Scheduling Two Competing Agents When One Agent Has Significantly Fewer Jobs

Danny Hermelin[1], Judith-Madeleine Kubitza[2], Dvir Shabtay[1], Nimrod Talmon[2], and Gerhard Woeginger[3]

1 **Ben Gurion University of the Negev, Israel**
hermelin@bgu.ac.il, dvirs@bgu.ac.il
2 **TU Berlin, Germany**
judith-madeleine.kubitza@campus.tu-berlin.de, nimrodtalmon77@gmail.com
3 **Eindhoven University of Technology, The Netherlands**
gwoegi@win.tue.nl

Abstract

We study a scheduling problem where two agents (each equipped with a private set of jobs) compete to perform their respective jobs on a common single machine. Each agent wants to keep the weighted sum of completion times of his jobs below a given (agent-dependent) bound. This problem is known to be NP-hard, even for quite restrictive settings of the problem parameters.

We consider parameterized versions of the problem where one of the agents has a small number of jobs (and where this small number constitutes the parameter). The problem becomes much more tangible in this case, and we present three positive algorithmic results for it. Our study is complemented by showing that the general problem is NP-complete even when one agent only has a single job.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Parameterized Complexity, Multiagent Scheduling

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.55

1 Introduction

Scheduling is a well-studied research area which provides a fertile ground for combinatorial problems. In a typical scheduling problem, we are given a set of jobs that have to be scheduled on a set of machines, arranged according to a specific machine setting. The objective is to determine a schedule which minimizes a predefined scheduling criterion, such as the makespan, total weighted completion time, and total weighted tardiness. There are different machine settings such as a single machine, parallel machines, flow-shop and job-shop, and each scheduling problem may have different characteristics and constraints. We refer the reader, for example, to [22] for further examples of scheduling problems and for a detailed survey of classical results.

We consider a scheduling problem with two agents Alice and Bob who each own a set of jobs that are to be scheduled non-preemptively on a single machine. An instance of this scheduling problem consists of the following:

- two sets $\{J_1^A, \ldots, J_n^A\}$ and $\{J_1^B, \ldots, J_k^B\}$ of jobs;
- the processing times $p_1^A, \ldots, p_n^A$ and $p_1^B, \ldots, p_k^B$ of these jobs;
- the weights $w_1^A, \ldots, w_n^A$ and $w_1^B, \ldots, w_k^B$ of these jobs;
- two bounds A and B.


© Danny Hermelin, Judith Kubitza, Dvir Shabtay, Nimrod Talmon, and Gerhard Woeginger;
licensed under Creative Commons License CC-BY

All processing times, all weights, and both bounds A and B are positive integers (notice that A and B are used both as labels and as integers). A *schedule* is simply a permutation of the union of both job sets, specifying the ordering in which the machine is executing the jobs. A schedule assigns to every job a corresponding *job completion time*; we denote by C_i^A and C_i^B respectively the completion times of the ith job of Alice and the ith job of Bob in some given schedule. The goal is to find a so-called *feasible schedule*, which satisfies the conditions

$$\sum_i w_i^A C_i^A \leq A \qquad \text{and} \qquad \sum_i w_i^B C_i^B \leq B.$$

Throughout the paper we refer to this problem as the Two Agent Scheduling problem.

The Two Agent Scheduling problem belongs to the family of *competitive multi-agent scheduling problems*: the jobs are partitioned into subsets, each belonging to a different agent that has his own scheduling criterion to optimize. Multi-agent scheduling problems were first introduced by Baker and Smith [3] and Agnetis et al. [2], which focus on the case with only two agents. For various combinations of the scheduling criteria, the objective in [3] is to minimize the weighted sum of the agents criteria. On the other hand, the objective in [2] is to minimize the scheduling criterion of the first agent, subject to a hard upper bound on the value of the criterion of the second agent. Following these two fundamental papers, numerous researchers have studied various combinations of multi-agent scheduling problems (see for instance Yuan et al. [23], Leung et al. [16], Lee et al. [14], Mor and Mosheiov [18], and Kovalyov et al. [13]). Detailed surveys of these problems appear in Perez-Gonzalez and Framinan [21] and in a recent book by Agnetis *et al.* [1].

Agnetis et al. [2] established that Two Agent Scheduling with unit weights is NP-complete. Furthermore, they designed a pseudo-polynomial time algorithm (which becomes polynomial time if all processing times are given in unary) for the problem. Lee et al. [14] extended this pseudo-polynomial time result to the case where the number of agents is an arbitrary but fixed constant (which is not part of the input). Oron et al. [20] proved that Two Agent Scheduling is NP-complete even when the processing times of all jobs are unit. They also show that the special case where Alice's jobs have unit weights is solvable in polynomial time.

