Hereafter, we formally define the addressed problems and summarize our contributions.

From a centralized viewpoint, in bicriteria and multi-agent optimization one is usually interested in Pareto optimal solutions. Since different schedules may produce the same objective functions values, in this paper we consider such schedules as equivalent. Thus, for each pair of objective function values $(\bar{f}^A, \bar{f}^B)$ for which a Pareto optimal schedule exists, we are interested in finding just *one* Pareto optimal schedule σ such that $f^A(\sigma) = \bar{f}^A$ and $f^B(\sigma) = \bar{f}^B$, and not all of them, if several exist. In other words, we regard two Pareto optimal schedules σ', σ'' as *distinct* only if $(f^A(\sigma'), f^B(\sigma')) \neq (f^A(\sigma''), f^B(\sigma''))$. We denote the set of Pareto optimal solution value pairs as $\mathcal{P}$.

The first problem consists in counting the number of elements in $\mathcal{P}$, i.e. its size.

Problem $P_1(f^A, f^B)$ **(size of** $\mathcal{P}$**).** *Given the two task sets* $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, *and the two objective functions* f^A, f^B, *what is the size of the set* $\mathcal{P}$ *of Pareto optimal feasible schedules?*

Concerning the size of $\mathcal{P}$, note that for a given upper bound K on the size of the largest processing time, the range of values for each objective function is bounded by a polynomial in n and K. Thus, $\mathcal{P}$ will always have pseudopolynomial size. By the same argument, the size of $\mathcal{P}$ can only be exponential in the number of input values, if we allow processing times exponential in n. As an answer to problem $P_1(f^A, f^B)$ we will either show that a problem has a strictly polynomial number of Pareto optimal solutions or that there exists an instance (necessarily containing processing times exponential in n) where $|\mathcal{P}|$ is exponential in n.

Problem $P_2(f^A, f^B)$ **(Recognition problem).** *Given the two task sets* $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, *the two objective functions* f^A, f^B, *and two integers* Q_A *and* Q_B, *is there a feasible schedule* σ *such that* $f^A(\sigma) \leq Q_A$ *and* $f^B(\sigma) \leq Q_B$*?*

From the agent B perspective, when the submission sequence σ^A is given, it is clearly interesting to ask for the best response of B which is the subject of problem $P_3(f^A, f^B)$.

Problem $P_3(f^A, f^B)$ **(Offline problem).** *Given the two task sets* $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, *the submission sequence* σ^A, *and the objective function* f^B, *find a submission sequence for* B *that minimizes* $f^B(\sigma^A, \sigma^B)$, *i.e., a sequence* σ^* *such that*

$$\sigma^* = \arg\min_{\sigma^B}\{f^B(\sigma^A, \sigma^B)\}$$

However, if σ^A is not known to B, then agent B might look for a strategy that minimizes its costs in the face of the worst possible (for B) sequence σ^A, i.e., the *minimax* schedule.

Problem $P_4(f^A, f^B)$ **(Minimax problem).** *Given the two task sets* $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$ *and the objective function* f^B, *find a submission sequence* σ^* *for* B *such that*

$$\sigma^* = \arg\min_{\sigma^B}\{\max_{\sigma^A} f^B(\sigma^A, \sigma^B)\}$$

We will see that both problems $P_3(f^A, f^B)$ and $P_4(f^A, f^B)$ are trivially solvable when $f^B = C^B_{\max}$ or $f^B = \sum C^B_j$. When $f^B = \sum w_j C^B_j$, these problems are still polynomially solvable, through a more sophisticated algorithm, presented in Section 4.2. For this case, we also want to evaluate the quality of the schedule B attains, when its tasks are sequenced by applying a certain standard, single-agent heuristic rule H, compared to the optimal schedule for agent B. Letting σ^B_H denote the submission sequence of B obtained by applying H, we analyze the value of the performance ratio

$$r(H) \leq \max_{\sigma^A} \frac{f^B(\sigma^A, \sigma^B_H)}{\min_{\sigma^B}\{f^B(\sigma^A, \sigma^B)\}}. \tag{2}$$

As usual, we call a performance bound $r(H)$ *tight*, if no larger value than $r(H)$ exists which fulfills (2). We will provide bounds for $r(H)$ when H is either the shortest processing time first (SPT) or the weighted shortest processing time (WSPT) rule.

Tables 1, 2 and 3 summarize the results of this paper.

	$P_1(f^A, f^B)$	$P_2(f^A, f^B)$
(f^A, f^B)	size of $\mathcal{P}$	Complexity of Recognition Problem
$(C^A_{\max}, C^B_{\max})$	1	trivial (Section 3.1)
$(C^A_{\max}, \sum_j C^B_j)$	$O(n)$	trivial (Section 3.2)
$(C^A_{\max}, \sum_j w_j C^B_j)$	exponential	$\mathcal{NP}$-complete (Section 3.3)
$(\sum_j C^A_j, \sum_j C^B_j)$	exponential	$\mathcal{NP}$-complete (Section 3.4)

Table 1: Summary of results for the centralized perspective (Section 3).

3 Centralized perspective

In this section, we address the problem of characterizing the set $\mathcal{P}$ of Pareto optimal solutions for various objective functions, i.e., we address problems $P_1(f^A, f^B)$ and $P_2(f^A, f^B)$. Note that

(f^A, f^B)	$P_3(f^A, f^B)$ Offline Problem	$P_4(f^A, f^B)$ Minimax Problem
$(f^A, C^B_{\max})$	$O(n \log n)$ (SPT, Section 4.1)	$O(n \log n)$ (SPT, Section 4.1)
$(f^A, \sum_j C^B_j)$	$O(n \log n)$ (SPT, Section 4.1)	$O(n \log n)$ (SPT, Section 4.1)
$(f^A, \sum_j w_j C^B_j)$	$O(n^2)$ (Section 4.2)	$O(n^2)$ (Section 4.3)

Table 2: Complexity results for the agent perspective (Section 4).

H	$r(H)$
SPT	$r(SPT) = n$
WSPT	$\sqrt{n}/2 \leq r(WSPT) \leq n$

Table 3: Performance ratios for standard sequencing rules when $f^B = \sum w_j C^B_j$ (Section 4.4).

in Sections 3.1 and 3.2 problem $P_2(f^A, f^B)$ can be answered trivially by constructing all Pareto optimal solutions.

3.1 Makespan minimization

First assume that both agents want to minimize their makespan, i.e., consider the pair of objective functions $(C^A_{\max}, C^B_{\max})$. Since the largest task can never win against any opponent's task, the owner of the largest task, either a_n or b_b, will always have a makespan of $\sum_j a_j + \sum_j b_j$. Hence, there is only *one* Pareto optimal solution, precisely the one in which the largest task is played in the first round. As a consequence, the other agent wins the first n rounds (regardless of its submission sequence). So, both problems $P_1(C^A_{\max}, C^B_{\max})$ and $P_2(C^A_{\max}, C^B_{\max})$ are trivial.

3.2 Minimization of makespan and total completion time

Consider now the problems in which agent A wants to minimize its makespan, while agent B has the objective of minimizing the total completion time.

Proposition 1 *For problem $P_1(C^A_{\max}, \sum_j C^B_j)$ there are at most n Pareto optimal schedules.*

Proof. **Case 1.** $a_n > b_n$:
In this case A cannot avoid to reach $C^A_{\max} = \sum_j a_j + \sum_j b_j$ since all B-tasks will be processed before a_n. Hence, there is only *one* Pareto optimal solution arising when A submits a_n in the first round. It consists of the following schedule:

$$\langle b_1, \ldots, b_n, a_n, a_1, \ldots, a_{n-1} \rangle$$

The solution is clearly optimal for B since the B-tasks are scheduled in SPT order. Of course, tasks $a_1, \ldots, a_{n-1}$ can be scheduled in arbitrary order.

Case 2. $a_n \leq b_n$:
Let $t := \min\{j \mid b_j \geq a_n\}$. For any $k = 0, 1, \ldots, t-1$ the following Pareto optimal solution arises if B starts its submission sequence with tasks $b_1, \ldots, b_k, b_t$, while A starts with a_n :

$$\langle b_1, \ldots, b_k, a_n, a_1, \ldots, a_{n-1}, b_t, b_{k+1}, \ldots, b_{t-1}, b_{t+1}, \ldots, b_n \rangle$$

This means that B can split the tasks $b_1, \ldots, b_{t-1}$ into a first part scheduled before the A-tasks and a second part scheduled after the A-tasks (and after b_t), see Figure 1 for an example.

Figure 1: Example of a Pareto optimal solution when $f^A = C^A_{\max}$ and $f^B = \sum_j C^B_j$ in which $b_n = 20 > a_n = 15$.

Clearly, any deviation of B from the partial SPT ordering would only worsen the objectives for both agents as can be shown by a simple exchange argument. Moreover, if A submits any task before a_n, it would still have $C^A_{\max} = \sum_{j=1}^{n} a_j + \sum_{j=1}^{k} b_j$ as before, but B's objective might increase. Altogether, there are exactly $t \leq n$ Pareto optimal schedules. □

Since there are at most n Pareto optimal schedules, problem $P_2(C^A_{\max}, \sum_j C^B_j)$ is trivially solved in polynomial time.

3.3 Minimization of makespan and total weighted completion time

In this section we show that as soon as agent B assigns weights to the completion times the problem changes from easy to difficult. More precisely, we show that when $f^A = C^A_{\max}$ and $f^B = \sum w_j C^B_j$ there are exponentially many Pareto optimal solutions and problem $P_2(C^A_{\max}, \sum w_j C^B_j)$ is $\mathcal{NP}$-complete.

Theorem 2 *There are exponentially many PO solutions for problem $P_1(C^A_{\max}, \sum w_j C^B_j)$.*

Proof. Consider an instance of our problem with the following processing times:

$$a_i = \begin{cases} \varepsilon & i = 1, \ldots, n-1; \\ M & i = n \end{cases} \qquad b_i = w_i = \begin{cases} 2^i & i = 1, \ldots, n-1; \\ M + \varepsilon & i = n \end{cases}$$

where $\varepsilon \ll 1$ and $M \gg 2^n$. Note that for simplicity of presentation we relax the integrality assumption for processing times. However, the instance and its analysis could be easily adapted to integer values by scaling arguments. Let $S \subseteq \{1, \ldots, n-1\}$, $\bar{S} = \{1, \ldots, n-1\} \setminus S$. We build a solution sequence $\sigma(S)$ assuming that agent A submits its tasks in SPT order and agent B submits tasks b_i for all $i \in S$ (in any order) first, then b_n, and eventually tasks b_i for all $i \in \bar{S}$ (in any order). The resulting schedule and the lengths of its parts are illustrated hereafter:

$$\langle \underbrace{a_1, \ldots, a_{n-1}}_{(n-1)\varepsilon} \mid \underbrace{B\text{-tasks in } S}_{\sum_{i \in S} 2^i} \mid \underbrace{\quad a_n \quad}_{M} \mid \underbrace{\quad b_n \quad}_{M+\varepsilon} \mid \underbrace{B\text{-tasks in } \bar{S}}_{\sum_{i \in \bar{S}} 2^i} \rangle.$$

Note that, since for each B-task its processing time equals its weight, once the set S is chosen, the relative order of B-tasks in the blocks preceding and succeeding b_n is irrelevant for B.

