Scheduling with State-Dependent Machine Speed

Veerle Timmermans(✉) and Tjark Vredeveld

Maastricht University, Maastricht, The Netherlands
{v.timmermans,t.vredeveld}@maastrichtuniversity.nl

Abstract. We study a preemptive single machine scheduling problem where the machine speed is externally given and depends on the number unfinished jobs. The objective is to minimize the sum of weighted completion times. We develop a greedy algorithm that solves the problem to optimality when we work with either unit weights or unit processing times. If both weights and processing times are arbitrary, we show the problem is NP-hard by making a reduction from 3-partition.

1 Introduction

In queueing theory, many studies have been made about queues with state-dependent service speeds, see e.g. Bekker and Boxma [4] or Bekker, Borst, Boxma and Kella [13] and the references therein. This model is among others motivated by Bertrand and Van Ooijen [5] through human servers who may be slow when there is much work to do, due to stress, or when there is little work to do due to laziness. For state-dependent server speeds in packet-switched communication systems, we refer to [6,7,11,14].

Although these models have been extensively studied in queueing theory, not much is known about algorithms that solve these models to optimality nor the computational complexity of this type of problem. During the 2013 Scheduling workshop in Dagstuhl, Urtzi Ayesta [3] posed it as an open question how optimal policies look like and what the computational complexity is. In this paper, we settle this open problem for one variant of state-dependent machine speeds, namely when the speed of the machine varies with the number of jobs in the system. This number of jobs is a good measure for the total workload in the system, when the service requirements of the jobs are i.i.d. Moreover, in this setting the speed of the server only changes at discrete times, see e.g. [4].

1.1 Previous Work

Models with workload dependent server speeds originate from queueing theory. Bertrand and Van Ooijen [5] assume in their paper that the workload level affects the effective processing times in a job shop. This assumption is based on the results of empirical research on the relationship between workload and shop performance. Bekker, Borst, Boxma and Kella [13] consider two types of queues

L. Sanità and M. Skutella (Eds.): WAOA 2015, LNCS 9499, pp. 196–208, 2015.
DOI: 10.1007/978-3-319-28684-6_17

with workload dependent arrival rate and service speed. Bekker and Boxma [4] consider a queueing system where feedback information about the level of congestion is given right after arrival instants. When the amount of work right after arrival is at most some threshold, then the server works at a low speed until the next arrival instant.

Related work in deterministic scheduling with varying machine speeds includes the following. Research into speed scaling algorithms started with the work of Yao, Demers and Shenker [8], where each job is to be executed between its arrival time and deadline by a single processor with variable speed. The difference with the previous mentioned models as well as ours is that the scheduler in this setting also needs to decide on the speed of the machine at any time t. A review paper on speed scaling algorithms is written by Albers [1].

Megow and Verschae [12] also study scheduling problems on a machine of varying speed, but they assume a speed function that depends on the time which is known a priori. They developed a PTAS for minimizing the total weighted completion time.

The machine speed model we consider, was previously investigated by Gawiejnowicz [10] and Alidaee and Ahmadian [2]. Though both papers discussed the non-preemptive case instead of the preemptive case we are looking at. Here Gawiejnowicz considered the makespan objective whereas we study the goal to minimize the sum of (weighted) completion. Alidaee and Ahmadian studied the sum of completion times where jobs have equal weights.

1.2 Problem Definition

In the model under consideration, n jobs need to be scheduled on a single machine. A job j is associated with a strictly positive processing requirement denoted by p_j and depending on the variant that we consider also with a weight w_j. All jobs as well as the machine are available from the beginning. The machine is allowed to preempt a job, i.e., the processing of a job may be interrupted and resumed later on the machine. By allowing infinitesimally small processing on a job before preempting it, the preemption model can be viewed as one in which during each time interval the processing capacity of the machine is divided over one or more jobs. The goal is to minimize the total (weighted) completion time, $\sum w_j C_j$, where C_j denotes the completion time of job j. In our scheduling model, the speed at which the machine processes its jobs varies with the number of jobs in the system. For notational convenience, we represent the speed at which the machine is processing as a function of the number of *completed* jobs. Hereto, we are given a speed function s_i $(i = 1, \ldots, n)$, where s_i denotes the speed of the machine between the $(i-1)$th and ith completion. From now on we refer to this problem as JDMS.