1.1 Our contribution

It is natural to assume that the Two Agent Scheduling problem becomes more tangible if one of the agents has significantly fewer jobs than the other. Hence, we will parameterize the problem by the number k of jobs of agent Bob; throughout, we will tacitly assume that $k \ll n$. We will derive the following results:

1. The case where Alice's jobs have unit weights and the case where Alice's jobs have unit processing times are both fixed-parameter tractable with respect to parameter k. In other words, the problem can be solved in $f(k) \cdot n^{O(1)}$ time, where $f()$ is a function independent of n.
2. In stark contrast to the above positive results, the case where Alice's jobs have arbitrary weights and arbitrary processing times is NP-complete even if Bob has only a single job (that is, even if $k = 1$ holds).
3. If all job weights and processing times are given in unary and if Bob has a constant number of jobs, the problem is polynomial time solvable.

We also study another variant of Two Agent Scheduling where all jobs have unit weight: while Bob might have many jobs, we assume that his jobs have only a constant number t of different processing times. We show that this case is polynomial time solvable.

For obtaining the algorithms mentioned above, we will carefully analyze the combinatorial structure of Two Agent Scheduling. In the case where Alice's jobs have unit weights or unit processing times, it turns out that there is a very simple $n^{O(k)}$-time algorithm; however, it takes considerable work to improve this to a fixed-parameter algorithm. We model the problem in a non-trivial fashion as a mixed integer linear program (MILP), and then apply Lenstra's celebrated algorithm to it. This MILP modelling approach does not work for the general case where Alice's jobs have arbitrary weights and arbitrary processing times; in this general case we resort to dynamic programming to get an XP algorithm.

1.2 Further related work

The main contribution of this paper is a fixed-parameter algorithm for a natural scheduling problem. While scheduling theory and parameterized complexity (the study of fixed-parameter algorithms [8, 10, 19]) are two well-studied research fields, it seems that up to now the research on the interface between both areas has been rather limited. In fact, we found only a few research papers that study classical scheduling problems from the perspective of parameterized complexity. Bodlaender and Fellows [6] study the so-called precedence constrained k-processor scheduling problem. Fellows and McCartin [9] study the problem of scheduling unit length jobs on a single machine with precedence constraints. Both papers [6, 9] contain only hardness results. Some papers which include positive results on scheduling problems in the perspective of parameterized complexity are the paper by Halldórsson and Karlsson [12], the papers by van Bevern et al. [4, 5], and the paper by Mnich and Wiese [17], which shows that various classical scheduling problems are fixed-parameter tractable with respect to certain natural parameters.

2 Unit Weights and Unit Processing Times

In this section we show that Two Agent Scheduling is fixed-parameter tractable with respect to parameter k, the number of jobs that Bob has, in case Alice's jobs have either unit weights or unit processing times. We begin with the unit weight case.

2.1 Unit weights

▶ **Theorem 1.** Two Agent Scheduling *where Alice's jobs have unit weights is fixed-parameter tractable with respect to* k.

Agnetis *et al.* [2] observed that if an instance of Two Agent Scheduling where both agents have unit weights has a feasible schedule, then there is always a feasible schedule where the relative order amongst the jobs of each agent is according to the SPT (Shortest Processing Time first) rule. That is, we can assume that each job of each agent proceeds all jobs with greater processing times of the same agent in a feasible schedule. This remains true for Alice's jobs even if Bob's jobs have arbitrary weights. Thus, we assume $J_1^A, \ldots, J_n^A$ are already indexed according to the SPT order (that is, that $p_i^A \leq p_{i+1}^A$ for all $i \in \{1, \ldots, n-1\}$). Since Bob has only k jobs, we can iterate through all relative orderings of his jobs and "guess" his relative order. Thus, by allowing an additional multiplicative factor of $k!$ to the running time of our algorithm, we can assume that we know this ordering, and that $J_1^B, \ldots, J_k^B$ are already sorted accordingly.

Therefore, to determine whether a feasible schedule is actually possible, we only need to figure out if it is possible to interleave the two ordered sets of jobs together in a way that satisfies both Alice's and Bob's bounds on their total weighted completion times. Towards

this aim, we formalize the problem as a mixed integer linear program (MILP) where the number of integer variables is k. We complete the proof by using the celebrated result of Lenstra [15] which states that determining whether a given MILP has a feasible solution is fixed-parameter tractable with respect to the number of integer variables.