We claim that, for all $S \subseteq \{1, \ldots, n-1\}$, $\sigma(S)$ is Pareto-Optimal. In fact, let $w(S) = \sum_{i \in S} 2^i$ be the total weight (equal to the total duration) of the B-tasks corresponding to set S and, similarly, $w(\bar{S})$ that of tasks corresponding to $\bar{S}$, while W denotes the total weight of *all* B-tasks.

For simplicity, but w.l.o.g. we will set $\varepsilon = 0$ in the following. The objective function values for the two agents are:

$$f^A(\sigma(S)) = w(S) + M \tag{3}$$

$$f^B(\sigma(S)) = \sum_{j\in S}\left(w_j \sum_{\substack{i\in S\\ i\le j}} b_i\right) + M(w(S)+2M) + \sum_{j\in\bar{S}}\left(w_j\left(w(S)+2M+\sum_{\substack{i\in\bar{S}\\ i\le j}} b_i\right)\right)$$

$$= 2M^2 + M\underbrace{(w(S)+2w(\bar{S}))}_{=W+w(\bar{S})} + \sum_{j\in S}\left(w_j \sum_{\substack{i\in S\\ i\le j}} b_i\right) + \sum_{j\in\bar{S}}\left(w_j\left(w(S)+\sum_{\substack{i\in\bar{S}\\ i\le j}} b_i\right)\right) \tag{4}$$

If we consider a different subset $S' \neq S$ with $w(S) < w(S')$, then $f^A(\sigma(S)) < f^A(\sigma(S'))$ and clearly $w(\bar{S}) > w(\bar{S}')$. Comparing $f^B(\sigma(S))$ to $f^B(\sigma(S'))$ it suffices to consider terms depending on M, since M can be chosen arbitrarily large and thus obliterates the influence of the sum terms in (4). Hence we get that $f^B(\sigma(S)) - f^B(\sigma(S'))$ is of order $M(w(\bar{S}) - w(\bar{S}')) > 0$ and thus M can be chosen such that $f^B(\sigma(S)) > f^B(\sigma(S'))$.

Since there are exponentially many sets S all with different weights the statement follows. □

The coefficients of the instance used in the proof of Theorem 2 are exponential in n, which might be seen as a drawback. However, as we pointed out in Section 2.2 any constant bound K on the coefficients would immediately imply that the number of PO solutions is polynomially bounded in K.

For the next proof we use the classical, weakly $\mathcal{NP}$-complete knapsack problem:

KNAPSACK:
Instance: a set of *items* $I = \{1,\dots,n\}$, having nonnegative integer *utilities* $\{u_1,\dots,u_n\}$ and *volumes* $\{v_1,\dots,v_n\}$, two integer target values U^T and V^T.
Question: Is there a subset $S \subseteq I$ of items such that

$$\sum_{i\in S} u_i \ge U^T \ ? \tag{5}$$

$$\text{and} \qquad \sum_{i\in S} v_i \le V^T \tag{6}$$

Theorem 3 *Problem $P_2(C^A_{\max}, \sum w_j C^B_j)$ is $\mathcal{NP}$-complete.*

Proof. We reduce KNAPSACK to $P_2(C^A_{\max}, \sum w_j C^B_j)$. Denote by $U = \sum_{i=1}^n u_i$ the total utility and by $V = \sum_{i=1}^n v_i$ the total volume of all items in KNAPSACK.

Given an instance of KNAPSACK, define the following instance of $P_2(C^A_{\max}, \sum w_j C^B_j)$ with $n+1$ tasks for each agent, where $M \gg UV$ is a large enough integer and $\varepsilon \ll 1$ (again we allow processing times to deviate from the integers w.l.o.g.).

$$a_i = \begin{cases} \varepsilon & i = 1,\dots,n;\\ M & i = n+1 \end{cases} \qquad b_i = \begin{cases} v_i & i = 1,\dots,n;\\ M+\varepsilon & i = n+1 \end{cases}$$

$$w_i = \begin{cases} u_i & i = 1, \ldots, n; \\ \varepsilon & i = n+1 \end{cases} \qquad \begin{aligned} Q_A &= V^T + M \\ Q_B &= 2M(U - U^T) + UV. \end{aligned}$$

For simplicity, we will set $\varepsilon = 0$ and assume w.l.o.g. that A submits its tasks in SPT order. We first observe that any feasible solution has the following structure:

$$\langle \underbrace{a_1, \ldots, a_n}_{n\varepsilon} \mid \text{first block of } B\text{-tasks} \mid \underbrace{\qquad a_{n+1} \qquad}_{M} \mid \underbrace{\qquad b_{n+1} \qquad}_{M+\varepsilon} \mid \text{second block of } B\text{-tasks} \rangle.$$

Hence, a feasible solution is completely defined by the position of b_{n+1} in the submission sequence of B. We call *first block* and *second block* the two sets of B-tasks scheduled before and, respectively, after the two long tasks a_{n+1} and b_{n+1}.

We want to show that $P_2(C^A_{\max}, \sum w_j C^B_j)$ has a solution if and only if KNAPSACK has a solution. Given a yes-solution σ to $P_2(C^A_{\max}, \sum w_j C^B_j)$, take as set S in the KNAPSACK instance the set of items corresponding to the tasks scheduled in the first block in σ. Hence, the total length of the B-tasks in the first block in σ equals the total volume of these items, and since $C^A_{\max}(\sigma) \leq Q_A = V^T + M$, one has that (6) is satisfied. If we let U_S denote the total utility of the items corresponding to the tasks in the first block in σ, the total weight of the tasks in the second block is $U - U_S$. Hence, the contributions to f^B of the first, resp. second block, do not exceed VU_S, resp. $(2M + V)(U - U_S)$, which yields

$$f^B(\sigma) \leq 2M(U - U_S) + UV \tag{7}$$

Now we can show that $U_S \geq U^T$. Assume by contradiction that $U_S \leq U^T - 1$: Then the contribution of the second block would be at least $2M(U - (U^T - 1)) = 2M(U - U^T) + 2M$, and this would exceed Q_B, since $M > UV$. Hence, $U_S \geq U^T$ and (5) is verified.

Viceversa, given the set S in a yes-instance of KNAPSACK, we build a schedule σ as indicated in (3.3), identifying the first block of B-tasks with the tasks in S. Clearly, (6) implies that $C^A_{\max} \leq Q_A$. Similarly, from (5), the total weight U_S of the tasks in the first block (i.e., the total utility of the items in S) is at least U^T and hence, from (7), one has $f^B(\sigma) \leq Q_B$. □

The above reduction from KNAPSACK only shows that $P_2(C^A_{\max}, \sum w_j C^B_j)$ is weakly $\mathcal{NP}$-complete. In the following we develop a pseudopolynomial dynamic programming algorithm for the associated optimization problem $\tilde{P}$ which consists in finding a feasible solution minimizing $\sum w^B_j C^B_j$ subject to $C^A_{\max} \leq Q_A$. This shows that problem $P_2(C^A_{\max}, \sum w_j C^B_j)$ is not strongly $\mathcal{NP}$-complete.

If $a_n > b_n$, then task a_n will always lose against any task of agent B, and as a consequence it will be processed only after all B-tasks. As a consequence, in any schedule $C^A_{\max} = \sum_j a_j + \sum_j b_j$. Hence, $\tilde{P}$ has a feasible solution if and only if $Q_A \geq \sum_j a_j + \sum_j b_j$, and if so, the optimal schedule is obtained when A submits a_n first, and the other tasks in any order. This allows agent B to schedule all of its tasks in an optimal way, namely in WSPT order before all A-tasks.

From now on we consider the case $a_n \leq b_n$. Our algorithm is based on the following observation. If A completes all its tasks before time Q_A, the most favorable situation for B occurs when all A-tasks are consecutively performed to complete as close as possible to (but not later than) time Q_A. This can be reached if agent A submits a_n as first task. As long as B schedules tasks *shorter than* a_n, these are consecutively performed until the submission of any task $b_k \geq a_n$. At this point, all A-tasks can be consecutively performed, after a_n, in arbitrary order. Thereafter,

the remaining B-tasks will be scheduled. Hence, the optimal solution of $\tilde{P}$ has the following structure:

$$\langle\ B^{(1)}\ A\ \ \bar{b}\ \ B^{(2)}\ \rangle$$

All B-tasks are grouped into two blocks $B^{(1)}$ and $\{\bar{b}\} \cup B^{(2)}$ respectively preceding and following the single block of A-tasks. A task $\bar{b} \geq a_n$ is scheduled at the beginning of the second B-block. B-tasks in $B^{(1)}$ and $B^{(2)}$ are scheduled in SPT order. All B-tasks with length at least a_n are in the second block.

Hence, once the task $\bar{b}$ is fixed, the problem consists in deciding how to partition the remaining B-tasks into the two blocks $B^{(1)}$ and $B^{(2)}$, given that the A-block must complete within Q_A and that $B^{(2)}$ contains all B-tasks larger than a_n. The whole set of A-tasks can be regarded as a single block of length $t(A) := \sum_j a_j$ (although of course it will not behave as a single task from the viewpoint of the scheduling mechanism). In what follows, for simplicity we temporarily remove $\bar{b}$ from B (leaving it with $n-1$ tasks) and suppose that the remaining tasks in B are numbered according to WSPT.

We now define the dynamic programming array $F(k, t_1, t_2)$ to represent the optimal value of a subproblem restricted to the first k B-tasks plus A and $\bar{b}$, in which the machine *continuously* works between 0 and t_1, it is idle from t_1 to t_2, and block A starts at time t_2. Note that this means that each B-task is either processed between time 0 and t_1 (i.e., in $B^{(1)}$), or after $t_2 + t(A) + \bar{b}$ (i.e., in $B^{(2)}$). In particular, for each $k = 1, \ldots, n-1$, a decision on task b_k based on the values of $F(k-1, \cdot, \cdot)$ must be taken.

Because of the WSPT ordering and based on the definition of F there are two possible cases in an optimal solution to the restricted problem:

Case (i): task b_k completes at time t_1.
Note that if $b_k > a_n$, then it must necessarily belong to the second block, and one has:

$$F(k, t_1, t_2) = F_{(i)}(k, t_1, t_2) \doteq \begin{cases} F(k-1, t_1 - b_k, t_2) + w_k t_1 & \text{if } b_k < a_n \\ +\infty & \text{otherwise.} \end{cases} \tag{8}$$

Case (ii): task b_k is the last task in the schedule.
In this case b_k completes at time $(t_2 - t_1) + t(A) + \bar{b} + \sum_{j=1}^{k} b_j$, and therefore:

$$F(k, t_1, t_2) = F_{(ii)}(k, t_1, t_2) \doteq F(k-1, t_1, t_2) + w_k \left((t_2 - t_1) + t(A) + \bar{b} + \sum_{j=1}^{k} b_j \right). \tag{9}$$

As a consequence, from (8) and (9), one has:

$$F(k, t_1, t_2) = \min \left\{ F_{(i)}(k, t_1, t_2),\ F_{(ii)}(k, t_1, t_2) \right\}. \tag{10}$$

We have the following boundary conditions for recursion (10):

- $F(0, 0, t_2) = \bar{w}(t_2 + t(A) + \bar{b})$ for any $t_2 > 0$
- $F(k, t_1, t_2) = +\infty$ for any $t_1 < 0$ or $t_1 > t_2$
- $F(0, t_1, t_2) = +\infty$ for any $t_1 > 0$, since the machine must be continuously working between 0 and t_1
- $F(k, t_1, t_2) = +\infty$ for any $t_2 > Q_A - t(A)$, since in this case $C^A_{\max}$ would exceed Q_A.

