



Shared processor scheduling

Dariusz Dereniowski¹ · Wiesław Kubiak²

Published online: 13 April 2018
© The Author(s) 2018

Abstract

We study the shared processor scheduling problem with a single shared processor to maximize total weighted overlap, where an overlap for a job is the amount of time it is processed on its private and shared processor in parallel. A polynomial-time optimization algorithm has been given for the problem with equal weights in the literature. This paper extends that result by showing an $O(n \log n)$ -time optimization algorithm for a class of instances in which non-decreasing order of jobs with respect to processing times provides a non-increasing order with respect to weights—this instance generalizes the unweighted case of the problem. This algorithm also leads to a $\frac{1}{2}$ -approximation algorithm for the general weighted problem. The complexity of the weighted problem remains open.

Keywords Divisible jobs · Scheduling · Shared processor

1 Introduction

We begin with a motivating example for the shared processor scheduling problem before we describe it generally and define formally. Consider three orders (jobs) of size $p_1 = 52$, $p_2 = 12$, and $p_3 = 26$ units. These orders belong to three competing agents (manufactures), each with its own *private* processor \mathcal{P}_1 , \mathcal{P}_2 , and \mathcal{P}_3 , respectively. If each processor produces one unit per unit of time, the order's earliest *completion* times will be 52, 12, and 26, respectively, provided that the private processors start processing their orders right away, i.e., at time 0, and experience no preemptions resulting in undesirable delays. Naturally these completion times cannot be shortened without additional processors. Now suppose a subcontractor with a single processor \mathcal{M} , capable of processing at most one order at a time, offers \mathcal{M} to be shared by all agents. We refer to \mathcal{M} as the *shared* processor. Then the agents may consider subcontracting parts of their orders to the subcontractor but only if this reduces their order's completion times. Otherwise, they will obviously do better by

using only their own private processors. Therefore, schedules that allow the subcontractor to process an order on \mathcal{M} while the order's private processor remains idle are undesirable to the order's agent since they do not reduce the order's completion time, and moreover they are more expensive to the agent who needs to unnecessarily pay for using \mathcal{M} when its private processor is idle. These requirements imply that each order is either done on its private processor only or on its private processor and in *parallel* on \mathcal{M} (resulting in *overlap*) at any time in a desirable schedule. Hence, only the overlap can reduce completion times and thus agents strive to have (collectively) as much of it as possible. Clearly, the overlap is beneficial to the agents, by reducing completion times, and at the same time to the subcontractor, by payments from agents for using \mathcal{M} . Observe that while preemptions on private processor are not advantageous, one cannot *a priori* rule out possible advantages from preemptions on the shared processor where an order could be done in disjoint time intervals, each overlapping with the execution on the order's private processor. However, we prove later that such preemptions are not advantageous either. Thus, feasible schedules can be limited to non-preemptive schedules only. Finally, the subcontractor needs to find a feasible schedule that maximizes the total (weighted) overlap. In the motivating example such schedule would have order 2 scheduled in (0,6), order 3 in (6,16), and order 1 in (16,34) on \mathcal{M} . Observe that order 2 is also done in (0,6) on \mathcal{P}_2 , order 3 in (0,16) on \mathcal{P}_3 , and order 1 in (0,34) on \mathcal{P}_1 , see Fig. 1a. This schedule is optimal for the

✉ Dariusz Dereniowski
deren@eti.pg.edu.pl

Wiesław Kubiak
wkubiak@mun.ca

¹ Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk, Poland

² Faculty of Business Administration, Memorial University, St. John's, Canada

unweighted case. Generally, the orders may have individual weights w_1 , w_2 , and w_3 , respectively. This may be because the payoffs to agents resulting from order completion time savings may be order-dependent, or the subcontractor may charge different rates to different orders for using the shared processor.

Generally, we consider a subcontracting system where each agent j has a job with processing time p_j to be executed. Such an agent can perform the work by itself on its private processor, in which case the job completes after p_j units of time, or it can send (subcontract) a part (overlap) of length $t_j \leq p_j/2$ of this job to a subcontractor for processing on a shared processor. The subcontractor needs to complete this part of agent's j job by $p_j - t_j$ bearing in mind that the shared processor can do at most one job at a time which obviously constrains the subcontractor. The speedup in terms of the completion time that the agent achieves in this scenario is exactly t_j , or in other words, the work of agent j is completed at time moment $p_j - t_j$. Whenever $t_j > 0$, the subcontractor is paid by agent j : the payoff for executing t_j units of j -th agent's job is $t_j w_j$. The goal of the subcontractor is to maximize its total weighted overlap (or total weighted payoff to subcontractor, or total weighted time savings to agents depending on interpretation). Thus, in this subcontracting system all agents try to minimize completion times of their jobs (the parameters p_j and w_j are fixed for each agent j) by commissioning the biggest possible part of their jobs to the subcontractor. The subcontractor is the party that decides the overlap t_j to maximize the total weighted overlap.

The shared processor scheduling problem can be placed in a wider context of scheduling with the presence of private (local) processors (machines), available only to a particular job or a set of jobs, and shared (global) processors that are available to all jobs. Then, some additional rules are given in order to specify the conditions under which a job can gain access to a shared processor in such systems. These systems can be run as either centralized or decentralized. The former typically has a single optimization criterion forcing all parties to achieve the same goal. The latter emphasizes that each party is trying to optimize its own goal, which may (and often does) lead to problems having no solutions which are optimal for each agent (job) individually. These problems can be seen as multi-criteria optimization or coordination problems. The latter can be further subdivided into problems in which agents have complete knowledge about resources of other agents (complete information games) and problems without such a complete knowledge (distributed systems) in the search for coordinating mechanisms. This research falls into the category of centralized problems as the subcontractor is deciding on the schedule that reflects its best interest.

The outline of this paper is as follows. In the next section, we briefly survey the related work to provide a state of the art overview. Section 3 gives a formal statement of the

shared processor scheduling problem, and it introduces the necessary notation. Then, in Sect. 4, we recall some facts related to the problem, mainly the fact that when computing optimal schedules one may restrict attention to schedules that are called *synchronized*. This generally greatly simplifies the formal arguments and algorithmic approach. Section 5 considers a restricted version of the problem in which it is assumed that for any pair of jobs, neither of the jobs can have weight and processing time to be strictly smaller than the other. We give an $O(n \log n)$ -time optimization algorithm for this case, and we use it subsequently as a building block to obtain an $O(n \log n)$ -time $1/2$ -approximation algorithm for the general case in Sect. 6.

2 Related work

The shared processor scheduling problem has recently been studied by Vairaktarakis and Aydinliyim (2007), Hezarkhani and Kubiak (2015), and Dereniowski and Kubiak (2017). Vairaktarakis and Aydinliyim (2007) consider the unweighted problem with a single shared processor and 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. Vairaktarakis and Aydinliyim (2007) prove 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. Hezarkhani and Kubiak (2015) observes that this algorithm also gives optimal solutions to the preemptive unweighted problem where more than one interval can be used by a job on the shared processor. Dereniowski and Kubiak (2017) considers shared multi-processor problem proving its strong NP-hardness and giving an efficient, polynomial-time algorithm for the shared multi-processor problem with equal weights. Also, it is shown in Dereniowski and Kubiak (2017) that synchronized optimal schedules always exist for weighted multi-processor instances. Vairaktarakis and Aydinliyim (2007), Vairaktarakis (2013), and Hezarkhani and Kubiak (2015) also study decentralized subcontracting systems focusing on coordinating mechanisms to ensure their efficiency. Hezarkhani and Kubiak (2015) show such coordination mechanism for the unweighted problem and give examples where such schemes do not exist for the weighted problem.

