Shared Processor Scheduling of Multiprocessor Jobs

Dariusz Dereniowski[*]
Faculty of Electronics,
Telecommunications and Informatics,
Gdańsk University of Technology,
Gdańsk, Poland

Wiesław Kubiak
Faculty of Business Administration,
Memorial University,
St. John's, Canada

April 27, 2022

Abstract

We study shared processor scheduling of *multiprocessor* weighted jobs where each job can be executed on its private processor and simultaneously on possibly *many* processors shared by all jobs in order to reduce their completion times due to processing time overlap. Each of m shared processors may charge different fee but otherwise the processors are identical. The total weighted overlap of all jobs is to be maximized. This problem is key to subcontractor scheduling in extended enterprises and supply chains, and divisible load scheduling in computing. We prove that, quite surprisingly, *synchronized* schedules that complete each job using shared processors at the same time on its private and shared processors include optimal schedules. We show that optimal α*-private* schedules that require each job to use its private processor for at least $\alpha = 1/2 + 1/(4(m+1))$ of the time required by the job guarantee more than an α fraction of the total weighted overlap of the optimal schedules. This gives an α-approximation algorithm that runs in strongly polynomial time for the problem, and improves the $1/2$-approximation reported recently in the literature to $5/8$-approximation for a single shared processor problem. The computational complexity of the problem, both single and multi-shared processor, remains open. We show however an LP-based optimal algorithm for *antithetical* instances where for any pair of jobs j and i, if the processing time of j is smaller than or equal to the processing time of i, then the weight of j is greater than or equal to the weight of i.

Keywords: discrete optimization, subcontracting, supply chains, extended enterprises, shared processors

1 Introduction

Quick-response industries are characterized by volatile demand and inflexible capacities. The agents (companies) in such industries need to supplement their *private* capacity by adapting their extended enterprises and supply chains to include subcontractors with their own capacity. This capacity of subcontractors however is often *shared* between other independent supply chains which can cause undesirable and difficult to control bottlenecks in those supply chains. A well-documented real-life example of this issue has been reported in Boeing's Dreamliner supply chain where the overloaded schedules of subcontractors, each working with multiple suppliers, resulted in long delays in the overall production due dates, see Vairaktarakis [11].

[*]Corresponding author. Email: deren@eti.pg.edu.pl

The use of subcontractor's shared processor (capacity) benefits an agent only if it can reduce the agent's job (order) completion time at a competitive enough cost. Hence, the subcontractor's shared processor should never be used, and paid for by the agent, as long as the agent's private processor remains available. We reasonably assume that the cost of using subcontractor's processor is higher than this of the agent's private processor. Moreover the agent's private processor should never remain idle as long as the agent's job remains unfinished. Therefore, only a simultaneous execution, or *overlap*, on both private and shared processors reduces completion time. The total (weighted) overlap is the objective function studied in this paper. This objective function is closely related to the total completion time objective traditionally used in scheduling. The total completion time can be reduced by an increase of the total overlap resulting from the simultaneous execution of jobs on private and shared processors. However, we need to emphasize that the two objectives exist for different practical reasons. The minimization of total completion time minimizes mean flow time and thus by Little's Law minimizes average inventory in the system. The maximization of the total overlap on the other hand maximizes the total net payoff resulting from completing jobs earlier thanks to the use of shared processors (subcontractors). This different focus sets the total overlap objective apart from the total completion time objective, and makes it a key objective in scheduling shared processors, Dereniowski and Kubiak [5].

The reduction of completion time of a job due to the overlap depends on whether only a single shared processor or multiple shared processors can be used simultaneously by the job. For instance, an order of size 12 can be completed in 6 units of time (assuming it takes one unit of time to complete the order of size one) at the earliest if only a single shared processor is allowed to process the order simultaneously with private processor. The order is then split in half between the private and the shared processor both working simultaneously on the job in the time interval $(0, 6)$. The resulting overlap equals 6, and the job is not executed by any other shared processor in the interval. This constraint has been imposed in the literature thus far, see Vairaktarakis and Aydinliyim [12], Hezarkhani and Kubiak [8], and Dereniowski and Kubiak [6], [5]. We refer to the constraint as a *single processor* (*SP*) job mode. This paper relaxes the constraint and permits a job to be processed simultaneously on its private and possibly *more* than one shared processor. For instance, the job of size 12 can be completed in 4 units of time by executing it in the interval $(0, 4)$ on its private processor and simultaneously in the intervals $(0, 4)$, $(0, 3)$ and $(1, 2)$ on three different shared processors. The resulting total overlap equals 8. We refer to this relaxation as a *multiprocessor* (*MP*) job mode. To our knowledge this mode of execution of jobs has been first studied by Blazewicz, Drabowski and Weglarz [4] and refereed to as multiprocessor jobs in the literature. The multiprocessor jobs gained prominence in distributed computing where the processing by the nodes of a shared network of processors as well as possible communications between the nodes overlap in time so that the completion time (makespan) for the whole job (referred to as divisible load) is shorter than the processing of the whole load by a single node, Bharadwaj, Ghose, and Robertazzi [3]. Bharadwaj, Ghose, and Robertazzi [3] and Drozdowski [7] survey many real-life applications that satisfy the divisibility property.

The shared processors may also charge different fees, c_i, and the jobs may have different weights w_j. Then, the contribution to the total payoff of a job piece of length l is l times the difference between its weight and the shared processors' fee. Thus, the execution in the interval $(0, 4)$ on one shared processor may cost the same as the execution in the interval $(1, 2)$ on another shared processor if the latter is four times more expensive than the former. The shared processors with different costs will studied in this paper. Figure 1 illustrates the difference between the two modes, SP and MP.

It is quite remarkable that regardless of the mode of job execution on shared processors, and job weights there always exist optimal schedules that are *synchronized*, i.e., each agent using shared processors has its job completed on private and shared processors at the same time (for a formal definition of synchronized

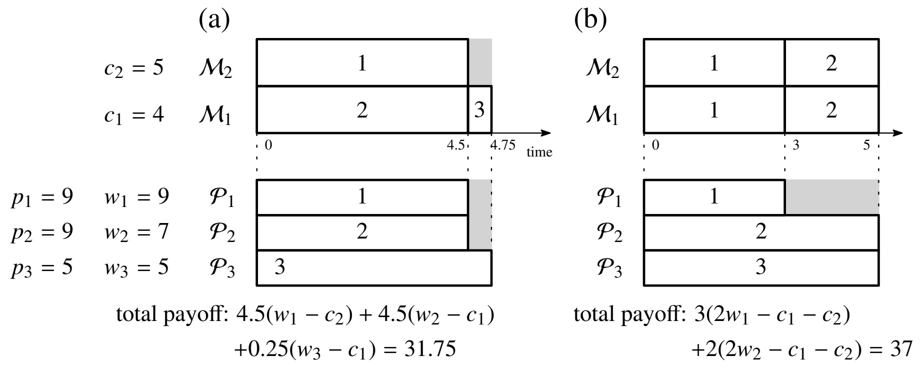


Figure 1: The example illustrates that allowing jobs to be executed in *MP* mode simultaneously on several shared processors ($\mathcal{M}_1$ and $\mathcal{M}_2$) may be beneficial for some problem instances: (a) an optimal (synchronized) schedule for the *SP* mode; (b) an optimal (synchronized) schedule for the *MP* mode. In this input instance, the shared processors' fees are $c_1 = 4$ and $c_2 = 5$, the jobs' processing times are $p_1 = p_2 = 9$, $p_3 = 5$ and their weights are $w_1 = 9$, $w_2 = 7$, $w_3 = 5$.

schedules for the *MP* mode see Section 1.3). This has been shown for the *SP* job mode by Vairaktarakis and Aydinliyim [12], Hezarkhani and Kubiak [8], and Dereniowski and Kubiak [5], and [6]. In this paper we show it for the *MP* job mode. We return to the paper outline later in the introduction in Section 1.3 to give more details.

1.1 Related Work and Applications

The shared processor scheduling problem with a *single* shared processor has been studied by Vairaktarakis and Aydinliyim [12], Hezarkhani and Kubiak [8], and Dereniowski and Kubiak [6]. Vairaktarakis and Aydinliyim [12] consider the *unweighted* problem with each job allowed to use at most one time interval on the shared processor. This case is sometimes referred to as *non-preemptive* since jobs are not allowed preemption on the shared processor. [12] proves that there are optimal schedules that complete job execution on its private and the shared processor at the same time, we call such schedules *synchronized*. It further shows that this guarantees that sequencing jobs in non-decreasing order of their processing times leads to an optimal solution for the case. We refer to such schedules as *processing time ordered*, see [6]. Interestingly, the processing time ordered schedules guarantee that each job uses exactly one nonempty time interval on the shared processor. [8] observes that the processing time ordered schedules also give optimal solutions to the *preemptive* unweighted problem, where more than one interval can be used by a job on the shared processor. [6] considers the *weighted* problem. It observes that for the weighted problem it no longer holds that each job occupies a non-empty interval on the shared processor in optimal schedules, there may exist jobs processed on their private processors only in each optimal schedule. It shows that there always exist optimal schedules that are synchronized, gives a $\frac{1}{2}$- approximation algorithm for the problem, and shows that the $\frac{1}{2}$ bound for the algorithm is tight. It also extends earlier result for the unweighted problem by proving that the processing time ordered schedules are optimal for antithetical instances, i.e., the ones for which there exists ordering of jobs that is simultaneously non-decreasing with respect to processing times and non-increasing with respect to the weights. The complexity status of the weighted problem with a single shared processor remains open.

Vairaktarakis and Aydinliyim [12], Vairaktarakis [11], and Hezarkhani and Kubiak [8] focus on the *tension* between the agents and the subcontractor in the decentralized system where each agent strives to complete its job as early as possible and needs to compete with other agents for the shared processor, and the subcontractor who strives to have the shared processor occupied as long as possible to maximize its payoff. The tension calls for coordinating mechanisms to ensure the efficiency. Hezarkhani and Kubiak [8] show such coordination mechanism for the unweighted problem, and give examples to prove that such mechanisms do not exist for the problem with weighted jobs.

Dereniowski and Kubiak [5] consider shared *multi-processor* problem. They however, contrary to this paper, assume that each job can only be processed by its private processor and at most *one* out of many shared processors, the *SP* mode. Besides no distinction is made between the shared processors, in particular the costs of using shared processors are the same for all of them. [5] proves that synchronized optimal schedules always exist for weighted multi-processor instances. However, the wighted problem is NP-hard in the strong sense. For the multi-processor problem with equal weights for all jobs, [5] gives an efficient, polynomial-time algorithm running in time $O(n \log n)$.

The motivation to study the shared processor scheduling problem comes from diverse applications. Vairaktarakis and Aydinliyim [12], [2] consider it in the context of supply chains and extended enterprises where subcontracting allows jobs to reduce their completion times by using a shared subcontractor's processor. Bharadwaj et. al. [3], and Drozdowski [7] use the divisible load scheduling to reduce a job completion time in parallel and distributed computer systems, and Anderson [1] argues for using batches of potentially infinitely small items that can be processed independently of other items of the batch in scheduling job-shops. We refer the reader to Dereniowski and Kubiak [5] for more details on these applications.

1.2 Problem Formulation

We are given a set $\mathcal{J}$ of n preemptive jobs with non-negative processing times p_j and weights w_j, $j \in \mathcal{J}$. With each job $j \in \mathcal{J}$ we associate its *private* processor denoted by $\mathcal{P}_j$. Moreover, $m \geq 1$ *shared* processors $\mathcal{M}_1, \ldots, \mathcal{M}_m$ are available for all jobs; processor $\mathcal{M}_i$ has *cost* c_i, $i \in \{1, \ldots, m\}$. Without loss of generality we always assume $c_1 \leq \cdots \leq c_m$ in this paper.

A schedule $\mathcal{S}$ selects for each job $j \in \mathcal{J}$:

(i) a (possibly empty) collection of open and pairwise disjoint maximal time intervals $I_{i,j}^{1}, \ldots, I_{i,j}^{l(i,j)}$ in which the job j executes on shared processor $\mathcal{M}_i$ for each $i \in \{1, \ldots, m\}$, any $I_{i,j}^{k}$ is called a *piece* of job j on processor i or simply *piece* of job j if the processor is obvious from the context, and

(ii) a *single* time interval $(0, C_{\mathcal{S}}^{\mathcal{P}}(j))$ in which j executes on its private processor $\mathcal{P}_j$.

In a *feasible* schedule, the total length of all these intervals (the ones in (i) and the one in (ii)) equals p_j:

$$p_j = C_{\mathcal{S}}^{\mathcal{P}}(j) + \sum_{i=1}^{m} \sum_{k=1}^{l(i,j)} \left| I_{i,j}^{k} \right| \tag{1}$$

for each $j \in \mathcal{J}$. Moreover, each shared processor i can execute at most one job at a time, i.e. the unions of all pieces of different jobs j and j' on processor i are disjoint for each shared processor $\mathcal{M}_i$, $i \in \{1, \ldots, m\}$, or formally

$$\left(\bigcup_{k=1}^{l(i,j)} I_{i,j}^{k} \right) \cap \left(\bigcup_{k'=1}^{l(i,j')} I_{i,j'}^{k'} \right) = \emptyset, \tag{2}$$

for any two different jobs j and j' and any shared processor $\mathcal{M}_i$, $i \in \{1,\ldots,m\}$. Without loss of generality we also assume that

$$I^k_{i,j} \subseteq (0, C^{\mathcal{P}}_{\mathcal{S}}(j)) \tag{3}$$

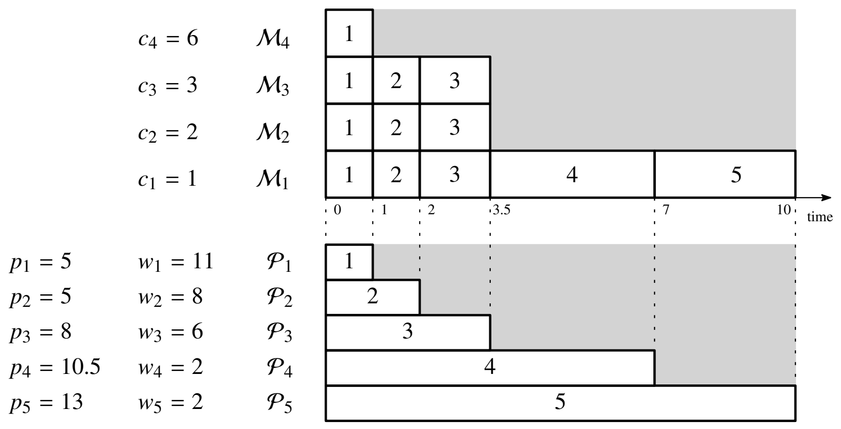
for each i, j, and k in a feasible schedule.

