SRPT is 1.86-Competitive for Completion Time Scheduling

Christine Chung* Tim Nonner† Alexander Souza‡

Abstract

We consider the classical problem of scheduling preemptible jobs, that arrive over time, on identical parallel machines. The goal is to minimize the total completion time of the jobs. In standard scheduling notation of Graham et al. [5], this problem is denoted $\mathrm{P} \mid r_j, \mathrm{pmtn} \mid \sum_j c_j$. A popular algorithm called SRPT, which always schedules the unfinished jobs with shortest remaining processing time, is known to be 2-competitive, see Phillips et al. [13, 14]. This is also the best known competitive ratio for any online algorithm. However, it is conjectured that the competitive ratio of SRPT is significantly less than 2. Even breaking the barrier of 2 is considered a significant step towards the final answer of this classical online problem. We improve on this open problem by showing that SRPT is 1.86-competitive. This result is obtained using the following method, which might be of general interest: We define two dependent random variables that sum up to the difference between the cost of an SRPT schedule and the cost of an optimal schedule. Then we bound the sum of the expected values of these random variables with respect to the cost of the optimal schedule, yielding the claimed competitiveness. Furthermore, we show a lower bound of 21/19 for SRPT, improving on the previously best known 12/11 due to Lu et al. [11].

1 Introduction

In this paper, we study the classical problem of online scheduling preemptible jobs, arriving over time, on identical machines. The goal is to minimize the total completion time of the jobs. Our performance measure is the competitive ratio, i.e., the worst-case ratio of the objective value achieved by an online algorithm and the offline optimum. Specifically, we are given m identical machines and jobs $J = \{1, \ldots, n\}$, which arrive over time, where each job j becomes known at its *release time* $r_j \geq 0$. At time r_j we also learn the *processing time* $p_j > 0$ of job j. Preemption is allowed, i.e., at any time we may interrupt any job that is currently running and resume it later, possibly on a different machine. A *schedule* σ assigns (pieces of) jobs to time-intervals on machines, and the time when job j completes is denoted c_j. We seek to minimize the total completion time $\sum_j c_j$. In the standard scheduling notation due to Graham et al. [5], this problem is denoted $\mathrm{P} \mid r_j, \mathrm{pmtn} \mid \sum_j c_j$.

For roughly 15 years, the best known competitive ratio for this fundamental scheduling problem was due to Phillips, Stein, and Wein [13, 14]. They proved that the algorithm SRPT, which always schedules the unfinished jobs with shortest remaining processing time, is 2-competitive. To achieve this, they showed that, at any time $2t$, SRPT has completed as many jobs as any other schedule could complete by time t. It was an open problem to prove that the competitive ratio of SRPT is bounded by a constant strictly smaller than 2 for any number of processors m, as conjectured by Stein [20] and Lu, Sitters, and Stougie [11].

Contributions. We show in Section 3 that SRPT is 1.86-competitive, which also improves upon the best known competitive ratio for $\mathrm{P} \mid r_j, \mathrm{pmtn} \mid \sum_j c_j$. As the makespan argument of [13, 14] is tight, we need another approach. We make use of the following general method.

Consider an arbitrary optimization problem, and let OPT be the cost of an optimal solution for a fixed but arbitrary instance. Moreover, let ALG also denote the cost of the solution returned by some deterministic algorithm ALG for the same instance. Hence, to obtain an approximation guarantee for ALG, we need to bound ALG/OPT. Let now X and Y be two dependent random variables with $X + Y = \mathrm{ALG} - \mathrm{OPT}$. In fact, they need to be dependent, since they sum up to a constant value depending on the given instance. Assume now that we have the bounds $\mathbb{E}[X] \leq \alpha\mathrm{OPT}$ and $\mathbb{E}[Y] \leq \beta\mathrm{OPT}$ for two positive constants α, β. In this case, by linearity of expectation, we obtain that

$$\begin{aligned} \mathrm{ALG} - \mathrm{OPT} &= \mathbb{E}[\mathrm{ALG} - \mathrm{OPT}] = \mathbb{E}[X] + \mathbb{E}[Y] \\ &\leq (\alpha + \beta)\mathrm{OPT}, \end{aligned}$$

and hence $\mathrm{ALG}/\mathrm{OPT} \leq 1 + \alpha + \beta$. In our case $\mathrm{ALG} = \mathrm{SRPT}$, and we obtain the two random variables X and Y by randomly transforming some schedules.

*Department of Computer Science, University of Pittsburgh, USA, chung@cs.pitt.edu

†Department of Computer Science, Albert Ludwigs University of Freiburg, Germany, nonner@informatik.uni-freiburg.de (corresponding author), Supported by DFG research program No 1103 *Embedded Microsystems*

‡Department of Computer Science, Humboldt University of Berlin, Germany, souza@informatik.hu-berlin.de

To the best of our knowledge, the approach described in the last paragraph is novel in the area of scheduling and may be of independent interest. In fact, the authors are not aware of the use of such a technique in the analysis of any algorithm, since the usual approach is to either analyze a deterministic algorithm in a deterministic way, or to analyze a randomized algorithm using the randomness provided by the algorithm. We can also think of this as applying the *probabilistic method*: this is a non-constructive method pioneered by Paul Erdős for proving the existence of a prescribed kind of mathematical object. If one can show that choosing objects from a specified class at random yields strictly positive probability that the result is of the prescribed kind, then the prescribed object exists. Hence, although the proof is probabilistic, the final conclusion is determined for certain. In our case, we are interested in algorithms that transform an optimal schedule into an SRPT-schedule. By concatenating the transformations used to obtain the random variables X and Y, we obtain such a random transformation yielding an increase in objective value by a factor of at most 1.86 in expectation. Thus, by the probabilistic method, such a transformation always exists and shows the competitiveness of at most 1.86 of SRPT.