The motivation to study the shared processor scheduling problem comes from diverse applications. Vairaktarakis and Aydinliyim (2007) consider it in the context of supply chains where subcontracting allows jobs to reduce their completion times by using a shared subcontractor's processor. Bharadwaj et al. (2003) use the divisible load scheduling to reduce

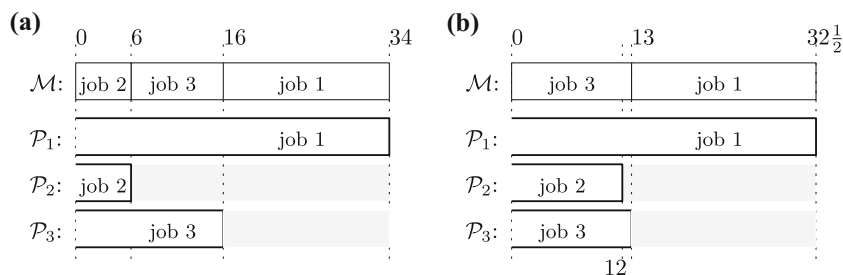


Fig. 1 **a** An optimal schedule for three unweighted jobs with of processing times $p_1 = 52, p_2 = 12$ and $p_3 = 26$ (the total weighted overlap of this schedule is 34); **b** an optimal schedule when the jobs have weights $w_1 = 2, w_2 = 1, w_3 = 3$ (the total weighted overlap equals $\frac{p_3}{2}w_3 + \frac{p_1-p_3/2}{2}w_1 = 78$)

a job completion time in parallel and distributed computer systems, and Anderson (1981) 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 (2017) for more details on these applications.

We also remark multi-agent scheduling models in which each agent has its own optimality criterion and performs actions aimed at optimizing it. In these models, being examples of decentralized systems, agents usually have a number of non-divisible jobs to execute (depending on the optimization criterion this may be seen as having one divisible job, but restricted by allowing preemptions only at certain specified points). For minimization of weighted total completion time in such models see Lee et al. (2009) and weighted number of tardy jobs see Cheng et al. (2006). Bukchin and Hanany (2007) give an example of a game-theoretic analysis to a problem of this type. For overviews and further references on the multi-agent scheduling we refer to the book by Agnetis et al. (2014).

3 Problem formulation

We are given a set \mathcal{J} of n preemptive jobs. Each job $j \in \mathcal{J}$ has its processing time p_j and weight w_j . With each job $j \in \mathcal{J}$ we associate its *private* processor denoted by \mathcal{P}_j . Moreover, there exists a single shared processor, denoted by \mathcal{M} , that is available for all jobs. We follow the convention and notation from Dereniowski and Kubiak (2017) to formulate the problem in this paper.

A schedule \mathcal{S} is *feasible* if it satisfies the following conditions:

- each job $j \in \mathcal{J}$ executes non-preemptively in a *single* time interval $(0, C_{\mathcal{S}}^{\mathcal{P}}(j))$ on its private processor and there is a (possibly empty) collection \mathcal{I}_j of open, non-empty intervals such that j executes non-preemptively in each time interval $I \in \mathcal{I}_j$ on the shared processor,

- for each job $j \in \mathcal{J}$,

$$C_{\mathcal{S}}^{\mathcal{P}}(j) + \sum_{I \in \mathcal{I}_j} |I| = p_j,$$

- the time intervals in $\bigcup_{j \in \mathcal{J}} \mathcal{I}_j$ are pairwise disjoint (i.e., at most one job on \mathcal{M} at a time).

Given a feasible schedule \mathcal{S} , for each job $j \in \mathcal{J}$ we call any maximal time interval in which j executes on both private \mathcal{P}_j and shared \mathcal{M} simultaneously an *overlap*. Observe that this definition allows $I \in \mathcal{I}_j$ not to be an overlap. Thus, the definition allows for feasible schedules that have jobs done on the shared processor but not on their private processors at the same time. These schedules may not be desirable (because they are not optimal) for the jobs, but they are included by the definition to remain consistent with the definition in Dereniowski and Kubiak (2017). The total overlap t_j of job j equals the sum of lengths of all overlaps for j . The *total weighted overlap* of \mathcal{S} equals

$$\Sigma(\mathcal{S}) = \sum_{j \in \mathcal{J}} t_j w_j.$$

A feasible schedule that maximizes the total weighted overlap is called *optimal*. Figure 1 gives examples of optimal schedules.

The formulation of our *weighted single-processor scheduling* problem (WSPPS) is as follows.

Instance A set of weighted jobs \mathcal{J} with arbitrary given processing times.

Goal Find an optimal schedule for \mathcal{J} .

4 Preliminaries

The section provides a brief discussion of the main characteristics of optimal schedules on the shared processor. These

characteristics simplify the formal arguments and algorithmic approach in Sects. 5 and 6.

Let \mathcal{S} be a feasible schedule. For j such that $\mathcal{I}_j \neq \emptyset$, we denote by $s_{\mathcal{S}}^{\mathcal{M}}(j)$ and $C_{\mathcal{S}}^{\mathcal{M}}(j)$ the earliest and the latest time points, respectively, in which j executes on \mathcal{M} , i.e.,

$$s_{\mathcal{S}}^{\mathcal{M}}(j) = \inf \bigcup_{I \in \mathcal{I}_j} I, \quad C_{\mathcal{S}}^{\mathcal{M}}(j) = \sup \bigcup_{I \in \mathcal{I}_j} I.$$

For brevity we take $s_{\mathcal{S}}^{\mathcal{M}}(j) = C_{\mathcal{S}}^{\mathcal{M}}(j) = 0$ if $\mathcal{I}_j = \emptyset$, i.e., j executes on its private processor only. Whenever $s_{\mathcal{S}}^{\mathcal{M}}(j) < C_{\mathcal{S}}^{\mathcal{M}}(j)$, i.e., some non-empty part of a job j executes on \mathcal{M} , then we say that the job j appears on \mathcal{M} in schedule \mathcal{S} . If, in a schedule \mathcal{S} , there is no idle time on the shared processor in time interval

$$\left[0, \max\{C_{\mathcal{S}}^{\mathcal{M}}(j) \mid j \in \mathcal{J}\} \right],$$

then we say that \mathcal{S} has no gaps. We have the following results from the literature.

Observation 1 (Dereniowski and Kubiak 2017) *There exists an optimal schedule that has no gaps.*

A schedule \mathcal{S} is called *non-preemptive* if each job j executes in \mathcal{S} in time interval $[s_{\mathcal{S}}^{\mathcal{M}}(j), C_{\mathcal{S}}^{\mathcal{M}}(j)]$ on the shared processor. We say that a schedule \mathcal{S} is *synchronized*¹ if it satisfies the following conditions:

- (i) \mathcal{S} is non-preemptive and has no gaps,
- (ii) for each job j that appears on the shared processor it holds $C_{\mathcal{S}}^{\mathcal{M}}(j) = C_{\mathcal{S}}^{\mathcal{P}}(j)$.