We re-emphasize that, contrary to earlier literature on shared multi-processor scheduling [5], we allow the pieces of the same job *not* to be disjoint (or simply to overlap) on different shared processors like in the *MP* job mode introduced in [4] and used in [3] and [7] for instance. Observe that the private processor $\mathcal{P}_j$ can only execute job j but none of the other jobs.

Given a feasible schedule $\mathcal{S}$, for each job $j \in \mathcal{J}$ we call any pair $(\mathcal{M}_i, I)$ an *overlap of* j *on* $\mathcal{M}_i$ if I is a time interval of maximum length where j executes on both its private processor $\mathcal{P}_j$ and the shared processor $\mathcal{M}_i$ simultaneously; we say that $|I|$ is the *length* of the overlap $(\mathcal{M}_i, I)$. Note that $I \subseteq (0, C^{\mathcal{P}}_{\mathcal{S}}(j))$. Then, the *total overlap* $\texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i)$ of j on $\mathcal{M}_i$ equals the sum of lengths of all overlaps of j on $\mathcal{M}_i$. The *total weighted overlap* of $\mathcal{S}$ equals

$$\Sigma(\mathcal{S}) = \sum_{i=1}^{m} \sum_{j \in \mathcal{J}} \texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i)(w_j - c_i). \tag{4}$$

To illustrate we give an example in Figure 2. The example also gives some intuitions as to how optimal schedules look like — for more details see a discussion at the end of the next section.



$c_4 = 6$
$\mathcal{M}_4$
$c_3 = 3$
$\mathcal{M}_3$
$c_2 = 2$
$\mathcal{M}_2$
$c_1 = 1$
$\mathcal{M}_1$
1
2
3
4
5
0
1
2
3.5
7
10
time
$p_1 = 5$ $w_1 = 11$ $\mathcal{P}_1$
$p_2 = 5$ $w_2 = 8$ $\mathcal{P}_2$
$p_3 = 8$ $w_3 = 6$ $\mathcal{P}_3$
$p_4 = 10.5$ $w_4 = 2$ $\mathcal{P}_4$
$p_5 = 13$ $w_5 = 2$ $\mathcal{P}_5$

$$X_1 = \textstyle\sum_{i=1}^{4} 1 \cdot (w_1 - c_i)$$
$$= 5 + 8 + 9 + 10 = 32$$
$$X_2 = \textstyle\sum_{i=1}^{3} 1 \cdot (w_2 - c_i)$$
$$= 5 + 6 + 7 = 18$$
$$X_3 = \textstyle\sum_{i=1}^{3} 1.5 \cdot (w_3 - c_i)$$
$$= 1.5 \cdot (3 + 4 + 5) = 18$$
$$X_4 = 3.5 \cdot (2 - 1) = 3.5$$
$$X_5 = 3$$
$$\Sigma(\mathcal{S}) = 74.5$$

Figure 2: An optimal schedule for an input instance with $\mathcal{J} = \{1,\ldots,5\}$ and 4 shared processors $\mathcal{M}_1,\ldots,\mathcal{M}_4$. Here X_j denotes the contribution of a job j to the total weighted overlap of the schedule, $X_j = \sum_{i=1}^{m} \texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i)(w_j - c_i)$

1.3 Outline

The main structural property of optimal schedules proved in this paper is schedule synchronization. We say that a feasible schedule $\mathcal{S}$ is *synchronized* if there exists a subset of jobs $\{j_1,\ldots,j_k\} \subseteq \mathcal{J}$ such that:

- each job $j \notin \{j_1,\ldots,j_k\}$ executes only on its private processor $\mathcal{P}_j$ in $(0, p_j)$ in $\mathcal{S}$,
- there exist $m \geq m_1 \geq m_2 \geq \cdots \geq m_k \geq 1$ and $0 = t_0 \leq t_1 \leq \cdots \leq t_k$ such that job j_i, $i \in \{1,\ldots,k\}$, executes non-preemptively in time interval (t_{i-1}, t_i) on each shared processor $\mathcal{M}_l$, $l \in \{1,\ldots,m_i\}$, and in $(0, t_i)$ on $\mathcal{P}_{j_k}$ in $\mathcal{S}$.

Figure 2 gives an example of a synchronized schedule of five jobs where $m_1 = 4$, $m_2 = m_3 = 3$, and $m_4 = m_5 = 1$, and $t_1 = 1$, $t_2 = 2$, $t_3 = 3.5$, $t_4 = 7$, and $t_5 = 10$. Observe that the total weighted overlap equals

$$\Sigma(\mathcal{S}) = \sum_{i=1}^{k}(t_i - t_{i-1})\left(m_i w_{j_i} - \sum_{l=1}^{m_i} c_l\right),$$

for a synchronized $\mathcal{S}$, and by the feasibility of $\mathcal{S}$

$$t_i + m_i(t_i - t_{i-1}) = p_{j_i}.$$

From these two we get a natural interpretation of the objective function for synchronized $\mathcal{S}$ which is summarized in the following formula

$$\Sigma(\mathcal{S}) = \sum_{i=1}^{k}(p_{j_i} - t_i)\left(w_{j_i} - \frac{\sum_{l=1}^{m_i} c_l}{m_i}\right),$$

where $p_{j_i} - t_i$ is the *total* time j_i is executed on the m_i cheapest shared processors $\mathcal{M}_1, \dots, \mathcal{M}_{m_i}$, and the $(\sum_{l=1}^{m_i} c_l)/m_i$ is the *average* cost of execution on these processors which must not exceed the weight, w_{j_i}, of the job j_i if $\mathcal{S}$ is to be optimal. Our main structural result in this paper is as follows.

Theorem 1.1. *There always exists an optimal schedule that is synchronized.*

The theorem follows immediately from Corollary 4.2 and Lemma 6.8 in Section 6. Though our algorithmic results, both optimization and approximation, do not require the schedule synchronization property directly, the property comes useful to prove some key properties of the algorithms. Therefore we start with synchronization here but leave technical details of its proof for Section 6.

The existence of optimal schedules that are synchronized is consistent with earlier results [12, 5, 6, 8] for the *SP* job mode. However, the proof for the *MP* job mode turns out to be more challenging. We note that both *SP* and *MP* modes are equivalent in the single processor case, $m = 1$.

The computational complexity status of the problem remains a challenging open question. However, we show in Section 2 that for any given permutation of job completions on private processors, recall that by definition there is exactly one job on any private processor, an LP can be formulated that finds a schedule maximizing the total weighted overlap among all schedules that respect this permutation. This result points at a certain strategy in solving the problem which is to search for the duration each job must be executed on its private processor. The difficulty in establishing computational complexity status for the problem however indicates that the optimal durations will be also difficult to find. We therefore propose to *relax* the strategy by requiring that each duration be at *least* $\alpha > 1/2$ fraction of job processing time, i.e., we require that the completion of a job j is limited to occur in the interval $[\alpha p_j, p_j]$. We naturally refer to such schedules as *α-private* since they require that at least α fraction of each job is executed of its private processor. The optimal α-private schedules can be found in strongly polynomial time by solving a minimum-cost network flow problem, see Orlin [9] for a strongly polynomial algorithm for the minimum-cost network flow problem. These schedules are important since they guarantee that their total weighted overlaps are *not less* than α of the maximum total weighted overlaps which results in a α-approximation strongly polynomial time algorithm for the problem. This is shown in Section 3, where we also show that the best α we can find in this paper is $\frac{1}{2} + \frac{1}{4(m+1)}$. The α-approximation algorithm improves a $1/2$-approximation algorithm for a single processor [6] to a $5/8$-approximation algorithm for $m = 1$.

Section 5 considers *antithetical* instances where $p_j \geq p_{j'}$ implies $w_j \leq w_{j'}$ for each pair of jobs j and j'. We prove that a permutation of job completions on private processors that coincides with non-decreasing order of processing times (or equivalently with non-increasing order of weights) is optimal for the antithetical instances. The LP of Section 2 then finds an optimal schedule for the permutation which gives a polynomial time algorithm for the antithetical instances. Note that even though the permutation of job completions on private processors is fixed, the LP still needs to optimally decide how many shared processors should a job use. In particular, generally using more of them may decrease the job's contribution to the total weighted overlap of the schedule since the job uses more costly shared processors, on the other hand using more shared processors may reduce the job completion time thus allowing jobs that follow it in the permutation to start earlier and thus to contribute proportionally more to the total weighted overlap of the schedule. Figure 2 illustrates this tradeoff for an antithetical instance, for example job 1 executed on all 4 shared processors finishes at $t_1 = 1$ and contributes 32 to the total weighted overlap, the same job executed on the cheapest three shared processors would finish later at $t_1 = \frac{5}{4}$ however it would contribute $33\frac{3}{4}$ to the total weighted overlap. The optimal solution for the instance choses the former which speeds up starting time of the remaining four jobs by $\frac{1}{4}$ in comparison to the latter. Generally this tradeoff is optimally handled by the LP, however it remains open whether an optimal schedule can be found by a more efficient or strongly polynomial time algorithm different from LP.

2 An LP Formulation

In this section we give an LP formulation that takes as an input an instance of the problem with n jobs $\mathcal{J} = \{1, \dots, n\}$ having weights $w\colon \mathcal{J} \to \mathbb{R}_+$, processing times $p\colon \mathcal{J} \to \mathbb{R}_+$, m shared processors with costs $c_1, \dots, c_m$, and a permutation $A = (1, \dots, n)$ of the jobs in $\mathcal{J}$. The permutation A provides an order according to which the jobs finish on their private processors. The LP finds a schedule that maximizes the total weighted overlap among all A-compatible schedules. We say that, for a permutation of jobs $A = (1, \dots, n)$, a schedule $\mathcal{S}$ is *A-compatible* if $C^{\mathcal{P}}_{\mathcal{S}}(j) \leq C^{\mathcal{P}}_{\mathcal{S}}(j+1)$ for each $j \in \{1, \dots, n-1\}$.

The variables used in the LP are as follows. For each $i \in \{1, \dots, n\}$, a variable t_j is the completion time of job j on its private processor. For each $j \in \{1, \dots, n\}$, $k \in \{1, \dots, j\}$ and $i \in \{1, \dots, m\}$, a variable x_{jik} is the total amount of job j executed on the shared processor $\mathcal{M}_i$ in the time interval (t_{k-1}, t_k).

The LP is as follows:

$$\text{maximize} \quad f = \sum_{j=1}^{n} \sum_{i=1}^{m} \sum_{k=1}^{j} (w_j - c_i) x_{jik} \tag{5}$$

subject to:

$$t_0 = 0 \leq t_1 \leq \dots \leq t_n, \tag{6}$$

$$\sum_{j=k}^{n} x_{jik} \leq t_k - t_{k-1}, \quad i \in \{1, \dots, m\}, k \in \{1, \dots, n\}, \tag{7}$$

$$\sum_{i=1}^{m} \sum_{k=1}^{j} x_{jik} = p_j - t_j, \quad j \in \{1, \dots, n\}, \tag{8}$$

$$x_{jik} \geq 0, \quad j \in \{1, \dots, n\}, k \in \{1, \dots, j\}, i \in \{1, \dots, m\}. \tag{9}$$

For a solution to the LP, we define the following *corresponding* schedule $\mathcal{S}$. For each job $j \in \{1, \dots, n\}$, let $C^{\mathcal{P}}_{\mathcal{S}}(j) = t_j$. For each $k \in \{1, \dots, j\}$ and $i \in \{1, \dots, m\}$, execute a piece of job j of duration x_{jik} on shared processor $\mathcal{M}_i$ in time interval (t_{k-1}, t_k). These job pieces are executed in (t_{k-1}, t_k) on $\mathcal{M}_i$ so that there is no overlap between them.

Lemma 2.1. *For each feasible solution to LP the corresponding schedule $\mathcal{S}$ is feasible, A-compatible and such that $f = \Sigma(\mathcal{S})$, and for each feasible and A-compatible schedule $\mathcal{S}$ there is a feasible solution to LP such that $\Sigma(\mathcal{S}) = f$.*

Proof. Let x_{jik}, $j \in \{1, \ldots, n\}$, $k \in \{1, \ldots, j\}$, $i \in \{1, \ldots, m\}$ and t_k, $k \in \{1, \ldots, n\}$ be a solution to the LP. We first prove that a corresponding schedule $\mathcal{S}$ is feasible and A-compatible. For each $k \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, m\}$, executing a piece of job j of duration x_{jik} on shared processor $\mathcal{M}_i$ in time interval (t_{k-1}, t_k) is feasible since (7) ensures that the duration x_{jik} does not exceed the length of the interval. Also by (7) and by (9), the length of the interval (t_{k-1}, t_k) is sufficient to execute all job pieces of length x_{jik}, $j \in \{1, \ldots, n\}$, on each shared processor $\mathcal{M}_i$. Hence (2) is satisfied by $\mathcal{S}$. By (8), the total execution time of all pieces of a job j on all shared processors equals $p_j - t_j$, for each job $j \in \{1, \ldots, n\}$. Since $C^{\mathcal{P}}_{\mathcal{S}}(j) = t_j$ in $\mathcal{S}$, we obtain that the total length of all pieces of j in $\mathcal{S}$ equals $C^{\mathcal{P}}_{\mathcal{S}}(j) + p_j - t_j = p_j$ as required by (1). This proves that $\mathcal{S}$ is feasible and (6) implies that it is A-compatible. Finally f in (5) equals the total weighted overlap in (4) since it can be readily verified that $\sum_{k=1}^{j} x_{jik} = \mathtt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i)$ for each $i \in \{1, \ldots, m\}$.