When the machine speed does not vary, i.e. $s_i = 1$ for all i, the problem is a standard problem where the total weighted completion time should be minimized on a single machine. This problem is solved by Smith's rule [15].

1.3 Our Results

When we know the order in which the jobs complete, we can formulate the problem as an LP with two types of constraints. The first type of constraints ensures that the amount of processing done during an interval is not more than the capacity of the machine in that interval. The second type of constraints ensures that the jobs complete in the order that is assumed. Any solution to our scheduling problem can be represented as a solution to the LP and vice versa. Using this fact, we can prove that there exists an optimal solution such that at every completion time C_j, any job k is already completed by time C_j or it has not received any processing yet by this time.

Intuitively this lemma means that there exists an optimal solution where the jobs are partitioned in groups and all jobs in a group start and finish at the same time. Given the order of job completions, we still need to decide on how to partition the jobs into groups. Although this follows from the optimal LP solution, we also develop a combinatorial greedy algorithm. For the case of the total completion time objective, it is easy to show that jobs will complete in shortest processing time (SPT) order, whereas the case of unit processing times the optimal order will be sorting according to non-increasing weight. When both weight and processing times can be arbitrary, the problem is strongly NP-hard and we make a reduction from 3-partition.

2 Structural Property

In this section we show that an optimal schedule has a certain block structure in which the jobs are processed in groups and that the jobs in one group start and finish at the same time. Note that we view the preemption model as one in which the processing capacity of the machine is divided over one or more jobs. Therefore, we may assume that a set of jobs can complete at exactly the same time. In case that the order in which the jobs need to complete is given, we can formulate the problem of minimizing the total weighted completion time as a linear program. Assuming w.l.o.g. that the order of completion is given by the index of jobs, i.e., $C_1 \leq \cdots \leq C_n$, the variables in the LP denote the time between the ith and the $i+1$st completion. That is, we use variable Δ_i, where:

$$\Delta_i = \begin{cases} C_1 & \text{if } i = 1, \\ C_i - C_{i-1} & \text{if } 1 < i \leq n. \end{cases}$$

Hence, the completion time of job j can be written as $C_j = \sum_{i=1}^{j} \Delta_i$. At the interval $[0, C_1]$, the machine is operating at speed s_1 and during the intervals $[C_{i-1}, C_i]$, the machine is operating at speed s_i. The sum of weighted completion times can be rewritten as:

$$\sum_{j=1}^{n} w_j C_j = \sum_{j=1}^{n} w_j \sum_{i=1}^{j} \Delta_i = \sum_{i=1}^{n} \left(\sum_{j=i}^{n} w_j \cdot \Delta_i \right).$$

To make sure the requested order on the completion times is enforced, we have the constraint $\Delta_i \geq 0$ for all i. Lastly we want to make sure that by time C_i at least jobs $1, \ldots, i$ have been fully processed. Thus the total amount of processing up to time C_i needs to be at least the processing requirements of the first i jobs:

$$\sum_{k=1}^{i} \Delta_k \cdot s_k \geq \sum_{k=1}^{i} p_k, \qquad 0 \leq i \leq n.$$

The LP is as follows:

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{n} \left(\sum_{j=i}^{n} w_j \cdot \Delta_i \right) \\ \text{subject to } & \sum_{k=1}^{i} \Delta_k \cdot s_k \geq \sum_{j=1}^{i} p_j, & 1 \leq i \leq n \\ & \Delta_i \geq 0, & 1 \leq i \leq n. \end{aligned}$$

Note that a feasible solution for this LP does not correspond to a unique schedule, but with a non-empty set of schedules for which the completion times are set. Any feasible schedule leads to a unique solution of the LP. For any given order on the completion times, we can find an optimal schedule in polynomial time using this LP.

We use this LP to show that an optimal solution to the general JDMS has a block structure. We first prove that there exists an optimal solution, such that at all completion times, each job that has started is finished.