Theorem 1 (Dereniowski and Kubiak 2017) *There exists an optimal synchronized schedule.*

Consider a synchronized schedule \mathcal{S} . Let $A = \{j_1, \dots, j_k\} \subseteq \mathcal{J}$ be the set of all jobs that appear on \mathcal{M} in \mathcal{S} , where the jobs are ordered according to increasing order of their completion times in \mathcal{S} , i.e., $C_{\mathcal{S}}^{\mathcal{M}}(j_1) < \dots < C_{\mathcal{S}}^{\mathcal{M}}(j_k)$. Note that the set A and the order are enough to determine the schedule \mathcal{S} . Indeed, given the order (j_1, \dots, j_k) we obtain that for each $i \in \{1, \dots, k\}$ (by proceeding with increasing values of i in a synchronized schedule),

$$s_{\mathcal{S}}^{\mathcal{M}}(j_i) = C_{\mathcal{S}}^{\mathcal{M}}(j_{i-1}) \quad \text{and} \\ C_{\mathcal{S}}^{\mathcal{M}}(j_i) = C_{\mathcal{S}}^{\mathcal{P}}(j_i) = \left(p_i + s_{\mathcal{S}}^{\mathcal{M}}(j_i) \right) / 2,$$

¹ We remark that the definition of synchronized schedule that appears in Dereniowski and Kubiak (2017) uses as an intermediate step in the analysis a weaker condition $C_{\mathcal{S}}^{\mathcal{M}}(j) \leq C_{\mathcal{S}}^{\mathcal{P}}(j)$ [such schedules are called *normal* in Dereniowski and Kubiak (2017)] which can be omitted here due to the stronger condition in (ii) in our definition of synchronized schedule. Also, it has been proved, cf. Observation 3.2 in Dereniowski and Kubiak (2017), that normal schedule has no gaps which justifies our condition (i).

where $C_{\mathcal{S}}^{\mathcal{M}}(j_0) = 0$. Hence, the start times and completion times can be iteratively computed for all jobs in a synchronized schedule. Thus, for synchronized schedules we write for brevity $\mathcal{S} = (j_1, \dots, j_k)$ to refer to the schedule computed above.

5 A $O(n \log n)$ -time optimal algorithm for antithetical instances

We call an instance \mathcal{J} of the problem *antithetical* if for any two jobs i and j it holds: $p_i \leq p_j$ implies $w_i \geq w_j$. We call a schedule \mathcal{S} *processing time ordered* if $\mathcal{S} = (j_1, \dots, j_n)$, where $p_{j_i} \leq p_{j_{i+1}}$ for each $i \in \{1, \dots, n - 1\}$. In other words, all jobs are present on the shared processor, they are arranged according to non-decreasing order of their processing times and the schedule is synchronized. The definition is correct since we observe that by the construction at the end of Sect. 4, \mathcal{S} is synchronized and all jobs from \mathcal{J} appear on the shared processor, see Vairaktarakis and Aydinliyim (2007) and Hezarkhani and Kubiak (2015). We now prove that any optimal solution for an antithetical instance is a processing time ordered schedule. We remark that this algorithm generalizes the previously known solutions for the unweighted case ($w_1 = \dots = w_n$) from Hezarkhani and Kubiak (2015), Vairaktarakis and Aydinliyim (2007).

Before giving the main result of this section in Lemma 3, we prove a technical lemma which shows how to transform a schedule with $k - 1$ synchronized jobs, i.e., jobs completing on the shared and private processors at the same time, into a schedule that has k jobs synchronized. This transformation will be used in the proof of Lemma 3.

Lemma 2 *Let \mathcal{S}' be a non-preemptive schedule with no gaps. Consider a subset of jobs j_1, \dots, j_k for some $k \in \{1, \dots, n\}$ and let $i \in \{1, \dots, k - 1\}$. Suppose that $p_{j_i} \leq p_{j_{i+1}} \leq \dots \leq p_{j_k}$, and $C_{\mathcal{S}'}^{\mathcal{M}}(j_{i'}) = C_{\mathcal{S}'}^{\mathcal{P}}(j_{i'})$ for each $i' \in \{1, \dots, k\} \setminus \{i\}$, and one of the two cases holds:*

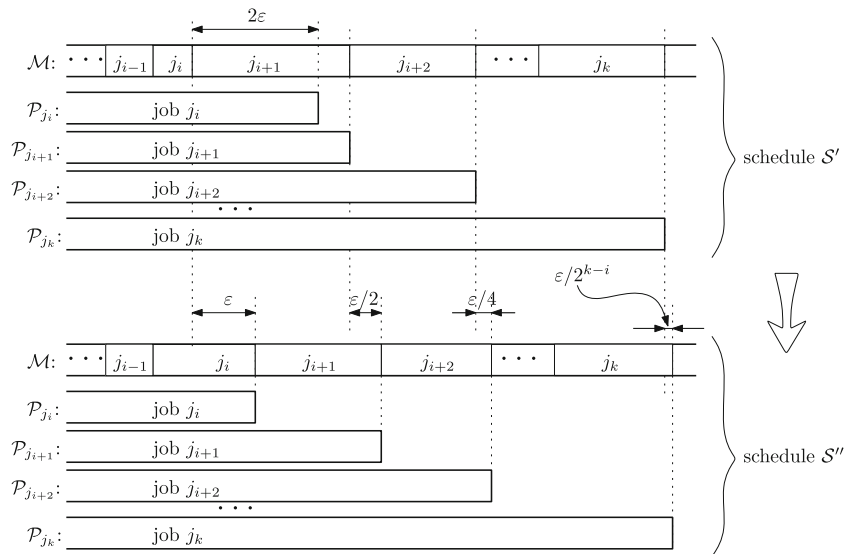
- (i) *exactly the jobs j_1, \dots, j_k appear on the shared processor in \mathcal{S}' in this order and $C_{\mathcal{S}'}^{\mathcal{M}}(j_i) < C_{\mathcal{S}'}^{\mathcal{P}}(j_i)$, or*
- (ii) *exactly the jobs $j_1, \dots, j_{i-1}, j_{i+1}, \dots, j_k$ appear on the shared processor in \mathcal{S}' in this order, $p_{j_i} > s_{\mathcal{S}'}^{\mathcal{M}}(j_{i+1})$ and j_i does not appear on \mathcal{M} .*

Then, the synchronized schedule $\mathcal{S}'' = (j_1, \dots, j_k)$ satisfies

$$\Sigma(\mathcal{S}'') = \Sigma(\mathcal{S}') + \varepsilon w_{j_i} - \varepsilon \sum_{i'=i+1}^k \frac{w_{j_{i'}}}{2^{i'-i}}$$

for some $\varepsilon > 0$.

Fig. 2 Transformation from S' to S'' in proof of Lemma 2



Proof The transformation described in the proof is shown in Fig. 2. Informally speaking, we obtain S'' by moving a part of j_i from its private processor to the shared processor \mathcal{M} , so that j_i becomes synchronized, i.e., it ends on both of these processors at the same time. This move forces all jobs that follow j_i on \mathcal{M} , i.e., the jobs j_{i+1}, \dots, j_k , to be postponed on \mathcal{M} as described below. Note that the transformation is exactly the same for both case (i) and case (ii) but Fig. 2 depicts case (i) only.