Now for an A-compatible feasible schedule $\mathcal{S}$, set $t_j = C^{\mathcal{P}}_{\mathcal{S}}(j)$ for each $j \in \mathcal{J}$ and set x_{jik} to be the total execution time of job $j \in \mathcal{J}$ on shared processor $\mathcal{M}_i$, $i \in \{1, \ldots, m\}$, in time interval (t_{k-1}, t_k), $k \in \{1, \ldots, n\}$, where $t_0 = 0$. Since $\mathcal{S}$ is A-compatible, (6) is satisfied. The constraint (7) is satisfied since (2) holds in a feasible $\mathcal{S}$. The constraint (8) is satisfied since by (1) the total execution time of each job in a feasible $\mathcal{S}$ equals its processing time. Finally, (9) follows directly from the definition of x_{jik}'s. Thus the solution is a feasible solution to LP, and the total weighted overlap of $\mathcal{S}$ equals f in (5). □

Theorem 2.2. *Given a permutation A of jobs, a feasible and A-compatible schedule that maximizes the total weighted overlap can be computed in polynomial time.*

Proof. By Lemma 2.1, the schedule $\mathcal{S}$ corresponding to an optimal solution to LP is feasible and A-compatible, and it maximizes the total weighted overlap. The optimal solution to LP can be found in polynomial time, see for instance [10]. □

3 Approximation Algorithm

In this section we show a strongly polynomial α-approximation algorithm for the problem. The idea is to find a natural class of schedules for each instance of the problem such that optimal schedules in the class can be found in strongly polynomial time and such that those optimal schedules guarantee the required approximation α.

To that end we limit ourselves to *α-private* schedules in this section. In an α-private schedule each job j executes for at least αp_j time on its private processor $\mathcal{P}_j$, and in time interval $(0, \alpha p_j)$ only on shared processors. Although the optimal order of completion times of jobs on their *private* processors in α-private schedules is still difficult to find, and thus the LP from Section 2 may not be used to find an optimal α-private, we use another key property of those schedules which is that each job j completes by αp_j on all *shared* processors in formulating another linear program, we call it LA, to find optimal α-private schedules in this section. Therefore, each job j must complete by αp_j, $j \in \{1, \ldots, n\}$, on all shared processors in an α-private schedule, and thus the order of these *limiting* time points is clearly the same as the order of job processing times $p_1 \leq \cdots \leq p_n$. The completion time of j on its private processor $\mathcal{P}_j$ will be set to

$C^{\mathcal{P}}_{\mathcal{S}}(j) = (1-\alpha)p_j + \tilde{t}_j$ in α-private schedules, where $\tilde{t}_j \geq 0$ is referred to as the *remainder* of j. The choice of α needs to guarantee that $\alpha p_j \leq (1-\alpha)p_j + \tilde{t}_j$ so that each piece of job j on shared processors counts for an overlap in an α-private schedule. This inequality imposes an upper bound on α. On the other hand we wish α to be as large as possible, in particular greater than a half, to guarantee as good as possible an approximation offered by α-private schedules. This imposes a lower bound on α. The compromise used in our LA is $\alpha = \frac{2m+3}{4(m+1)} = \frac{1}{2} + \frac{1}{4(m+1)}$. It remains open whether a higher value of α that meets both conditions can be found. We are now ready to show that the LA with this alpha finds an optimal α-private schedule, i.e. an α-private schedule that maximizes the total weighted overlap among all α-private schedules. The variables used in the LA are as follows. For each $j \in \{1,\dots,n\}$, a variable $\tilde{t}_j$ is the reminder of job j to be executed on its private processor $\mathcal{P}_j$. For each $j \in \{1,\dots,n\}$, $k \in \{1,\dots,j\}$ and $i \in \{1,\dots,m\}$, a variable x_{jik} is the total amount of job j to be executed on the shared processor $\mathcal{M}_i$ in the time interval $(\alpha p_{k-1}, \alpha p_k)$. We take $p_0 = 0$, and assume the order $(1,\dots,n)$, $p_1 \leq \cdots \leq p_n$ in the program.

The LA is as follows:

$$\text{maximize} \quad \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{k=1}^{j}(w_j - c_i)x_{jik} \tag{10}$$

subject to:

$$\frac{p_j}{2(m+1)} \leq \tilde{t}_j \leq \alpha p_j, \quad j \in \{1,\dots,n\}, \tag{11}$$

$$\sum_{j=k}^{n} x_{jik} \leq \alpha(p_k - p_{k-1}), \quad i \in \{1,\dots,m\}, k \in \{1,\dots,n\}, \tag{12}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{j} x_{jik} = \alpha p_j - \tilde{t}_j, \quad j \in \{1,\dots,n\}, \tag{13}$$

$$x_{jik} \geq 0, \quad j \in \{1,\dots,n\}, k \in \{1,\dots,j\}, i \in \{1,\dots,m\}. \tag{14}$$

For a feasible solution to the LA, we define the following *corresponding* schedule $\mathcal{S}$. For each job $j \in \{1,\dots,n\}$, set the completion time of j on its private processor $\mathcal{P}_j$ to

$$C^{\mathcal{P}}_{\mathcal{S}}(j) = (1-\alpha)p_j + \tilde{t}_j. \tag{15}$$

For each $k \in \{1,\dots,n\}$ and $i \in \{1,\dots,m\}$, execute a piece of job j of duration x_{jik} on shared processor $\mathcal{M}_i$ in time interval $(\alpha p_{k-1}, \alpha p_k)$ in such a way that no two job pieces overlap. We now prove that $\mathcal{S}$ is feasible.

Lemma 3.1. *For a feasible solution to LA, the corresponding schedule $\mathcal{S}$ is feasible, and the value of objective function of the solution equals the total weighted overlap of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ be a schedule corresponding to a solution x_{jik}, $j \in \{1,\dots,n\}$, $k \in \{1,\dots,j\}$, $i \in \{1,\dots,m\}$ and $\tilde{t}_j$, $j \in \{1,\dots,n\}$. For each $k \in \{1,\dots,j\}$ and $i \in \{1,\dots,m\}$, executing a piece of job j of duration x_{jik} on shared processor $\mathcal{M}_i$ in time interval $(\alpha p_{k-1}, \alpha p_k)$ is feasible since (12) guarantees that the duration x_{jik} does not exceed the length of the interval, and by (14) the duration x_{jik} is non-negative. Moreover, again by (12) and (14), the length of the interval $(\alpha p_{k-1}, \alpha p_k)$ is sufficient to execute all pieces of jobs of length x_{jik}, $j \in \{1,\dots,n\}$, on each shared processor $\mathcal{M}_i$. By (13), the total execution time of all pieces of j on all shared processors equals $\alpha p_j - \tilde{t}_j$, for each job $j \in \{1,\dots,n\}$. Since $C^{\mathcal{P}}_{\mathcal{S}}(j) = (1-\alpha)p_j + \tilde{t}_j$, we obtain that the total length of all pieces of j in $\mathcal{S}$ equals p_j as required. Moreover, by (12) and (13) all the pieces of job j that execute on shared processors end by αp_j, and thus they end by $C^{\mathcal{P}}_{\mathcal{S}}(j) = (1-\alpha)p_j + \tilde{t}_j$ since by (11),

$\tilde{t}_j \geq p_j/(2(m+1))$, and $\alpha = \frac{1}{2} + \frac{1}{4(m+1)}$. This proves that $\mathcal{S}$ is feasible. Finally, by (15), each job piece of j on shared processor ends by its completion on the private one which shows that (10) is the total weighted overlap of $\mathcal{S}$. □

We now show that an optimal solution to the LA gives an α-approximation of the optimum.

Lemma 3.2. *For each input instance, the schedule $\mathcal{S}$ that corresponds to an optimal solution to LA satisfies $\Sigma(\mathcal{S}) \geq \alpha\Sigma(\mathcal{S}_{\mathrm{opt}})$, where $\mathcal{S}_{\mathrm{opt}}$ is an optimal solution.*

Proof. Suppose that an input instance consists of n jobs $\{1,\ldots,n\}$ with processing times $p_1 \leq \cdots \leq p_n$, weights $w_1,\ldots,w_n$ and m shared processors with costs $c_1 \leq \cdots \leq c_m$. Let $\mathcal{S}_{\mathrm{opt}}$ be an optimal schedule for the instance. By Theorem 1.1 we may assume synchronized $\mathcal{S}_{\mathrm{opt}}$.

For $\mathcal{S}_{\mathrm{opt}}$, let y_{jik} be equal to the total execution time of job j on shared processor i in time interval (p_{k-1}, p_k) for each $j \in \{1,\ldots,n\}$, $i \in \{1,\ldots,m\}$ and $k \in \{1,\ldots,j\}$, where $p_0 = 0$. Denote for brevity

$$e_j = \sum_{i=1}^{m}\sum_{k=1}^{j} y_{jik}$$

to be the total amount of job j executed on all shared processors in $\mathcal{S}_{\mathrm{opt}}$. From the synchronization we observe

$$0 \leq e_j \leq \frac{mp_j}{(m+1)}. \tag{16}$$

We assign values to the variables in the LA as follows:

$$x_{jik} = \alpha y_{jik}, \quad j \in \{1,\ldots,n\}, k \in \{1,\ldots,j\}, i \in \{1,\ldots,m\}, \tag{17}$$

$$\tilde{t}_j = \alpha(p_j - e_j), \quad j \in \{1,\ldots,n\}. \tag{18}$$

We prove that this assignment gives a feasible solution to the LA. By (16) and $\alpha > \frac{1}{2}$, we have (11) satisfied. For each shared processor $\mathcal{M}_i$ and each interval (p_{k-1}, p_k), the total execution time of all job pieces executed in this interval on $\mathcal{M}_i$ in the schedule $\mathcal{S}_{\mathrm{opt}}$ is $\sum_{j=1}^{n} y_{jik}$. Thus, since $\mathcal{S}_{\mathrm{opt}}$ is feasible (and in particular we use the fact that no job j executes on a shared processor after time point p_j in $\mathcal{S}_{\mathrm{opt}}$), we have for each $i \in \{1,\ldots,m\}$

$$\sum_{j=k}^{n} y_{jik} \leq p_k - p_{k-1}.$$

This proves, by (17), that (12) holds. The total amount of a job j that executes on all shared processors in $\mathcal{S}_{\mathrm{opt}}$ is e_j and hence

$$e_j = \sum_{i=1}^{m}\sum_{k=1}^{j} y_{jik} = \frac{1}{\alpha}\sum_{i=1}^{m}\sum_{k=1}^{j} x_{jik},$$

which by (18) gives (13). Finally, (14) follows directly from (17) and the feasibility of $\mathcal{S}_{\mathrm{opt}}$.

Let $\mathcal{S}$ be the schedule corresponding to the above LA solution. By Lemma 3.1, $\mathcal{S}$ is indeed a feasible schedule, and the total weighted overlap of $\mathcal{S}$ equals by (17):

$$\Sigma(\mathcal{S}) = \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{k=1}^{j} x_{jik}(w_j - c_i) = \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{k=1}^{j} \alpha y_{jik}(w_j - c_i) = \alpha\Sigma(\mathcal{S}_{\mathrm{opt}}),$$

which completes the proof since $\Sigma(\mathcal{S}^*) \geq \Sigma(\mathcal{S})$, where $\mathcal{S}^*$ is the schedule that corresponds to an optimal solution to the LA. □

We now argue that the LA can be recast as a minimum-cost network flow problem which can be solved more efficiently than a general linear program, see Orlin [9]. The directed flow network $D = (V, A)$ can be constructed as follows. For each $i \in \{1, \dots, m\}$ and $k \in \{1, \dots, n\}$ introduce two nodes v_{ik}, v'_{ik}. For each job $j \in \{1, \dots, n\}$, introduce a node u_j, and let s and t be the source and sink nodes in the network. We add the following arcs to D:

(a) for each $j \in \{1, \dots, n\}$, let $(s, u_j) \in A$ be an arc of capacity $\mathtt{cap}((s, u_j)) = \frac{mp_j}{2(m+1)}$ and cost $\mathtt{cost}((s, u_j)) = 0$,

(b) for each $i \in \{1, \dots, m\}$, $j \in \{1, \dots, n\}$ and $k \in \{1, \dots, j\}$, let $(u_j, v_{ik}) \in A$ be an arc of capacity $\mathtt{cap}((u_j, v_{ik})) = +\infty$ and cost $\mathtt{cost}((u_j, v_{ik})) = w_j - c_i$,

(c) for each $i \in \{1, \dots, m\}$ and $k \in \{1, \dots, n\}$, let $(v_{ik}, v'_{ik}) \in A$ be an arc of capacity $\mathtt{cap}((v_{ik}, v'_{ik})) = \alpha(p_k - p_{k-1})$ and cost $\mathtt{cost}((v_{ik}, v'_{ik})) = 0$,

(d) for each $i \in \{1, \dots, m\}$ and $k \in \{1, \dots, n\}$, let $(v'_{ik}, t) \in A$ be an arc of capacity $\mathtt{cap}((v'_{ik}, t)) = +\infty$ and cost $\mathtt{cost}((v'_{ik}, t)) = 0$.