We conjecture that 1.86 is not the final answer, but as with many other problems, e.g., Vertex Cover, giving an approximation guarantee of 2 is relatively easy, but improving on this is far more involved.

In terms of negative results we improve the previously best known lower bound for SRPT due to Lu, Sitters, and Stougie [11] from 12/11 to 21/19 in Section 2. We believe that 21/19 is the right answer for SRPT.

Related work. We restrict this review to the following variants of completion time scheduling with release times: preemptive or non-preemptive, unweighted or weighted, and single machine or identical parallel machines.

The single machine problem $1 \mid r_j, \mathrm{pmtn} \mid \sum_j c_j$ is the only variant that is known to be solvable in polynomial time. Indeed, Schrage [17] proved that SRPT is optimal for this problem. For all other variants considered here, the offline versions are already NP-hard, see [8, 9, 4], but they all admit a PTAS [1].

For the weighted and preemptive case, $1 \mid r_j, \mathrm{pmtn} \mid \sum_j w_j c_j$, i.e., each job j has a non-negative weight w_j, and we seek to minimize $\sum_j w_j c_j$, Goemans, Williamson, and Wein observed in unpublished work that preemptively scheduling in order of non-decreasing p_j/w_j values is 2-competitive. A proof was provided by Schulz and Skutella [18], where also a 4/3-competitive randomized algorithm was given. Finally, Sitters [19] gave a deterministic 1.56-competitive algorithm. On the other hand, for the weighted and non-preemptive case, $1 \mid r_j \mid \sum_j w_j c_j$, Anderson and Potts [2] extended an algorithm of Hoogeveen and Vestjens [7] and proved that it is 2-competitive. This is best-possible, since no deterministic online algorithm can be better than 2-competitive for this variant [7].

For identical parallel machines and even for the weighted and preemptive case, $\mathrm{P} \mid r_j, \mathrm{pmtn} \mid \sum_j w_j c_j$, Megow and Schulz [12] gave a 2-competitive algorithm. The best known bound for the weighted and non-preemptive case, $\mathrm{P} \mid r_j \mid \sum_j w_j c_j$, is due to Correa and Wagner [3], who gave a 2.62-competitive algorithm. They also found a randomized algorithm with competitive ratio strictly smaller than 2, but approaching 2 as m grows. Furthermore, Liu and Lu [10] gave a 2-competitive algorithm for the unweighted and non-preemptive case, $\mathrm{P} \mid r_j \mid \sum_j c_j$.

To the best of our knowledge, the following table summarizes the currently best known upper bounds of deterministic algorithms for the preemptive case. The unreferenced bound is due to this work.

Machines	$\sum_j c_j$	$\sum_j w_j c_j$
Single	1 [17]	1.56 [19]
Identical	1.86	2 [12]

On the negative side, no deterministic algorithm can be better than 22/21-competitive for $\mathrm{P} \mid r_j, \mathrm{pmtn} \mid \sum_j c_j$, which was shown by Vestjens [21]. Without preemption, 1.309 is the best known lower bound. A comprehensive survey on further online scheduling models is given by Pruhs, Torng, and Sgall [15].

2 Notation and lower bound

An instance consists of a set of jobs $J = \{1, \ldots, n\}$, where each job j is characterized by its *release time* $r_j \geq 0$ and *processing time* $p_j > 0$. We assume w.l.o.g. that the processing and release times are integral. Therefore, we may also assume that time is divided into time slots $1, 2, 3, \ldots$ of unit length one. A *schedule* σ assigns each job j to a set of distinct time slots denoted σ_j, and $\sigma(t) := \{j \mid t \in \sigma_j\}$ are hence the jobs scheduled at some time slot t, such that the following three feasibility properties are satisfied:

(1) each job j is scheduled only at time slots not earlier than r_j, i.e., for all $t \in \sigma_j$, $t \geq r_j$,

(2) at most m jobs are scheduled at each time slot t, i.e., $|\sigma(t)| \leq m$,

(3) each job j is scheduled at no more than p_j time slots, i.e., $|\sigma_j| \leq p_j$.

Note that σ_j is a set, and consequently, only one unit of the processing time p_j of j may be scheduled at each time slot t, which corresponds to the requirement that a job is never scheduled in parallel on different machines. If the inequality in feasibility property (3) is tight for each job j, then we say that σ is *complete*. However, we will also use incomplete schedules in what follows, and we refer to $p_j(\sigma) := |\sigma_j|$ as the *processing time of job* j in σ. Feasibility property (1) implies that, in our notation, the release time of a job is the first time slot when it may be scheduled. Finally, feasibility property (2) corresponds to the requirement that we have m identical machines.

Let $c_j(\sigma) := \max \sigma_j$ denote the last time slot when a job j is scheduled by σ, to which we refer as its *completion time*. Furthermore, let $f_{jt}(\sigma) := \min\{s \geq t \mid s \in \sigma_j\} \leq c_j(\sigma)$ denote the first time slot at which j is scheduled by σ after time slot t. Note that we include time slot t whenever we say *after time slot* t. If there is no time slot $s \geq t$ with $s \in \sigma_j$, then define $f_{jt}(\sigma) := 0$. Let $p_{jt}(\sigma) := |\{s \in \sigma_j \mid s \geq t\}|$ be the remaining processing time of a job j after time slot t, and let $n_t(\sigma) := |\{j \mid p_{jt}(\sigma) > 0\}|$ be the number of unfinished jobs at time slot t. We say that two schedules σ and σ' *define* the same instance after time slot t if $p_{jt}(\sigma) = p_{jt}(\sigma')$, for each job j. Finally, let $T(\sigma) := \max\{t \mid \sigma(t) \neq \emptyset\}$ be the last time slot when σ schedules a job.