Let

$$\varepsilon = \frac{1}{2} \left(C_{S'}^{\mathcal{P}}(j_i) - s_{S'}^{\mathcal{M}}(j_{i+1}) \right).$$

Note that $\varepsilon > 0$ by the lemma assumption. We now obtain a schedule S'' as follows. For $i' \in \{i + 1, \dots, k\}$, the start of the job $j_{i'}$ is postponed by $\varepsilon/2^{i'-i-1}$ on \mathcal{M} , i.e.,

$$s_{S''}^{\mathcal{M}}(j_{i'}) = s_{S'}^{\mathcal{M}}(j_{i'}) + \frac{\varepsilon}{2^{i'-i-1}} = s_{S'}^{\mathcal{M}}(j_{i'}) + \frac{2\varepsilon}{2^{i'-i}}, \quad (1)$$

the completion of job $j_{i'}$ is postponed by $\varepsilon/2^{i'-i}$ on \mathcal{M} and on $\mathcal{P}_{j_{i'}}$, i.e.,

$$C_{S''}^{\mathcal{M}}(j_{i'}) = C_{S'}^{\mathcal{P}}(j_{i'}) = C_{S'}^{\mathcal{M}}(j_{i'}) + \frac{\varepsilon}{2^{i'-i}}. \quad (2)$$

Thus, $j_{i'}$ completes on both \mathcal{M} and $\mathcal{P}_{j_{i'}}$ at the same time for $i' \in \{i + 1, \dots, k\}$. Moreover, $s_{S''}^{\mathcal{M}}(j_i) = s_{S'}^{\mathcal{M}}(j_i)$ in case (i) and $s_{S''}^{\mathcal{M}}(j_i) = s_{S'}^{\mathcal{M}}(j_{i+1})$ in case (ii). Then, let $C_{S''}^{\mathcal{M}}(j_i) = s_{S''}^{\mathcal{M}}(j_{i+1})$ (thus in both cases the duration of j_i on \mathcal{M} increases by ε) and the remaining part of length $C_{S'}^{\mathcal{P}}(j_i) - \varepsilon$ of j_i executes on \mathcal{P}_{j_i} so that j_i completes on both \mathcal{M} and \mathcal{P}_{j_i} at the same time as well. Finally, S'' and S' are identical in time interval $[0, C_{S'}^{\mathcal{M}}(j_i)]$. From the transformation we obtain that

$$\Sigma(S'') = \Sigma(S') + \varepsilon w_{j_i} - \varepsilon \sum_{i'=i+1}^k \frac{w_{j_{i'}}}{2^{i'-i}}. \quad (3)$$

Indeed, the amount of j_i present on \mathcal{M} increases by ε , and the amount of $j_{i'}$ decreases by $\varepsilon/2^{i'-i}$ for each $i' \in \{i + 1, \dots, k\}$.

Clearly the transformation ensures that S'' is non-preemptive and has no gaps as long as S' is non-preemptive and has no gaps, which holds by assumption. By lemma's assumption, the execution time of j_i in S'' is well defined. Thus, it remains to argue that the jobs j_{i+1}, \dots, j_k are present on \mathcal{M} in S'' , i.e., that the execution times of these jobs are properly defined. To that end we need to prove that

$$C_{S''}^{\mathcal{M}}(j_{i'}) - s_{S''}^{\mathcal{M}}(j_{i'}) = C_{S'}^{\mathcal{M}}(j_{i'}) - s_{S'}^{\mathcal{M}}(j_{i'}) - \frac{\varepsilon}{2^{i'-i}} > 0,$$

for each $i' \in \{i + 1, \dots, k\}$. By induction on i' we get

$$C_{S'}^{\mathcal{M}}(j_{i'}) - s_{S'}^{\mathcal{M}}(j_{i'}) = \frac{p_{j_{i'}}}{2} - \sum_{\ell=i+1}^{i'-1} \frac{p_{j_\ell}}{2^{i'-\ell+1}} - \frac{x}{2^{i'-i}}$$

for each $i' \in \{i + 1, \dots, k\}$, where $x = s_{S'}^{\mathcal{M}}(j_{i+1})$. Since $p_{j_i} \leq p_{j_{i+1}} \leq \dots \leq p_{j_k}$, we have

$$C_{S'}^{\mathcal{M}}(j_{i'}) - s_{S'}^{\mathcal{M}}(j_{i'}) \geq \frac{p_{j_{i'}}}{2^{i'-i}} - \frac{x}{2^{i'-i}}.$$

Thus, it suffices to show that

$$p_{j_{i'}} - x > \varepsilon.$$

By definition of ε this boils down to showing that

$$2p_{j_{i'}} > C_{S'}^{\mathcal{P}}(j_i) + x.$$

This inequality holds since $p_{j_i} \geq C_{S'}^{\mathcal{P}}(j_i) > x$ and thus we get

$$2p_{j_{i'}} \geq 2p_{j_i} > C_{S'}^{\mathcal{P}}(j_i) + x,$$

which completes the argument and proves that S'' is feasible. This implies that S'' is feasible and synchronized. Observe that though we assumed $C_{S'}^{\mathcal{M}}(j_i) - s_{S'}^{\mathcal{M}}(j_i) > 0$ in case (i) of the lemma, we have not used this assumption in the proof. Hence, the proof also works for case (ii) in which one can equivalently take $C_{S'}^{\mathcal{M}}(j_i) - s_{S'}^{\mathcal{M}}(j_i) = 0$. We use this observation in the proof of Lemma 3 where the case of j_i absent from \mathcal{M} in S' needs to be considered (i.e., when we refer to case (ii) of this lemma). \square

We remark that it is still premature to conclude from Lemma 2 that no job is missing on \mathcal{M} in an optimal schedule S . This would simplify the proof of Lemma 3 as it would eliminate case (5) in the proof. However, an insertion of a job missing on \mathcal{M} in S requires the jobs on \mathcal{M} to the right of the insertion point to be ordered according to the non-decreasing order of processing times. Otherwise, if this key assumption in Lemma 2 is not met, then the synchronized schedule S'' from Lemma 2 may not be feasible or may not satisfy the formula for its total weighted overlap given in the lemma. Unfortunately, at this point we cannot guarantee that S satisfies the assumption.

Lemma 3 *An optimal schedule for an antithetical instance of the problem WSPS is a processing time ordered schedule.*

Proof Let S be an optimal schedule for an antithetical instance \mathcal{J} . By Theorem 1 we can assume that S is synchronized. We assume without loss of generality that the jobs in $\mathcal{J} = \{1, \dots, n\}$ are ordered in non-decreasing order of their processing times, i.e., $p_1 \leq p_2 \leq \dots \leq p_n$. Let $A \subseteq \mathcal{J}$ be the set of jobs that appear on the shared processor in S . Let $\pi(1), \pi(2), \dots, \pi(k), k = |A|$, be the order of jobs on the shared processor in S , i.e., $S = (\pi(1), \pi(2), \dots, \pi(k))$. We have $n \in A$. Indeed, by Vairaktarakis and Aydinliyim (2007) and Hezarkhani and Kubiak (2015) regardless of the ordering of the jobs $1, \dots, n - 1$, it is always possible to add the job n as the last one on \mathcal{M} and increase the total overlap.

If for an index $i \in \{1, \dots, k - 1\}$ it holds

$$p_{\pi(i)} > p_{\pi(i+1)} \tag{4}$$

or if for an index $i \in \{1, \dots, k\}$ it holds

$$\bar{j} = \pi(i) - 1 \notin A, \tag{5}$$

then the job $\pi(i)$ is called a *violator*. Informally speaking, the condition (4) detects whether some jobs in S do not follow the non-decreasing order of the processing times required

by the processing time ordered schedule. The condition (5) detects whether some job is missing on the shared processor in S , namely, the job \bar{j} should precede the job $\pi(i)$ on \mathcal{M} in the processing time ordered schedule but \bar{j} does not appear on \mathcal{M} in S .

Since $n \in A$, we have that if there is no violator, then S is a processing time ordered schedule and the proof of the lemma is thus completed. Therefore, we assume in the following that at least one violator exists. Then, the *maximal* index $i \in \{1, \dots, k\}$ such that $\pi(i)$ is the violator is called the *violation point* in S .