Clearly, the optimal schedule for a fixed task $\bar{b}$ must be searched among solutions corresponding to values of $F(k, t_1, t_2)$ having $t_1 = t_2$, since there should be no idle times. Its value is given by

$$\min_{0 \le t \le Q_A - t(A)} F(n-1, t, t).$$

The whole procedure has to be repeated for each B-tasks as a candidate for $\bar{b}$.

Let us now turn to complexity issues. Once the B-tasks are ordered, the computation of each value $F(k, t_1, t_2)$ can be done in constant time. Considering that for all t_1, t_2 there is $t_1 \le t_2 \le Q_A - t(A)$, and that the whole procedure has to be repeated for all n possible choices of $\bar{b}$, we have the following result.

Theorem 4 *Problem $\tilde{P}$ and thus $P_2(C^A_{\max}, \sum w_j C^B_j)$ can be solved in $O(n^2 Q_A^2)$ time.*

Unfortunately, choosing $\bar{b}$ is not an obvious task and we currently see no way of avoiding the n iterations over all B-tasks. Clearly, a natural choice would be to select among all B-task larger than a_n the first one in WSPT order. However, the following example shows that this is not always optimal.

Example 5 *Consider a scenario where $B^{(2)} \cup \bar{b}$ consists of 3 tasks with lengths $b_1 = 1$, $b_2 = 10$, $b_3 = 2$, and weights $w_1 = 10$, $w_2 = 11$ and $w_3 = 2$. Suppose $a_n = 1 + \varepsilon$ so that the only possible choices for $\bar{b}$ are b_2 and b_3. If we choose $\bar{b} = b_2$ according to WSPT ($w_2/b_2 = 1.1 > w_3/b_3 = 1$), the resulting schedule*

$$\sigma = \langle B^{(1)} \;\; A \;\; b_2 \;\; b_1 \;\; b_3 \rangle$$

– neglecting the contributions of $B^{(1)}$ and the A block – would cost $f(\sigma) \approx 11 \times 10 + 10 \times 11 + 2 \times 13 = 246$. If, on the other hand, $\bar{b} = b_3$ is chosen, then the schedule

$$\sigma' = \langle B^{(1)} \;\; A \;\; b_3 \;\; b_1 \;\; b_2 \rangle$$

would cost $f(\sigma') \approx 2 \times 2 + 10 \times 3 + 11 \times 13 = 177$ and, obviously, σ' is better than σ.

3.4 Minimization of total completion times

As in the previous section, we show that there are exponentially many Pareto optimal solutions when both agents want to minimize their total completion times and that $P_2(\sum C^A_j, \sum C^B_j)$ is $\mathcal{NP}$-complete. However, the proofs are far more complicated due to the more complex objective function of agent A.

Regarding the characterization of Pareto optimal solutions, it is easy to note that, when both agents submit their tasks in SPT order, the outcome is a Pareto optimal solution. Note that because of the selection mechanism, this results in an overall SPT schedule, thereafter denoted as σ_{SPT}, which is obviously the schedule minimizing $\sum_j C^A_j + \sum_j C^B_j$.

Let us first address problem $P_1(\sum C^A_j, \sum C^B_j)$. Consider the following instance denoted as POEXP. Let M be a large enough number. Each agent owns n tasks, defined as follows:

POEXP:

$a_1 = 1 + M$	$b_1 = 2 + M$	$b_2 = 3 + M$	$a_2 = 4 + M$
$a_3 = 1 + M^2$	$b_3 = 2 + M^2$	$b_4 = 3 + M^2$	$a_4 = 4 + M^2$
$\dots$	$\dots$	$\dots$	$\dots$
$a_{2i-1} = 1 + M^i$	$b_{2i-1} = 2 + M^i$	$b_{2i} = 3 + M^i$	$a_{2i} = 4 + M^i$
$\dots$	$\dots$	$\dots$	$\dots$
$a_{n-1} = 1 + M^{n/2}$	$b_{n-1} = 2 + M^{n/2}$	$b_n = 3 + M^{n/2}$	$a_n = 4 + M^{n/2}$.

We call each subset of 4 tasks $a_{2i-1}, a_{2i}, b_{2i-1}, b_{2i}$ a *quartet* Q_i. We call *quasi-SPT* a schedule in which the tasks of Q_i are all scheduled before the tasks of Q_{i+1}, for all $i = 1, \ldots, n/2 - 1$.

Proposition 6 *There exist* $2^{n/2}$ *quasi-SPT schedules for instance* POEXP.

Proof. In any quasi-SPT schedule, for each quartet Q_i there are exactly two nondominated feasible *modes* of being scheduled, namely $\sigma_{SPT}(Q_i) = \langle a_{2i-1}, b_{2i-1}, b_{2i}, a_{2i} \rangle$ (which we denote as *SPT-mode*, since it is obtained when A submits its tasks in SPT order) and $\sigma_{LPT}(Q_i) = \langle b_{2i-1}, b_{2i}, a_{2i}, a_{2i-1} \rangle$ (denoted as *LPT-mode*, since it is obtained when A submits its tasks in LPT order). See Figure 2 for a pictorial representation of the two modes. Since each quartet can be scheduled independently of the others, the thesis follows. □

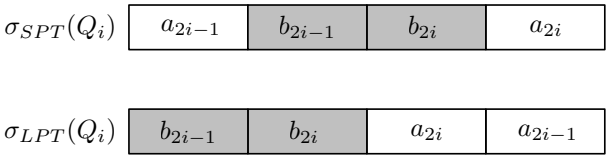

Figure 2: Structure of a quartet Q_i scheduled in SPT-mode and LPT-mode.

Proposition 7 *No quasi-SPT schedule is dominated by a non-quasi-SPT schedule for instance* POEXP.

Proof. We compare the total cost of schedules, i.e. $f^A + f^B$. Clearly, the optimal total cost f^* is obtained by scheduling all tasks by SPT. Now we first bound the difference between f^* and the total cost of the worst possible quasi-SPT schedule, call it $\hat{\sigma}$, which is obtained by choosing LPT-mode $\sigma_{LPT}(Q_i)$ for all i. A trivial calculation yields $f^A(\hat{\sigma}) + f^B(\hat{\sigma}) - f^* \leq 3n$.

On the other hand considering the combined sequence of tasks of both agents any non-quasi-SPT schedule must have at least one position where a task from some quartet Q_{i+p} is followed by a task from a quartet Q_i with $p \geq 1$. Calculating the difference of this configuration from an optimal SPT schedule with respect to the total completion times we get in the best case for the non-quasi-SPT schedule, i.e. $p = 1$, a deviation of $1 + M^{i+1} - 4 - M^i \geq M(M-1) - 3$ which is strictly larger than $3n$ for M sufficiently large. Hence, any non-quasi-SPT schedule has a total cost larger than that of a quasi-SPT schedule and the statement follows. □

Proposition 8 *All quasi-SPT schedules are Pareto optimal for instance* POEXP.

Proof. Given a quasi-SPT schedule σ, it is easy to show that, letting $X_i = \sum_{k=1}^{i}(a_{2k-1} + a_{2k} + b_{2k-1} + b_{2k})$, $i = 1, \ldots, n/2$, for each quartet scheduled in SPT-mode one has

$$f^A(\sigma_{SPT}(Q_i)) = 2X_{i-1} + 11 + 5M^i \text{ and } f^B(\sigma_1(Q_i)) = 2X_{i-1} + 9 + 5M^i \tag{11}$$

while, for each quartet scheduled in LPT-mode

$$f^A(\sigma_{LPT}(Q_i)) = 2X_{i-1} + 19 + 7M^i \text{ and } f^B(\sigma_2(Q_i)) = 2X_{i-1} + 7 + 3M^i. \tag{12}$$

Now let $\mathcal{Q}^{SPT}(\sigma)$ and $\mathcal{Q}^{LPT}(\sigma)$ denote the set of quartets scheduled in SPT and LPT modes in σ respectively. From (11) and (12) one can then compute

$$f^A(\sigma) = 2\sum_{i=1}^{n/2-1} X_i + 11|\mathcal{Q}^{SPT}(\sigma)| + 19|\mathcal{Q}^{LPT}(\sigma)| + 5\sum_{i\in\mathcal{Q}^{SPT}(\sigma)} M^i + 7\sum_{i\in\mathcal{Q}^{LPT}(\sigma)} M^i \qquad (13)$$

$$f^B(\sigma) = 2\sum_{i=1}^{n/2-1} X_i + 9|\mathcal{Q}^{SPT}(\sigma)| + 7|\mathcal{Q}^{LPT}(\sigma)| + 5\sum_{i\in\mathcal{Q}^{SPT}(\sigma)} M^i + 3\sum_{i\in\mathcal{Q}^{LPT}(\sigma)} M^i. \qquad (14)$$

In what follows, consider two *distinct* quasi-SPT schedules σ, σ'. Clearly $\mathcal{Q}^{SPT}(\sigma) \neq \mathcal{Q}^{SPT}(\sigma')$ and $\mathcal{Q}^{LPT}(\sigma) \neq \mathcal{Q}^{LPT}(\sigma')$. Let k be the *largest* index of a quartet such that Q_k is processed in different modes in σ and σ', and suppose that $Q_k \in \mathcal{Q}^{SPT}(\sigma)$ and $Q_k \in \mathcal{Q}^{LPT}(\sigma')$. From (13) and (14), even if $Q_i \in \mathcal{Q}^{LPT}(\sigma)$ and $Q_i \in \mathcal{Q}^{SPT}(\sigma')$ for all $i = 1, \ldots, k-1$, switching from σ to σ' implies that the cost for B decreases by at least $2(M^k - M^{k-1} - M^{k-2} - \cdots - M)$ and the cost of A increases by at most the same amount (disregarding terms of smaller order). For large enough M, this means that $f^B(\sigma') < f^B(\sigma)$ and $f^A(\sigma') > f^A(\sigma)$. On the contrary, if $Q_k \in \mathcal{Q}^{LPT}(\sigma)$ and $Q_k \in \mathcal{Q}^{SPT}(\sigma')$, then $f^B(\sigma') > f^B(\sigma)$ and $f^A(\sigma') < f^A(\sigma)$. In conclusion, if the cost to one agent decreases, the cost to the other agent necessarily increases. So, no two quasi-SPT schedules dominate each other. From Proposition 7 it follows that no quasi-SPT schedule is dominated by a non-quasi-SPT schedule and the thesis follows. □

Note that Proposition 8 does *not* imply that all Pareto optimal schedules are quasi-SPT schedules for instance POexp. Summarizing, Propositions 6 and 8 imply the following result.