Alice's total completion time bound

For each job J_i^B, define an integer variable x_i representing the number of jobs belonging to Alice that are scheduled before J_i^B. For each $1 \leq i \leq k$, we add a pair of constraints ensuring that $0 \leq x_i \leq n$, and for $i \neq n$, we add the constraint $x_i \leq x_{i+1}$. Using the variables x_i, we encode the bound on the total completion time of Alice's jobs by adding the linear constraint

$$\left(\sum_{i=1}^{n}(n-i+1)\cdot p_i^A\right) + \left(\sum_{i=1}^{k}(n-x_i)\cdot p_i^B\right) \leq A. \tag{1}$$

▶ **Lemma 2.** *Assuming the value of each variable x_i equals the number of Alice's jobs that are scheduled before J_i^B, the left-hand side of constraint (1) is precisely the total completion time of Alice's jobs.*

Proof. Observe that the first term in the left-hand side of constraint (1) is precisely the sum of completion times of Alice's jobs when no job of Bob is scheduled at all. We now add Bob's jobs according to the intended meaning of the variables $x_1, \ldots, x_k$. If there are x_i jobs of Alice prior to J_i^B in the presumed schedule, then J_i^B causes an increase of p_i^B to the completion time of $n - x_i$ jobs of Alice. Thus, adding all of Bob's jobs causes an additional increase of $\sum_{i=1}^{k}(n-x_i)\cdot p_i^B$ to the total completion time of Alice's jobs. ◀

Bob's total completion time bound

The encoding of Bob's bound is a bit more involved. Specifically, for each J_i^B, we introduce a real-valued variable y_i which we would like to be equal to the contribution of Alice's jobs to the completion time of J_i^B. Note that by our intended meaning for variable x_i, this is precisely $\sum_{j=1}^{x_i} p_j^A$. However, we cannot encode this directly as a linear constraint. We therefore introduce n additional real-valued variables corresponding to J_i^B, denoted as $y_{i,j}$ for $j \in \{1, \ldots, n\}$, which are ensured to be non-negative by adding constraints $y_{i,j} \geq 0$ for each j. The $y_{i,j}$ variables are used to provide upper-bounds to the "steps" in the contribution of Alice's jobs as depicted in Fig. 1. Accordingly, we add the constraints $y_{i,j} \geq (x_i - j + 1)\cdot(p_j^A - p_{j-1}^A)$, for each $j \in \{1, \ldots, n\}$. (Naturally, we set here $p_0^A = 0$.) Furthermore, we add the constraint $y_i \geq \sum_{j=1}^{n} y_{i,j}$ so that y_i will equal its intended meaning.

Finally, to complete the construction of our MILP, we add the following constraint which encodes the bound on Bob's total weighted completion time.

$$\left(\sum_{i=1}^{k} w_i^B \cdot \sum_{j=1}^{i} p_j^B\right) + \left(\sum_{i=1}^{k} w_i^B \cdot y_i\right) \;\leq\; B. \tag{2}$$

▶ **Lemma 3.** *Assuming the value of each variable x_i equals the number of Alice's jobs that are scheduled before J_i^B, the left-hand side of constraint (2) is an upper-bound to the total completion time of Bob's jobs in any feasible solution.*

Proof. Observe that the first term in the left-hand side of constraint (2) is the total weighted completion time of Bob's jobs ordered $J_1^B, \ldots, J_k^B$, assuming no job of Alice has been

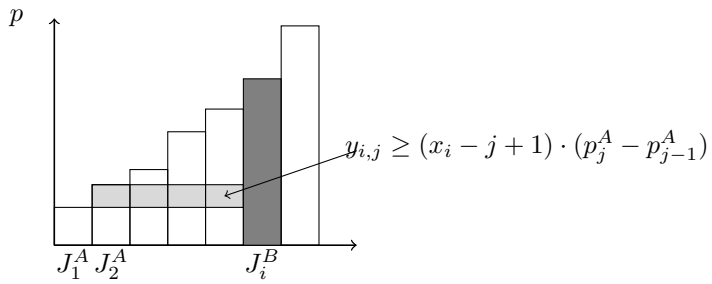

Figure 1 The contribution of x_i jobs that belong to Alice which are scheduled prior to J_i^B increases the completion time of this job by $\sum_{j=1}^{x_i} p_j^A$. The variable $y_{i,j}$ is intended to capture j'th "step" of this contribution.