Among all optimal and synchronized schedules for \mathcal{J} we take S to satisfy the following:

- (a) $|A|$ is maximum, and
- (b) with respect to (a): the violation point of S is minimum.

Let $i > 0$ be the violation point of S . By definition, we have that one of the cases (4) or (5) holds. We should arrive at a contradiction in both cases, and we start by analyzing the case of (4), that is, we assume that (4) holds for the violation point i . For antithetical instances we have $w_{\pi(i)} \leq w_{\pi(i+1)}$. Also, for convenience and without loss of generality we denote $j = \pi(i + 1)$ and $j + 1 = \pi(i)$. Thus, $p_{j+1} > p_j$ by (4). In the next two paragraphs we describe a transition from S to a new schedule S' . This transition is depicted in Fig. 3.

Consider the intervals in which the two jobs j and $j + 1$ execute on \mathcal{M} in S and suppose that $j + 1$ starts at t on \mathcal{M} , $s_S^{\mathcal{M}}(j + 1) = t$. Since S is synchronized, the length $l = C_S^{\mathcal{M}}(j) - s_S^{\mathcal{M}}(j + 1)$ of this sequence on the shared processor equals

$$\begin{aligned} l &= \left(C_S^{\mathcal{M}}(j) - s_S^{\mathcal{M}}(j) \right) + \left(C_S^{\mathcal{M}}(j + 1) - s_S^{\mathcal{M}}(j + 1) \right) \\ &= \frac{p_j - t}{2} + \frac{p_{j+1} - t}{4}, \end{aligned} \tag{6}$$

and its contribution x to the total weighted overlap equals

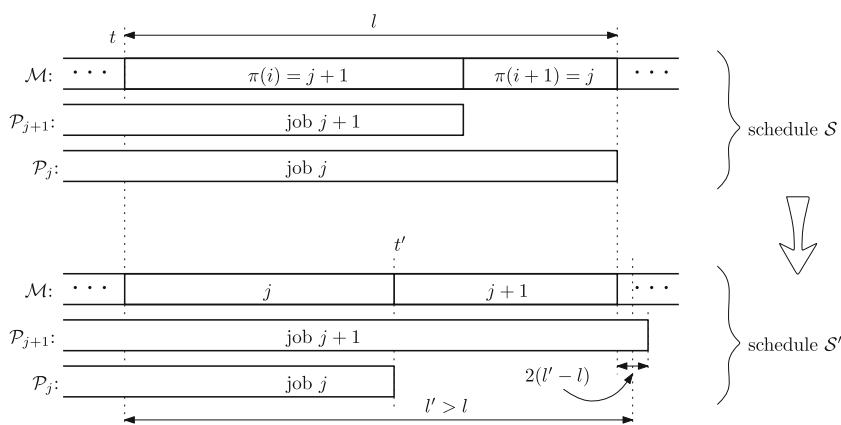
$$\begin{aligned} x &= \left(C_S^{\mathcal{M}}(j + 1) - s_S^{\mathcal{M}}(j + 1) \right) w_{j+1} \\ &\quad + \left(C_S^{\mathcal{M}}(j) - s_S^{\mathcal{M}}(j) \right) w_j \\ &= \frac{p_{j+1} - t}{2} \left(w_{j+1} - \frac{w_j}{2} \right) + \frac{p_j - t}{2} w_j. \end{aligned} \tag{7}$$

Thus, we can express the total weighted overlap of S as follows:

$$\Sigma(S) = c + x \quad \text{for some } c \in \mathbb{R}. \tag{8}$$

Before we formally define S' , we analyze the impact the reversed order of the two jobs j and $j + 1$ in the interval $(t, t + l)$ has on S and its total weighted overlap. Suppose for

Fig. 3 Transition from \mathcal{S} to \mathcal{S}' in proof of Lemma 3



the time being that j starts at t on \mathcal{M} and is followed by the job $j + 1$, and that each of these jobs completes both on \mathcal{M} and its private processor at the same time. Then the length l' of the interval $(t, t + l')$ occupied by these two jobs on the shared processor equals

$$l' = \frac{p_{j+1} - t}{2} + \frac{p_j - t}{4}$$

and its contribution x' to the total weighted overlap equals

$$x' = \frac{p_j - t}{2} \left(w_j - \frac{w_{j+1}}{2} \right) + \frac{p_{j+1} - t}{2} w_{j+1}. \tag{9}$$

Clearly, $l' > l$ for $p_{j+1} > p_j$ by

$$l' - l = \frac{p_{j+1} - p_j}{4}. \tag{10}$$

The job j completes at

$$t' = t + \frac{p_j - t}{2} < t + \frac{p_j - t}{2} + \frac{p_{j+1} - t}{4} = t + l \tag{11}$$

after the exchange. We construct the schedule \mathcal{S}' as follows: \mathcal{S} and \mathcal{S}' are identical in time intervals $[0, t)$ and $(t + l, +\infty)$, the job j executes in time interval

$$(t, t') = (t, t + (p_j - t)/2) \tag{12}$$

in \mathcal{S}' and the job $j + 1$ executes in time interval

$$(t', t + l) = (t + (p_j - t)/2, t + l) \tag{13}$$

on processor \mathcal{M} in \mathcal{S}' . Note that j finishes at the same time on its private and shared processor in \mathcal{S}' while the job $j + 1$ does not have this property. Since $p_{j+1} > p_j$, $j + 1$ completes $(p_{j+1} - p_j)/4$ units later on its private processor than on \mathcal{M} . Thus, \mathcal{S}' is not synchronized; see also Fig. 3. The total

weighted overlap of \mathcal{S}' is then by (6), (12) and (13):

$$\begin{aligned} \Sigma(\mathcal{S}') &= c + (t' - t)w_j + (t + l - t')w_{j+1}, \\ &= c + \frac{p_j - t}{2}w_j + \frac{p_{j+1} - t}{4}w_{j+1} \end{aligned} \tag{14}$$

where c is defined in (8). By (8) and (14) we obtain that the difference (in total weighted overlaps) between \mathcal{S}' and \mathcal{S} is

$$\Sigma(\mathcal{S}') - \Sigma(\mathcal{S}) = \frac{p_{j+1} - t}{4}(w_j - w_{j+1}). \tag{15}$$

Since $p_{j+1} - t > 0$ (this holds since the job $j + 1$ appears on the shared processor in \mathcal{S}) and $w_j \geq w_{j+1}$, we have that $\Sigma(\mathcal{S}') - \Sigma(\mathcal{S}) \geq 0$. Note that if w_j is strictly greater than w_{j+1} , then we obtain the desired contradiction with the optimality of \mathcal{S} . However, if $w_j = w_{j+1}$, then $\Sigma(\mathcal{S}) = \Sigma(\mathcal{S}')$ and we need to use different arguments to arrive at a contradiction.