Theorem 9 *There can be exponentially many Pareto optimal solutions for* $P_1(\sum C_j^A, \sum C_j^B)$.

In the following, we will show the $\mathcal{NP}$-completeness of $P_2(\sum C_j^A, \sum C_j^B)$ by reducing Even-Odd Partition [11] to the latter problem.

> Even-Odd Partition:
> **Instance**: an even integer $n = 2m$, nonnegative integers $S = \{s_1, \ldots, s_n\}$, with $s_j < s_{j+1}$ $j = 1, 2, \ldots, n-1$ and $\sum_{j=1}^n s_j = U$.
> **Question**: Is there a subset $S' \subseteq S$ such that exactly one between s_{2i-1} and s_{2i} belongs to S' for each i, and
>
> $$\sum_{j\in S'} s_j = \sum_{j\in S\setminus S'} s_j = U/2 \ ?$$

Theorem 10 *Problem* $P_2(\sum C_j^A, \sum C_j^B)$ *is* $\mathcal{NP}$*-complete.*

Proof. Problem $P_2(\sum C_j^A, \sum C_j^B)$ trivially belongs to $\mathcal{NP}$.

Given an instance of Even-Odd Partition we assume w.l.o.g. that the items are multiplied by a suitable positive constant so that $s_i \gg 1$. We define the corresponding instance of our problem $P_2(\sum C_j^A, \sum C_j^B)$ as follows:

Extending the construction of instance POexp by adding the value s_i to each quartet Q_i and duplicating all quartets we get an instance (POcomp) where each agent has $2n = 4m$ tasks. Here, an odd quartet Q_{2h-1} is the subset of 4 tasks $a_{2h-1}, b_{2h-1}, b_{2h}, a_{2h}$ and an even quartet Q_{2h}

Table 4: Instance POcomp

octet	quartet	tasks			
O_1	Q_1:	$a_1 = 1 + s_1 + M^1$	$b_1 = 2 + s_1 + M^1$	$b_2 = 3 + s_1 + M^1$	$a_2 = 4 + s_1 + M^1$
	Q_2:	$\bar{a}_1 = 1 + s_2 + M^1$	$\bar{b}_1 = 2 + s_2 + M^1$	$\bar{b}_2 = 3 + s_2 + M^1$	$\bar{a}_2 = 4 + s_2 + M^1$
...	...	...	...	...	...
O_h	Q_{2h-1}:	$a_{2h-1} = 1 + s_{2h-1} + M^h$	$b_{2h-1} = 2 + s_{2h-1} + M^h$	$b_{2h} = 3 + s_{2h-1} + M^h$	$a_{2h} = 4 + s_{2h-1} + M^h$
	Q_{2h}:	$\bar{a}_{2h-1} = 1 + s_{2h} + M^h$	$\bar{b}_{2h-1} = 2 + s_{2h} + M^h$	$\bar{b}_{2h} = 3 + s_{2h} + M^h$	$\bar{a}_{2h} = 4 + s_{2h} + M^h$
...	...	...	...	...	...
O_m	Q_{n-1}:	$a_{n-1} = 1 + s_{n-1} + M^{n/2}$	$b_{n-1} = 2 + s_{n-1} + M^{n/2}$	$b_n = 3 + s_{n-1} + M^{n/2}$	$a_n = 4 + s_{n-1} + M^{n/2}$
	Q_n:	$\bar{a}_{n-1} = 1 + s_n + M^{n/2}$	$\bar{b}_{n-1} = 2 + s_n + M^{n/2}$	$\bar{b}_n = 3 + s_n + M^{n/2}$	$\bar{a}_n = 4 + s_n + M^{n/2}$

is the subset of 4 tasks $\bar{a}_{2h-1}, \bar{b}_{2h-1}, \bar{b}_{2h}, \bar{a}_{2h}$, for $h = 1, \ldots, n/2$. The subset of tasks consisting of quartets Q_{2h-1} and Q_{2h} is called *octet* O_h. Table 4 summarizes the structure and tasks lengths of instance POcomp.

Similarly to the schedules introduced in Section 3.4, we consider solution schedules for POcomp in which the tasks of Q_i are all scheduled before the tasks of Q_{i+1}, for all $i = 1, \ldots, n/2 - 1$. In any such schedule, for each quartet Q_i there are exactly two nondominated feasible *modes* of being scheduled, namely

1. *SPT mode*: $\sigma_{SPT}(Q_i) = \langle a_i, \quad b_i, \quad b_{i+1}, \quad a_{i+1} \rangle$ or $\sigma_{SPT}(Q_i) = \langle \bar{a}_{i-1}, \quad \bar{b}_{i-1}, \quad \bar{b}_i, \quad \bar{a}_i \rangle$ obtained when A submits its tasks in Q_i in SPT order or
2. *LPT mode*: $\sigma_{LPT}(Q_i) = \langle b_i, \quad b_{i+1}, \quad a_{i+1}, \quad a_i \rangle$ or $\sigma_{LPT}(Q_i) = \langle \bar{b}_{i-1}, \quad \bar{b}_i, \quad \bar{a}_i, \quad \bar{a}_{i-1} \rangle$ obtained when A submits its Q_i tasks in LPT order.

Consider the schedule σ_{SPT} obtained by scheduling all tasks in SPT order (note that this schedule is organized in quartets which are all in SPT-mode). We indicate by $f^A_{SPT} = \sum_j C^A_j(\sigma_{SPT})$ and $f^B_{SPT} = \sum_j C^B_j(\sigma_{SPT})$ the values of the objective functions of the two agents for schedule σ_{SPT}. Now we ask whether there is a schedule σ such that

$$f^A(\sigma) = \sum_j C^A_j(\sigma) \leq P^A = f^A_{SPT} + 4n + U + 2\sum_{i=1}^{n/2} M^i \tag{15}$$

$$f^B(\sigma) = \sum_j C^B_j(\sigma) \leq P^B = f^B_{SPT} - n - U - 2\sum_{i=1}^{n/2} M^i \tag{16}$$

Now we will prove the following statement: The given instance of Even-Odd Partition is a yes-instance if and only if POcomp with (15) and (16) is a yes-instance.

($\Longrightarrow$) Given the solution S' to Even-Odd Partition, we can derive a solution to POcomp by building a quasi-SPT-plus schedule in which the items of set S' correspond to the tasks of the quartets in LPT mode, i.e., for each item s_i in S', the quartet Q_i is scheduled in LPT mode (conversely, for each item s_i not in S', the quartet Q_i is scheduled in SPT mode). If we compare the resulting schedule with σ_{SPT} we observe that, if a quartet Q_i is scheduled differently from σ_{SPT}, i.e. in LPT mode, A's objective increases by $b_i + b_{i+1} + a_{i+1} - a_i$ or $\bar{b}_{i-1} + \bar{b}_i + \bar{a}_i - \bar{a}_{i-1}$ which are both equal to $8 + 2s_i + 2M^{\lceil i/2 \rceil}$, while B's objective decreases by $2a_i$ or $2\bar{a}_{i-1}$, i.e. by $2 + 2s_i + 2M^{\lceil i/2 \rceil}$. Recalling that $|S'| = n/2$, by summing up on all n quartets, A's objective,

with respect to σ_{SPT}, is increased by

$$\sum_{i \in S'} \left(8 + 2s_i + 2M^{\lceil i/2 \rceil}\right) = 4n + U + 2\sum_{i=1}^{n/2} M^i.$$

Analogously, B's objective, with respect to σ_{SPT}, decreases of

$$\sum_{i \in S'} \left(2 + 2s_i + 2M^{\lceil i/2 \rceil}\right) = n + U + 2\sum_{i=1}^{n/2} M^i.$$

Hence, given a solution S' to Even-Odd Partition we can build in polynomial time a schedule σ satisfying conditions (15) and (16).

($\Longleftarrow$) Given a feasible schedule σ for POcomp fulfilling (15) and (16), we have to show that a solution to Even-Odd Partition exists.

Here, a schedule is called *quasi-SPT-plus* if, (1) the tasks of quartet Q_i precede those of Q_{i+1} for all $i = 1, \ldots, n-1$ and (2) in each octect O_h its quartets Q_{2h-1} and Q_{2h} are scheduled one in SPT mode and the other in LPT mode for all $h = 1, \ldots, n/2$. The proof proceeds by showing that if our instance is a yes-instance, then σ must be quasi-SPT-plus. In particular, we prove that:

Fact 1. The last eight tasks of schedule σ must be those of octet $O_{n/2}$, i.e. tasks with processing times $1 + s_{n-1} + M^{n/2}$, $2 + s_{n-1} + M^{n/2}$, $3 + s_{n-1} + M^{n/2}$, $4 + s_{n-1} + M^{n/2}$, $1 + s_n + M^{n/2}$, $2 + s_n + M^{n/2}$, $3 + s_n + M^{n/2}$, $4 + s_n + M^{n/2}$;

Fact 2. The tasks of the last octet $O_{n/2}$ are arranged in a quasi-SPT-plus schedule.

This argument can be repeated for the second last octet and so on down to the first one, proving that any solution of a yes-instance of our problem must be a quasi-SPT-plus schedule. To this purpose the M's are chosen so that the durations of the tasks of octet O_i are suitably larger than those of any task in octet O_k with $k < i$.

Proof of Fact 1. Given any schedule σ, let $f(\sigma) = f^A(\sigma) + f^B(\sigma)$ indicate its total cost.

Any schedule σ satisfying Equations (15) and (16) must have $f(\sigma) - f(\sigma_{SPT}) \leq 3n$. It is easy to observe that any schedule σ such that the last eight tasks are not those of the last octet, has $f(\sigma) - f(\sigma_{SPT}) \geq M^{n/2} - M^{n/2-1} >> 3n$. In fact, disregarding whether tasks belong to agent A or B, if a task whose processing time is order of $M^{n/2}$ precedes another one whose processing time is order of $M^{n/2-1}$ the total cost of the schedule increases by at least (order of) $M^{n/2} - M^{n/2-1}$ with respect to $f(\sigma_{SPT})$.

Proof of Fact 2. In order to obtain a schedule satisfying (15) and (16), we observe that the contribution of the tasks in octet $O_{n/2}$ (neglecting the constant and the s_i values) must be order of $20M^{n/2} + 4T_{n/2}$ for agent A and $16M^{n/2} + 4T_{n/2}$ for agent B, where T_i is a constant indicating the starting time of octet O_i in σ_{SPT}. This means that, with respect to σ_{SPT} (in which A and B tasks contribute order of $18M^{n/2} + 4T_{n/2}$ each), in the last octet A's objective is increased by order of $2M^{n/2}$ and B's objective is decreased by the same quantity. This argument can be applied from the last octet down to the first one, i.e. tasks of octet O_i are sequenced so that their contribution to A's objective is order of $20M^i$, while B's objective is order of $16M^i$.