We say that a schedule σ is SRPT-*scheduled after time slot* t if, for each time slot $s \geq t$, the jobs $\sigma(s)$ are the (up to) m unfinished jobs j with minimum remaining processing time $p_{js}(\sigma) > 0$, where ties are broken arbitrarily. Therefore, a schedule produced by the SRPT algorithm is complete and SRPT-scheduled after time slot 1.

We consider the problem of finding a complete schedule σ that minimizes the objective function

$$\mathrm{cost}(\sigma) := \sum_{j=1}^{n} c_j(\sigma).$$

Note that we can write this objective function as

$$\mathrm{cost}(\sigma) = \sum_{t=1}^{\infty} n_t(\sigma). \tag{2.1}$$

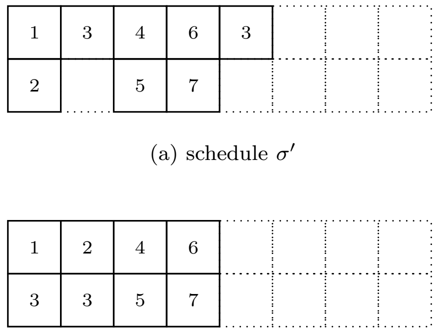
We will use this representation of the objective function in the sections to follow. Finally, the following theorem gives the lower bound for the competitiveness of SRPT.

THEOREM 2.1. *If* SRPT *is c-competitive, then* $c \geq 21/19$.

Proof. Consider the following instance, which is based closely on the lower-bound instance of [11]. We have $m = 2$ and $n = 7$, with the release time and processing time of each job as listed in the following table.

j	1	2	3	4	5	6	7
p_j	1	1	2	1	1	1	1
r_j	1	1	1	3	3	3	3

An SRPT schedule σ' gives total completion time $\mathrm{cost}(\sigma') = \sum_{j=1}^{7} c_j(\sigma') = 1+1+3+3+4+4+5 = 21$. On the other hand, an optimal schedule σ^* gives $\mathrm{cost}(\sigma^*) = \sum_{j=1}^{7} c_j(\sigma^*) = 1+2+2+3+3+4+4 = 19$. Two such schedules are depicted in Figure 1. Our intuition behind this lower bound instance is given in the following paragraph.



(a) schedule σ'

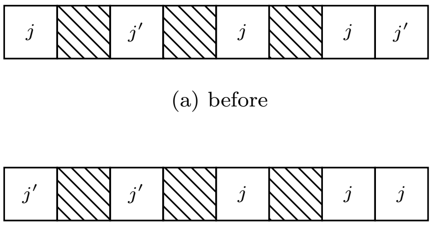
(b) schedule σ^*

Figure 1: An SRPT schedule σ' and an optimal schedule σ^*. Each column represents a time slot, and such a column contains a box labeled with j if job j is scheduled at the corresponding time slot. Since $m = 2$, each column contains room for two boxes, and we depict eight time slots in total, although not all of them are used.

It may happen that, at some time slot t, some SRPT schedule σ' has not finished a certain job j which an optimal schedule σ^* has already completed. This will be problematic if at least m jobs with processing time smaller than the remaining processing time of j arrive at time t. Then, the completion of j will be further delayed. However, as the arriving jobs also contribute to $\mathrm{cost}(\sigma^*)$, their effect on the competitive ratio is bounded. This suggests that we seek for an instance where $m > 1$ is minimal, i.e., $m = 2$, and as many small jobs arrive at time t that cause the competitive ratio to grow. In the lower bound instance given above, we use $j = 3$ and $t = 3$. $\square$

3 Competitive analysis of SRPT

Consider an optimal schedule σ^* with $\mathrm{cost}(\sigma^*) = \mathrm{OPT}$ and an SRPT schedule σ' with $\mathrm{cost}(\sigma') = \mathrm{SRPT}$.



(a) before

(b) after

Figure 2: Schedule σ before and after scheduling job j' before j at time slots $\overline{W}$. We do not know anything about the shaded boxes, but the unshaded boxes correspond exactly to the time slots $\overline{W}$. Hence, we have here that $p_{j'} = 2$ and $p_j = 3$. The sum of completion times $\mathrm{cost}(\sigma)$ does clearly decrease.

Moreover, abbreviate $T = T(\sigma^*)$ and $n_t = n_t(\sigma^*)$, for each time slot t.

We want to upper bound SRPT/OPT, and we know that SRPT/OPT = 1 for $m = 1$ [17]. This can be easily shown by iteratively transforming σ^* into σ' without increasing $\mathrm{cost}(\sigma^*)$. However, this transformation works as well for any schedule σ. We illustrate this for the first time slot 1, whereas we assume that σ and σ' both schedule some job at this time slot: Let j and j' be the jobs scheduled by σ and σ' at time slot 1, respectively. If $j = j'$, then σ and σ' are identical at time slot 1, i.e., $\sigma(1) = \sigma'(1)$. Therefore, assume that $j \neq j'$. Since $m = 1$, we have that $W \cap W' = \emptyset$, where $W := \sigma_j$ and $W' := \sigma_{j'}$. Moreover, since σ' is SRPT-scheduled, we find that $p_{j'} \leq p_j$. By combining these facts, we conclude that, without increasing $\mathrm{cost}(\sigma)$, we can transform σ at time slots $\overline{W} := W \cup W'$ such that job j' is scheduled at these time slots before job j, i.e., such that $i' \leq i$, for each pair $i' \in \sigma_{j'} \cap \overline{W}$ and $i \in \sigma_j \cap \overline{W}$. We illustrate this with an example in Figure 2. This gives that σ' and σ are afterwards identical at time slot 1. This scheme can then be iterated for all following time slots $2, 3, \ldots, T(\sigma)$ such that finally $\sigma = \sigma'$. However, observe that this transformation does not work if $m > 1$, since it is then possible that $W' \subseteq W$, and hence, we cannot transform σ as described above. This is in particular the case for the job pair $j = 3$ and $j' = 2$ in the optimal schedule σ^* in Figure 1.