To that end denote

$$q = \sum_{i'=i+2}^k \frac{w_{\pi(i')}}{2^{i'-i-1}}.$$

We show that

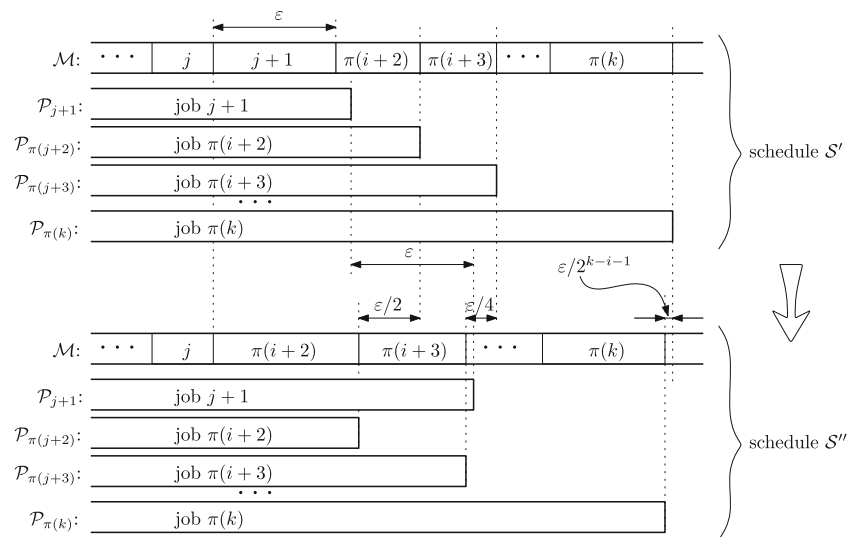
$$w_{j+1} \geq q. \tag{16}$$

Suppose for a contradiction that $w_{j+1} < q$. To obtain a contradiction with this assumption we will convert \mathcal{S}' into a schedule \mathcal{S}'' with strictly greater total weighted overlap, which will contradict the optimality of the original schedule \mathcal{S} . This conversion is described in the next two paragraphs and depicted in Fig. 4. Let

$$\varepsilon = C_{\mathcal{S}'}^{\mathcal{M}}(j + 1) - s_{\mathcal{S}'}^{\mathcal{M}}(j + 1).$$

Observe that $\varepsilon > 0$ by (10). By assumption, \mathcal{S} is synchronized and \mathcal{S} and \mathcal{S}' are identical on \mathcal{M} in time intervals $[C_{\mathcal{S}'}^{\mathcal{M}}(j + 1), +\infty)$. Thus,

Fig. 4 Transition from S' to S'' in proof of Lemma 3 when proving (16)



$$C_{S'}^M(i') = C_{S''}^M(i') = C_{S'}^P(i') = C_{S''}^P(i')$$

for each job i' that appears on \mathcal{M} and completes in S' later than the job $j + 1$. The schedule S'' is defined as follows. Let, S'' and S' be identical on \mathcal{M} in time interval

$$[0, C_{S'}^M(j + 1) - \varepsilon) = [0, s_{S''}^M(j + 1)).$$

Then, the job $j + 1$ is not present on \mathcal{M} in S' , i.e., it executes only on \mathcal{P}_{j+1} in S'' . Finally, for each job $\pi(i')$, $i' \in \{i + 2, \dots, k\}$, we set:

$$\begin{aligned} s_{S''}^M(\pi(i')) &= s_{S'}^M(\pi(i')) - \frac{\varepsilon}{2^{i'-i-2}}, \\ C_{S''}^M(\pi(i')) &= C_{S'}^M(\pi(i')) - \frac{\varepsilon}{2^{i'-i-1}}, \\ C_{S''}^P(\pi(i')) &= C_{S'}^P(\pi(i')) - \frac{\varepsilon}{2^{i'-i-1}}. \end{aligned}$$

Both S'' and S' are the same on $\mathcal{P}_{i'}$ for each $i' \in \mathcal{J} \setminus \{j + 1, \pi(i + 2), \dots, \pi(k)\}$, i.e., on each processor not specified by the formulas above.

Clearly, S'' is feasible and synchronized. To compare its total weighted overlap to that of S' , note that on the one hand the total weighted overlap of S'' decreases by εw_{j+1} in comparison with S' since the job $j + 1$ does not appear on \mathcal{M} in S'' , on the other hand it increases since the subintervals with the jobs that follow $j + 1$ on \mathcal{M} get longer due to disappearance of $j + 1$ from \mathcal{M} . Precisely, we have

$$\begin{aligned} \Sigma(S'') &= \Sigma(S') - \varepsilon w_{j+1} + \varepsilon \sum_{i'=i+2}^k \frac{w_{i'}}{2^{i'-i-1}} \\ &= \Sigma(S') + \varepsilon(q - w_{j+1}) > \Sigma(S') \end{aligned}$$

because $\varepsilon > 0$ and $w_{j+1} < q$ by assumption. Thus, we obtain a contradiction with the optimality of \mathcal{S} (recall that $\Sigma(\mathcal{S}) \leq \Sigma(S')$). Hence, (16) holds.

By Lemma 2, we obtain a synchronized schedule S^* from S' . Moreover by (16) we have $\Sigma(S^*) \geq \Sigma(S')$. In particular, if $\Sigma(S^*) > \Sigma(S')$, then we have immediately a contradiction with the optimality of \mathcal{S} since $\Sigma(S') \geq \Sigma(\mathcal{S})$. On the other hand, if $\Sigma(S^*) = \Sigma(S')$, then the contradiction comes from the selection of \mathcal{S} to be a schedule that minimizes the violation point i . Observe that since the intervals in (12) and (13) are non-empty then all \mathcal{S} , S' , and S^* have the same number $|A|$ of jobs that appear on the shared processor.

We now consider case (5). We can assume

$$p_{\pi(i-1)} \leq p_{\pi(i)} \tag{17}$$

since if the inequality does not hold we get case (4) which we already considered. We assume $p_{\pi(0)} = 0$ in (17). Recall that $\mathcal{S} = (\pi(1), \dots, \pi(k))$ and $\bar{j} \notin A$. By (17) and (5) we have

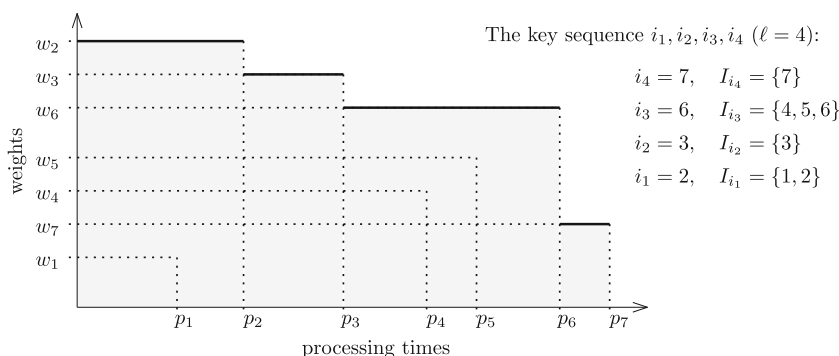
$$C_{S'}^M(\pi(i - 1)) < p_{\pi(i-1)} \leq p_{\bar{j}} \leq p_{\pi(i)} \tag{18}$$

for $i > 1$. We apply Lemma 2(ii) to S' with $j_1 = \pi(1), \dots, j_{i-1} = \pi(i - 1), j_i = \bar{j}$ and $j_{i+1} = \pi(i), \dots$, when $i > 1$ and with $j_1 = \bar{j}, j_2 = \pi(1), j_3 = \pi(2), \dots$ when $i = 1$. Observe that since i is the violation point of \mathcal{S} and (5) holds, we have $w_{\bar{j}} \geq w_{\pi(i')}$ for each $i' \in \{i, \dots, k\}$ in the antithetical instance \mathcal{J} . Therefore,

$$\sum_{i'=i}^k \frac{w_{\pi(i')}}{2^{i'-i+1}} \leq w_{\bar{j}} \sum_{i'=i}^k \frac{1}{2^{i'-i+1}} < w_{\bar{j}}.$$