To consider all such schedules, we note that O_i consists of 8 tasks all with completion times order of $h(M^i + T_i)$ for $h = 1, \ldots, 8$. It easy to see that only subsets $S \subseteq \{1, \ldots, 8\}$ having cardinality 4 and summing up to 20 correspond to possible position assignments for

A-tasks. Here is a list of all such subsets: $S_1 = \{1, 4, 7, 8\}$, $S_2 = \{1, 5, 6, 8\}$, $S_3 = \{2, 3, 7, 8\}$, $S_4 = \{2, 4, 6, 8\}$, $S_5 = \{2, 5, 6, 7\}$, $S_6 = \{3, 4, 5, 8\}$, $S_7 = \{3, 4, 6, 7\}$. Clearly, the position assignments for B is given by the complement of S_i.

Observe that subsets S_5 and S_7 do not correspond to any feasible schedule since agent A, owning the largest task, will always occupy the last position of the octet, i.e. position 8. Moreover, also S_2 and S_4 do not correspond to feasible schedules since they cannot be obtained with any submission list of A and B.

The remaining subsets correspond to feasible schedules depicted in Figure 3, namely S_1 corresponds to schedules σ_1 and σ_2, S_3 corresponds to schedule σ_3, and finally S_6 corresponds to schedules σ_4 and σ_5. Clearly, these five schedules are not the only feasible ones, however they are the only non dominated ones. In fact, by swapping some tasks of the same agent it is possible to obtain different but still feasible schedules (in particular b_{2i} can be swapped with b_{2i-1}, $\bar{b}_{2i}$ with $\bar{b}_{2i-1}$, and a_{2i} with $\bar{a}_{2i-1}$). Note that σ_1 and σ_4 are the desired quasi-SPT-plus schedules.

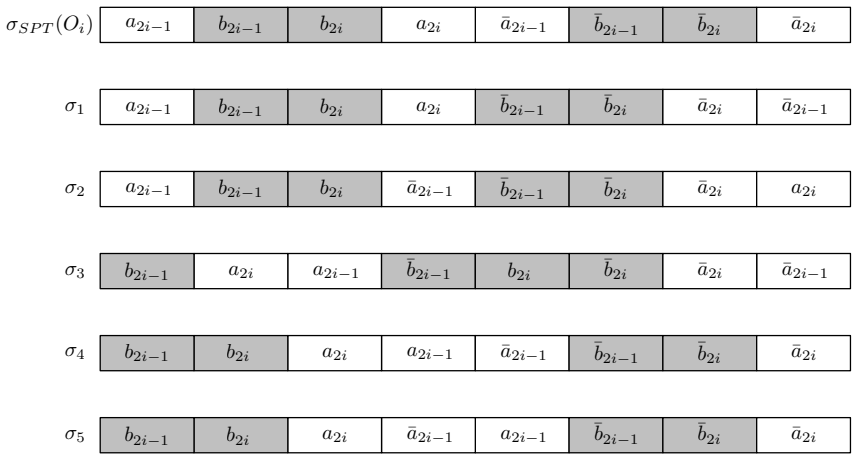

Figure 3: Pictorial representation of scheduling modes of octet O_i in the proof of Theorem 10.

Hereafter we compute for each of these five schedules the contributions of octet O_i to the objective functions of both agents, $f_i^A(\sigma_j)$ and $f_i^B(\sigma_j)$, and report the differences to the corresponding values of σ_{SPT}, for $j = 1, \ldots 5$.

schedule σ of octet O_i	agent X	$f_i^X(\sigma)$	$\Delta_i^X(\sigma) = f_i^X(\sigma) - f_i^X(\sigma_{SPT})$
σ_{SPT}	A	$4T_i + 18M^i + 13s_{2i-1} + 5s_{2i} + 42$	0
	B	$4T_i + 18M^i + 13s_{2i-1} + 5s_{2i} + 38$	0
σ_1	A	$4T_i + 20M^i + 13s_{2i-1} + 7s_{2i} + 50$	$2M^i + 2s_{2i} + 8$
	B	$4T_i + 16M^i + 13s_{2i-1} + 3s_{2i} + 36$	$-2M^i - 2s_{2i} - 2$
σ_2	A	$4T_i + 20M^i + 17s_{2i-1} + 3s_{2i} + 44$	$2M^i + 4s_{2i} - 2s_{2i-1} + 2$
	B	$4T_i + 16M^i + 13s_{2i-1} + 3s_{2i} + 30$	$-2M^i - 2s_{2i} - 8$
σ_3	A	$4T_i + 20M^i + 13s_{2i-1} + 7s_{2i} + 52$	$2M^i + 2s_{2i} + 10$
	B	$4T_i + 16M^i + 12s_{2i-1} + 4s_{2i} + 38$	$-2M^i - s_{2i} - s_{2i-1}$
σ_4	A	$4T_i + 20M^i + 15s_{2i-1} + 5s_{2i} + 50$	$2M^i + 2s_{2i-1} + 8$
	B	$4T_i + 16M^i + 11s_{2i-1} + 5s_{2i} + 36$	$-2M^i - 2s_{2i-1} - 2$
σ_5	A	$4T_i + 20M^i + 14s_{2i-1} + 6s_{2i} + 50$	$2M^i + s_{2i} + s_{2i-1} + 8$
	B	$4T_i + 16M^i + 11s_{2i-1} + 5s_{2i} + 36$	$-2M^i - 2s_{2i-1} - 2$

Consider now a schedule σ satisfying (15) and (16). Assume each octet is scheduled in one of the five modes above. Let F_j indicate the set of octets scheduled according to mode σ_j, $j = 1, \ldots, 5$. Neglecting the higher order terms M^i, Equations (15) and (16) can be rewritten as follows:

$$\begin{aligned} f^A(\sigma) - f^A_{SPT} = &\sum_{i\in F_1}(2s_{2i}+8) + \sum_{i\in F_2}(4s_{2i} - 2s_{2i-1} + 2) + \sum_{i\in F_3}(2s_{2i}+10) + \\ &+ \sum_{i\in F_4}(2s_{2i-1}+8) + \sum_{i\in F_5}(s_{2i} + s_{2i-1} + 8) \leq 4n + U \end{aligned} \tag{17}$$

$$\begin{aligned} f^B(\sigma) - f^B_{SPT} = &- \sum_{i\in F_1}(2s_{2i-1}+2) - \sum_{i\in F_2}(2s_{2i}+8) - \sum_{i\in F_3}(s_{2i} + s_{2i-1}) + \\ &- \sum_{i\in F_4}(2s_{2i-1}+2) - \sum_{i\in F_5}(2s_{2i}+2) \leq -n - U \end{aligned} \tag{18}$$

Summing up inequalities (17) and (18), we obtain

$$6|F_1| + \sum_{i\in F_2} 2(s_{2i} - s_{2i-1} - 3) + \sum_{i\in F_3}(s_{2i} - s_{2i-1} + 10) + 6|F_4| + \sum_{i\in F_5}(s_{2i} - s_{2i-1} + 6) \leq 3n \tag{19}$$

Since $s_{2i} - s_{2i-1} >> 1$, all terms in the summations of the left-hand-side of (19) are greater than 6 which yields a contradiction unless $F_2 = F_3 = F_5 = \emptyset$, i.e. only schedules σ_1 and σ_4 can occur.

At this point, we have proved that any solution of a yes-instance of our problem must be structured into quartets $\langle Q_1, Q_2, \ldots Q_n \rangle$ and, for each octet one quartet is in LPT mode while the other is in SPT mode, i.e. it must be a quasi-SPT-plus schedule.

Hence, given a yes-instance and a solution schedule σ, we can build a solution S' to Even-Odd Partition in which S' contains all items s_i such that the quartet Q_i is scheduled in LPT mode in σ.

In fact, σ satisfies (15) and (16), then A's objective, with respect to σ_{SPT} is increased by at most $4n + U + 2\sum_{i=1}^{n/2} M^i$, and B's objective, with respect to σ_{SPT} is decreased by

$n + U + 2\sum_{i=1}^{n/2} M^i$. Moreover, since for each quartet in LPT mode, A's objective increases by $8 + 2s_i + 2M^{\lceil i/2 \rceil}$ and since for each octet there is exactly one quartet in LPT mode, we have that Equation (15) becomes

$$\sum_{i \in S'} \left(8 + 2s_i + 2M^{\lceil i/2 \rceil}\right) = 4n + \sum_{i \in S'} 2s_i + 2\sum_{i=1}^{n/2} M^i \leq 4n + U + 2\sum_{i=1}^{n/2} M^i.$$

Analogously, since for each quartet in LPT mode, B's objective decreases by at least $2 + 2s_i + 2M^{\lceil i/2 \rceil}$, then Equation (16) becomes

$$\sum_{i \in S'} \left(2 + 2s_i + 2M^{\lceil i/2 \rceil}\right) = n + \sum_{i \in S'} 2s_i + 2\sum_{i=1}^{n/2} M^i \geq n + U + 2\sum_{i=1}^{n/2} M^i.$$

So, $\sum_{i \in S'} 2s_i = U$ and this completes the proof. □

This result implies that the same recognition version of problem where both agents minimize their *weighted* sum of completion times, i.e. $P_2(\sum w_j^A C_j^A, \sum w_j^B C_j^B)$, is $\mathcal{NP}$-complete. Note that Theorem 10 shows the weak $\mathcal{NP}$-completeness of $P_2(\sum C_j^A, \sum C_j^B)$. We leave it as an open problem to derive a pseudopolynomial algorithm for the problem and thus rule out strong $\mathcal{NP}$-completeness similar to Theorem 4.

4 Single agent perspective

4.1 Minimization of makespan or total completion time

We now focus our attention to the design of algorithms for suggesting agent B which task to submit at each round, in order to optimize its objective function.

When B's objective is the minimization of the makespan or total completion time, finding an optimal strategy for B is trivial. Since in every round the shorter between the two candidate tasks wins, there is no advantage in deviating from the SPT sequence. More precisely, independently from A's submission sequence, by standard pairwise interchange arguments, it is easy to show that an optimal solution for agent B is attained by submitting tasks in SPT order. Hence, SPT is both a minimax strategy and an optimal offline algorithm for agent B when $f^B = C_{\max}^B$ and $f^B = \sum_j C_j^B$, for any objective f^A.

This argument does not hold anymore for the minimization of the *weighted* sum of completion times of B-tasks, for which different aspects will be considered in the subsequent sections.

4.2 Minimization of weighted total completion time: optimal offline response

In this section we present an algorithm for solving problem $P_3(f^A, \sum_j w_j^B C_j^B)$, i.e. it builds a sequence which minimizes the total weighted completion time of B-tasks, for a given sequence of submissions of A. Recall from (1) that the schedule is a sequence of blocks.

Scheduling all the tasks of B in SPT order has the advantage that the tasks are scheduled as early as possible. This was beneficial in Section 4.1 for the makespan and the total completion time criteria. However, task weights are not taken into account. It is well known that, in the classical (single agent) case, a schedule minimizing the total weighted completion time is obtained by sorting tasks according to the WSPT rule, i.e. in nonincreasing order of weight over processing time ratios. However, when considering multiple agents, sequencing the tasks of one

agent in WSPT order may result in a schedule that is not optimal due to the presence of the other agent's tasks, which may delay the heavier (and possibly long) tasks. (See Section 4.4 for a worst case analysis of the performances of SPT and WSPT algorithms as single agent strategies when B's objective is $f^B = \sum_j w_j C_j^B$.)