Since iteratively transforming σ^* into σ' as described in the last paragraph does not work if $m > 1$, we show in Subsection 3.2 how to merge σ^* and σ' to form an incomplete schedule κ with $\mathrm{cost}(\kappa) \leq \mathrm{cost}(\sigma^*)$ and $\mathrm{cost}(\kappa) \leq \mathrm{cost}(\sigma')$. Note that $\mathrm{cost}(\kappa) < \mathrm{cost}(\sigma^*)$ is possible, since κ is incomplete, and hence, the processing times of some jobs j might be smaller in κ than in σ^*, i.e., $p_j(\kappa) < p_j(\sigma^*) = p_j$. More specifically, we define an algorithm, called **MRG**, that simultaneously constructs two schedules κ and κ' from σ^* and σ', respectively, and then we show that $\kappa = \kappa'$. To this end, motivated by the transformation described in the last paragraph, algorithm **MRG** also iterates over the time slots $1, 2, \ldots, T$. Let

$$\Delta^{\sim} := \mathrm{cost}(\sigma^*) - \mathrm{cost}(\kappa)$$

and

$$\Delta^{-} := \mathrm{cost}(\sigma') - \mathrm{cost}(\kappa')$$

be the corresponding cost differences. Since consequently

$$\text{(3.2)} \qquad \mathrm{SRPT} - \mathrm{OPT} = \Delta^{-} - \Delta^{\sim},$$

it suffices to bound $\Delta^{-} - \Delta^{\sim}$ with respect OPT in order to bound SRPT/OPT. We do so in Subsection 3.2, but this only gives that SRPT is 2-competitive, which matches the result of Philipps, Stein, and Wein [13].

To break the barrier of 2, we use randomization. Specifically, in Subsection 3.3, we construct a schedule σ from σ^* with $\mathrm{cost}(\sigma) \geq \mathrm{cost}(\sigma^*)$ by randomly inserting some empty time slots B. Let

$$\Delta^{+} := \mathrm{cost}(\sigma) - \mathrm{cost}(\sigma^*)$$

be the cost difference between σ and σ^*, which is a random variable. We can think of the time slots B as additional *buffer slots*. These buffer slots are then used in Subsection 3.4 to merge σ and σ' to form an incomplete schedule κ with $\mathrm{cost}(\kappa) \leq \mathrm{cost}(\sigma)$ and $\mathrm{cost}(\kappa) \leq \mathrm{cost}(\sigma^*)$ such that we can better control the cost decreases during this merging process than in Section 3.2. Specifically, we extend algorithm **MRG** to a new algorithm, called **MRG'**, which simultaneously construct two random schedules κ and κ' from σ and σ', respectively, and then we show that $\kappa = \kappa'$. For simplicity, as in the last paragraph, we also denote these schedules κ and κ'. Moreover, we define Δ^{-} as in the last paragraph, but we define now

$$\Delta^{\sim} := \mathrm{cost}(\sigma) - \mathrm{cost}(\kappa).$$

Note that, in contrast to the last paragraph, $\Delta^{\sim}$ and Δ^{-} are now random variables. Using these definitions, we obtain that

$$\text{(3.3)} \qquad \mathrm{SRPT} - \mathrm{OPT} = \Delta^{+} + \Delta^{-} - \Delta^{\sim}.$$

As explained in Section 1, we separately upper bound $\mathbb{E}[\Delta^{+}]$ and $\mathbb{E}[\Delta^{-} - \Delta^{\sim}]$ with respect to OPT, where Δ^{+} and $\Delta^{-} - \Delta^{\sim}$ correspond to the random variables X and Y, respectively. By Equation (3.3), this gives us then an improved upper bound for SRPT/OPT.

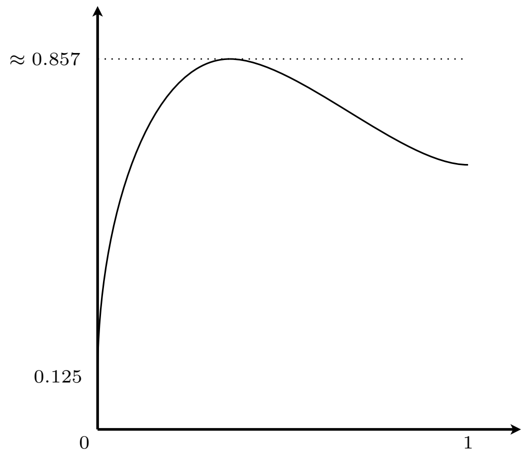


Figure 3: The function $a \mapsto \mathbb{E}[X]+\mathbb{B}_a[X]$ for a random variable X with density function $x \mapsto 7(1-x)^6$ with support $(0,1]$. Hence, we have that $1+\mathbb{E}[X]+\mathbb{B}[X] < 1.86$.

We first introduce some basic operations on a given schedule in Subsection 3.1, and we will mostly modify schedules with these operations in the sections to follow. For a random variable X with $0 < X \leq 1$ and some $0 < a \leq 1$, define

$$\mathbb{B}_a[X] := \frac{1}{1+a}\Big(\Pr[0 < X \leq a] + \Pr[a < X \leq 1]\,\mathbb{E}[X \mid a < X \leq 1]\Big)$$

and

$$\mathbb{B}[X] := \max_{0<a\leq 1} \mathbb{B}_a[X].$$

Using the strategy explained above, we prove the following theorem in Section 3.5.