Lemma 1. *There exists an optimal schedule such that at every completion time C_i, for every job j one of the following holds:*

1. *Job j is already completed at time C_i.*
2. *At time C_i job j has not received any processing yet.*

Proof. Given an order of completion times, we make a corresponding LP and reformulate this lemma in terms of this linear program. For this linear program we prove that for each job k one of the following holds:

- $\Delta_{k+1} = 0$, and thus $C_k = C_{k+1}$.
- If $C_{k+1} > C_k$, then it has to hold that $\sum_{j=1}^{k} s_j \Delta_j = \sum_{j=1}^{k} p_j$. If that is not the case, we use the fact that a solution must satisfy the requested order on the completion times in combination with the assumption that a machine is always working at full speed. Then there exists a job $l \geq k+1$ that is not yet completed, but already received some processing.

Thus we want to prove that for $k = 1, \ldots, n-1$ either:

$$\Delta_{k+1} = 0 \quad (1) \qquad \text{or} \qquad \sum_{j=1}^{k} s_j \Delta_j = \sum_{j=1}^{k} p_j. \quad (2)$$

Suppose we have an optimal solution σ such that there exists a k with:

$$\Delta_{k+1} > 0 \quad \text{and} \quad \sum_{j=1}^{k} s_j \Delta_j > \sum_{j=1}^{k} p_j.$$

We define ℓ as:

$$\ell = \max\{j \leq k | \Delta_j > 0\}.$$

Note that ℓ exists as prosessing times are strictly positive. We define two new feasible solutions, for some $\epsilon > 0$:

1. We define σ' as the solution where $\Delta_\ell^{\sigma'} = \Delta_\ell^{\sigma} - \frac{\epsilon}{s_\ell}$ and $\Delta_{k+1}^{\sigma'} = \Delta_{k+1}^{\sigma} + \frac{\epsilon}{s_{k+1}}$. The change in the objection value is:

$$\begin{aligned}\sum_{i=1}^{n} w_i C_i^{\sigma'} - \sum_{i=1}^{n} w_i C_i^{\sigma} &= \sum_{1 \leq i \leq j \leq n} w_j \Delta_i^{\sigma'} - \sum_{1 \leq i \leq j \leq n} w_j \Delta_i^{\sigma} \\ &= \left(\sum_{i=k+1}^{n} w_i\right) \frac{\epsilon}{s_{k+1}} - \left(\sum_{i=\ell}^{n} w_i\right) \frac{\epsilon}{s_\ell}.\end{aligned}$$

   Note that σ' is still feasible, as we can do ϵ amount of work less in $\Delta_\ell^{\sigma'}$ compared to Δ_ℓ^{σ}, but ϵ amount of work extra in $\Delta_{k+1}^{\sigma'}$ compared to Δ_{k+1}^{σ}.
2. We define σ'' as the solution where $\Delta_\ell^{\sigma''} = \Delta_\ell + \frac{\epsilon}{s_\ell}$ and $\Delta_{k+1}^{\sigma''} = \Delta_{k+1} - \frac{\epsilon}{s_{k+1}}$. The change in the objection value is:

$$\begin{aligned}\sum_{i=1}^{n} w_i C_i^{\sigma''} - \sum_{i=1}^{n} w_i C_i^{\sigma} &= \sum_{1 \leq i \leq j \leq n} w_j \Delta_i^{\sigma''} - \sum_{1 \leq i \leq j \leq n} w_j \Delta_i^{\sigma} \\ &= \left(\sum_{i=\ell}^{n} w_i\right) \frac{\epsilon}{s_\ell} - \left(\sum_{i=k+1}^{n} w_i\right) \frac{\epsilon}{s_{k+1}}.\end{aligned}$$

   Note that σ'' is still feasible, as we can do ϵ amount of work extra in $\Delta_\ell^{\sigma''}$ compared to Δ_ℓ^{σ}, but ϵ amount of work less in $\Delta_{k+1}^{\sigma''}$ compared to Δ_{k+1}^{σ}.

As:

$$\sum_{i=1}^{n} w_i C_i^{\sigma'} - \sum_{i=1}^{n} w_i C_i^{\sigma} = -\left(\sum_{i=1}^{n} w_i C_i^{\sigma''} - \sum_{i=1}^{n} w_i C_i^{\sigma}\right),$$

at least one of the two solutions is better than or equal to σ. As σ is optimal, we actually know that both new solutions are also optimal.