We introduce the following notation: The *density* ρ_j of a B-task b_j is given by the usual weight over processing time ratio w_j/b_j. Extending this concept to a sequence Δ of consecutive tasks, we define

$$\rho(\Delta) = \frac{w(\Delta)}{t(\Delta)} = \frac{\sum_{i \in \Delta \cap B} w_i}{\sum_{i \in \Delta \cap A} a_i + \sum_{i \in \Delta \cap B} b_i}.$$

Clearly, when Δ contains only a single B-task b_j, we have $\rho(\Delta) = \rho_j = w_j/b_j$.

The algorithm works as follows: First, it is easy to see (by a simple exchange argument) that, in any solution minimizing B's total weighted completion time, the tasks within each block of B must be ordered in nonincreasing order of density, i.e. in WSPT order. Hence, it remains to find the optimal arrangement of B-tasks into blocks. To this purpose the algorithm starts from a partial schedule containing a certain number of auxiliary blocks D_k, initially containing only A-tasks. In iteration $j = 1, \ldots, n$, the most dense b_j task among the unscheduled B-tasks is included in (appending it at the end of) a suitable block $D_{k(j)}$. Eventually, the algorithm returns the sequence $\langle D_1, D_2, \ldots, D_q \rangle$ which is proven to be an optimal solution schedule.

Algorithm 1 Algorithm of agent B for $P_3(f^A, \sum_j w_j C_j^B)$

WSPTINSERT

1: Build schedule $\tilde{\sigma}$ in which B submits its tasks in SPT order while those of A are submitted in the given order;
2: Consider block structure $\tilde{\sigma} = \langle A_1, B_1, A_2, B_2, \ldots A_q, B_q \rangle$;
3: Keep a label ℓ for each task of B pointing to its *original* block, e.g. $\ell(j) := k$ if $b_j \in B_k$;
4: Generate new sequence of initially empty blocks $\langle D_1, D_2, \ldots D_q \rangle$;
5: Initialize $D_k := A_k$ for all $k = 1, \ldots, q$;
6: /* Inserting the tasks of B at the correct (=best) position */
7: Renumber tasks of B in WSPT order: $\rho_1 \geq \rho_2 \geq \ldots, \geq \rho_n$;
8: **for** all tasks b_j from $j := 1$ to n **do**
9: **for** all blocks D_k, $k = \ell(j), \ldots, q$ **do**
10: Compute the objective function if b_j is appended at the end of block D_k
11: **end for**
12: Append b_j at the end of block $D_{k(j)}$ where the smallest objective value is obtained;
13: **end for**
14: Return the resulting schedule $\langle D_1, D_2, \ldots D_q \rangle$

The core of the algorithm is to establish which block $D_{k(j)}$ the current task b_j should be appended to. To this aim, we first build an *initial block structure* as follows. We order all tasks of B in SPT order, and feed the SPT sequence along with σ^A. The resulting schedule has structure $\langle A_1, B_1, A_2, B_2, \ldots A_q, B_q \rangle$ and we let $\ell(j)$ be the index of the block task b_j belongs to in such schedule. We will show that, when assigning task b_j to a block it suffices to consider appending b_j to a block D_k where $\ell(j) \leq k \leq q$. Note also that, since the initial block structure (Step 2) is obtained when B submits its tasks in SPT order when b_j is placed in block $B_{\ell(j)}$, the sequence composed of the first task of all blocks has nondecreasing processing times. As a consequence, the schedule obtained appending task b_j at the end of block D_h, with $h \geq \ell(j)$, is

always feasible, i.e., it can be obtained by a specific submission sequence of B task through the selection mechanism. In conclusion, the algorithm correctly considers all available positions for the generic task b_j (Step 9).

Let Δ_h^k be the sequence of all tasks of both agents in blocks $D_{h+1}, \ldots, D_k$ for $k > h$. The comparison between two insertion positions for some task b_j (cf. Step 10) is given by the following condition, which can be verified by elementary calculations (see Figure 4).

$$| \underbrace{A_h,\ B_h \mid b_j}_{D_h} \mid \overbrace{\underbrace{A_{h+1},\ B_{h+1}}_{D_{h+1}} \mid \ldots \mid \underbrace{A_k,\ B_k}_{D_k}}^{\Delta_h^k} \mid$$

Figure 4: Definition of Δ_h^k and insertion of b_j at the end of D_h (when $\rho_j > \rho(\Delta_h^k)$).

Proposition 11 *Comparing two blocks D_h and D_k with $k > h$, it is better to append b_j at the end of block D_h rather than at the end of block D_k if and only if*

$$\rho_j > \rho(\Delta_h^k)\,. \tag{20}$$

Theorem 12 *Given the sequence of submissions of agent A, algorithm* WSPTINSERT *finds an optimal solution of problem $P_3(f^A, \sum_j w_j C_j^B)$ for agent B.*

Proof. The proof is given by induction over the insertion of B-tasks into the D-blocks. The first, i.e. best task by WSPT will always be scheduled by B as early as possible due to the initial SPT structure. Indeed, in the first iteration of the **for** loop starting in Step 8, task b_1 will remain in block $D_{\ell(1)}$, as in the original SPT structure.

Considering iteration j with task b_j for some $j > 1$, we assume by induction that after iteration $j-1$ an optimal sequence is represented by the D-blocks. Clearly, task b_j is inserted at the best possible position by construction. It remains to be shown, that the insertion of j does not change the optimality of the position of any task $i < j$ inserted in one of the previous iterations. Since tasks are considered in nonincreasing order of densities and always appended at the end of a D-block, all B-tasks are scheduled in WSPT order in each D-block, but we have to show that each task i was put into the best possible D-block.

Recall $k(j)$ is the index of the D-block containing task j in the schedule returned by the algorithm. We first show that task $j > i$ ($\rho_i > \rho_j$) will never be placed in blocks $D_{\ell(i)}, \ldots, D_{k(i)-1}$, i.e. any task j with a lower density than i will be placed either in a D-block before $\ell(i)$, or in the same D-block $k(i)$ as i or in a later one.

Assume otherwise, i.e. j is appended to some block D_s with $\ell(i) \le s < k(i)$: At iteration i, it was better for task i to be placed in block $D_{k(i)}$ than in D_s. Proposition 11 implies that

$$\rho(\Delta_s^{k(i)}) > \rho_i \ge \rho_j\,.$$

Hence, block $D_{k(i)}$ would be a better choice for task j than block D_s. In fact, by the same argument it follows that no other tasks between i and j were appended to $D_{\ell(i)}, \ldots, D_{k(i)}$ before considering j, which implies $\Delta_s^{k(i)}$ is not changed between iteration i and j.

Thus, there remain three cases to be considered:

Case 1: $k(j) < \ell(i)$, i.e. j was positioned before block $D_{\ell(i)}$. In this case, b_j has the same effect on all positions available for i and its contribution does not affect the criterion of Proposition 20. Hence, the original decision for i is still the best choice.

Case 2: $k(j) > k(i)$, i.e. j was positioned in a block later than $D_{k(i)}$. Clearly, the insertion of b_j does not modify the current schedule until the position of b_j in block $D_{k(j)}$. Hence, if in iteration i it was better to place task i in block $D_{k(i)}$ rather than in D_s, $s \leq k(j)$[†], the same still applies in the j-th iteration. Consider then the possibility to place task i now in block D_r with $r > k(j)$. In iteration i, from the criterion of Proposition 11, one had that

$$\rho_i \geq \rho(\Delta^r_{k(i)}) \tag{21}$$

since otherwise block r was preferable to $k(i)$. Between iterations i and j, additional tasks with density lower than ρ_i were possibly inserted between the position of i in $D_{k(i)}$ and D_r. Let $w(I)$ resp. $t(I)$ denote their total weight resp. processing times. Clearly $\rho_i \geq w(I)/t(I)$, so from (21) and recalling $\rho(\Delta^r_{k(i)}) = w(\Delta^r_{k(i)})/t(\Delta^r_{k(i)})$, by a straightforward calculation one obtains

$$\rho_i \geq \frac{w(\Delta^r_{k(i)}) + w(I)}{t(\Delta^r_{k(i)}) + t(I)}\,. \tag{22}$$

Hence, the decision for $k(i)$ remains optimal.

Case 3: $k(j) = k(i)$, i.e. j is positioned in the same block as i. As before, b_j has no effect on the D-blocks before $D_{k(i)}$. Placing task i in some block D_s with $s > k(i)$ can be shown to be unprofitable by the same argument given in Case 2.

Summarizing, we have shown that reconsidering that decision of placing any task $i < j$ in iteration j would lead to exactly the same placement of i as before in iteration i. Thus, the optimality of the sequence remains valid also after iteration j. □

Theorem 13 *Algorithm* WSPTINSERT *can be implemented to run in* $O(n^2)$ *time.*

Proof. Clearly, the running time of WSPTINSERT is dominated by the $O(n^2)$ executions of Step 10. In a naive approach, each computation of the objective function would require linear time yielding an $O(n^3)$ overall running time. However, we can do better by storing for each block D_k its overall processing time $T_k := \sum_{j \in B_k \cap A} a_j + \sum_{j \in B_k \cap B} b_j$ and weight (w.r.t. B) $W_k := \sum_{j \in B_k \cap B} w_j$, which can be trivially kept updated.

Given the objective function value z over all D-blocks before inserting b_j, we can iteratively compute the candidate objective function value z_k arising if b_j is appended at the end of block D_k by $z_k = z_{k-1} + w_j T_k - W_k b_j$ for $k = \ell(j)+1, \ldots, q$ starting with $z_{\ell(j)} = z + w_j \left(b_j + \sum_{i=1}^{\ell(j)} T_i\right)$. Thus, we can go through all iterations of the **for** loop in Steps 9, 10 in linear time. □

4.3 Minimization of weighted total completion time: minimax strategy

We have already observed in Section 4.1 that SPT is a minimax strategy of agent B for problems $P_4(f^A, C^B_{\max})$ and $P_4(f^A, \sum_j C^B_j)$. Hereafter, we show how to exploit the offline algorithm described in Section 4.2 to derive a minimax strategy of B for problem $P_4(f^A, \sum_j w_j C^B_j)$.

Lemma 14 *For any strategy* σ^B *played by* B*, the maximum value of* B*'s objective* $\sum w_j C^B_j$ *is attained if* A *plays its tasks in SPT order.*

[†]Since anyway i would be placed before j in block $D_{k(j)}$, these arguments hold also for $s = k(j)$.

Proof. Let π^B be the sequence of tasks played by B. Assume by contradiction that B's worst outcome occurs when A plays a sequence $\pi^A = \langle a_1, a_2, \ldots, a_n \rangle$ where there exists a pair of tasks a_i, a_{i+1} with $a_i > a_{i+1}$. As usual, the resulting schedule can be seen as sequence of blocks: $\sigma(\pi^A, \pi^B) = \langle A_1, B_1, A_2, B_2, \ldots A_q, B_q \rangle$ where the first and the last blocks A_1 and B_q are possibly empty.