Theorem 3.1. *For any random variable X with $0 < X \leq 1$,* SRPT *is $(1+\mathbb{E}[X]+\mathbb{B}[X])$-competitive for completion time scheduling.*

Corollary 3.1. SRPT *is* 1.86*-competitive for completion time scheduling.*

Proof. Consider a random variable X with the density function $x \mapsto 7(1-x)^6$ with support $(0,1]$. It can then be easily computed that $1+\mathbb{E}[X]+\mathbb{B}[X] < 1.86$, which, with Theorem 3.1, proves the claim. We schematically depict the function $a \mapsto \mathbb{E}[X]+\mathbb{B}_a[X]$ in Figure 3. $\square$

3.1 Basic operations.

In this subsection, we introduce some basic operations on a given schedule σ. We additionally explain for each operation which prerequisites are needed such that the three feasibility properties given in Section 2 are conserved. Moreover, we sometimes add a prerequisite simply for the sake of exposition. Whenever we apply such an operation, it will always be clear from the context that all prerequisites are satisfied. Finally, we give bounds on the respective change of $\mathrm{cost}(\sigma)$.

We can *move* some job j from time slot t' to time slot t by scheduling this job at time slot t instead of time slot t'. Clearly, to apply this operation, we need the prerequisites $t' \in \sigma_j$ and $t \notin \sigma_j$. Moreover, to ensure that feasibility properties (1) and (2) are conserved, we need the prerequisites $r_j \leq t$ and $|\sigma(t)| < m$, respectively. If $c_j(\sigma) > \max\{t,t'\}$, then $\mathrm{cost}(\sigma)$ does not change. On the other hand, $\mathrm{cost}(\sigma)$ decreases by at least 1 if $t' = c_j(\sigma) > t$. Finally, if $t' = f_{jt}(\sigma) > t$, then $\mathrm{cost}(\sigma)$ decreases by at least 1 if and only if $p_{jt}(\sigma) = 1$. We abbreviate this operation as $\mathbf{move}(\sigma, j, t', t)$.

Consider the scenario that we have a pair of jobs j, j' with $p_{jt}(\sigma) = p_{j't}(\sigma)$ and the additional property that $t \in \sigma_j$. In this case, if the prerequisite $r_{j'} \leq t$ is satisfied, then we can simply *swap* the time slots when these jobs are scheduled after time slot t without modifying $\mathrm{cost}(\sigma)$ such that job j' is scheduled at time slot t afterwards. We abbreviate this operation as $\mathbf{swap}(\sigma, j, j', t)$. We also allow $j = j'$, but in this case, σ is not modified at all.

As explained in Section 2, we also allow incomplete schedules. To obtain such a schedule, we can *remove* some job j from some time slot t, which decreases $p_{jt}(\sigma)$ by 1. Of course, this requires the prerequisite $t \in \sigma_j$. We abbreviate this operation as $\mathbf{remove}(\sigma, j, t)$. Note that if $p_{jt}(\sigma)$ decreases by 1, then so does $p_j(\sigma)$. Consider now an extended scenario where, for some job j and time slot t, we want to decrease $p_{jt}(\sigma)$ by 1 without modifying σ at the earlier time slots $1, 2, \ldots, t-1$ as well. Moreover, assume that σ is SRPT-scheduled after time slot t, and we also want to conserve this property. In such a scenario, we can first decrease $p_{jt}(\sigma)$ by 1, and then simply reschedule σ after time slot t with the SRPT-policy, whereas the remaining processing times of all other jobs remain the same. This operation is abbreviated as $\mathbf{trickle}(\sigma, j, t)$. We add the prerequisite $p_{jt}(\sigma) > 1$, and hence, job j is still scheduled after time slot t afterwards, i.e., $p_{jt}(\sigma) > 0$. Moreover, for technical reasons, we add the prerequisite $r_j \leq t$. In the remainder of this subsection, we prove the following lemma, which upper bounds the decrease of $\mathrm{cost}(\sigma)$ due to operation **trickle**.

Lemma 3.1. *Operation* $\mathbf{trickle}(\sigma, j, t)$ *decreases* $\mathrm{cost}(\sigma)$ *by at most* $n_t(\sigma)/m + 1$.

We prove Lemma 3.1 by explicitly defining operation **trickle**. Note that, due to the fact that ties are broken arbitrarily when selecting jobs, there might be several ways to reschedule σ after time slot t with the SRPT-policy. However, we only need to show that the explicit definition of operation **trickle** results in one such way. Recall that we require the prerequisites $r_j \leq t$ and $p_{jt}(\sigma) > 1$, and moreover that σ is SRPT-scheduled after time slot t.

---

trickle(σ, j, t)

---

1. Set $s \leftarrow t - 1$.
2. Loop over the following steps at least once for job j from the input, and after this first iteration, repeat this while there is a job j with $r_j \leq s$ and $f_{js}(\sigma) \geq s + 1$:
   (a) If this is not the first iteration, set j to the job with minimal $p_{js+1}(\sigma)$ subject to $r_j \leq s$ and $f_{js}(\sigma) \geq s + 1$, where ties are broken arbitrarily.
   (b) Set j' to the job with minimal $f_{j's+1}(\sigma)$ subject to $p_{j's+1}(\sigma) = p_{js+1}(\sigma)$, where ties are broken arbitrarily. Set $s' \leftarrow f_{j's+1}(\sigma)$, and then **swap**$(\sigma, j', j, s')$.
   (c) If this is the first iteration, then **remove**(σ, j, s'), and otherwise, **move**(σ, j, s', s). Finally, set $s \leftarrow s'$.