Suppose that f is an s-t flow in D. We define a feasible LA solution by taking $x_{jik} = f(u_j, v_{ik})$ for each $j \in \{1, \dots, n\}$, $i \in \{1, \dots, m\}$, $k \in \{1, \dots, j\}$ and $\tilde{t}_j = \alpha p_j - f(s, u_j)$ for each $j \in \{1, \dots, n\}$. Constraint (11) is satisfied because $\mathtt{cap}((s, u_j)) = \frac{mp_j}{2(m+1)}$. Constraint (12) follows directly from $\mathtt{cap}((v_{ik}, v'_{ik})) = \alpha(p_k - p_{k-1})$ since the flow through the arc (v_{ik}, v'_{ik}) equals $\sum_{j=k}^{n} f(u_j, v_{ik}) = \sum_{j=k}^{n} x_{jik}$. For (13) we have for each $j \in \{1, \dots, n\}$,

$$\sum_{i=1}^{m} \sum_{k=1}^{j} x_{jik} = \sum_{i=1}^{m} \sum_{k=1}^{j} f(u_j, v_{ik}) = f(s, u_j) = \alpha p_j - \tilde{t}_j.$$

Then, (14) is due to the fact that the flow is non-negative. Since all arcs except for (u_j, v_{ik})'s have cost zero, we obtain that the objective function in (10) equals the total cost of the flow f. The construction leading from an LA solution to a flow in D is straightforward and analogous and hence we skip its description.

We have proved the following.

Theorem 3.3. *For any input instance with $m \geq 1$ processors, there exists a strongly polynomial approximation algorithm with approximation ratio $\alpha = \frac{1}{2} + \frac{1}{4(m+1)}$.* □

Observe that by Lemma 3.2 the LA is $\frac{5}{8}$- approximation for a single shared processor problem, $m = 1$, which improves the $\frac{1}{2}$- approximation in [6] but at a cost of computational complexity which however still remains strongly polynomial.

4 Processor-descending and sequential schedules

As a stepping stone towards the proof of Theorem 1.1 and towards the algorithm for antithetical instances we show that there always exist optimal schedules that are *processor-descending* and *sequential* in this section.

For a given schedule $\mathcal{S}$, we say that an interval I is a *segment* in $\mathcal{S}$ if I is a maximal interval such that each shared processor is either idle in I or has no idle time in I. We say that a job is *present* in a segment if some non-empty part of the job executes in this segment. We then say that a segment I is *sequential* if each job j is either not present in I or there exists an interval $I' \subseteq I$, called the *interval of j in I*, such that each processor that is not idle in I executes j exactly in the interval I'. If each segment in a schedule is sequential, then the schedule is called *sequential*. We say that a schedule is *processor-descending* if each shared processor has no idle times between any two job pieces it executes, and for any two processors $\mathcal{M}_i$ and $\mathcal{M}_{i'}$ with $c_i < c_{i'}$ it holds that $\mathcal{M}_i$ completes executing all job pieces not later than $\mathcal{M}_{i'}$. A job j is *synchronized* in a processor-descending and sequential schedule $\mathcal{S}$ if the last piece of j ends on shared processors at $C^{\mathcal{P}}_{\mathcal{S}}(j)$.

We now describe a simple schedule modification that, without increasing the total weighted overlap, arrives at a schedule that is processor-descending and sequential (see Figure 3 for an illustration). We name this transformation as a procedure as it will be used later.

Procedure `MakeSequential`($\mathcal{S}$)
Input: A feasible schedule $\mathcal{S}$.
Output: A processor-descending and sequential schedule $\mathcal{S}'$ with $\Sigma(\mathcal{S}') \geq \Sigma(\mathcal{S})$.

(M1) As long as there exists a processor $\mathcal{M}_i$ and an idle time (a, b) (take this idle time to have maximum duration) followed by a piece (a', b') of a job j do the following: move the piece of j to be executed in time interval $(a, a + b' - a')$.

(M2) For each segment I in $\mathcal{S}$ do the following:

(M2a) Let $j^I_1, \ldots, j^I_{l(I)}$ be the jobs present in segment I sorted according to non-decreasing order of their completion times on private processors, $C^{\mathcal{P}}_{\mathcal{S}}(j^I_1) \leq \cdots \leq C^{\mathcal{P}}_{\mathcal{S}}(j^I_{l(I)})$. Let a^I_i be the total amount of job $j^I_i \in \{j^I_1, \ldots, j^I_{l(I)}\}$ executed in the segment I. Let m' be the number of processors used by I in $\mathcal{S}$.

(M2b) Replace the segment I in $\mathcal{S}$ with one in which the job j^I_t, $t \in \{1, \ldots, l(I)\}$, executes in time interval

$$\left(L + \frac{1}{m'} \sum_{t'=1}^{t-1} a^I_{t'}, L + \frac{1}{m'} \sum_{t'=1}^{t} a^I_{t'}\right)$$

on all these m' shared processors, where L is the left endpoint of I.

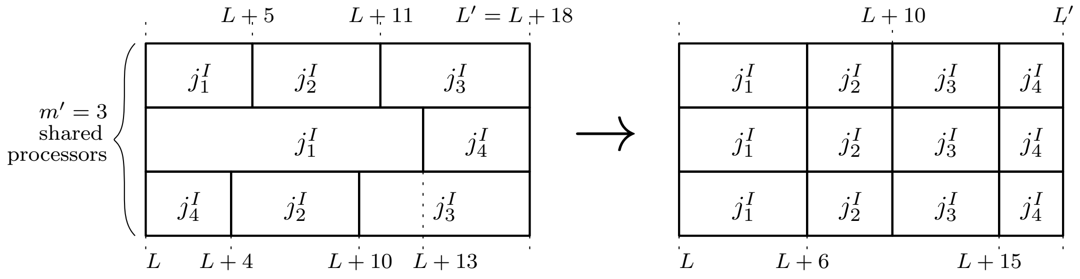


Figure 3: Transformation performed in Step (M2) for a segment $I = (L, L')$ with total job executions times in this segment being $a^I_1 = 18$, $a^I_2 = 12$, $a^I_3 = 15$, $a^I_4 = 9$

Lemma 4.1. *Procedure* `MakeSequential` *transforms an input schedule $\mathcal{S}$ into a schedule that is processor-descending, sequential and has the same total weighted overlap.*

Proof. Note that Step (M1) of Procedure `MakeSequential` makes $\mathcal{S}$ to be processor-descending because after this transformation, each shared processor is busy in a single time interval that starts at 0. Recall that, if in a given segment of the new schedule it holds that m' shared processors are used, then since they are ordered according to their costs, we may without loss of generality assume that these processors are $\mathcal{M}_1, \ldots, \mathcal{M}_{m'}$.

Consider now Step (M2) of Procedure `MakeSequential`. The fact that the total execution time of each job on shared processors does not change within the segment follows directly from the formula in Step (M2b). We need to prove that for each $t \in \{1, \ldots, l(I)\}$, the job j_t^I completes its piece in segment I in the output schedule $\mathcal{S}'$ not later than its completion time on its private processor:

$$C_{\mathcal{S}'}^{\mathcal{P}}(j_t^I) \geq L + \frac{1}{m'} \sum_{t'=1}^{t} a_{t'}^I = e_t \tag{19}$$

since this implies that $\Sigma(\mathcal{S}') = \Sigma(\mathcal{S})$. For each $t \in \{1, \ldots, l(I)\}$, if there exists a shared processor $\mathcal{M}_i$ such that a piece of j_t^I ends on $\mathcal{M}_i$ in $\mathcal{S}$ at e_t or later, then we are done. Suppose for a contradiction that (19) does not hold, i.e., $C_{\mathcal{S}'}^{\mathcal{P}}(j_t^I) < e_t$ for some t. Since $C_{\mathcal{S}}^{\mathcal{P}}(j_t^I) = C_{\mathcal{S}'}^{\mathcal{P}}(j_t^I)$, we have that the job j_t^I completes *before* e_t on each shared processor in $\mathcal{S}$. Thus, since there is no idle time in interval (L, e_t) in $\mathcal{S}'$, there exists $1 \leq \bar{t} < t$ such that the job $j_{\bar{t}}^I$ completes on some shared processor at time $\bar{e}$, after the time point e_t in $\mathcal{S}$, $\bar{e} > e_t$. But according to the job ordering picked in Step (M2a), $C_{\mathcal{S}}^{\mathcal{P}}(j_{\bar{t}}^I) \leq C_{\mathcal{S}}^{\mathcal{P}}(j_t^I)$. By assumption $C_{\mathcal{S}}^{\mathcal{P}}(j_t^I) = C_{\mathcal{S}'}^{\mathcal{P}}(j_t^I) < e_t$. This gives that $\bar{e} > C_{\mathcal{S}}^{\mathcal{P}}(j_{\bar{t}}^I)$, which violates (3) and gives the required contradiction since $\mathcal{S}$ is feasible. This proves (19) and completes the proof of the lemma. □

Thus we conclude.

Corollary 4.2. *There exists an optimal processor-descending and sequential schedule.* □

5 Antithetical Instances

An instance $\mathcal{J}$ is *antithetical* if for any two jobs j and j' it holds: $p_j \leq p_{j'}$ implies $w_j \geq w_{j'}$. Our main goal in this section is to show a polynomial time algorithm for antithetical instances. The algorithm relies on the LP given in Section 2 which however requires an optimal job order to produce an optimal solution. We prove that the ascending order of processing times is such an order, and that all jobs occur on shared processors in optimal schedules produced by the algorithm. The proof relies on a transformation, called j-filling, of a schedule which we now define. The transformation may produce schedules which are not synchronized even if applied to a synchronized schedule, however those schedules must then be processor-descending and sequential with synchronized suffixes inherited from the original synchronized schedule.

For a synchronized schedule $\mathcal{S}$ we say that it is *processing-time ordered* if $p_{j_1} \leq \cdots \leq p_{j_k}$, where $j_1, \ldots, j_k$ are the jobs that appear, in $\mathcal{S}$ in this order, on the shared processors. We then for brevity say that $(j_1, \ldots, j_k)$ is the *ordering* of jobs in $\mathcal{S}$. Consider an arbitrary processor-descending and sequential schedule $\mathcal{S}$. We say that a suffix $(j_1, \ldots, j_k)$, is *processing-time ordered* and *synchronized* in $\mathcal{S}$ if $p_{j_1} \leq \cdots \leq p_{j_k}$, there exists a time point t such that exactly the jobs $j_1, \ldots, j_k$ execute in time interval $(t, +\infty)$ in this order, and the jobs $j_1, \ldots, j_k$ are synchronized. The synchronization and processing-time ordering are thus not required for the entire schedule but only for some suffix of $\mathcal{S}$.

Consider a processor-descending and sequential schedule $\mathcal{S}$ such that there exists a job j which satisfies one of the following.

(1) j is present on the shared processors, j is not synchronized, and j is followed by a processing-time ordered synchronized suffix $(j_1, \ldots, j_k)$ in $\mathcal{S}$.

(2) j is not present on the shared processors, and $\mathcal{S}$ has a processing-time ordered synchronized suffix $(j_1, \ldots, j_k)$ that starts at time $t_1 < p_j$.

Let t_1 and t'_1 be such that the piece of the job j_1 executes in (t_1, t'_1) on the shared processors. We define an operation of *j-filling* in $\mathcal{S}$ as follows (see Figure 4(a)): for some t, $t_1 < t \leq t'_1$, the part of j_1 executing in time interval (t_1, t) is moved from each shared processor $\mathcal{M}_z$, $z \in \{1, \ldots, m'\}$ to its private processor $\mathcal{P}_{j_1}$, and it is replaced on $\mathcal{M}_z$ by j so that the completion time of j on $\mathcal{P}_j$ decreases by $m'(t - t_1)$, where m' is the number of shared processors used in the interval (t_1, t'_1). The $t \in (t_1, t_2]$ is chosen to be maximum to ensure that the completion time of j on shared processors, which equals t, is smaller than or equal to the completion time of j on $\mathcal{P}_j$. We remark that if $t < t'_1$, then the job j becomes synchronized as a result of j-filling —

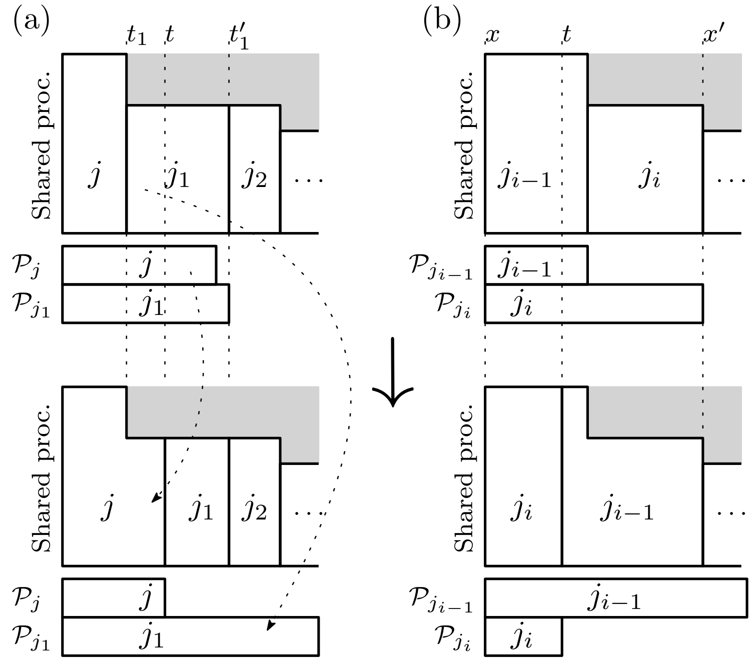


Figure 4: (a) j-filling for the case when j is present on the shared processors; (b) changing the order of jobs j_{i-1} and j_i on the shared processors when $p_{j_{i-1}} > p_{j_i}$

informally speaking this follows from observation that further increase of t is not possible due to the fact that there is not enough of j in time interval $(t_1, C^{\mathcal{P}}_{\mathcal{S}}(j))$ on private processor $\mathcal{P}_j$ in $\mathcal{S}$ to fill out the interval $(t_1, t + \varepsilon)$ on the shared processors for any $\varepsilon > 0$ (this case is depicted in Figure 4(a)). On the other hand, if $t = t_2$, then j may not be synchronized as a result of j-filling.