Consider now the new schedule in which A plays $\bar{\pi}^A$ obtained from π^A by swapping a_i and a_{i+1}. If a_i and a_{i+1} are in the same block A_j of $\sigma(\pi^A, \pi^B)$, then the objective of B is unchanged in the new schedule $\sigma(\bar{\pi}^A, \pi^B)$. Else, if a_i is in block A_j and a_{i+1} in block A_{j+1}, then in schedule $\sigma(\bar{\pi}^A, \pi^B)$, a_i and a_{i+1} are both in block A_j, thus delaying all the tasks of block B_j. This contradicts the fact that π^A is the worst opposing strategy for B. Hence, the thesis follows. □

The above results implies that a minimax strategy for B can be attained by applying the offline WSPTINSERT algorithm of Section 4.2 assuming that A submits its tasks in SPT order.

4.4 Minimization of weighted total completion time: performance of standard algorithms

When the objective of B is the minimization of the weighted total completion time, neither SPT nor WSPT are optimal submission strategies for B. Hereafter, we analyze the worst-case performance of these natural algorithmic strategies compared to an optimal strategy as defined in Section 2.2.

Recall that it was shown in Lemma 14 that the SPT rule pursued by agent A yields the worst outcome for any strategy of B. However, this does not imply that SPT of A also maximizes the worst-case performance ratio $r(H)$ for some heuristic H.

For the SPT strategy our analysis provides a tight performance bound. As can be expected for weighted total completion times, the performance of SPT can be very bad.

Theorem 15 *For the minimization of $\sum_j w_j C_j^B$ the SPT heuristic yields the performance bound*

$$r(SPT) = n.$$

Proof. To prove the upper bound, let C_j^B resp. $\tilde{C}_j^B$ be the completion time of a task b_j in the solution resulting from an optimal strategy resp. from an SPT strategy of B against an arbitrary strategy of A.

For any b_j let S_j^A denote the A-tasks processed before b_j. Since B follows an SPT strategy, any other strategy of B can not avoid to process all tasks in S_j^A before b_j, but possibly also others. Thus, the tasks of A cannot contribute more to $\tilde{C^B}_j$ than to C_j^B.

However, $\tilde{C}_j^B$ may be larger than C_j^B because of tasks of B scheduled before b_j by SPT, but not by the optimal strategy. In the worst case, all tasks (i.e. at most $n-1$) with processing time at most b_j might precede b_j in SPT, but not in the optimal strategy. Thus we get the following fairly rough bound:

$$\tilde{C}_j^B \leq C_j^B + (n-1)\, b_j \leq n\, C_j^B$$

This suffices to prove the upper bound of n stated in the theorem.

To prove the lower bound we consider the following instance for some large value M:
$a_1 = \ldots = a_n = 1$
$b_1 = \ldots = b_{n-1} = M$, $b_n = M + 1$
$w_1 = \ldots = w_{n-1} = M$, $w_n = M^2$

Since all tasks of A are identical, A has nothing to decide. No matter what B does, A will win the first n rounds and thus all its tasks are processed before any task of B.

The items of B are sorted according to SPT, which yields:

$$z^B_{SPT} = \sum_{k=1}^{n-1} (n + kM) \cdot M + (n + nM + 1) \cdot M^2$$

Considering only the largest powers of M we get a very rough estimation of $z^B_{SPT} \geq nM^3$.

An optimal solution for B would submit its item by WSPT and thus start with item b_n, which means:

$$z^B = (n + M + 1) \cdot M^2 + \sum_{k=1}^{n-1} (n + M + 1 + kM) \cdot M = M^3 + M^2(\frac{1}{2}n^2 + 2n - 1) + M(n^2 - 1)$$

Another estimation of the upper bound for z^B yields:

$$\frac{z^B_{SPT}}{z^B} \geq \frac{nM^3}{M^3 + (n^2/2 + 2n + 1)M^2 + Mn^2} \overset{M\to\infty}{\longrightarrow} n.$$

□

It should be noted that in the proof of Theorem 15 the tasks of A do not contribute to the constructed bounds at all. Thus, the same arguments could be used to show the following statement for the classical scheduling problem without agents.[‡]

Corollary 16 *Using the SPT heuristic for the classical minimization of the weighted sum of completion times on a single machine yields a tight worst-case performance ratio of n compared to the optimal solution obtained by the WSPT rule.*

If B pursues the WSPT strategy one could expect a better performance than for SPT. Indeed, the best lower bound we could find for $r(WSPT)$ is $O(\sqrt{n})$. Unfortunately, we could not find a matching upper bound but only a very rough estimate.

Theorem 17 *For the minimization of $\sum_j w_j C^B_j$ the performance ratio of the WSPT heuristic can be bounded as follows:*

$$\sqrt{n}/2 \leq r(WSPT) \leq n - 1.$$

Proof. To show an upper bound we first recall that the WSPT heuristic produces an optimal schedule for the tasks of B assuming that no A-tasks are present. Thus, we can restrict our analysis on bounding the possible increase of each term $w_j C^B_j$ caused by tasks of agent A. For convenience we denote the optimal solution for agent B for a given sequence of A as $Opt^B = \min_{\sigma^B}\{f^B(\sigma^A, \sigma^B)\}$.

Let S^A_j denote the A-tasks preceding b_j in the schedule produced by the WSPT heuristic, but not in the optimal schedule of B. Recalling the block structure of the sequence of tasks let $b_{j'}$ be the largest B-task of the block containing b_j, not scheduled later than b_j. This implies that $b_{j'}$ is at least as large as the first task of the block containing b_j and also $b_{j'} \geq b_j$. Since the sequence consisting of all starting tasks of all blocks is strictly increasing, there must be $a_i \leq b_{j'}$

[‡]The result was probably proved elsewhere, but we are not aware of any reference.

for each task $a_i \in S_j^A$, not matter to which block a_i actually belongs to. Thus by the WSPT ordering we can bound the contribution of a_i to the increase of each term $w_j C_j^B$ as follows:

$$w_j\, a_i \leq \left(\frac{b_j}{b_{j'}}\; w'_j\right) a_i \leq w'_j\, a_i \tag{23}$$

Next we consider the completion time $C_{j'}^{*B}$ of $b_{j'}$ in the optimal solution. We claim that all $a_i \in S_j^A$ must be scheduled before $b_{j'}$ also in the optimal solution. Assume otherwise: Since $a_i \leq b_{j'}$ there must exist some $\bar{a}_i \geq b_{j'} \geq a_i$ preceding a_i in the sequence of A-tasks. But since the B-block, in which the WSPT heuristic scheduled $b_{j'}$, starts with a task at most as large as $b_{j'}$ and thus not greater than $\bar{a}_i$, it follows from above that $\bar{a}_i$ is at least as large as all first tasks of blocks preceding $b_{j'}$ and thus can not be scheduled before $b_{j'}$ in any feasible solution.

Using this observation we can summarize the contribution bounded in (23) over all $a_i \in S_j^A$ and get

$$\sum_{i \in S_j^A} w_j a_i \leq w'_j \sum_{i \in S_j^A} a_i \leq w'_j C_{j'}^{*B} \leq Opt^B. \tag{24}$$

Thus, for each of the n terms $w_j C_j^B$ we obtain an increase of the objective function value in the WSPT heuristic of at most Opt^B which proves an upper bound of n on the performance ratio. Considering that the task with smallest value of $\frac{b_j}{w_j}$ is scheduled as early as possible by the WSPT heuristic, it follows that its contribution to the objective function can not be improved by any strategy of B and thus only $n-1$ terms have to be accounted for.

To show the lower bound consider the following instance for some arbitrarily small $\varepsilon > 0$ and a parameter value V to be chosen later:

$a_1 = \ldots = a_n = V + \varepsilon$

$b_1 = V + 2\varepsilon$, $b_2 = \ldots = b_n = 1$

$w_1 = V + 3\varepsilon$, $w_2 = \ldots = w_n = 1$

Again, since all tasks of A are identical, A has nothing to decide. Obviously, the items of B are sorted according to WSPT. By the WSPT rule, B loses with b_1 against all tasks of A. Neglecting the value ε we have:

$$z_{WSPT}^B = (n+1)V \cdot V + \sum_{k=1}^{n-1}((n+1)V + k) \cdot 1 = (n+1)V^2 + (n^2-1)V + \frac{1}{2}(n^2-n) \geq nV^2 + n^2V$$

An optimal solution for B would first submit all items $b_2, \ldots, b_n$ and win against A. Only at the end b_1 is submitted. We get:

$$z^B = \sum_{k=1}^{n-1} k \cdot 1 + (n-1+(n+1)V) \cdot V = \frac{1}{2}(n^2-n) + (n-1)V + (n+1)V^2 \leq n^2 + nV + (n+1)V^2$$

The resulting lower bound for $r(WSPT)$ can be roughly maximized by choosing $V := \sqrt{n}$.

$$r(WSPT) \geq \frac{nV^2 + n^2V}{n^2 + nV + (n+1)V^2} = \frac{n^2 + n^{5/2}}{2n^2 + n^{3/2} + n} \geq \frac{\sqrt{n}}{2}$$

□

5 A modified selection mechanism

As already observed in Section 1, the adopted selection mechanism does reflect a sensible coordinator objective (minimizing average flow time), but it does not take any fairness issues into account. For instance, it may even happen that all the tasks of one agent are performed only after all the tasks of the other agent are completed, e.g. if $a_n \leq b_1$. In this section we modify the selection mechanism in order to incorporate some notion of fairness. In particular, we allow an agent to override the usual mechanism if it did not have access to the machine for at least T time units. More precisely, we consider the following *modified selection rule*:

1. Each agent submits one of its tasks.
2. Let X denote the agent who lost the previous round. If the most recently scheduled task of X was completed at least T time units ago, then the task submitted by X is selected. Otherwise (standard case) the shortest between the two submitted tasks is selected.

We will refer to T as the *tenure time*. Note that when T is sufficiently small, the modified selection mechanism becomes indeed the round robin rule, while the original selection rule is attained for large T.

We next show that unfortunately such a fairer rule does not allow to efficiently devise an optimal offline or minmax strategy, even when agent B holds the simplest possible objective functions. In the proofs we make use of the following variant of the classical bin packing problem, which allows one item in each bin to exceed the capacity. It was shown to be strongly $\mathcal{NP}$-complete by [15].

> Open-End Bin Packing:
> **Instance**: integers n, k, C, nonnegative integer weights $S = \{s_1, \ldots, s_n\}$ such that $s_j < C$ for $j = 1, \ldots, n$.
> **Question**: Is there a partition $\{\mathcal{B}_1, \ldots, \mathcal{B}_k\}$ of S into k bins such that for each bin the sum of all item weights *but the largest* is below C, i.e., for all $i = 1, \ldots, k$
>
> $$\sum_{j \in \mathcal{B}_i} s_j - \max_{j \in \mathcal{B}_i} \{s_j\} \leq C - 1 \; ?$$

Theorem 18 *Under the modified selection rule with tenure period T, problem $P_3(f^A, C^B_{\max})$ is strongly $\mathcal{NP}$-hard.*

Proof. Given an instance of Open-End Bin Packing, consider an instance of our problem in which $T = C$, agent A has k identical tasks of length Q, such that $Q > T$, and agent B owns n tasks, of size $b_j = s_j$, $j = 1, \ldots, n$. We want to show that there is a schedule such that $C^B_{\max} \leq \sum_j b_j + (k-1)Q$ if and only if the instance of Open-End Bin Packing is a yes-instance.