scheduled. We argue that the second term upper-bounds the contribution of Alice's jobs. For this, it suffices to show that the contribution of Alice's jobs to the completion time of J_i^B, for each $i \in \{1, \ldots, k\}$, is at most y_i. We know that this contribution is $\sum_{j=1}^{x_i} p_j^A$, assuming x_i is the number of Alice's jobs that are scheduled prior to J_i^B. Since we are concerned only with feasible solutions where the constraints on the variables $y_i, y_{i,1} \ldots, y_{i,n}$ are met, we have that the following hold, and we are done:

$$y_i \geq \sum_{j=1}^{n} y_{i,j} \geq \sum_{j=1}^{n} \max\{0, (x_i - j + 1) \cdot (p_j^A - p_{j-1}^A)\} \geq \sum_{j=1}^{x_i} (x_i - j + 1) \cdot (p_j^A - p_{j-1}^A)$$
$$= x_i p_1^A + (x_i - 1)(p_2^A - p_1^A) + \cdots + (p_{x_i}^A - p_{x_i-1}^A) = \sum_{j=1}^{x_i} p_j^A. \quad \blacktriangleleft$$

To summarize, combining Lemma 2 and Lemma 3 shows that, assuming there exists a feasible schedule where Bob's jobs are scheduled according to our assumed order, the MILP described above will have a feasible solution, and otherwise no such solution exists. Moreover, the MILP described has only k integer variables. Thus, iterating through all possible orderings of Bob's jobs, and using Lenstra's algorithm to determine whether the MILP corresponding to each order has a feasible solution, gives us the fixed-parameter algorithm promised in Theorem 1.

2.2 Unit processing times

The somewhat symmetric problem, where Alice's jobs have arbitrary weights and unit processing times can also be shown to be fixed-parameter tractable with respect to k using similar ideas as above. The first crucial observation in this case, which can be seen by a simple exchange argument, is that we can assume that in any feasible schedule Alice's jobs are sorted amongst themselves in a non-increasing weight order. Thus, assuming $w_1^A \geq \cdots \geq w_n^A$, we know that J_i^A will be scheduled before J_{i+1}^A for all $i \in \{1, \ldots, n-1\}$. As in the proof of Theorem 1, by guessing the relative order of Bob's job in the schedule, we can assume that the same applies for Bob's jobs as well. Again we introduce variables $x_1, \ldots, x_k$, but this time x_i represents the number of Alice's jobs to be scheduled after J_i^B. Similarly to before, we add all constraints $0 \leq x_i \leq n$ and $x_i \geq x_{i+1}$. The bound on the weighted sum of completion times of Bob's jobs can be expressed as

$$\left(\sum_{i=1}^{k} w_i^B \sum_{j=1}^{i} p_j^B \right) + \left(\sum_{i=1}^{k} w_i^B (n - x_i) \right) \leq B,$$

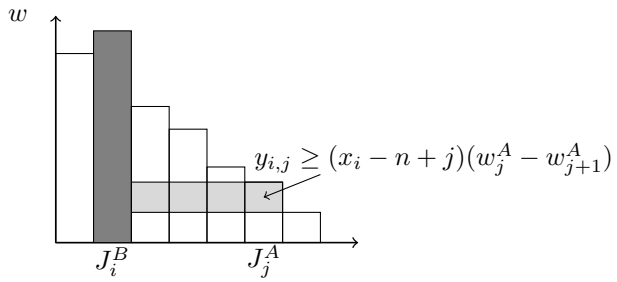

Figure 2 Inserting job J_i^B increases the weighted completion time of each following job J_j^A by $p_i^B w_j^A$. Thus, the contribution of J_i^B to the total weighted completion time of Alice's jobs is $p_i^B \sum_{j=n-x_i+1}^{n} w_j^A$. The variables $y_{i,j}$ are used to lower-bound the steps of the sum $\sum_{j=n-x_i+1}^{n} w_j^A$.

where the first term in the left-hand side is the contribution of Bob's jobs, and the second term in the left-hand side is the contribution of Alice's jobs.

For the total weighted completion time of Alice's jobs, we again introduce a set of real-valued variables $y_i, y_{i,1}, \ldots, y_{i,n}$ for each $i \in \{1 \ldots, k\}$. Here, variable y_i is meant to encode the contribution of J_i^B to the total weighted completion time of Alice's jobs. Note that this is precisely $p_i^B \sum_{j=n-x_i+1}^{n} w_j^A$ (see Fig. 2). We use the variables $y_{i,j}$ to encode lower-bounds on the steps of the sum $\sum_{j=n-x_i+1}^{n} w_j^A$, as done in the proof of Theorem 1, by adding the constraints $y_{i,j} \geq (x_i - n + j)(w_j^A - w_{j+1}^A)$ and $y_{i,j} \geq 0$ for all $j \in \{1, \ldots, n\}$. (Naturally we set $w_{n+1}^A = 0$.) Finally, we encode the bound on Alice's total weighted completion time by

$$\left(\sum_{i=1}^{n} i w_i^A\right) + \left(\sum_{i=1}^{k} p_i^B y_i\right) \leq A.$$

▶ **Theorem 4.** Two Agent Scheduling *where Alice's jobs have unit processing time is fixed-parameter tractable with respect to* k.