Note that the definition of j-filling is valid, it suffices to observe that the maximum t selected indeed satisfies $t_1 < t$. This follows from the assumption that $p_j > t_1$ when j is not present on the shared processors, and from the fact that j is not synchronized and directly precedes j_1 on the shared processors otherwise. Note that $w_j \geq w_{j_1}$ is sufficient to ensure that as a result of j-filling the total weighted overlap does not decrease in comparison to $\mathcal{S}$. Since the execution of jobs $j_2, \ldots, j_k$ does not change as result of j-filling, we obtain:

Observation 5.1. *Suppose that a schedule $\mathcal{S}$ and j satisfy the assumptions* (1) *or* (2) *of j-filling, and $w_j \geq w_{j_1}$. The operation of j-filling gives a feasible schedule $\mathcal{S}'$ such that $\Sigma(\mathcal{S}') \geq \Sigma(\mathcal{S})$, the job j_1 is either not synchronized in $\mathcal{S}'$ and present on the shared processors (this holds when $t < t'_1$), or the job j_1 executes on $\mathcal{P}_{j_1}$ only and $C^{\mathcal{P}}_{\mathcal{S}'}(j_1) > t'_1$ (this holds when $t = t'_1$) and the suffix $(j_{i+1}, \ldots, j_k)$ is processing-time ordered and synchronized in $\mathcal{S}'$.* □

We are now ready to prove the main result of this section.

Theorem 5.2. *For any antithetical instance, there exists an optimal schedule that is processing-time ordered and each job is present on the shared processors. Moreover, it can be computed in polynomial time.*

Proof. Consider an optimal schedule $\mathcal{S}$ for an antithetical instance. By Theorem 1.1 we may assume that $\mathcal{S}$ is synchronized. Let $(j_1, \ldots, j_k)$ be the ordering of jobs in $\mathcal{S}$. We first argue that $\mathcal{S}$ is processing-time ordered. We prove this by contradiction: take the largest index i such that $p_{j_{i-1}} > p_{j_i}$. Swap the jobs j_{i-1} and j_i on the shared processors as follows (see Figure 4(b)): Suppose that j_{i-1} and j_i occupy the interval (x, x') on the shared processors in $\mathcal{S}$. Find the t, $x < t \leq x'$, so that when replacing j_{i-1} by j_i in time interval (x, t) on each shared processor and executing j_i in $(0, t)$ on its private processor results in j_i having the total execution time equal to p_{j_i}. Thus, j_i remains synchronized. Finally execute j_{i-1} in time interval (t, x') on each shared processor on which j_{i-1} or j_i was initially present, and execute the remainder of j_{i-1} on its private processor. The fact that $p_{j_{i-1}} > p_{j_i}$ implies that this swap gives a feasible schedule and that j_{i-1} is no longer synchronized. By the maximality of j, the suffix $(j_{i+1}, \ldots, j_k)$ in the resulting schedule is processing-time ordered and synchronized (as nothing in the suffix has changed with respect to $\mathcal{S}$). Perform j_{i-1}-filling to $\mathcal{S}$, and then for each $i' := i+1, \ldots, k-1$ (in this order) apply $j_{i'}$-filling, obtaining a final schedule $\mathcal{S}'$. By Observation 5.1, $\mathcal{S}'$ is feasible, $\Sigma(\mathcal{S}') \geq \Sigma(\mathcal{S})$ and the job j_k is either not synchronized or not present on the shared processors in $\mathcal{S}'$. In both cases we obtain that $\mathcal{S}'$ is not optimal (observe that j_k completes later on its private processor than the last job completes on shared processors in $\mathcal{S}'$ which is obviously not optimal since some part of j_k can be moved to the cheapest shared processor and thus increase the overlap), which contradicts the optimality of $\mathcal{S}$. Thus, we have proved that $\mathcal{S}$ is processing-time ordered.

We now prove that $\mathcal{J} = \{j_1, \ldots, j_k\}$, i.e, all jobs are present on the shared processors in $\mathcal{S}$. By contradiction, let $j \notin \{j_1, \ldots, j_k\}$. If $p_j \geq p_{j_k}$, then $\mathcal{S}$ is not optimal and we immediately obtain a contradiction. Otherwise, since $\mathcal{S}$ is processing-time ordered and synchronized, as we showed earlier in the proof, there is a suffix $(j_i, \ldots, j_k)$ of $\mathcal{S}$ for which the condition (2) of j-filling is satisfied (i is the maximum index such that $p_j > p_{j_i}$). Perform the j-filling and then iteratively for $i' := i, \ldots, k-1$ (in this order) perform $j_{i'}$-filling obtaining the final $\mathcal{S}'$. Again by Observation 5.1, $\mathcal{S}'$ is feasible, $\Sigma(\mathcal{S}') \geq \Sigma(\mathcal{S})$ and j_k is either not present on shared processors or is not synchronized, giving us the required contradiction.

Finally, the LP (see Theorem 2.2) gives the optimal processing-time ordered schedule in polynomial time. □

6 Structure of Optimal Schedules

This section proves Theorem 1.1 that was announced earlier in the paper. By Corollary 4.2 we can limit ourselves to processor-descending and sequential schedules $\mathcal{S}$. Those schedules may not be synchronized for a number of reasons: a job may appear in more than one segment of $\mathcal{S}$, we call this a split of the job, or even if each job appears in at most one segment of $\mathcal{S}$ some jobs may not be synchronized by finishing on shared processors earlier than on their private processors. We need to show how to remove these undesirable configurations from processor-descending and sequential schedules to produce synchronized schedules without decreasing the total weighted overlap in the process. This removal affects schedules and their total weighted overlaps in a quite complicated way that requires sometimes delaying parts of the schedules whereas at other times their advancing in order not to reduce the total wighted overlap. We describe the main building block of the transformation, we call it *modification*, and its key properties in the next subsection. The modification will be used in Subsection 6.2 to remove the splits, and in Subsection 6.3 to synchronize jobs.

6.1 Towards Schedule Synchronization

Let $\mathcal{S}$ be a processor-descending and sequential schedule. Suppose $\mathcal{S}$ has ℓ segments, $S_1, \ldots, S_\ell$ and the i-th segment executes jobs $j_{i,1}, \ldots, j_{i,l(i)}$ in time intervals $(s_{i,1}, e_{i,1}), \ldots, (s_{i,l(i)}, e_{i,l(i)})$, respectively. Define $T(\mathcal{S}) = \{s_{i,k}, e_{i,k} : i = 1, \ldots, \ell; k = 1, \ldots, l(i)\}$ to be the set of all time points t such that some piece of a job starts or ends at t on a shared processor. Let $m_{i,k}$ be the number of shared processors used by $\mathcal{S}$ in the interval $(s_{i,k}, e_{i,k})$. Observe that this number remains the same for each interval in a segment i thus we denote it by m_i and we refer to it as the *width* of the interval $(s_{i,k}, e_{i,k})$. We define the *factor* $m^+_{i,k}$ and the *radius* $r_{i,k}$ of the interval $(s_{i,k}, e_{i,k})$ as follows. Let a job $j_{i,k}$ execute in an interval $(s_{i,k}, e_{i,k})$, if $e_{i,k} = C^{\mathcal{P}}_{\mathcal{S}}(j_{i,k})$, then $m^+_{i,k} = m_i + 1$ and $r_{i,k} = \min\{e_{i,k} - s_{i,k}, p_{j_{i,k}} - e_{i,k}\}$. Otherwise, if $e_{i,k} < C^{\mathcal{P}}_{\mathcal{S}}(j_{i,k})$, then $m^+_{i,k} = m_i$ and $r_{i,k} = \min\{e_{i,k} - s_{i,k}, C^{\mathcal{P}}_{\mathcal{S}}(j_{i,k}) - e_{i,k}\}$.

In this section we define a transformation of $\mathcal{S}$ that would be used to make it synchronized. The transformation is multi-step which in each step is defined as a function ξ that takes a schedule $\mathcal{S}$, a time point $t \in T(\mathcal{S})$ and a *shift* $\varepsilon \in \mathbb{R}$ as an input, and produces a schedule $\mathcal{S}'$ and a new shift ε' as output.

Before giving its formal description, we start with some informal intuitions. We consider three basic steps that make up the whole transformation. The first step is the *base* step of our transformation: this case simply moves the endpoint of the last job piece of the entire schedule, i.e., the piece that ends at $e_{\ell,l(\ell)}$. If $\varepsilon > 0$, then this piece is moved to the right (i.e., it completes later in $\mathcal{S}'$ than in $\mathcal{S}$), and if $\varepsilon < 0$, then this piece advances in $\mathcal{S}'$ with respect to $\mathcal{S}$.

The *main* step is subdivided into two subcases. In the *first* subcase we consider a piece of a job j that ends earlier on shared processors than on the private processor. Note that if this is the last piece of j, then it implies that j is not synchronized. However, it may also happen that this is not the last piece of j but j itself is synchronized as there may be another piece of j in one of the subsequent segments of $\mathcal{S}$. In this situation the endpoint of the piece of j is just moved (on each shared processor) to the right or to the left (according to whether $\varepsilon > 0$ or $\varepsilon < 0$, respectively).

In the *second* subcase we consider the last piece of a job j that is synchronized. Then, we shift both the endpoint of the piece of j on shared processors and, by the same amount, the endpoint on the private processor. In this way the job remains synchronized.

For each step we define a payoff value that tells how much the total weighted overlap of the schedule changes by doing the step. We need to keep in mind the multi-step nature of the entire transformation. Typically the transformation starts with some $\mathcal{S}$, time point $t = e_{i,b}$ and ε, then it will subsequently trigger changes for the same parameter $\mathcal{S}$, subsequent time points

$$e_{i,b+1}, \ldots, e_{i,l(i)}, \quad \ldots \quad , e_{i+1,1}, \ldots, e_{i+1,l(i+1)}, \quad \ldots \quad , e_{\ell,1}, \ldots, e_{\ell,l(\ell)},$$

and different values of ε.

We now give a description of these three steps and then an example that depicts all of them follows (see Figure 5). The changes introduced to $\mathcal{S}$ in each of these steps are referred to as one step *modifications*.

Base Step. The assumption of this step is that $t = e_{\ell,l(\ell)}$, i.e., t is the end of the last job $j_{\ell,l(\ell)}$ of the last segment of $\mathcal{S}$. We call this job the *job of the modification* and denote by j for convenience. We may assume without loss of generality that j is synchronized and hence $t = C^{\mathcal{P}}_{\mathcal{S}}(j)$. The modification is *doable* if

$$\varepsilon \in \left[-m^+_{\ell,l(\ell)} r_{\ell,l(\ell)}, m^+_{\ell,l(\ell)} r_{\ell,l(\ell)}\right]. \tag{20}$$

For the doable modification, we set the completion time of j on processors $\mathcal{M}_1, \ldots, \mathcal{M}_{m_\ell}$ and $\mathcal{P}_j$ in $\mathcal{S}'$ to

$$e_{\ell,l(\ell)} + \frac{\varepsilon}{m^+_{\ell,l(\ell)}}. \tag{21}$$

This transformation is denoted by $\xi(\mathcal{S}, t, \varepsilon)$ and its *payoff* equals

$$\texttt{payoff}(\xi(\mathcal{S}, t, \varepsilon)) = \frac{\varepsilon}{m^+_{\ell, l(\ell)}} \sum_{z=1}^{m_\ell} (w_j - c_z). \tag{22}$$

This completes the description of the base step.

In the two remaining steps, we assume $t = e_{i,b}$ where $i < \ell$ or $b < l(\ell)$ and use some common notation for both. Let for brevity $j = j_{i,b}$. Note that the interval that immediately follows $(s_{i,b}, e_{i,b})$ is either $(s_{i,b+1} = e_{i,b}, e_{i,b+1})$ when $b < l(i)$, i.e., when j is not the last in the segment S_i or $(s_{i+1,1} = e_{i,b}, e_{i+1,1})$ when $b = l(i)$, i.e., j is last in the segment S_i. Let $m' = m_i$ in the former case, and $m' = m_{i+1}$ in the latter case. Finally, let j' be the job in the interval that immediately follows $(s_{i,b}, e_{i,b})$.