Hence, by Lemma 2(ii), there exists a schedule S^* for which we have $\Sigma(S^*) > \Sigma(S')$. Thus, we again have a contradic-

Fig. 5 An example of the key sequence (the area of the gray part equals u^*)



tion by the optimality of \mathcal{S} since clearly $\Sigma(\mathcal{S}') = \Sigma(\mathcal{S})$. This completes the proof of the lemma. \square

Due to the fact that some jobs may have equal processing times yet different weights in an antithetical instance not all processing time ordered schedules are optimal for the instance. However any processing time ordered schedule $\mathcal{S} = (j_1, \dots, j_n)$ such that $w_{j_1} \geq \dots \geq w_{j_n}$ is optimal for the instance. To prove that we observe that any maximal sequence of jobs having the same processing times occupies the same interval (s, C) on \mathcal{M} in any processing time ordered schedule. However, the earlier the job appears in the sequence the longer is its overlap, moreover the overlaps in (s, C) remain the same regardless of the sequence of jobs having equal processing times. Hence, the first position should be occupied by the heaviest job, the second position by the second heaviest job etc. among the jobs with the same processing times to ensure total weighted overlap maximization. This gives an $O(n \log n)$ -time optimization algorithm for the antithetical instances.

6 A 1/2-approximation algorithm

Let $0 = q_0 < q_1 < \dots < q_\ell$ and $u_1, \dots, u_\ell \geq 0$ for some $\ell \geq 1$. An *envelope* for q_1, \dots, q_ℓ and u_1, \dots, u_ℓ is a step function of non-negative x defined as follows

$$e(q_1, \dots, q_\ell, u_1, \dots, u_\ell, x) = \begin{cases} u_1 & \text{if } q_0 \leq x \leq q_1 \\ u_2 & \text{if } q_1 < x \leq q_2 \\ \dots & \dots \\ u_\ell & \text{if } q_{\ell-1} < x \leq q_\ell \\ 0 & \text{if } q_\ell < x. \end{cases}$$

The *area* of the envelope e is

$$\sum_{i=1}^{\ell} u_i (q_i - q_{i-1}).$$

Let \mathcal{J} be a set of jobs. Without loss of generality we assume $p_1 \leq \dots \leq p_n$, and for any tie (i.e., for equal

processing times) we assume that the jobs are ordered in non-decreasing order of their weights, i.e., the heaviest tied job comes last in the tie. A sequence of jobs i_1, \dots, i_ℓ for some $\ell \geq 1$, where $1 \leq i_1 < \dots < i_\ell \leq n$, is called a *key sequence* for \mathcal{J} if it satisfies the following conditions:

- (i) $i_\ell = n$,
- (ii) $w_{i_1} > \dots > w_{i_\ell}$,
- (iii) $w_k \leq w_{i_j}$ for each $k \in I_{i_j} = \{i_{j-1} + 1, \dots, i_j\}$ and $j \in \{1, \dots, \ell\}$, where $i_0 = 0$.

Clearly $p_{i_1} < \dots < p_{i_\ell}$, thus $u(p_{i_1}, \dots, p_{i_\ell}, w_{i_1}, \dots, w_{i_\ell}, x)$ is an envelope; we refer to it as the *upper envelope* for \mathcal{J} . Let u^* be the area of the upper envelope for \mathcal{J} . See Fig. 5 for an example.

Note that the key sequence always exists. This follows from the fact that it can be constructed ‘greedily’ by starting with picking the last job of the sequence [(see (i))] and then iteratively selecting the predecessor of the previously selected job so that the predecessor has strictly bigger weight [see (ii)] and satisfies the condition (iii). Also, the key sequence is unique by the same argument.

We have the following simple observation.

Claim For each $k \in \{1, \dots, \ell\}$, $w_{i_k} = \max\{w_j \mid i_k \leq j \leq n\}$. \square

The key sequence for \mathcal{J} defines a synchronized schedule \mathcal{S}_{key} for \mathcal{J} with the set of jobs executed on the shared processor being $\mathcal{J}_{\text{key}} = \{i_1, \dots, i_\ell\}$ and the permutation of the jobs on the processor being $\pi(j) = i_j$ for $j \in \{1, \dots, \ell\}$. The jobs in $\mathcal{J} \setminus \mathcal{J}_{\text{key}}$ are executed on their private processors only. Following our notation introduced in Sect. 4, we get $\mathcal{S}_{\text{key}} = (i_1, \dots, i_\ell)$. We have the following lemma.

Lemma 4 For the schedule \mathcal{S}_{key} it holds $2\Sigma(\mathcal{S}_{\text{key}}) \geq u^*$.

Proof We argue that for each $k \in \{1, \dots, \ell\}$,

$$w_{i_k} \left(C_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) - s_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) \right) \geq w_{i_k} \frac{p_{i_k} - p_{i_{k-1}}}{2}, \tag{19}$$

where $p_{i_0} = 0$. Note that $s_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) \leq p_{i_{k-1}}$ for each $k \in \{1, \dots, \ell\}$. Thus,

$$\begin{aligned} C_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) - s_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) &= \frac{s_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) + p_{i_k}}{2} - s_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) \\ &\geq \frac{p_{i_k} - p_{i_{k-1}}}{2} \end{aligned}$$

for each $k \in \{1, \dots, \ell\}$, which proves (19).

By (19),

$$\begin{aligned} \Sigma(\mathcal{S}_{\text{key}}) &= \sum_{k=1}^{\ell} w_{i_k} \left(C_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) - s_{\mathcal{S}_{\text{key}}}^{\mathcal{M}}(i_k) \right) \\ &\geq \sum_{k=1}^{\ell} w_{i_k} \frac{p_{i_k} - p_{i_{k-1}}}{2} = \frac{u^*}{2}. \end{aligned}$$

□

We now prove that the area u^* of the upper envelope for \mathcal{J} is an upper bound on the total weighted overlap of an optimal solution for \mathcal{J} . By Theorem 1, there exists an optimal synchronized schedule \mathcal{S}_{opt} for \mathcal{J} . Let $\mathcal{J}_{\text{opt}} \subseteq \mathcal{J}$ be the set of jobs that appear on the shared processor in \mathcal{S}_{opt} and let π be the permutation of jobs on the shared processor in \mathcal{S}_{opt} . Thus, we have $\mathcal{S}_{\text{opt}} = (\pi(1), \dots, \pi(|\mathcal{J}_{\text{opt}}|))$. It holds $0 < C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(1)) < \dots < C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(|\mathcal{J}_{\text{opt}}|)) < p_n$ and therefore

$$e \left(C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(1)), \dots, C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(|\mathcal{J}_{\text{opt}}|)), w_{\pi(1)}, \dots, w_{\pi(|\mathcal{J}_{\text{opt}}|)}, x \right).$$

is an envelope for $C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(1)), \dots, C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(|\mathcal{J}_{\text{opt}}|))$ and $w_{\pi(1)}, \dots, w_{\pi(|\mathcal{J}_{\text{opt}}|)}$. Let the area of this envelope be e^* . We have the following key result.

Lemma 5 *It holds $e^* \leq u^*$.*

Proof Observe that for each index $j \in \{1, \dots, |\mathcal{J}_{\text{opt}}|\}$ there exists $\tau(j) \in \{1, \dots, n\}$ such that

$$C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(j)) \leq p_{i_{\tau(j)}}. \tag{20}$$

This follows from condition (i) in definition of key sequence. If, for a given j , there are several jobs $\tau(j)$ that satisfy the above, then take $\tau(j)$ to be the smallest one.