Consider a yes-instance of the scheduling problem. In such schedule, we call *block* the set of B-tasks between two consecutive A-tasks. Since $Q > T$, and all B-tasks are shorter than T, agent A can schedule one of its tasks only when the tenure period has elapsed (or when B has no more tasks to schedule). With no loss of generality, we can suppose that the last B-task in a block is also the longest B-task in the block. Because of the modified selection mechanism, such last B-task starts at the latest $C - 1$ time units after the first B-task of the block has started. Hence, for each block $\mathcal{B}$ one has

$$\sum_{j \in \mathcal{B}} b_j - \max_{j \in \mathcal{B}} \{b_j\} \leq C - 1$$

and therefore the block corresponds to a feasible packing of a bin. Since $C^B_{\max} \leq \sum_j b_j + (k-1)Q$, only $k-1$ A-tasks are performed before all B-tasks are completed, i.e., all B-tasks are partitioned in at most k blocks. Hence, the number of bins in the bin packing instance is at most k, i.e., it is a yes-instance.

Viceversa, given a yes-instance of the bin packing problem, clearly the bins define a feasible schedule in which B-tasks are partitioned into k blocks, i.e., the last B-task completes within $\sum_j b_j + (k-1)Q$. □

Note that in the construction used in the above theorem, all tasks of agent A are identical. This implies the $\mathcal{NP}$-completeness of the minimax version of the same problem. A very similar construction establishes that also the case in which agent B's objective is the minimization of total flow time, the offline problem is intractable.

Theorem 19 *Under the modified selection rule with tenure period T, problem $P_3(f^A, \sum_j C_j)$ is strongly $\mathcal{NP}$-hard.*

Proof. Given an instance of Open-End Bin Packing, we consider again an instance of our problem in which $T = C$ and agent B owns n tasks, of size $b_j = s_j$, $j = 1, \ldots, n$. Agent A has k tasks. The first $k-1$ tasks are identical, of length $Q > T$. The last task is very long with $a_k > n((k-1)Q + \sum_j b_j)$.

Agent A submits its tasks in SPT order, i.e., a_k is the last task to be submitted. We want to show that there is a schedule such that $\sum_j C_j \leq a_k$ if and only if the instance of Open-End Bin Packing is a yes-instance.

The reasoning is very similar to the previous theorem. Consider a yes-instance of the scheduling problem. In such a schedule, no B-task is scheduled after a_k, since otherwise the completion time of any such task would already exceed a_k. Conversely, if no task is scheduled after a_k, then certainly $\sum_j C_j \leq a_k$, since the completion time of any B-task cannot exceed $(k-1)Q + \sum_j b_j$. This means that a scheduling instance is a yes-instance if and only if all B-tasks are scheduled in k blocks. As in Theorem 18, since $Q > T$ and all B-tasks are shorter than T, it is easy to check that each block corresponds to a feasible packing of a bin. Since there are k blocks, a schedule with the desired property exists if and only if the bin packing instance is a yes-instance. □

6 Conclusions

In this paper we analyzed a two-agent scheduling situation with an external coordinator, who enforces a selection mechanism on the task sequence submitted by the two agents. We addressed the scenario in which, at each round, the shortest submitted task is selected for processing, while the agents pursue the most common scheduling performance indices (makespan, total weighted or unweighted completion time). We analyzed various optimization problems related to the agents' strategy and behavior.

We wish to underline that the results presented in this paper are strongly related to the fact that the coordinator uses SPT as a selection rule. This rule aims at minimizing the average time spent in the system by the tasks, and it is especially sensible in a context in which the system coordinator has no information on the tasks of the two agents before the tasks show up. Such selection rule is common knowledge. In Section 4 we have seen that under such mechanism, when the opponent's task sequence is known, the optimal strategy of an agent can be found in polynomial time, either by a simple rule (for makespan or total flow time objective) or by a fairly elaborate greedy-type algorithm (total weighted flow time objective).

To our knowledge, this is one of the first attempts to analyze this type of scheduling structure, hence more has to be done in this respect. Among future research directions, we can propose the following.

- Considering that the system coordinator has no information on the tasks of the two agents before the tasks show up, the SPT selection rule is sensible, since it aims at minimizing the average time spent in the system by the tasks, i.e., overall system productivity. However, this is by no means a fair rule, since in general it can result in very unbalanced schedules. Hence, a more neutral coordinator may decide to enforce different rules, that incorporate some idea of fairness, e.g. as in Section 5, where it was shown that the resulting offline problem (for an agent holding total completion time as objective) becomes NP-hard. Another rule taking into account fairness concepts is round robin, in which the agents simply take turns in using the resource (as in [4, 9]). We note here that the complexity of the offline problem with such seemingly simple selection mechanism, when pursuing total weighted completion time is still open.

  Different selection mechanisms may definitely change the agent's strategy. In this regard investigating the problems arising from the adoption of new and different coordinator selection mechanisms is certainly an interesting topic for future research.

- For a given selection rule, the situation described in this paper can be viewed as a classical noncooperative game, in which for each pair of strategies (σ^A, σ^B), the agents get payoffs $f^A(\sigma^A, \sigma^B)$ and $f^B(\sigma^A, \sigma^B)$ respectively. In a more classical setting, one could analyze the existence and the properties of equilibria of such game. These equilibria might be also used by the system owner to enforce solutions from which the agents have no incentive to deviate.

- When studying the problems faced by the individual agent B, we assumed that the number and length of the tasks of agent A are known in advance. In a different, perhaps more realistic setting, an agent may only have limited information on the other agent's tasks, e.g., the number and the total duration, or some information on the distribution of tasks' durations. In principle, this allows an agent to define strategies in a dynamic way, exploiting the information revealed up to a certain point. For instance, if agent A has already won several rounds, so that it has few tasks left, but the total duration of unscheduled tasks is still large, most likely the remaining tasks are long, and therefore agent B can decide that time has come to play longer tasks.

- Different variants of the problem arise if the losing task of each round is not forced to remain submitted in the next round, as in our linear conveyor model. One possibility would be a *circular conveyor* model where the losing task of each round is moved to the end of the submission sequence of the corresponding agent. This scenario was briefly discussed by the authors in [3] where a number of preliminary results are given. However, most questions remain open for these cases. Furthermore, a completely open environment may allow each agent to withdraw the losing task and submit any unprocessed task in the next round. In such a case, the worst-case analysis of a strategy for one agent may suffer from a completely erratic behavior of the opponent. The resulting scenario was studied in detail in the recent work [19].

Acknowledgments

This work was supported by the Austrian Science Fund (FWF): [P 23829-N13] and by the Italian Ministry MIUR project PRIN 2009XN4ZRR_002.

References

[1] Agnetis A., J.-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, A. Soukhal (2014). *Multiagent Scheduling: Models and Algorithms*, Springer, Berlin.

[2] Agnetis A., P.B. Mirchandani, D. Pacciarelli, A. Pacifici (2004). Scheduling problems with two Competing agents, *Operations Research*, 52, 2, 229-242.

[3] Agnetis A., G. Nicosia, A. Pacifici, U. Pferschy (2013). Two agents competing for a shared machine, *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT 2013)*, *Lecture Notes in Computer Science* 8176, 1–14, Springer.

[4] Agnetis A., D. Pacciarelli, A. Pacifici (2007). Combinatorial models for multi-agent scheduling problems, in *Multiprocessor Scheduling: Theory and Applications*, I-Tech Education and Publishing, Vienna, Austria.

[5] Agnetis A., G. De Pascale, M. Pranzo (2009). Computing the Nash solution for scheduling bargaining problems, *International Journal of Operational Research*, 6(1), 54–69.

[6] Arbib C., S. Smriglio, M. Servilio (2004). A competitive scheduling problem and its relevance to UMTS channel assignment, *Networks*, 44(2), 132–141.

[7] Baker K., J.C. Smith (2003). A multiple criterion model for machine scheduling, *Journal of Scheduling*, 6(1), 7–16.

[8] Cohen J., D. Cordeiro, D. Trystram, F. Wagner (2011). Multi-organization scheduling approximation algorithms, *Concurrency and Computation: Practice and Experience*, 23(17), 2220–2234.

[9] Darmann A., G. Nicosia, U. Pferschy, J. Schauer (2014). The subset sum game, *European Journal of Operational Research*, 233(3), 539–549.

[10] Ding G., S. Sun (2010). Single-machine scheduling problems with two agents competing for makespan, *Lecture Notes in Computer Science*, 6328, 244–255, Springer.

[11] Garey M.R., R.E. Tarjan, G.T. Wilfong (1988). One-processor scheduling with symmetric earliness and tardiness penalties, *Mathematics of Operations Research*, 13(2), 330–348.

[12] Hall N.G., Z. Liu (2013). Market good flexibility in capacity auctions, *Production and Operations Management*, 22(2), 459–472.

[13] Huynh Tuong N., A. Soukhal, J.-C. Billaut (2012). Single-machine multi-agent scheduling problems with a global objective function, *Journal of Scheduling*, 15(1), 311–321.

[14] Jayasudha, A.R., T. Purusothaman (2012). Job Scheduling Model with Job Sequencing and prioritizing strategy in Grid Computing, *International Journal of Computer Applications*, 46(24), 29–32.

[15] Leung J.Y.-T., M. Dror, G.H. Young (2001). A note on an open-end bin packing problem, *Journal of Scheduling*, 4, 201–207.

[16] Marini C., G. Nicosia, A. Pacifici, U. Pferschy (2013), Strategies in competing subset selection, *Annals of Operations Research*, 207(1), 181–200.

[17] Nicosia G., A. Pacifici, U. Pferschy (2009). Subset weight maximization with two competing agents, *Proceedings of Algorithmic Decision Theory: ADT 2009*, *Lecture Notes in Computer Science* 5783, 74–85, Springer.

[18] Nicosia G., A. Pacifici, U. Pferschy (2011). Competitive subset selection with two agents, *Discrete Applied Mathematics*, 159(16), 1865-1877.

[19] Nicosia G., A. Pacifici, U. Pferschy (2014). Scheduling the Tasks of Two Agents with a Central Selection Mechanism, submitted. available as: Optimization Online 2014-02-4222.

[20] Pessan C., J.-L. Bouquard, E. Neron (2008). An unrelated parallel machines model for an industrial production resetting problem, *European Journal of Industrial Engineering*, 2, 153-171.

[21] Soomer M.J., G.J. Franx (2008). Scheduling aircraft landings using airlines' preferences, *Mathematical Programming*, 190, 277–291.

[22] T'Kindt V., J.-C. Billaut (2006). *Multicriteria scheduling. Theory, models and algorithms*, Springer.

[23] Wellman M.P., W.E. Walsh, P.R. Wurman, J.K. MacKie-Mason (2001). Auction protocols for decentralized scheduling, *Games and Economic Behavior*, 35 (1/2), 271–303.

[24] Ye, D., G. Zhang (2012). Coordination Mechanisms for Selfish Parallel Jobs Scheduling, in M. Agrawal et al. (eds.), 9th Annual Conference on Theory and Applications of Models of Computation, 2012, *Lecture Notes in Computer Science* 7287, 225–236, Springer.