Suppose k is the smallest value such that neither (1) or (2) holds. Let:

$$\epsilon = \min\{\Delta_{k+1}, \sum_{j=1}^{k} s_j \Delta_j - \sum_{j=1}^{k} p_j\},$$

then either σ' or σ'' will give an optimal solution where either (1) or (2) holds for job k. We repeat this procedure until this property holds for all $k \in \{0 \dots n\}$.

Thus given an order on the completion times, we can find an optimal schedule satisfying the requested order such that for all i, j, where $1 \leq i, j \leq n$ it holds that at time C_j either job i is completed or did not receive any processing time yet. As this holds for any order, this will also hold for the order of some optimal solution. Therefore there exists an optimal solution such that or all i, j, where $1 \leq i, j \leq n$, it holds that $y_i(C_j) \in \{0, p_i\}$. □

Lemma 1 implies that we can divide the jobs into groups of consecutive jobs, such that all jobs in a group will start and end at the same time. We use G_i to denote the ith group of jobs and we denote an optimal solution as $[G_1, \dots, G_k]$ (Fig. 1).

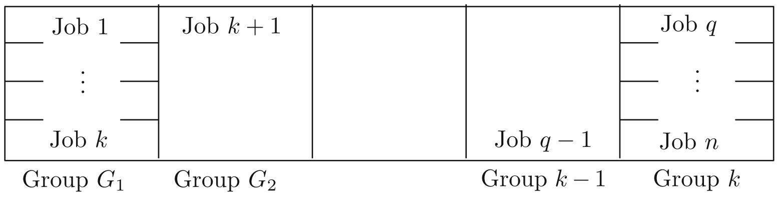

Fig. 1. Schedule in which the jobs are divided into k groups.

3 JDMS is Strongly NP-hard

From the previous section it follows that once we know the order in which the jobs complete in an optimal solution, we can find such an optimal solution in polynomial time. The only question that remains is how to find an order such that there exists an optimal solution that satisfies this order? When there are no restrictions on weights and processing times, it is strongly NP-hard to find such an order.

Theorem 1. *JDMS is strongly NP-hard.*

Proof. We make a reduction from 3-partition [9] to JDMS. We take an instance of 3-partition with $3m$ elements of size a_j, and $B = \frac{1}{m}\sum_{i=1}^{3m} a_j$. The reduction to JDMS is as follows: we define $3m$ jobs, with $p_j = w_j = a_j$. The machine speed is 1 when there are 0 (mod 3) jobs completed, and 0 otherwise. Now we claim that there is a solution for 3-partition if and only if we can find a schedule where the sum of weighted completion times is at most $\frac{1}{2}m(m+1)B^2$.

Suppose we have a yes-instance for 3-partition, then there are sets $S_1, \dots, S_m$ that contain exactly three elements, and $\sum_{i \in S_j} a_j = B$ for all j. Then for JDMS we make the schedule where we process the jobs in groups $[S_1, \dots S_m]$. In this schedule the machine runs always at speed 1, as we always process three jobs at the same time. Group S_i has weight B and completion time iB, thus the objective value is in this case $\sum_{i=1}^{m} iB^2$, which equals $\frac{1}{2}m(m+1)B^2$. Thus indeed there exists a schedule with completion time at most $\frac{1}{2}m(m+1)B^2$.

Suppose we have a schedule in JDMS with objective value at most $\frac{1}{2}m(m+1)B^2$. According to Lemma 1 the jobs are processed in groups, and the sizes of these groups have to be a multiples of three. Therefore we have at most m groups. Now let x_i be the length of group i, then the total weighted cost is:

$$\sum_{i=1}^{m} \left(x_i * \sum_{1 \le j \le i} x_j \right).$$

This can be rewritten as:

$$\frac{1}{2} \left((\sum_{i=1}^{m} x_i)^2 + \sum_{i=1}^{m} x_i^2 \right).$$

We know that $\sum_{i=1}^{m} x_i = mB$, thus we have objective value:

$$\frac{1}{2} \left(m^2 B^2 + \sum_{i=1}^{m} x_i^2 \right).$$

The sum of squares is minimized when $x_1 = \cdots = x_n = \frac{mB}{m} = B$. Note that this will give us exactly objective value $\frac{1}{2}m(m+1)B^2$, and is therefore the only schedule we could have found. This implies that every group has exactly processing time B. Note that we need to use all m groups, and we know that every group has to contain a multiple of 3 jobs. As we have $3m$ jobs in total, every group has to contain exactly three jobs. Thus these groups form a 3-partition of the jobs a_j.