Main Step I. The assumption of this step is that $e_{i,b} < C^{\mathcal{P}}_{\mathcal{S}}(j)$. We say that the modification is *doable* if

$$\varepsilon \in \left[-m^+_{i,b} r_{i,b}, m^+_{i,b} r_{i,b}\right], \tag{23}$$

For the doable modification set the completion time of j on processors $\mathcal{M}_1, \ldots, \mathcal{M}_{m_i}$ equal to the start time of j' on shared processors $\mathcal{M}_1, \ldots, \mathcal{M}_{m'}$ to

$$t' = e_{i,b} + \frac{\varepsilon}{m^+_{i,b}}, \tag{24}$$

and denote $\texttt{next}(\varepsilon) = \varepsilon \frac{m'}{m^+_{i,b}}$. The *payoff* is

$$\texttt{payoff}(\xi(\mathcal{S}, t, \varepsilon)) = \frac{\varepsilon}{m^+_{i,b}} \left(\sum_{z=1}^{m_i} (w_j - c_z) - \sum_{z=1}^{m'} (w_{j'} - c_z) \right). \tag{25}$$

Main Step II. The assumption of this step is:

$$e_{i,b} = C^{\mathcal{P}}_{\mathcal{S}}(j). \tag{26}$$

We say that the modification is *doable* if

$$\varepsilon \in \left[-m^+_{i,b} r_{i,b}, m^+_{i,b} r_{i,b}\right] \tag{27}$$

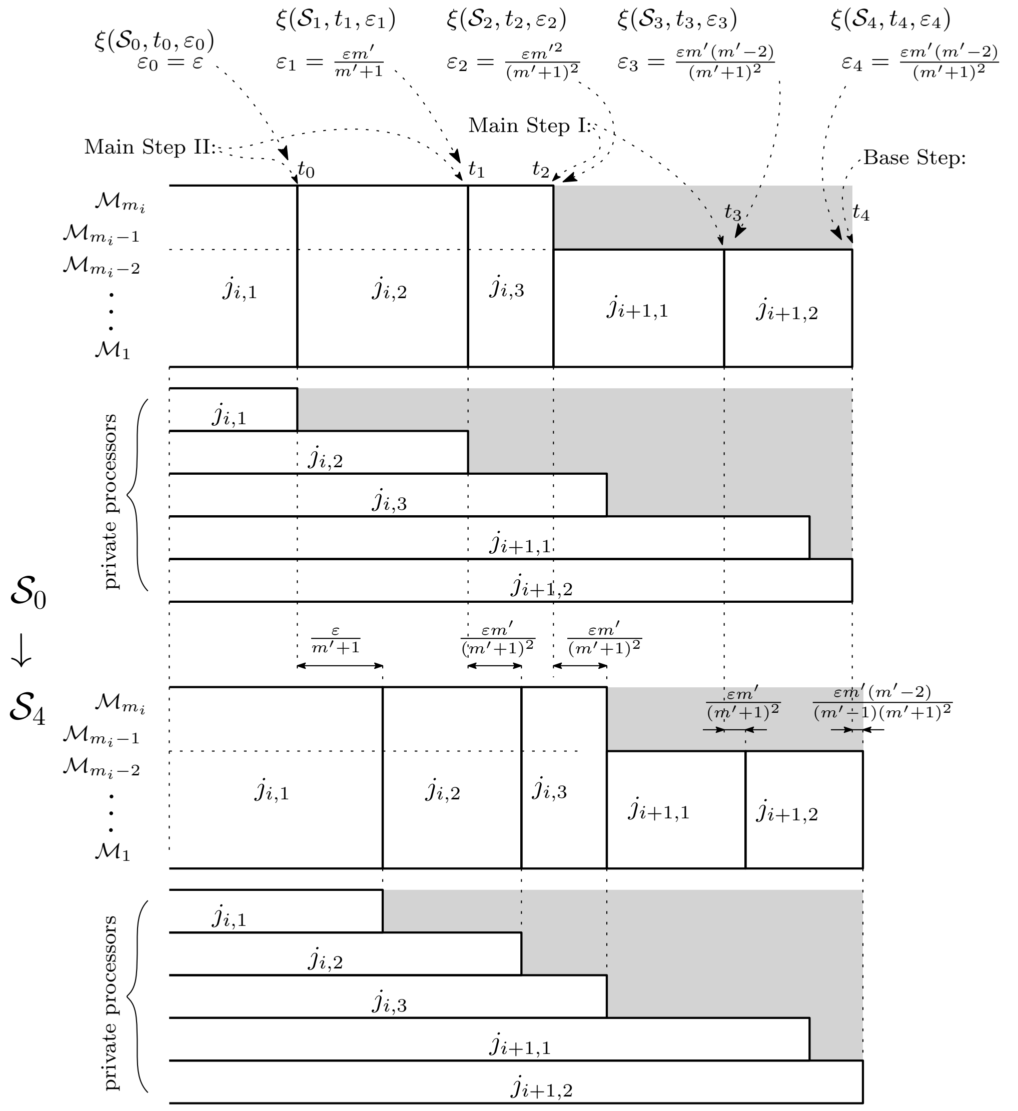
For the doable modification set the completion time of j on processors $\mathcal{M}_1, \ldots, \mathcal{M}_{m_i}$ and $\mathcal{P}_j$ equal to the start time of j' on shared processors $\mathcal{M}_1, \ldots, \mathcal{M}_{m'}$ to

$$t' = e_{i,b} + \frac{\varepsilon}{m^+_{i,b}}, \tag{28}$$

and denote $\texttt{next}(\varepsilon) = \varepsilon \frac{m'}{m^+_{i,b}}$. The *payoff* is then

$$\texttt{payoff}(\xi(\mathcal{S}, t, \varepsilon)) = \frac{\varepsilon}{m^+_{i,b}} \left(\sum_{z=1}^{m_i} (w_j - c_z) - \sum_{z=1}^{m'} (w_{j'} - c_z) \right). \tag{29}$$

This completes the description of all cases of our transformation — see Figure 5 for an example.



$\xi(\mathcal{S}_0, t_0, \varepsilon_0)$
$\varepsilon_0 = \varepsilon$
$\xi(\mathcal{S}_1, t_1, \varepsilon_1)$
$\varepsilon_1 = \frac{\varepsilon m'}{m'+1}$
$\xi(\mathcal{S}_2, t_2, \varepsilon_2)$
$\varepsilon_2 = \frac{\varepsilon m'^2}{(m'+1)^2}$
$\xi(\mathcal{S}_3, t_3, \varepsilon_3)$
$\varepsilon_3 = \frac{\varepsilon m'(m'-2)}{(m'+1)^2}$
$\xi(\mathcal{S}_4, t_4, \varepsilon_4)$
$\varepsilon_4 = \frac{\varepsilon m'(m'-2)}{(m'+1)^2}$
Main Step II:
Main Step I:
Base Step:
t_0 t_1 t_2 t_3 t_4
$\mathcal{M}_{m_i}$ $\mathcal{M}_{m_i-1}$ $\mathcal{M}_{m_i-2}$ $\mathcal{M}_1$
$j_{i,1}$ $j_{i,2}$ $j_{i,3}$ $j_{i+1,1}$ $j_{i+1,2}$
private processors
$\mathcal{S}_0$
$\downarrow$
$\mathcal{S}_4$
$\frac{\varepsilon}{m'+1}$
$\frac{\varepsilon m'}{(m'+1)^2}$
$\frac{\varepsilon m'}{(m'+1)^2}$
$\frac{\varepsilon m'}{(m'+1)^2}$
$\frac{\varepsilon m'(m'-2)}{(m'-1)(m'+1)^2}$

Figure 5: In this example we consider two consecutive segments, which have three and two job pieces, respectively. We consider executing $\xi(\mathcal{S}_i, t_i, \varepsilon_i)$ for $i = 0, \ldots, 4$, where $\varepsilon_0 = \varepsilon$ is positive. All five modifications are doable but note that the job $j_{i,3}$ is synchronized in $\mathcal{S}_4$ but is not synchronized in $\mathcal{S}$. Hence, according to Condition (23) of Main Step I that handles this modification, this is the maximum $\varepsilon > 0$ for which all five modifications are doable.

Let $t_0 < t_1 < \cdots < t_q$, $q \geq 0$, be the last $q + 1$ end points in the sequence $e_{1,1} < \cdots < e_{\ell,l(\ell)}$ of the schedule $\mathcal{S}$. Let $I_1 = (t_0, t_1), \ldots, I_q = (t_{q-1}, t_q)$ be the last q intervals of $\mathcal{S}$. Let j_i, m_i, m_i^+, and r_i be the job, the width, the factor, and the radius of the interval ending at t_i, $i = 0, \ldots, q$. The $q + 1$ step modification starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and an $\varepsilon = \varepsilon_0$ such that

$$0 < |\varepsilon| < \min_{i=0,\ldots,q} \{m_i r_i/2\}, \tag{30}$$

and recursively builds schedules $\mathcal{S}_1, \ldots, \mathcal{S}_{q+1}$ using the one step modifications just described such that

$\mathcal{S}_i = \xi(\mathcal{S}_{i-1}, t_{i-1}, \varepsilon_{i-1})$, where each subsequent value of ε_i is computed on the basis of the previous one as follows: $\varepsilon_i = \mathtt{next}(\varepsilon_{i-1})$ for each $i \in \{1, \dots, q+1\}$. Finally, the $\mathcal{S}_{q+1} = \xi(\mathcal{S}_q, t_q, \varepsilon_q)$ is always the Base Step. We say that the $q+1$ step modification is *doable* if all its one step modifications are doable, i.e. all $\varepsilon_0, \dots, \varepsilon_q$ satisfy appropriate condition in (20), (23) and (27). We later show that the initial choice of ε_0 that meets (30) guarantees that the $q+1$ step modification is doable. Observe that by definition

$$\varepsilon_i = \mathtt{next}(\varepsilon_{i-1}) = \varepsilon_{i-1} \frac{m_i}{m^+_{i-1}}. \tag{31}$$

Therefore the points $t_0, t_1, \dots, t_q$ and the shifts $\varepsilon_0, \dots, \varepsilon_q$ can be readily calculated from $\mathcal{S}$ and ε. We summarize this in the following corollary.

Corollary 6.1. *Consider a doable $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon = \varepsilon_0$. Then,*

$$\varepsilon_i = \varepsilon \prod_{z=1}^{i} \frac{m_z}{m^+_{z-1}}$$

for each $i \in \{1, \dots, q\}$. □

We remark that we will use later the fact that each ε_i is linearly dependent on the ε.

For a doable $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon = \varepsilon_0$, by (22), (25) and (29) the payoff can be written as follows

$$\Delta(\mathcal{S}_0, t_0, \varepsilon_0) = \sum_{k=0}^{q} \mathtt{payoff}(\xi(\mathcal{S}_k, t_k, \varepsilon_k)) = \sum_{i=0}^{q} \frac{\varepsilon_i}{m^+_i} \left(\sum_{z=1}^{m_i} (w_{j_i} - c_z) - \sum_{z=1}^{m_{i+1}} (w_{j_{i+1}} - c_z) \right), \tag{32}$$

where $m_{q+1} = 0$. We conclude from (32) the following.

Corollary 6.2. *For a doable $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon = \varepsilon_0$ it holds*

$$\Delta(\mathcal{S}_0, t_0, \varepsilon_0) = \frac{\varepsilon_0}{m^+_0} \sum_{z=0}^{m_0} (w_{j_0} - c_z) + \sum_{i=1}^{q} \left(\frac{\varepsilon_i}{m^+_i} - \frac{\varepsilon_{i-1}}{m^+_{i-1}} \right) \sum_{z=1}^{m_i} (w_{j_i} - c_z).$$

□

Motivated by Corollaries 6.1 and 6.2, we introduce the following function for each $t \in T(\mathcal{S})$:

$$R(\mathcal{S}, t) = \frac{1}{m^+_0} \sum_{z=1}^{m_0} (w_{j_0} - c_z) + \sum_{i=1}^{q} \left(\frac{1}{m^+_i} \prod_{z=1}^{i} \frac{m_z}{m^+_{z-1}} - \frac{1}{m^+_{i-1}} \prod_{z=1}^{i-1} \frac{m_z}{m^+_{z-1}} \right) \sum_{z=1}^{m_i} (w_{j_i} - c_z),$$

which we call the *rate* of a doable $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon = \varepsilon_0$. We stress out that the rate is the same regardless of the value of ε chosen for the modification. In other words, the function R depends only on the schedule $\mathcal{S}$ and the time point $t \in T(\mathcal{S})$. By Corollaries 6.1 and 6.2 we obtain:

Corollary 6.3. *For a doable $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon = \varepsilon_0$, $\Delta(\mathcal{S}_0, t_0, \varepsilon_0) = \varepsilon \cdot R(\mathcal{S}_0, t_0)$.* □

Note that a doable $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon = \varepsilon_0$ does not produce a feasible schedule $\mathcal{S}'$. More precisely, the schedule $\mathcal{S}'$ is not feasible since the total amount of the job j_0 equals $p_{j_0} + \varepsilon$ (note that this is the job $j_{i,1}$ in the example from Figure 5) in $\mathcal{S}'$. We summarize the properties of $\mathcal{S}'$ in the following lemmas.

Lemma 6.4. *Let $\mathcal{S}$ be a processor-descending and sequential schedule. A doable $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon = \varepsilon_0$ that meets (30) results in $\mathcal{S}'$ that satisfies the following conditions:*

(i) *the completion time of each job j on each shared processor is smaller than or equal to $C^{\mathcal{P}}_{\mathcal{S}'}(j)$,*

(ii) *the total execution time of each job $j \neq j_0$ equals p_j in $\mathcal{S}'$,*

(iii) *the total execution time of j_0 in $\mathcal{S}'$ is $p_{j_0} + \varepsilon$,*

(iv) *no two pieces of jobs overlap in $\mathcal{S}'$.*

Proof. Assume $\varepsilon_0 > 0$ in the proof, the proof for $\varepsilon_0 < 0$ is similar and thus will be omitted. The ends of the interval $I_i = (s_i, e_i)$ change to $(s'_i, e'_i) = I'_i$ as a result of the $q+1$ step modification as follows:

$$s'_i = s_i + \frac{\varepsilon_{i-1}}{m^+_{i-1}} \text{ and } e'_i = e_i + \frac{\varepsilon_i}{m^+_i} \tag{33}$$

for $i = 1, \dots, q$. Thus

$$e'_i - s'_i = e_i - s_i + \frac{\varepsilon_{i-1}}{m^+_{i-1}}\left(\frac{m_i}{m^+_i} - 1\right). \tag{34}$$

For $m^+_i = m_i$, we have $e'_i - s'_i = e_i - s_i > 0$. For $m^+_i = m_i + 1$, we have

$$e'_i - s'_i = e_i - s_i - \frac{\varepsilon_{i-1}}{m^+_{i-1} m^+_i}. \tag{35}$$

Since $\varepsilon > \frac{\varepsilon_{i-1}}{m^+_{i-1}}$, and by (30) $m^+_i(e_i - s_i) > \varepsilon$, we have $e'_i - s'_i > 0$ for $\varepsilon > 0$. Thus, by the one step modifications, (iv) holds.