3 Single Job Bob

In the previous section we showed that when Alice's jobs have either unit weights or unit processing times, Two Agent Scheduling becomes fixed-parameter tractable in the number of Bob's jobs. In this section we complement this result by showing that when Alice's jobs have arbitrary weights and processing times, the problem becomes NP-complete already when Bob has a single job. Note that this rules out a similar fixed-parameter algorithm for general Two Agent Scheduling in the strongest possible sense (indeed, even an XP-algorithm cannot exist); such an algorithm would imply P=NP.

▶ **Theorem 5.** Two Agent Scheduling *where Bob has a single job is* NP*-complete.*

Proof. We provide a reduction from the NP-complete Partition problem [11]: Given a set $X = \{x_1, \ldots, x_n\}$ of positive integers (encoded in binary) with $\sum_{i=1}^{n} x_i = 2Z$, determine whether X can be partitioned into two sets S_1 and S_2 such that $\sum_{x_i \in S_1} x_i = \sum_{x_i \in S_2} x_i = Z$.

Given an instance X to the Partition problem, we create for each element x_i a job J_i^A for Alice with both processing time and weight equal to x_i. This gives n jobs for Alice with $p_i^A = w_i^A = x_i$ for all $i \in \{1, \ldots, n\}$. For Bob, we create a single job with both unit processing time and unit weight. Thus, $p_1^B := w_1^B := 1$. We set the bound A on the total weighted completion time of Alice's jobs to be $A = \sum_{i=1}^{n} \sum_{j=1}^{i} x_i x_j \; + \; Z$. We set the bound B for

Bob to be $B = Z + 1$. This completes our construction of the Two Agent Scheduling instance.

Suppose that X can be partitioned into two sets S_1 and S_2 with $\sum_{x_i \in S_1} x_i = \sum_{x_i \in S_2} x_i = Z$. We create a schedule σ where we first schedule all of Alice's jobs corresponding to elements of S_1 in an arbitrary order, followed by Bob's job, followed by all of Alice's jobs corresponding to the elements of S_2 in an arbitrary order. Observe that Bob's job completes in σ after $\sum_{x_i \in S_1} x_i + 1 = Z + 1$ time units, and so his total weighted completion time bound is met. To see that Alice's bound is met, observe that without Bob's job the total weighted completion time of Alice's jobs in σ is precisely $\sum_{i=1}^{n} \sum_{j=1}^{i} x_i x_j$. Adding Bob's job increases the competition time of Alice's jobs that correspond to elements of S_2 by a unit. Thus, it contributes precisely $\sum_{x_i \in S_2} x_i = Z$ to the total weighted completion time of Alice, and so Alice's bound is met as well.

For the other direction, suppose there is a feasible schedule σ to our Two Agent Scheduling problem. Let $\mathcal{J}_1$ denote the set of Alice's jobs that are placed before Bob's job in σ, and let $\mathcal{J}_2$ denote her remaining jobs. Since Bob's bound is satisfied in σ, it must be that $\sum_{J_i^A \in \mathcal{J}_1} p_i^A \leq B - p_1^B = Z$. Moreover, note that the total weighted completion time of Alice is $\sum_{i=1}^{n} \sum_{j=1}^{i} x_i x_j + \sum_{J_i^A \in \mathcal{J}_2} p_i^A$. Thus, since Alice's bound is also met by σ, it must be that $\sum_{J_i^A \in \mathcal{J}_2} p_i^A \leq Z$. Since the sum of all processing times of Alice's jobs is $2Z$, we get that $\sum_{J_i^A \in \mathcal{J}_1} p_i^A = \sum_{J_i^A \in \mathcal{J}_2} p_i^A = Z$, and so $S_1 = \{x_i : J_i^A \in \mathcal{J}_1\}$ and $S_2 = \{x_i : J_i^A \in \mathcal{J}_2\}$ is a solution to our Partition instance. ◀

4 Unary Encoded Weights and Processing Times

Agnetis *et al.* [1] showed that Two Agent Scheduling is strongly NP-hard, even when the input is given in unary. The unit-weight variant of Two Agent Scheduling, however, can be solved by a dynamic programming that is based on the SPT ordering of the jobs of both agents [2]. Roughly speaking, the algorithm computes a table $T[\cdot, \cdot, \cdot]$, where the entry $T[C, i, j]$ stores the minimum total completion time of Bob's jobs when $\{J_1^A, \ldots, J_i^A\} \cup \{J_1^B, \ldots, J_j^B\}$ are scheduled together and Alice's total completion time is at most C. Note that this works because the SPT order of Alice's and Bob's jobs are preserved.