This implies that the decision variant of JDMS is NP-complete, and therefore JDMS is NP-hard. □

4 Combinatorial Algorithm for Special Cases

In the previous section we proved that JDMS is strongly NP-hard. As the partitioning problem can be solved within polynomial time, the problem lies in finding a suitable order on the job completions. In some special cases, when working with either unit weights or unit processing requirements, such an order can be found within polynomial time.

4.1 Solving the Sequencing Problem in Special Cases

In the first special case we assume all jobs have an equal weight.

Theorem 2. *Suppose $w_1 = \cdots = w_n$, then there exists an optimal schedule for JDMS in which the jobs are completed in order of non-decreasing processing time.*

Proof. Assume w.l.o.g that $p_1 \leq \cdots \leq p_n$ and we have an optimal schedule σ for which it does not hold that $C_1^\sigma \leq \cdots \leq C_n^\sigma$. Then we look at the smallest i such that $C_{i+1}^\sigma < C_i^\sigma$. We change σ to σ^* by only changing σ at job i and $i+1$. We let job i finish at time $C_i^{\sigma^*} = C_{i+1}^\sigma$ and job $i+1$ finish at time $C_{i+1}^{\sigma^*} = C_i^\sigma$. As $p_i \leq p_{i+1}$ and $C_{i+1}^\sigma < C_i^\sigma$, we know that there needs to be a time t at which in schedule σ the remaining processing time of a job i is the same as the remaining processing time of job $i+1$. We define the schedule σ^* as follows. It processes all jobs $j \neq i, i+1$ the same as in σ as it does with jobs i and $i+1$ up to time t. From time t onwards, σ^* processes job $i+1$ whenever σ processes job i and it processes job i whenever job $i+1$ is processed by σ. All other jobs are processed in σ^* the same way as in σ.

As the time points where jobs finish stay the same, it holds that the speed of the machine is equal in σ and σ^* at every point in time. Therefore per time unit the same amount of work can be done in σ and σ^*. So there is exactly enough space for job $i+1$ to be finished at time C_i^σ. Thus $\sum_{i=1}^n C_i^\sigma = \sum_{i=1}^n C_i^{\sigma^*}$ and σ^* will remain optimal.

Iterating this process implies that there is an optimal schedule σ' such that $C_1^{\sigma'} \leq \cdots \leq C_n^{\sigma'}$. □

Thus for JDMS with equal weights there exists an optimal solution that satisfies the SPT order.

When we work with unit processing requirements instead of weights, we again can find an order on the completion times that guarantees an optimal solution.

Theorem 3. *Suppose $p_1 = \cdots = p_n$, then there exists an optimal schedule for JDMS in which the jobs completed in order of non-increasing weights*

Proof. Suppose we have an optimal schedule σ and it does not hold that $C_1^\sigma \leq \cdots \leq C_n^\sigma$. Then we look at the smallest i such that $C_{i+1}^\sigma < C_i^\sigma$ and $w_i > w_{i+1}$ (if $w_i = w_{i+1}$ then job i and $i+1$ are identical and therefore switching job i with job j will result in an optimal schedule as well). We change σ to σ^* by processing job $i+1$ whenever σ processes job i and job i whenever σ processes job $i+1$. All other jobs are processed as in σ. As $p_i = p_{i+1}$, we know that $C_{i+1}^{\sigma^*} = C_i^\sigma$ and $C_i^{\sigma^*} = C_{i+1}^\sigma$ and $C_j^{\sigma^*} = C_j^\sigma$ for all $j \neq i, i+1$. Thus $w_j C_j^\sigma = w_j C_j^{\sigma^*}$ for all $j \neq i, i+1$. Furthermore, as $w_i > w_{i+1}$, $C_{i+1}^{\sigma^*} > C_i^{\sigma^*}$, $C_i^\sigma = C_{i+1}^{\sigma^*}$ and $C_{i+1}^\sigma = C_i^{\sigma^*}$, some simple rewriting learns us that:

$$\begin{aligned}
w_i C_i^\sigma + w_{i+1} C_{i+1}^\sigma &= w_{i+1} C_i^{\sigma^*} + w_i C_{i+1}^{\sigma^*} \\
&= w_{i+1} C_i^{\sigma^*} + w_i C_{i+1}^{\sigma^*} - (w_i C_i^{\sigma^*} + w_{i+1} C_{i+1}^{\sigma^*}) \\
&\quad + (w_i C_i^{\sigma^*} + w_{i+1} C_{i+1}^{\sigma^*}) \\
&= (w_i - w_{i+1})(C_{i+1}^{\sigma^*} - C_i^{\sigma^*}) + (w_i C_i^{\sigma^*} + w_{i+1} C_{i+1}^{\sigma^*}) \\
&> w_i C_i^{\sigma^*} + w_{i+1} C_{i+1}^{\sigma^*}.
\end{aligned}$$

Thus $\sum_{i=1}^n w_i C_i^\sigma > \sum_{i=1}^n w_i C_i^{\sigma^*}$, which contradicts the fact that σ is optimal. □

So when we number the jobs in order of non-increasing weights we know that there exists an optimal solution satisfying this order.

4.2 Combinatorial Algorithm

We have split the problem up in two parts: we need to find a good sequence for the order on the job completions (secuencing problem) and then decide how to make the groups (partitioning problem). In Sect. 4.1 we solved the sequencing problem for some special cases. To find an optimal solution it remains to solve the partitioning problem. In Sect. 2 we showed that we can rewrite the problem as an LP to find the exact completion times, and hence make an optimal solution. In this section we develop a combinatorial algorithm that solves the partitioning problem in linear time. We first give a intuitive explanation of this algorithm, and thereafter a formal definition of the algorithm in pseudo code is given.

The algorithm finds an sequence of groups that fits in the block structure of an optimal solution for the given order. This sequence indicates which groups of jobs should be processed in what order. Again we assume w.l.o.g. that the order on the completion times is $C_1 \leq \cdots \leq C_n$. The algorithm determines for each job i, whether it is better to schedule this job in the same group as job $i-1$ or to start a new group for job i. Hereto, we determine what the effect of job i is on the total weighted completion time is when it is scheduled in the same group as job $i-1$ and also for the case when a new group is started for job i. When job i is scheduled in the same group as job $i-1$ the contribution is $(\sum_{j \in G_k} w_j + \sum_{j=i}^{n} w_j)(p_i/s_{G_k})$, as it delays all jobs processed in or after group G_k. Here we use s_{G_k} to refer to the speed at which G_k is processed. When job i starts a new group the contribution is $(\sum_{j=i}^{n} w_j)(p_i/s_i)$, as it delays job i and all jobs that complete after job i. Therefore we determine whether or not the following equation holds:

$$\frac{\sum_{j \in G_k} w_j + \sum_{j=i}^{n} w_j}{s_{G_k}} \leq \frac{\sum_{j=i}^{n} w_j}{s_i}. \tag{1}$$

If so, then job i is scheduled to be processed together with the group of $i-1$. Otherwise a new group is started. The pseudo code of this greedy algorithm can be found in Algorithm 1.

Theorem 4. *Algorithm 1 finds an optimal solution for a given order of job completions in JDMS.*

Proof. Suppose σ is an optimal schedule for an instance of JDMS with n jobs satisfying $C_1 \leq \cdots \leq C_n$. Let σ^* be the solution according to the algorithm. Then we want to show that a schedule according to the algorithm will give a solution with an equal objective value. Let job i be the first job in σ that is not scheduled according to the algorithm. Then there are two possible situations:

1. In σ, job i is scheduled in a new group G_k, whereas in σ^* it is still processed in group G_{k-1}.

```
Input: $n$ jobs with processing requirements $p_1, \dots, p_n$, weights $w_1, \dots w_n$,
       speeds $s_1, \dots, s_n$ and an order on the completion times $C_1 \leq \dots \leq C_n$.
Output: an optimal sequence of groups $[G_1, \dots, G_k]$
initialization: $G_1 = \{1\}, k = 1, E = w_1, s = s_1$ ;
for $i = 2$ until $i = n$ do
    if $(E + \sum_{j=i}^{n} w_j)s_i \leq (\sum_{j=i}^{n} w_j)s$ then
        $E \to E + w_i$, ;
        $G_k \to G_k \cup \{i\}$;
    else
        $E \to w_i$;
        $s \to s_i$;
        $G_{k+1} \to \{i\}$;
        $k \to k + 1$;
    end
end
Return $[G_1, \dots, G_k]$
```