By (33) the execution of j_i is reduced (this does not happen for j_0 for which the reduction is 0) by

$$\varepsilon_{i-1}\frac{m_i}{m^+_{i-1}} = \varepsilon_i, \tag{36}$$

and it increases by

$$\varepsilon_i \frac{m_i}{m^+_i} \tag{37}$$

on shared processors. For $m^+_i = m_i$ the two are equal, and for $m^+_i = m_i + 1$, the private processor $\mathcal{P}_{j_0}$ of job j_0 gets $\varepsilon_i \frac{1}{m^+_i}$ of that job. Thus (ii) and (iii) hold.

In Base Step, the job j_ℓ is synchronized due to (21). Similarly, in Main Step II, also j_i completes both on shared processor and on its private processor at the same time according to (28). In Main Case I, the completion time of j is set in (24) and this does not exceed $C^{\mathcal{P}}_{\mathcal{S}'}(j)$ by definition of r_i in the right hand side inequality in (23). For all remaining jobs their completion times on all processors remain unchanged, which proves (i). □

The second lemma shows a sufficient condition for ε to make $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon_0 = \varepsilon$ doable. Recall that $\mathcal{S}'$ produced by the $q+1$ step modification is not feasible however, by Lemma 6.4, the only reason for that is that the total execution time of the job j_0 is incorrect in $\mathcal{S}'$, i.e., it equals $p_{j_0} + \varepsilon$ instead of p_{j_0}. For this reason we introduce notation $\mathcal{S}'_{-j}$, for a job j, to denote a schedule obtained from $\mathcal{S}'$ by removing all pieces of j from shared processors and by removing the private processor of j. Note that $\mathcal{S}'_{-j}$ is then a feasible schedule for the instance $\mathcal{J} \setminus \{j\}$. Hence, the second lemma also shows the difference between the total weighted overlap $\Sigma(\mathcal{S}'_{-j})$ of $\mathcal{S}'_{-j}$, which gives the sum of total overlaps of all jobs in $\mathcal{S}'$ except of j, and the total weighted overlap $\Sigma(\mathcal{S})$ of $\mathcal{S}$.

Lemma 6.5. *Let $\mathcal{S}$ be a processor-descending sequential schedule. Let ε meet (30). Then, both $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and $\varepsilon_0 = -\varepsilon$ and $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t = t_0$ and or $\varepsilon_0 = \varepsilon$ are doable, and we have*

$$\Sigma(\mathcal{S}'_{-j_0}) = \Sigma(\mathcal{S}) + \Delta(\mathcal{S}_0, t_0, \varepsilon_0) - \sum_{i=1}^{m} \mathtt{ovlp}_{\mathcal{S}}(j_0, \mathcal{M}_i)(w_{j_0} - c_i) - \frac{\varepsilon}{m_0^+} \sum_{z=1}^{m_0} (w_{j_0} - c_z), \tag{38}$$

for the resulting schedule $\mathcal{S}'$.

Proof. Assume $\varepsilon_0 > 0$ in the proof, the proof for $\varepsilon_0 < 0$ is similar and thus will be omitted. We first prove that

$$m_i^+(e'_i - s'_i) > \varepsilon_i \tag{39}$$

for $i \in \{1, \ldots, q\}$. This holds for $m_i^+ = m_i$ since then $e_i - s_i = e'_i - s'_i$ and by $\varepsilon \geq \varepsilon_i$. Suppose $m_i^+ = m_i + 1$. By (35) and (39) we need to show

$$m_i^+(e_i - s_i) > \varepsilon_i + \frac{\varepsilon_{i-1}}{m_{i-1}^+ m_i^+}. \tag{40}$$

To that end we observe that

$$\frac{\varepsilon_{i-1} m_i^+}{m_{i-1}^+} > \varepsilon_i + \frac{\varepsilon_{i-1}}{m_{i-1}^+ m_i^+} = \frac{\varepsilon_{i-1} m_i}{m_{i-1}^+} + \frac{\varepsilon_{i-1}}{m_{i-1}^+ m_i^+}. \tag{41}$$

Thus it suffices to show that

$$m_i^+(e_i - s_i) > \frac{\varepsilon_{i-1} m_i^+}{m_{i-1}^+}, \tag{42}$$

or equivalently

$$e_i - s_i > \frac{\varepsilon_{i-1}}{m_{i-1}^+}. \tag{43}$$

By multiplying both sides of the last inequality by m_i we get

$$m_i(e_i - s_i) > \frac{\varepsilon_{i-1} m_i}{m_{i-1}^+} = \varepsilon_i. \tag{44}$$

This last inequality holds since $\varepsilon \geq \varepsilon_i$ and (30) holds for ε. We also prove, a similar proof for $m_i^+ C_{\mathcal{S}}^{\mathcal{P}}(j) - e'_i > \varepsilon_i$ will be omitted, that

$$m_i^+(p_j - e'_i) > \varepsilon_i. \tag{45}$$

Since $m_i^+ \geq m_i$, it suffices to show that

$$m_i(p_j - e_i) > \varepsilon_i + \varepsilon_i \frac{m_i}{m_i^+}. \tag{46}$$

The last inequality holds since $m_i(p_j - e_i) \geq 2(m_i r_i/2) > 2\varepsilon \geq \varepsilon_i$ by (30). This completes the proof of the first part of the lemma. Observe that ε_i does not reach neither end of doable intervals.

For the proof of the second part, let for brevity $\mathcal{J}' = \{j_0, j_1, \ldots, j_q\}$. We have

$$\begin{aligned}\Sigma(\mathcal{S}'_{-j_0}) &= \sum_{j\in\mathcal{J}\setminus\{j_0\}} \sum_{i=1}^{m} \mathtt{ovlp}_{\mathcal{S}'_{-j_0}}(j, \mathcal{M}_i)(w_j - c_i) \\ &= \sum_{j\in\mathcal{J}\setminus\mathcal{J}'} \sum_{i=1}^{m} \mathtt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i)(w_j - c_i) + \sum_{j\in\mathcal{J}'\setminus\{j_0\}} \sum_{i=1}^{m} \mathtt{ovlp}_{\mathcal{S}'_{-j_0}}(j, \mathcal{M}_i)(w_j - c_i).\end{aligned} \tag{47}$$

Consider any $j_i \in \mathcal{J}' \setminus \{j_0\}$. The total weighted overlap of j_i is the same in $\mathcal{S}_{-j_0}$ as in $\mathcal{S}$ except for the shift in its piece performed by the i-th and $(i-1)$-st modifications (see also Corollary 6.2):

$$\begin{aligned}\sum_{z=1}^{m} \mathtt{ovlp}_{\mathcal{S}_{-j_0}}(j_i, \mathcal{M}_z)(w_{j_i} - c_z) = &\sum_{z=1}^{m} \mathtt{ovlp}_{\mathcal{S}}(j_i, \mathcal{M}_z)(w_{j_i} - c_z) \\ &+ \left(\frac{\varepsilon_i}{m_i^+} - \frac{\varepsilon_{i-1}}{m_{i-1}^+}\right) \sum_{z=1}^{m_i} (w_{j_i} - c_z).\end{aligned} \tag{48}$$

By (47) and (48) applied to all jobs in $\mathcal{J}' \setminus \{j_0\} = \{j_1, \ldots, j_q\}$ we obtain

$$\begin{aligned}\Sigma(\mathcal{S}'_{-j_0}) = \Sigma(\mathcal{S}) - &\sum_{i=1}^{m} \mathtt{ovlp}_{\mathcal{S}}(j_0, \mathcal{M}_i)(w_{j_0} - c_i) \\ &+ \sum_{i=1}^{q} \left(\frac{\varepsilon_i}{m_i^+} - \frac{\varepsilon_{i-1}}{m_{i-1}^+}\right) \sum_{z=1}^{m_i} (w_{j_i} - c_z).\end{aligned} \tag{49}$$

By Corollary 6.2, (49) and $\varepsilon_0 = \varepsilon$,

$$\begin{aligned}\Sigma(\mathcal{S}'_{-j_0}) = \Sigma(\mathcal{S}) - &\sum_{i=1}^{m} \mathtt{ovlp}_{\mathcal{S}}(j_0, \mathcal{M}_i)(w_{j_0} - c_i) \\ &+ \Delta(\mathcal{S}_0, t_0, \varepsilon_0) - \frac{\varepsilon}{m_0^+} \sum_{z=1}^{m_0} (w_{j_0} - c_z).\end{aligned} \tag{50}$$

which proves (38) and completes the proof of the lemma. □

6.2 Splits

Suppose that $\mathcal{S}$ is a processor-descending and sequential schedule. We say that a job j has a (I, I')-*split* if I and I' are two pieces of j executing in two different segments. We assume that I' is to the right of I. Given that $\mathcal{S}$ has such a job j with a (I, I')-split, we introduce the following schedule transformation that we call a (I, I', ε)-*transfer*. Although this modification works for an arbitrary split, we will be particularly interested in our analysis in the case when the (I, I')-split is the rightmost. Let $q \geq 0$ be the number of intervals (job pieces) to the right of I' in $\mathcal{S}$. Consider ε such that

$$0 < |\varepsilon| < \min\left\{|I|, \frac{|I'|}{1 + 1/m_0^+}, \min_{i=0,\ldots,q}\{m_i r_i/2\}\right\} \tag{51}$$

where m_0^+ is the factor of interval $I' = (s_0, e_0)$. Let m_0 be the width of I'. The modification is composed of the following steps.

(T1) Obtain a schedule $\mathcal{S}'$ by performing $q+1$ step modification with $\mathcal{S}$, $t = e_0$, and ε.

(T2) If $\varepsilon > 0$, then change in $\mathcal{S}'$ the completion time of the piece in I of the job j from y to $y - \varepsilon$ on the processor $\mathcal{M}_{m_0+1}$, where y is the right endpoint of I.

(T3) If $\varepsilon < 0$, then add a piece of the job j of length $|\varepsilon|$ to the shared processor $\mathcal{M}_{m_0+1}$ in time interval $(s_0, s_0 + |\varepsilon|)$.

(T4) Call `MakeSequential`$(\mathcal{S}')$ to make each segment of the new schedule sequential, and return $\mathcal{S}'$.

The (I, I')-transfer is illustrated in Figure 6.

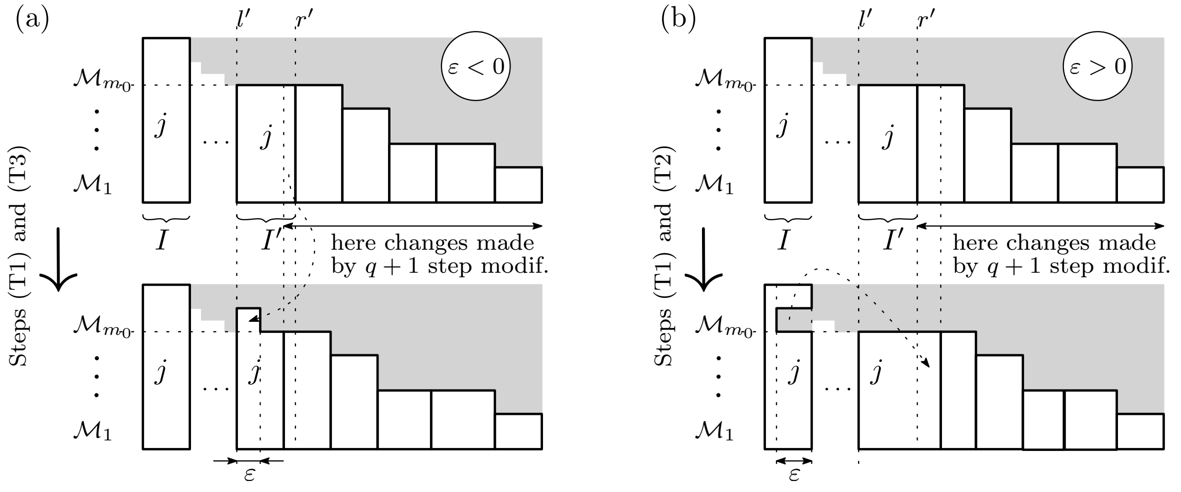


Figure 6: A (I, I', ε)-transfer: (a) if $\varepsilon < 0$, then Steps (T1) and (T3) modify the schedule shown on top to the one on bottom; (b) if $\varepsilon > 0$, then Steps (T1) and (T2) are applied.