When jobs have arbitrary weights the SPT rule no longer applies, and we do not know the relative order of the jobs of each agent in advance. Nevertheless, we can easily extend the algorithm above to an algorithm which is fixed-parameter in k when only Bob's jobs have arbitrary weights, by guessing the relative order of his jobs. For unary encoding, this gives a faster algorithm than the algorithm proposed in Theorem 1. However, when the jobs of both Alice and Bob have arbitrary weights, this strategy fails. In the remainder of the section we present a more elaborate dynamic program that will give the following:

▶ **Theorem 6.** Two Agent Scheduling *can be solved in* $O(n \cdot k! \cdot B(W_A P_A)^{k+1})$ *time, where* $W_A = \sum_{i=1}^{n} w_i^A$ *and* $P_A = \sum_{i=1}^{n} p_i^A$. *The algorithm is polynomial if Alice's jobs' weights and processing times, and Bob's bound are given in unary, and* k *is a fixed constant.*

In order to derive the result in Theorem 6 above, we start, as in Section 2, by fixing the sequence in which Bob's jobs are scheduled. We then renumber Bob's jobs according to this sequence such that J_i^B is scheduled before J_{i+1}^B for $i = 1, ..., k-1$, and are left with the following *feasibility subproblem*: can Alice's jobs be sequenced and the two ordered sets of jobs be interleaved such that the total weighted completion time of both Alice and Bob will not exceed the bounds A and B, respectively? Clearly, there is a feasible schedule to

our Two Agent Scheduling problem if and only if at least one of our $k!$ instances of the feasibility subproblem has a solution.

Let A_i be the set of Alice's jobs that are scheduled between jobs J_i^B and J_{i+1}^B for $i \in \{1, ..., k-1\}$, and let A_0 and A_k be the set of Alice's jobs that are scheduled before job J_1^B and after job J_k^B, respectively. The following lemma, easily proven by a simple pair-wise interchange argument, helps us to partially answer the sequencing part of the feasibility subproblem.

▶ **Lemma 7.** *If the answer to the feasibility problem is yes, then there is a schedule that yields a yes-answer where the jobs in each A_i are scheduled in a non-decreasing order of their processing time by weight ratio (that is, according to the* WSPT *rule).*

We next use the above lemma to construct a dynamic programming algorithm to answer the feasibility subproblem in $O(n \cdot B(W_A P_A)^{k+1})$ time. We start by renumbering Alice's jobs in a non-increasing order of p_i^A/w_i^A, such that $p_i^A/w_i^A \geq p_{i+1}^A/w_{i+1}^A$ for $i = 1, \ldots, n-1$.

Now, let $F_j[C, W_0, ..., W_k, P_0, ..., P_k]$ be the minimum weighted sum of completion time of Alice's jobs in a partial schedule that includes jobs $J_1^A, ..., J_j^A$ where W_i and P_i represents the total weight and processing time of Alice's jobs that are assigned to A_i, and C corresponds to the weighted sum of completion times of Bob's jobs.

Initially, we have that none of Alice's jobs are scheduled. Thus, $A_i = \emptyset$ for $i = 0, \ldots, k$, and the total weighted completion time of Bob's jobs is given by $C = \Sigma_{i=1}^k w_i^B \Sigma_{j=1}^i p_j^B$. Thus, the initial condition for our recursion is:

$$F_0[C, W_0, ..., W_k, P_0, ..., P_k] = \begin{cases} 0, & \text{if } C = \Sigma_{i=1}^k w_i^B \Sigma_{j=1}^i p_j^B \\ & \text{and } W_i = P_i = 0 \text{ for all } i, \\ \infty, & \text{otherwise.} \end{cases} \tag{3}$$

Consider now a state value $F_j[C, W_0, ..., W_k, P_0, ..., P_k]$ and assume that job J_j^A is assigned to A_i. Following Lemma 7, job J_j^A is scheduled first in A_i, which leads to an increase of p_j^A units of time in the completion time of Alice's jobs that have already been assigned to sets $A_i, A_{i+1}, ..., A_k$, and in the completion time of all of Bob's jobs J_j^B for $j = i+1, \ldots, k$. Given that J_j^A is completed at time $\Sigma_{l=0}^{i-1}(P_l + p_l^B) + p_j^A$, we obtain the following recursion:

$$\begin{aligned} F_j[C, W_0, \ldots, W_k, P_0, ..., P_k] = \\ \min_{i=0,\ldots,k} \{ & F_{j-1}[C - p_j^A \Sigma_{l=i+1}^k w_l^B, W_0', \ldots, W_k', P_0', ..., P_k'] \\ & + p_j^A \Sigma_{l=i}^k W_l + w_j^A \Sigma_{l=0}^{i-1}(P_l + p_l^B) \}, \end{aligned} \tag{4}$$

where $W_l' = W_l - w_j^A$ and $P_l' = P_l - p_j^A$ if $l = i$; and $W_l' = W_l$ and $P_l' = P_l$, otherwise.