Observe that the remaining processing time $p_{jt}(\sigma)$ is decreased by 1. This is done by removing job j from time slot s' with operation **remove** in Step 2c of the first iteration. Because of the prerequisite $p_{jt}(\sigma) > 1$, we still have then that job j is scheduled after time slot t. However, we obtain some free scheduling capacity at time slot s', which we use to schedule some other job there. We do this by moving some job to time slot s' with operation **move** in Step 2c of the next iteration if possible. This results again in some free scheduling capacity at a later time slot, and so on. More specifically, let r be the number of iterations of the loop in Step 2 including the last iteration when the stopping condition applies. Hence, we can label these iterations $1, 2, \ldots, r$, and let **remove**(σ, j_1, s_2), **move**(σ, j_2, s_3, s_2), **move**(σ, j_3, s_4, s_3), $\ldots$, **move**$(\sigma, j_r, s_r, s_{r-1})$ be an ordering of the used **remove**- and **move**-operations in this loop with $s_1 := t-1 < s_2 < \ldots < s_r$. Recall here that we use the **remove**-operation only in iteration 1 and no such operation in iteration r. Therefore, just before each iteration $1 < i < r$, we have that $j = j_i$, $s = s_i$, and $s' = s_{i+1}$. Moreover, just before

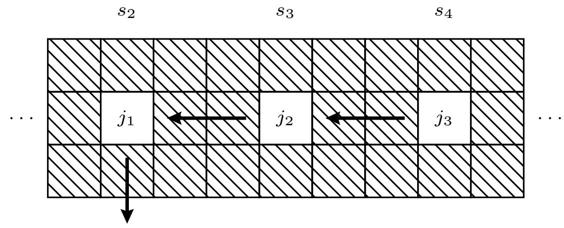


Figure 4: This figure schematically depicts the execution of algorithm **trickle**. First, we use the **remove**-operation to remove job j_1 from time slot s_2 in iteration 1, and afterwards, in each iteration $1 < i < r$, we move job j_i from time slot s_{i+1} to time slot s_i with the **move**-operation.

iteration 1, we have that $j = j_1$ and $s' = s_2$, and just before iteration r, we have that $s = s_r$. Using these definitions, we illustrate operation **trickle** in Figure 4. On the other hand, Step 2b is necessary to preserve the SRPT-property during this process. Before proving Lemma 3.1, we need the following preliminary lemma, which shows that the explicit definition of operation **trickle** given above is indeed correct.

LEMMA 3.2. *After applying operation* **trickle**(σ, j, t), *schedule σ is still* SRPT-*scheduled after time slot t*

Proof. To show that σ is finally SRPT-scheduled after time slot t, we have to show that, for each time slot $t' \geq t$, the jobs $\sigma(t')$ are the (up to) m unfinished jobs j with minimum remaining processing time $p_{jt'}(\sigma) > 0$. We say that σ is SRPT-*scheduled at some time slot t'*, if this property holds only for t'. Hence, σ is finally SRPT-scheduled after time slot t if and only if σ is SRPT-scheduled at each time slot $t' \geq t$. To prove the claim, we will show via induction on the iterations that the following two properties hold just before each iteration i:

(1) schedule σ is SRPT-scheduled at each time slot $t' \geq t$ with $t' \neq s_i$,

(2) if $i > 1$, then $|\sigma(s_i)| \leq m - 1$, and $\sigma(s_i)$ are the jobs with minimum remaining processing time at time slot s_i, where ties are broken arbitrarily, i.e., for each job $j \in \sigma(s_i)$ and each job $j' \notin \sigma(s_i)$ with $r_{j'} \leq s_i$ and $p_{j's_i}(\sigma) > 0$, we have that $p_{js_i}(\sigma) \leq p_{j's_i}(\sigma)$. Moreover, if even $|\sigma(s_i)| < m - 1$, then there is at most one job j with $r_j \leq s_i$ and $f_{js_i}(\sigma) \geq s_i + 1$.

Hence, these properties hold as well just before iteration r, which is the iteration when the stopping condition of the loop applies. By combining these facts, we obtain that σ is finally SRPT-scheduled after time slot t, which proves the claim of the lemma.

Induction start. Property (1) holds just before iteration 1, since σ is initially SRPT-scheduled after time slot $t = s_1 + 1$. Property (2) trivially holds then as well, since we only consider here the case $i > 1$.

Induction step. Assume as induction hypothesis that properties (1) and (2) hold just before iteration i. We will separately show that these properties then hold just before iteration $i + 1$ as well.

Property (2): Since $s_{i+1} > s_i$, we know from property (1) of the induction hypothesis that σ is SRPT-scheduled at time slot s_{i+1} just before iteration i. Consider the job j' selected in Step 2b of iteration i with $f_{j's_i+1}(\sigma) = s_{i+1}$. We change the set of jobs $\sigma(s_{i+1})$ scheduled at time slot s_{i+1} by scheduling job j_i instead of job j' at this time slot using operation **swap**$(\sigma, j', j_i, s_{i+1})$. However, Since $p_{js_{i+1}}(\sigma) = p_{j's_{i+1}}(\sigma)$, we have that σ is then still SRPT-scheduled at time slot s_{i+1}. Consequently, since we remove job j_i from time slot s_{i+1} with operation **move**$(\sigma, j_i, s_{i+1}, s_i)$ (or operation **remove**(σ, j_i, s_{i+1}) if $i = 1$) in Step 2c, we immediately obtain that property (2) holds just before iteration $i + 1$.

Property (1): Consider a fixed $t' \geq t$ with $t' \neq s_{i+1}$, and distinguish four cases:

Case $t' < s_i$: If $i = 1$, then $t' < t$, and hence, this case is not possible. On the other hand, if $i > 1$, then, for any job j, we do modify the remaining processing time $p_{jt'}(\sigma)$ at time slot t' during iteration i, which gives with property (1) of the induction hypothesis that σ is still SRPT-scheduled at time slot t' just before iteration $i + 1$.