Algorithm 1. Greedy algorithm

2. Job i is scheduled in the previous group G_k, while it should be scheduled in a new group.

Suppose we are in the first situation: in σ, job i is scheduled in a new group G_k, whereas in σ^* it is still processed in group G_{k-1}. Then we change σ to σ' by merging G_k and G_{k-1}. Then in σ' , jobs $1, \dots, i$ are scheduled the same as in σ^*.

We look at the total value that groups G_k, G_{k-1} and $G_k \cup G_{k-1}$ contribute to the objective value of σ and σ'. Here the contribution of a group S is not $\sum_{j \in S} w_j C_j$, but instead the time it takes to process all jobs in the group multiplied by the weight of all unfinished jobs at that point. As the other groups in σ' are the same as in σ, the contribution of these groups are the same in both σ and σ'. Let c be the function that computes the value that a group contributes to the objective value. Let W be the total weight of all the jobs scheduled after G_k. Then:

$$c_\sigma(G_k) + c_\sigma(G_{k-1}) = \left(\sum_{j \in G_{k-1}} w_j + \sum_{j \in G_k} w_j + W \right) \frac{\sum_{j \in G_{k-1}} p_j}{s_{G_{k-1}}} + \left(\sum_{j \in G_k} w_j + W \right) \frac{\sum_{j \in G_k} p_j}{s_{G_k}}, \quad (2)$$

$$c_{\sigma'}(G_k \cup \ G_{k-1}) = \left(\sum_{j \in G_{k-1}} w_j + \sum_{j \in G_k} w_j + W \right) \frac{\sum_{j \in G_k \cup \ G_{k-1}} p_j}{s_{G_{k-1}}}. \quad (3)$$

According to Algorithm 1 it should hold that:

$$\frac{\sum_{j\in G_{k-1}} w_j + \sum_{j=i}^{n} w_j}{\sum_{j=i}^{n} w_j} \le \frac{s_{G_{k-1}}}{s_i}. \tag{4}$$

Combining the fact that $\sum_{j\in G_k} w_j + W = \sum_{j=i}^{n} w_j$ and (4) we know that:

$$\frac{\sum_{j\in G_{k-1}} w_j + \sum_{j\in G_k} w_j + W}{\sum_{j\in G_k} w_j + W} \le \frac{s_{G_{k-1}}}{s_i}. \tag{5}$$

We rewrite (5), and as job i is the first job of G_k we can replace s_i by s_{G_k}:

$$\frac{\sum_{j\in G_{k-1}} w_j + \sum_{j\in G_k} w_j + W}{s_{G_{k-1}}} \le \frac{\sum_{j\in G_k} w_j + W}{s_{G_k}}. \tag{6}$$

Multiplying both sides with $\sum_{j\in G_k} p_j$:

$$\left(\sum_{j\in G_{k-1}} w_j + \sum_{j\in G_k} w_j + W\right)\frac{\sum_{j\in G_k} p_j}{s_{G_{k-1}}} \le \left(\sum_{j\in G_k} w_j + W\right)\frac{\sum_{j\in G_k} p_j}{s_{G_k}}. \tag{7}$$

Adding $\left(\sum_{j\in G_{k-1}} w_j + \sum_{j\in G_k} w_j + W\right)\frac{\sum_{j\in G_{k-1}} p_j}{s_{G_{k-1}}}$ yields:

$$\begin{aligned}\left(\sum_{j\in G_{k-1}} w_j + \sum_{j\in G_k} w_j + W\right)\frac{\sum_{j\in G_k\cup\, G_{k-1}} p_j}{s_{G_{k-1}}} \le{}& \left(\sum_{j\in G_k} w_j + W\right)\frac{\sum_{j\in G_k} p_j}{s_{G_k}}\\ &+\left(\sum_{j\in G_{k-1}} w_j + \sum_{j\in G_k} w_j + W\right)\frac{\sum_{j\in G_{k-1}} p_j}{s_{G_{k-1}}}.\end{aligned} \tag{8}$$