Starting from the initial condition in (3), we compute $F_j[C, W_0, ...W_k, P_0, ..., P_k]$ by using (4), for any $j \in \{1, ..., n\}$, $W_i \in \{0, 1, \ldots, \Sigma_{l=1}^j w_l^A\}$, $P_i \in \{0, \ldots, \Sigma_{l=1}^j p_l^A)$, and $C \leq B$, where $i \in \{0, \ldots, k\}$. At the end of the dynamic programming procedure, we output a yes-answer for the feasibility subproblem if there exists a computed state with $F_n[C, W_0, ...W_k, P_0, \ldots, P_k] \leq A$. Otherwise, we output a no-answer.

Given that $W_i \in [0, \Sigma_{l=1}^j w_l^A]$, $W_i \in [0, \Sigma_{l=1}^j p_l^A]$, for $i = 0, \ldots, k$, and $C \leq B$, it follows that our dynamic program algorithm runs in $O(n \cdot B(W_A P_A)^{k+1})$ time. The fact that there are only $k!$ possible ways to sequence Bob's jobs yields the result in Theorem 6.

5 Bob Has Only a Few Job Types

In this section, we do not restrict the number k of Bob's jobs, but the number t of types for his jobs. Specifically, we consider Two Agent Scheduling when all jobs are of unit

weight and there is only a small number t of different processing times for Bob's jobs. Let B_i be the set of Bob's jobs of type i, that is, the set of Bob's jobs of the ith processing time, for $1 \leq i \leq t$.

We say that schedule σ_1 *dominates* schedule σ_2, if the sums of completion times for both Alice and Bob under σ_1 are smaller or equal to the sums of completion times for them under σ_2. A *Pareto-optimal point* (or, *Pareto-optimal schedule*) is a schedule which is not dominated by any other schedule. Crucially, if there is a feasible schedule to an instance of Two Agent Scheduling, then there is a feasible schedule to this instance which is a Pareto-optimal schedule.

The next lemma will be used to construct a polynomial-time algorithm for Two Agent Scheduling when all jobs are of unit weights and when t is a constant, based on iterating over a representative set of Pareto-optimal points.

We say that job J_j^A interleaves with B_i in schedule σ, if σ includes a subschedule of type $\{b_1, J_j^A, b_2\}$, where $b_1, b_2 \in B_i$ are non-empty sets (that is, the job J_j^A is scheduled before each job of b_2 and after each job of b_1).

▶ **Lemma 8.** *For any Pareto-optimal point, there exists a Pareto-optimal schedule in which, for each $1 \leq i \leq t$, at most a single job of Alice interleaves with each B_i.*

Proof. Consider a Pareto-optimal schedule σ that includes, for some $1 \leq i \leq t$, at least two jobs of Alice, J_j^A and J_{j+1}^A, both interleaving with B_i. Accordingly, σ includes a subschedule $\{b_1, J_j^A, b_2, J_{j+1}^A, b_3\}$, where $b_1, b_2, b_3 \in B_i$ are non-empty sets. Construct an alternative schedule σ' as follows: move J_j^A, $\min\{|b_1|, |b_3|\}$ positions to the left and J_{j+1}^A, $\min\{|b_1|, |b_3|\}$ positions to the right. Since all jobs in B_i have the same processing time, the sum of completion times of Alice's jobs remains the same. Moreover, the sum of completion times of Bob's jobs decreases by $\min\{|b_1|, |b_3|\} \cdot \left(p_{j+1}^A - p_j^A\right)$. The lemma now follows from the fact that the jobs are numbered according to the SPT rule and that in σ' at least one job out of the pair $\{J_j^A, J_{j+1}^A\}$ does not interleave with B_i. ◀

Following Lemma 8 above, we can construct a set of Pareto-optimal points which represents all Pareto-optimal points (in the sense that we have a representative for each equivalence class of the Pareto-optimal points, where two Pareto-optimal points are in the same equivalence class if the sums of completion times for both Alice and Bob are the same in both schedules) by constructing the entire set of schedules in which, for each $1 \leq i \leq t$, at most a single job of Alice interleaves with each B_i. We call any schedule of this type a *normal* schedule, and note that any such schedule can be concisely described as: "$a_1 b_1 a_{1^*} b_{1^*} a_2 b_2 a_{2^*} b_{2^*} \cdots a_t b_t a_{t^*} b_{t^*} a_{\text{rest}}$", where $a_{\text{rest}} + \sum_{i=1}^{t}(a_i + a_{i^*}) = n$, $0 \leq a_{i^*} \leq 1$ for each $1 \leq i \leq t$, and $b_i + b_{i^*} = |B_i|$ for each $1 \leq i \leq t$, with the intended meaning that a_1 of Alice's jobs are scheduled first, then b_1 of Bob's jobs of the first type, then a_{1^*} of Alice's jobs, then b_{1^*} of Bob's jobs of the first type, then a_2 of Alice's jobs, then b_2 of Bob's jobs of the second type, then a_{2^*} of Alice's jobs, then b_{2^*} of Bob's jobs of the second type, and so on, until, finally, a_{rest} of Alice's jobs are scheduled.