Lemma 6.6. *If $\mathcal{S}$ is an optimal processor-descending and sequential schedule with a job j having the right-most (I, I')-split, then both the $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t_0 = e_0$, $\varepsilon_0 = \varepsilon$, and the $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t_0 = e_0$, $\varepsilon_0 = -\varepsilon$, where ε satisfies condition (51), are doable and produce a processor-descending and sequential schedule $\mathcal{S}'$. Moreover, for $\varepsilon < 0$ we have $\mathcal{S}'$ shorter than $\mathcal{S}$.*

Proof. In Step (T2), ε must not exceed $|I|$ as this is the length of the piece of j executing in the interval I. By (21), (24) and (28) in Step (T3), we need $|\varepsilon| \le |I'| - |\varepsilon| / m_1^+$. Thus, we obtain a condition

$$|\varepsilon| \le \frac{|I'|}{1 + 1/m_0^+}.$$

By Lemma 6.5 both $q+1$ step modifications are doable since (51) implies (30). Moreover, for $\varepsilon < 0$ we have $\mathcal{S}'$ shorter that $\mathcal{S}$ due to Base Step of the modification. □

Lemma 6.7. *Suppose that $\mathcal{S}$ is a processor-descending and sequential schedule with a job j having the right-most (I, I')-split and consider any ε that meets (51). Then, the processor-descending and sequential schedule $\mathcal{S}'$ resulting from the (I, I', ε)-transfer satisfies*

$$\Sigma(\mathcal{S}') = \Sigma(\mathcal{S}) + \varepsilon \cdot (R(\mathcal{S}, t) - w_j + c_{m_0+1}),$$

where m_0 is the width of the interval $I' = (s', e' = t)$ in $\mathcal{S}$. Moreover, for an optimal $\mathcal{S}$,

$$R(\mathcal{S}, t) - w_j + c_{m_0+1} = 0.$$

Proof. Consider first an arbitrary ε that meets (51). By Lemma 6.6, the (I, I', ε)-transfer is doable. Let j be the job with (I, I')-split modified in Step (T2) of Procedure `Transfer`. We first calculate the sum of $\texttt{ovlp}_{\mathcal{S}'}(j, \mathcal{M}_i)$ taken over all shared processors $\mathcal{M}_i$. This value is similar to that in $\mathcal{S}$, except for the two changes introduced to j in Steps (T1), (T2) and (T3) of Procedure `Transfer`. In Steps (T2) and (T3), the total weighted overlap of j changes by

$$-\varepsilon(w_j - c_{m_0+1}).$$

In Step (T1), it changes by

$$\frac{\varepsilon}{m_0^+}\sum_{z=1}^{m_0}(w_j - c_z),$$

where m_0 and m_0^+ are the width and the factor of I'. Indeed, this follows from the fact that the transformation changes only the right endpoint of the piece of j executing in $I' = (s', e')$, and due to (21), (24) and (28) this value changes, on each machine $\mathcal{M}_1, \ldots, \mathcal{M}_{m_1}$, by ε/m_1^+ since this is done in the first step of the $q+1$ step modification that starts with $\mathcal{S}_0 = \mathcal{S}$, $t_0 = e'$, and $\varepsilon_0 = \varepsilon$. Thus we obtain

$$\begin{aligned}\sum_{z=1}^{m}\texttt{ovlp}_{\mathcal{S}'}(j, \mathcal{M}_z)(w_j - c_z) = \sum_{z=1}^{m}&\texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_z)(w_j - c_z)\\ &- \varepsilon(w_j - c_{m_0+1}) + \frac{\varepsilon}{m_0^+}\sum_{z=1}^{m_0}(w_j - c_z).\end{aligned} \tag{52}$$

The total weighted overlap of $\mathcal{S}'$ can be expressed as

$$\Sigma(\mathcal{S}') = \Sigma(\mathcal{S}'_{-j}) + \sum_{z=1}^{m}\texttt{ovlp}_{\mathcal{S}'}(j, \mathcal{M}_z)(w_j - c_z).$$

By Lemma 6.5 (where j_0 is taken to be j) and (52),

$$\Sigma(\mathcal{S}') = \Sigma(\mathcal{S}) + \Delta(\mathcal{S}, t_0, \varepsilon) - \varepsilon(w_j - c_{m'+1}).$$

By Corollary 6.3,

$$\Sigma(\mathcal{S}') = \Sigma(\mathcal{S}) + \varepsilon \cdot (R(\mathcal{S}, t_0) - w_j + c_{m'+1}).$$

Note that the value of the expression $R(\mathcal{S}, t_0) - w_j + c_{m_0+1}$ depends only on the schedule $\mathcal{S}$ and the point e', where $I' = (s', e' = t_0)$. If this value is negative, then by Lemma 6.6, $(I, I', \varepsilon < 0)$-transfer is doable and results in a feasible schedule $\mathcal{S}'$, which satisfies by Lemma 6.7: $\Sigma(\mathcal{S}') > \Sigma(\mathcal{S})$. Thus, a contradiction. If this value is positive, then again by Lemma 6.6, $(I, I', \varepsilon > 0)$-transfer is doable and results in a feasible schedule $\mathcal{S}'$, which satisfies by Lemma 6.7 again the desired inequality: $\Sigma(\mathcal{S}') > \Sigma(\mathcal{S})$. Thus, again a contradiction. If, however this value equals 0, then we can arbitrarily perform either $(I, I', \varepsilon > 0)$-transfer or $(I, I', \varepsilon < 0)$-transfer and Lemmas 6.7 and 6.6 guarantee that we obtain some schedule $\mathcal{S}'$ with $\Sigma(\mathcal{S}') = \Sigma(\mathcal{S})$. This proves the lemma. □

Lemma 6.8. *There exists an optimal schedule that is processor-descending, sequential and has no job splits.*

Proof. Consider an optimal schedule $\mathcal{S}$ that is processor-descending and sequential. Without loss of generality we may assume that $\mathcal{S}$ has the minimum makespan among all optimal processor-descending and sequential schedules. Suppose for a contradiction that (I, I') is the rightmost split in $\mathcal{S}$. By Lemma 6.7, the processor-descending and sequential schedule $\mathcal{S}'$ resulting from the $(I, I', \varepsilon < 0)$-transfer satisfies $\Sigma(\mathcal{S}') = \Sigma(\mathcal{S})$. However, since $\varepsilon < 0$, $\mathcal{S}'$ is shorter than $\mathcal{S}$ which contradicts our choice of $\mathcal{S}$. □

6.3 Synchronization

Consider a processor-descending and sequential schedule $\mathcal{S}$ that has no splits and let j be the *last* job in $\mathcal{S}$ that is not synchronized, i.e., the job that has the greatest completion time on shared processors among jobs that are not synchronized. Suppose that j is present on $m_0 \geq 1$ shared processors. Since $\mathcal{S}$ is sequential, j starts and ends on $\mathcal{M}_1, \ldots, \mathcal{M}_{m_0}$ at time points s and e, respectively. Let q be the number of intervals to the right of the interval $I = (s, e)$ in which the piece of j executes. Define

$$0 < |\varepsilon| < \min\left\{m_0(e-s), \frac{m_0}{m_0+1}\left(C^{\mathcal{P}}_{\mathcal{S}}(j) - e\right), \min_{i=0,\ldots,q}\{m_i r_i/2\}\right\} \tag{53}$$

The following operation that we call a *j-synchronization*, performs a transition from $\mathcal{S}$ to a schedule $\mathcal{S}'$.

(S1) If $R(\mathcal{S}, t = e) > 0$, then let $\varepsilon > 0$ and otherwise let $\varepsilon < 0$, where ε satisfies (53).

(S2) Perform $q + 1$ step modification that starts with $\mathcal{S}$, e, and ε.

(S3) Obtain $\mathcal{S}'$ by setting the completion time of j on the private processor to $C^{\mathcal{P}}_{\mathcal{S}'}(j) := C^{\mathcal{P}}_{\mathcal{S}}(j) - \varepsilon$.

Lemma 6.9. *Suppose $\mathcal{S}$ is an optimal processor-descending and sequential schedule with no splits, and with job j which is not synchronized and done in $I = (s, e)$ on shared processors. Then $R(\mathcal{S}, t = e) = 0$*

Proof. By Lemma 6.5, the $q+1$ step modification called by the j-synchronization is doable since (53) implies (30). Also no more than $m_0(e - s) > |\varepsilon|$ of j can be moved from the m_0 shared processors in the interval $I = (s, e)$ to the job's private processor, and no more than $\frac{m_0}{m_0+1}\left(C^{\mathcal{P}}_{\mathcal{S}}(j) - e\right) > |\varepsilon|$ can be moved from the job's private processor to the m_o shared processors. Thus the choice of ε guarantees that $\mathcal{S}'$ is feasible. For each shared processor $i \in \{1, \ldots, m_0\}$, the execution time of j on $\mathcal{M}_i$ changes by ε/m_0 (if $\varepsilon < 0$, then the execution time decreases, otherwise it increases). Hence, for each such i, $\texttt{ovlp}_{\mathcal{S}'}(j, \mathcal{M}_i) = \texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i) + \varepsilon/m_0$. We can hence represent the total weighted overlap of $\mathcal{S}'$ as follows:

$$\begin{aligned}\Sigma(\mathcal{S}') &= \Sigma(\mathcal{S}'_{-j}) + \sum_{i=1}^{m_0} \texttt{ovlp}_{\mathcal{S}'}(j, \mathcal{M}_i) \cdot (w_j - c_i) \\ &= \Sigma(\mathcal{S}'_{-j}) + \sum_{i=1}^{m_0} \texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i) \cdot (w_j - c_i) + \frac{\varepsilon}{m_0}\sum_{i=1}^{m_0}(w_j - c_i).\end{aligned}$$

By Lemma 6.5,

$$\Sigma(\mathcal{S}') = \Sigma(\mathcal{S}) + \Delta(\mathcal{S}, t, \varepsilon).$$

Note that in the above $\texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i) = 0$ for each $i > m_0$ and hence $\sum_{i=1}^{m} \texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i)(w_j - c_i) = \sum_{i=1}^{m_0} \texttt{ovlp}_{\mathcal{S}}(j, \mathcal{M}_i) \cdot (w_j - c_i)$. Thus, by Corollary 6.3, $\Sigma(\mathcal{S}') = \Sigma(\mathcal{S}) + \varepsilon R(\mathcal{S}, t)$. By definition of j-synchronization we have $R(\mathcal{S}, t) = 0$ since otherwise $\Sigma(\mathcal{S}') > \Sigma(\mathcal{S})$ which contradicts our choice of $\mathcal{S}$. This proves the lemma. □

We are now ready to complete the proof that there exist optimal schedules that are synchronized.

Proof of Theorem 1.1. Consider an optimal schedule $\mathcal{S}$ that is processor-descending, sequential and without splits. Without loss of generality we may assume that $\mathcal{S}$ has minimum makespan among all optimal processor-descending, sequential schedules and without splits. Suppose for a contradiction that $\mathcal{S}$ is not synchronized. Let j be the last job that is not synchronized, and let a piece of j be executed in the interval $I = (s, e)$ on shared processors in $\mathcal{S}$. By Lemma 6.9, $R(\mathcal{S}, t = e) = 0$. Do the j-synchronization with $\varepsilon < 0$ and meeting the condition (53). For the resulting schedule we have $\Sigma(\mathcal{S}') = \Sigma(\mathcal{S})$ according to Corollary 6.3. Moreover, $\mathcal{S}'$ is processor-descending, sequential schedule and without splits. However, since $\varepsilon < 0$, $\mathcal{S}'$ is shorter than $\mathcal{S}$ which contradicts our choice of $\mathcal{S}$. □

7 Conclusions and Open Problems

Our first open problem regards the complexity of the problem. The complexity question remains open even for the single machine case, i.e., the $m = 1$ case [6]. Note however that the problem with *SP* jobs mode (recall that this is the problem variant where each job may use at most one shared processor) is NP-complete in the strong sense [5], and no approximation algorithm with guaranteed worst case ratio is know for the problem. The structural characterization shown in this paper for the *MP* job mode (recall that this is the problem variant where each job may use many, possibly all, shared processor simultaneously) indicates, intuitively speaking, that in this mode schedules for $m > 1$ shared processors 'resemble' schedules on a single shared processor in the sense that in both cases the jobs that appear on the shared processors have certain ordering: once one job finishes on all shared processors it uses, another job starts exclusively using all shared processors it requires. However the numbers of shared processors used by the jobs may be different since the jobs later in the sequence may consider some shared processor too expensive to use. Therefore, with respect to that the *SP* and *MP* modes behave very differently.

Our approximation ratio of $\frac{1}{2} + \frac{1}{4(m+1)}$ obtained for arbitrary number $m \geq 1$ of shared processors improves the previously known approximation ratio, see [6], from $\frac{1}{2}$ to $\frac{5}{8}$ in the single shared processor case. We leave an open question whether the approximation ratio provided by Theorem 3.3 is the best possible, both for multiple shared processors and for a single shared processor.

Acknowledgements

This research has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant OPG0105675.

References

[1] E. J. Anderson. A new continuous model for job–shop scheduling. *International Journal of System Science*, 12:1469–1475, 1981.

[2] T. Aydinliyim and G.L. Vairaktarakis. *Planning Production and Inventories in the Extended Enterprise*, chapter Sequencing Strategies and Coordination Issues in Outsourcing and Subcontracting Operations, pages 269–320. Springer, 2011.

[3] V. Bharadwaj, D. Ghose, and T.G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6:7–17, 2003.

[4] J. Blazewicz, M. Drabowski, and Weglarz J. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. Comput.*, C-35:389–393, 1986.

[5] D. Dereniowski and W. Kubiak. Shared multi-processor scheduling. *European Journal of Operational Research*, 261(2):503–514, 2017.

[6] D. Dereniowski and W. Kubiak. Shared processor scheduling. *Journal of Scheduling*, doi.org/10.1007/s10951-018-0566-0, 2018.

[7] M. Drozdowski. *Scheduling for parallel processing*. Springer, 2009.

[8] B. Hezarkhani and W. Kubiak. Decentralized subcontractor scheduling with divisible jobs. *J. Scheduling*, 18(5):497–511, 2015.

[9] J.B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41:338–350, 1993.

[10] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[11] G. L. Vairaktarakis. Noncooperative games for subcontracting operations. *Manufacturing and Service Operations Management*, 15:148–158, 2013.

[12] G.L. Vairaktarakis and T. Aydinliyim. Centralization versus competition in subcontracting operations. Technical Memorandum Number 819, Case Western Reserve University, 2007.