Combining (2), (3) and (8) yields:

$$c_{\sigma'}(G_k \cup\, G_{k-1}) \le c_\sigma(G_k) + c_\sigma(G_{k-1}). \tag{9}$$

Thus the value of our changed schedule σ' is smaller or equal than the objective value of σ, thus σ' is optimal as well. The proof for the second situation is similar and will therefore not be fully written out here. □

5 Concluding Remarks

In this paper, we considered the problem JDMS in which jobs need to be scheduled preemptively on a single machine of which the speed varies with the number of jobs that have been completed. We showed this problem to be NP-hard and that the main issue is to decide in which order the jobs need to complete. Once this order is known, we can find the optimal schedule in polynomial time through a greedy algorithm. This algorithm uses a structural property that tells that the jobs are processed in blocks. For two special cases, JDMS with unit weights or unit processing times, we found the orders that guarantee us to find an optimal solution. The optimal order to complete unit weight jobs is shortest processing time and the one for unit processing times is largest weight.

One question that remains is how well the general problem can be approximated. In case that the speed is constant, it is well known that the WSPT rule that processes the jobs in order of non-increasing ratio of weight over processing time is optimal [15]. However, the following example shows that this order can be arbitrarily bad for JDMS.

Example 1. In this example, there are two jobs with $w_1 = 0$, $p_1 = \epsilon$ and $w_2 = p_2 = A$. According to the WSPT order job 2 precedes job 1, and the optimal schedule has value A^2. If we consider the opposite order the optimal schedule has value $\frac{A}{1+\epsilon}$. Letting ϵ go to 0 and A be arbitrarily large, we see that this ratio can be arbitrarily large.

Acknowledgements. We thank Urtzi Ayesta for helpful discussion after posing this open question during the Daghstuhl Seminar 13111 “Scheduling” in 2013. Furthermore, we thank the organizers of this seminar and Schloss Daghstuhl for providing the right atmosphere to facilitate research.

References

1. Albers, S.: Review articles: energy-efficient algorithms. Commun. ACM **53**(5), 86–96 (2010)
2. Alidaee, B., Ahmadian, A.: Scheduling on a single processor with variable speed. Inf. Process. Lett. **60**, 189–193 (1996)
3. Ayesta, U.: Scheduling (dagstuhl seminar 13111): scheduling with time-varying capacities. Dagstuhl Rep. **3**(3), 29 (2013)
4. Bekker, R., Boxma, O.J.: An M/G/1 queue with adaptable service speed. Stochastic Models **23**(3), 373–396 (2007)
5. Bertrand, J.W.M., Ooijen, H.P.G.: Workload based order release and productivity : a missing link. Prod. Plan. Control : The Manage. Oper. **13**(7), 665–678 (2002)
6. Ewalid, A., Mintra, D.: Analysis and design of rate-based congestion control of high-speed networks, i: stochastic fluid models, access regulation. Queueing Syst. **9**, 29–64 (1991)
7. Ewalid, A., Mintra, D.: Statistical multiplexing with loss priorities in rate-based congestion control of high-speed networks. IEEE Trans. Commun. **42**, 2989–3002 (1994)
8. Demers, A.J., Yao, F.F., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS, pp. 374–382 (1995)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., New York (1979)
10. Gawiejnowicz, S.: A note on scheduling on a single processor with speed dependent on a number of executed jobs. Inf. Process. Lett. **57**, 297–300 (1996)
11. Mandjes, M., Mintra, D.: A simple model of network access: feedback adaptation of rates and admission control. In: Proceedings of Infocom, pp. 3–12 (2002)

12. Megow, N., Verschae, J.: Dual techniques for scheduling on a machine with varying speed. In: Fomin, F.V., Freivalds, R.U., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 745–756. Springer, Heidelberg (2013)
13. Boxma, O.J., Bekker, R., Borst, S.C., Kelly, O.: Queues with workload-dependent arrival and service rates. Queueing Syst. **46**(3–4), 537–556 (2004)
14. Ramanan, K.A., Weiss, A.: Sharing bandwidth in ATM. In: Proceedings of the Allerton Conference, pp. 732–740 (1997)
15. Smith, W.E.: Various optimizers for single-stage production. Naval Res. Logist. Q. **3**, 59–66 (1956)