Case $t' = s_i$: As in the previous case, if $i = 1$, then $t' < t$, and hence this case is not possible. On the other hand, if $i > 1$, then property (2) of the induction hypothesis allows us to distinguish two subcases:

Case $|\sigma(s_i)| = m - 1$: Because of the selection of j_i in Step 2a of iteration i, we have that σ is SRPT-scheduled at time slot t' after operation **move**$(\sigma, j_i, s_{i+1}, s_i)$ in Step 2c, and therefore just before iteration $i + 1$.

Case $|\sigma(s_i)| < m - 1$: There is at most one job j with $r_j \leq s_i$ and $f_{js_i}(\sigma) \geq s_i+1$. If there is no such job, then the stopping condition of the loop implies that $i = r$, and hence, iteration $i + 1$ does not exist. Otherwise, it follows that $j_i = j$, and hence, by using the same arguments as in the last case, we obtain that σ is SRPT-scheduled at time slot t' just before iteration $i + 1$.

Case $s_i < t' < s_{i+1}$: Assume for contradiction that σ is not SRPT-scheduled at time slot t' just before iteration $i + 1$. By the fact that j_i is the only job whose remaining processing time $p_{j_it'}(\sigma)$ at time slot t' decreases during iteration i due to operation **move**$(\sigma, j_i, s_{i+1}, s_i)$ (or operation **remove**(σ, j_i, s_{i+1}) if $i = 1$) in Step 2c, we must have that, just before iteration $i + 1$, there is a job $j \in \sigma(t')$ such that $p_{jt'}(\sigma) > p_{j_it'}(\sigma) > 0$. If even $p_{jt'}(\sigma) > p_{j_it'}(\sigma) + 1$, then, since we only decrease $p_{j_it'}(\sigma)$ by 1, it already holds just before iteration i that $p_{jt'}(\sigma) > p_{j_it'}(\sigma)$. But because also $r_{j_i} \leq s_i \leq t'$, this gives that σ was not SRPT-scheduled at time slot t' just before iteration i, since we would have scheduled j_i there instead of j, which contradicts to property (1) of the induction hypothesis. Therefore, it must hold that $p_{jt'}(\sigma) = p_{j_it'}(\sigma) + 1$ after Step 2c of iteration i, and hence

$$p_{jt'}(\sigma) = p_{j_it'}(\sigma) \tag{3.4}$$

just before iteration i. On the other hand, we know from property (1) of the induction hypothesis that σ is SRPT-scheduled after time slot $s_i + 1$ just before iteration i. By combining this with Equation (3.4) and the facts that $f_{js_i+1}(\sigma) \leq t' < s_{i+1} \leq f_{j_is_i+1}(\sigma)$ and $r_{j_i} \leq s_i$, it is easy to see that $f_{js_i+1}(\sigma) = t'$. Thus, we find that even $p_{js_i+1}(\sigma) = p_{j_is_i+1}(\sigma)$ just before iteration i. This gives a contradiction, since $t' < s_{i+1}$, and hence, we would have applied operation **swap**(σ, j, j_i, t') in Step 2b of iteration i.

Case $t' > s_{i+1}$: By property (1) of the induction hypothesis, we have that σ is SRPT-scheduled at time slot t' just before iteration i. Note that it might happen in iteration i that σ is modified after time slot t' with the single **swap**-operation in Step 2b. However, this clearly does not affect the property that σ is SRPT-scheduled at time slot t', and hence, σ is still SRPT-scheduled at time slot t' just before iteration $i + 1$.

□

Proof. [Lemma 3.1] We know from Lemma 3.2 that the explicit definition of operation **trickle** given above is indeed correct.

Consider now operation **trickle** without the **remove**- and **move**-operation in Step 2c. We refer to this modified operation as **trickle'**. Note that **trickle'** does not affect the SRPT-property or $\mathrm{cost}(\sigma)$, but simply swaps jobs with the same remaining processing time after some time slot. For simplicity, assume that operation **trickle'** has already been applied before operation **trickle** with the same input parameters, and that, instead of breaking ties arbitrarily, we always select the same job j in Step 2a in both operations. As a consequence, we still obtain the same final schedule σ even if we omit Step 2b in operation **trickle**, which helps us to simplify the arguments to follow. We can also think of this as dividing the original operation **trickle** in two operations.

The only step in operation **trickle** that might decrease $\mathrm{cost}(\sigma)$ in each iteration is Step 2c. However, since initially $p_{jt}(\sigma) > 1$, we have that the **remove**-operation in the first iteration 1 does not modify $\mathrm{cost}(\sigma)$. Therefore, we only have to consider the **move**-operations in the following iterations $2, 3, \dots, r-1$. We use *chunk* to refer to a maximal subinterval $C = \{a, \dots, b\}$ of interval $\{2, 3 \dots, r-1\}$ with the property that all elements in this subinterval index the same job, i.e., j_i is the same job for each $a \le i \le b$, and, if $a > 2$, then $j_{a-1} \ne j_a$, and, if $b < r-1$, then $j_{b+1} \ne j_b$. Note that, for each such chunk $C = \{a, \dots, b\}$, only the last **move**-operation, namely $\mathbf{move}(\sigma, j_b, s_{b+1}, s_b)$, can decrease $\mathrm{cost}(\sigma)$, and this happens if and only if $p_{j_b s_b+1}(\sigma) = 1$. Specifically, the decrease is then exactly

$$\Delta_C := s_{b+1} - s_b.$$

Let $\mathcal{C}$ be the chunks that decrease $\mathrm{cost}(\sigma)$ in this way. Using this, we obtain that the decrease of $\mathrm{cost}(\sigma)$ during operation $\mathbf{trickle}(\sigma, j, t)$, say Δ, satisfies

$$\Delta = \sum_{C \in \mathcal{C}} \Delta_C. \tag{3.5}$$

Moreover, we refer to the chunk C with $2 \in C$ as the *first chunk*. In what follows, we will define some pairwise disjoint job sets J_C, $C \in \mathcal{C}$, with $J_C \subseteq \{j \mid p_{jt}(\sigma) > 1\}$ and

$$|J_C| \ge \begin{cases} m(\Delta_C - 1) & C \text{ is the first chunk,} \\ m\Delta_C & \text{otherwise.} \end{cases} \tag{3.6}$$

By using these properties and Equation (3.5), we find that

$$\Delta \le \frac{\sum_{C \in \mathcal{C}} |J_C|}{m} + 1 = \frac{|\cup_{C \in \mathcal{C}} J_C|}{m} + 1 \le \frac{n_t(\sigma)}{m} + 1,$$

which proves the claim of the lemma. For each chunk $C \in \mathcal{C}$, we add jobs to J_C in two steps, first some *large jobs*, and then some *small jobs*. Recall that we consider here the state of σ before applying operation **trickle**, but after the application of operation **trickle'**.