We argue that

$$w_{\pi(j)} \leq w_{i_{\tau(j)}} \tag{21}$$

for each $j \in \{1, \dots, |\mathcal{J}_{\text{opt}}|\}$. Suppose for a contradiction that (21) does not hold. We consider two cases. In the first case let

$$p_{\pi(j)} \leq p_{i_{\tau(j)}}.$$

By condition (iii) in definition of the key sequence and the minimality of $\tau(j)$, $\pi(j)$ does not belong to the key sequence. But then, $w_{\pi(j)} > w_{i_{\tau(j)}}$ implies that there is t such that $p_{\pi(j)} < p_{i_t} < p_{i_{\tau(j)}}$, which contradicts the choice of $\tau(j)$. In the second case let

$$p_{\pi(j)} > p_{i_{\tau(j)}}.$$

Take the minimum index t such that $p_{i_t} \geq p_{\pi(j)}$. By condition (iii) in definition of the key sequence, $w_{i_t} \geq w_{\pi(j)}$. Since $w_{\pi(j)} > w_{i_{\tau(j)}}$, condition (ii) in definition of the key sequence implies that $i_{\tau(j)}$ does not belong to the key sequence — a contradiction. This completes the proof of (21).

Let $x \in [C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(j-1)), C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(j))]$ for some $j \in \{1, \dots, |\mathcal{J}_{\text{opt}}|\}$, where we take $C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(0)) = 0$. Denote for brevity

$$\tilde{e}(x) = e \left(C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(1)), \dots, C_{\mathcal{S}_{\text{opt}}}^{\mathcal{M}}(\pi(|\mathcal{J}_{\text{opt}}|)), w_{\pi(1)}, \dots, w_{\pi(|\mathcal{J}_{\text{opt}}|)}, x \right)$$

and

$$\tilde{u}(x) = u(p_{i_1}, \dots, p_{i_\ell}, w_{i_1}, \dots, w_{i_\ell}, x)$$

for each real x . By definition,

$$\tilde{e}(x) \leq w_{\pi(j)}.$$

By (21), $w_{\pi(j)} \leq w_{i_{\tau(j)}}$. By (20), $x \leq p_{i_{\tau(j)}}$ and hence by the monotonicity of u ,

$$w_{i_{\tau(j)}} \leq \tilde{u}(x).$$

Hence, we obtain:

$$\tilde{e}(x) \leq \tilde{u}(x). \tag{22}$$

Since the index j was selected arbitrarily, the above inequality holds for each $x \geq 0$. Note that the inequality in (22) implies that the integral of $\tilde{e}(x)$ is less than or equal to the integral of $\tilde{u}(x)$. Since the former equals e^* and the latter equals u^* , this completes the proof. □

Since $\Sigma(\mathcal{S}_{\text{opt}}) = e^*$ and $\Sigma(\mathcal{S}_{\text{key}}) \leq \Sigma(\mathcal{S}_{\text{opt}})$, Lemmas 4 and 5 give the following.

Corollary 1 *It holds $e^*/2 \leq \Sigma(\mathcal{S}_{\text{key}}) \leq e^*$.* □

Theorem 2 *The key sequence for \mathcal{J} provides a 1/2-approximate solution to the problem WSPS. This sequence can be found in time $O(n \log n)$ for any set of jobs \mathcal{J} , where $n = |\mathcal{J}|$. Moreover, the bound of 1/2 is tight, i.e., for each $\varepsilon > 0$ there exists a problem instance such that $\Sigma(\mathcal{S}_{\text{key}}) < (\frac{1}{2} + \varepsilon) \Sigma(\mathcal{S}_{\text{opt}})$.*

Proof The fact that the key sequence is a $1/2$ -approximation of the optimal solution follows from Corollary 1. The key sequence can be constructed directly from the definition and sorting the jobs in \mathcal{J} according to their processing times determines the $O(n \log n)$ running time.

To close we show that the $1/2$ bound for the key sequences is tight. Fix $\varepsilon > 0$ arbitrarily and assume without loss of generality that $\varepsilon < 1$. Take \mathcal{J} to contain $n = \lceil \log_2(3/(2\varepsilon)) \rceil$ jobs, each of the same weight $w > 0$ and the same processing time $p > 0$. The key sequence consists of one job and therefore the total weighted overlap of the corresponding schedule \mathcal{S}_{key} is $\Sigma(\mathcal{S}_{\text{key}}) = wp/2$. Take a schedule \mathcal{S} that places all jobs in \mathcal{J} on the shared processor. We have $\Sigma(\mathcal{S}) = wp(1 - 1/2^n)$. If \mathcal{S}_{opt} is an optimal schedule, then

$$\frac{\Sigma(\mathcal{S}_{\text{key}})}{\Sigma(\mathcal{S}_{\text{opt}})} \leq \frac{\Sigma(\mathcal{S}_{\text{key}})}{\Sigma(\mathcal{S})} \leq \frac{1}{2(1 - 2\varepsilon/3)} < \frac{1}{2} + \varepsilon$$

for $0 < \varepsilon < 1$. □

7 Open problems and further research

The complexity status of WSPS remains open. The generalized problem with multiple shared processors is strongly NP-hard (Dereniowski and Kubiak 2017) when the number of shared processors is a part of the input. However, it remains open whether the generalized problem with fixed number of processors is NP-hard or whether it is FPT, for instance when the parameter is defined to be the number of shared processors. This complexity result and the open complexity questions clearly underline the difficulty in finding efficient optimization algorithms for the shared processor scheduling problem. The development of an efficient branch-and-bound algorithm for the problem remains unexplored so far. The $1/2$ -approximation algorithm along with the structural properties of optimal schedules presented in this paper and in Dereniowski and Kubiak (2017) may prove useful building blocks of such an algorithm.

Acknowledgements This research has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant OPG0105675 and by Polish National Science Centre under Contract DEC-2011/02/A/ST6/00201. The authors are grateful to two anonymous reviewers for their insightful comments that have led to improvements in the paper's presentation.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli, D., & Soukhal, A. (2014). *Multiagent scheduling. Models and algorithms*. Berlin: Springer.
- Anderson, E. J. (1981). A new continuous model for job-shop scheduling. *International Journal of System Science*, 12, 1469–1475.
- Bharadwaj, V., Ghose, D., & Robertazzi, T. G. (2003). Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6, 7–17.
- Bukchin, Y., & Hanany, E. (2007). Decentralization cost in scheduling: A game-theoretic approach. *Manufacturing & Service Operations Management*, 9(3), 263–275.
- Cheng, T. C. E., Ng, C. T., & Yuan, J. J. (2006). Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical Computer Science*, 362(1–3), 273–281.
- Dereniowski, D., & Kubiak, W. (2017). Shared multi-processor scheduling. *European Journal of Operational Research*, 261(2), 503–514.
- Hezarkhani, B., & Kubiak, W. (2015). Decentralized subcontractor scheduling with divisible jobs. *Journal of Scheduling*, 18(5), 497–511.
- Lee, K., Choi, B.-C., Leung, J. Y.-T., & Pinedo, M. L. (2009). Approximation algorithms for multi-agent scheduling to minimize total weighted completion time. *Information Processing Letters*, 109(16), 913–917.
- Vairaktarakis, G. L. (2013). Noncooperative games for subcontracting operations. *Manufacturing and Service Operations Management*, 15, 148–158.
- Vairaktarakis, G. L., & Aydinliyim, T. (2007). Centralization versus competition in subcontracting operations. Technical Memorandum Number 819, Case Western Reserve University.