The number of normal schedules is upper-bounded by $k^t 2^t n^t$, since they are uniquely defined by all possible values of b_i, all possible (binary) values of a_{i^*}, and all possible values of a_i, for $1 \leq i \leq t$. Polynomial-time algorithm then follows by iteratively checking the feasibility all possible normal schedules.

6 Discussion

We would like to point out several directions for future research.

- While we determined the (parameterized) complexity of Two Agent Scheduling when the input is given in binary, our understanding of the complexity Two Agent Scheduling when the input is given in unary is lacking.
- It is natural to study Two Agent Scheduling when considering other objective functions. Similarly, it also makes sense to consider other, related, scheduling problems.
- Finally, we believe that our MILP formulation and the ideas underlying it, as described in the proof of Theorem 1, might be useful for other problems. This is done, to some extent, in [7, Theorem 1 and Theorem 2], and we would like to see other problems which have a similar structure that might be utilized in similar ways.

References

1 Alessandro Agnetis, Jean-Charles Billaut, Stanisław Gawiejnowicz, Dario Pacciarelli, and Ameur Soukhal. *Multiagent Scheduling: Models and Algorithms.* Springer, 2014.

2 Allesandro Agnetis, Pitu B. Mirchandani, Dario Pacciarelli, and Andrea Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, 2004.

3 Kenneth R. Baker and J. Cole Smith. A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6(1):7–16, 2003.

4 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. In *Proceedings of the 23th International Symposium on Algorithms and Computation (ISAAC'12)*, volume 7676 of *LNCS*, pages 247–256. Springer, 2012.

5 René van Bevern, Rolf Niedermeier, and Ondrej Suchý. A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *arXiv preprint arXiv:1005.4159*, 2015.

6 Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

7 Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Elections with few candidates: Prices, weights, and covering problems. In *Proceedings of the 4th International Conference on Algorithmic Decision Theory (ADT'15)*, pages 414–431. Springer, 2015.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Springer, 2013.

9 Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical computer science*, 298(2):317–324, 2003.

10 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer, 2006.

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

12 Magnús M. Halldórsson and Ragnar K. Karlsson. Strip graphs: Recognition and scheduling. In *Proceedings of the 32th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, volume 4271 of *LNCS*, pages 137–146. Springer, 2006.

13 Mikhail Y. Kovalyov, Ammar Oulamara, and Ameur Soukhal. Two-agent scheduling on an unbounded serial batching machine. In *Proceedings of the Second International Symposium on Combinatorial Optimization (ISCO'12)*, volume 3787 of *LNCS*, pages 427–438. Springer, 2012.

14 Kangbok Lee, Byung-Cheon Choi, Joseph Y.-T. Leung, and Michael L. Pinedo. Approximation algorithms for multi-agent scheduling to minimize total weighted completion time. *Information Processing Letters*, 109(16):913–917, 2009.

15 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

16 Joseph Y.-T. Leung, Michael Pinedo, and Guohua Wan. Competitive two-agent scheduling and its applications. *Operations Research*, 58(2):458–469, 2010.

17 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. In *Proceedings of the 17th International Conference on Integer Programming and Combinatorial Optimizationon (IPCO'14)*, volume 8494 of *LNCS*, pages 381–392. Springer, 2014.

18 Baruch Mor and Gur Mosheiov. Single machine batch scheduling with two competing agents to minimize total flowtime. *European Journal of Operational Research*, 215(3):524–531, 2011.

19 Rolf Niedermeier. Invitation to fixed-parameter algorithms. Habilitation thesis, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, September 2002.

20 Daniel Oron, Dvir Shabtay, and George Steiner. Single machine scheduling with two competing agents and equal job processing times. *European Journal of Operational Research*, 2015.

21 Paz Perez-Gonzalez and Jose M. Framinan. A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, 235(1):1–16, 2014.

22 Michael L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

23 Jinjiang Yuan, Weiping Shang, and Qi Feng. A note on the scheduling with two families of jobs. *Journal of Scheduling*, 8(6):537–542, 2005.