Adding large jobs. Consider a fixed chunk $C = \{a, \dots, b\} \in \mathcal{C}$, and assume that C is not the first chunk. In this case, it must hold that $|\sigma(s_a)| = m$. Otherwise, since σ is SRPT-scheduled after time slot $t \le s_a$, we have that j_a would have already been scheduled at time slot s_a before applying operation **trickle**. Note that j_a was not scheduled there because of the selection of j_a in Step 2a and the maximality-property in the definition of a chunk. Add then the jobs $(\sigma(s_a) \backslash \{j_{a-1}\}) \cup \{j_a\}$ to J_C. We refer to these jobs as the *large jobs*. Then, because σ is SRPT-scheduled after time slot $t \le s_a$ and $r_{j_a} \le s_a$, we have for each job $j \in J_C$ that $p_{js_a}(\sigma) \le p_{j_a s_a}(\sigma)$, and hence $p_{js_a+1}(\sigma) < p_{j_a s_a+1}(\sigma)$, since $j \in \sigma(s_a)$. Therefore, again since σ is SRPT-scheduled after time slot $t \le s_a$, it holds that $c_j(\sigma) \le s_b < s_{b+1}$. To see this, simply note that whenever j_a is scheduled at some time slot after s_a, then j is either already completed, or scheduled as well. We conclude that if we use this construction, then $J_C \cap J_{C'} = \emptyset$, for each $C' \in \mathcal{C}$ with $C' \ne C$. Moreover, we have so far that $|J_C| = m$, and this suffices to obtain Inequalities (3.6) if $\Delta_C = 1$.

Adding small jobs. If $\Delta_C = s_{b+1} - s_b > 1$, then the large jobs already added to J_C above are not sufficient to guarantee Inequalities (3.6). However, the definition of $\mathcal{C}$ gives that $p_{j_b s_b+1}(\sigma) = 1$. Therefore, because σ is SRPT-scheduled after time slot t, we have that $p_{js_b+1}(\sigma) = 1$, for each time slot t' with $s_b < t' < s_{b+1}$ and each job $j \in \sigma(t')$, i.e., each such job is scheduled at exactly one of the time slots $s_b+1, s_b+2, \dots, s_{b+1}-1$ and also completed there. That is why we refer to these job as the *small jobs*, and consequently, there are exactly $m(s_{b+1} - s_b - 1)$ many such small jobs, which we also add to J_C. Together with the already added large jobs, we hence have that $|J_C| = m(s_{b+1} - s_b)$, which satisfies Inequalities (3.6). As already mentioned above, each large job is completed before time slot s_b, and hence a large job is never also a small job. Clearly, it still holds then that $J_C \cap J_{C'} = \emptyset$, for each $C' \in \mathcal{C}$ with $C' \ne C$. Finally, if C is the first chunk, then we can only add the small jobs to J_C, and hence only $|J_C| \ge m(\Delta_C - 1)$. This completes the proof that Inequalities (3.6) are satisfied. We give an example for the adding of large and small jobs in Figure 5. $\square$

3.2 Merging σ^* and σ' to form κ. In this subsection, we use the operations from Section 3.1 to give a simple proof that SRPT is 2-competitive. To this end, we construct two schedules κ and κ' from σ^* and σ' with the following algorithm, respectively. Recall that $T = T(\sigma^*)$.

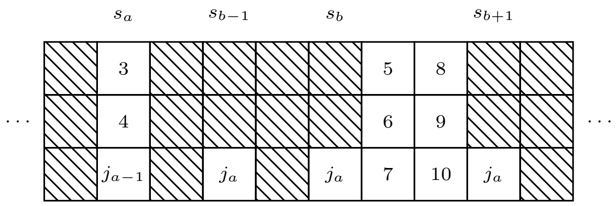


Figure 5: This figure depicts an example for the adding of large and small jobs, wherein we consider a chunk $C = \{a, \dots, b\} \in \mathcal{C}$ with $|C| = 3$. The two jobs 3 and 4 are the large jobs, and the jobs $5, 6 \dots, 10$ are the small jobs in J_C. We do not know anything about the shaded boxes. Note that since $j_a = j_{b-1} = j_b$, we only use j_a to label the job corresponding to chunk C.

MRG(σ^*, σ')

---

1. Set $\kappa \leftarrow \sigma^*$ and $\kappa' \leftarrow \sigma'$.
2. For $t = 1, \dots, T$:
   (a) Set $p \leftarrow \max_{j \in \kappa'(t)} p_{jt}(\kappa')$ to the maximal remaining processing time of a job scheduled by κ' at time slot t. While there are jobs $j \in \kappa(t) \backslash \kappa'(t)$ and $j' \in \kappa'(t) \backslash \kappa(t)$ with $p_{jt}(\kappa) = p_{j't}(\kappa) = p$, **swap**$(\kappa, j, j', t)$.
   (b) Set $K \leftarrow \kappa(t) \backslash \kappa'(t)$ and $K' \leftarrow \kappa'(t) \backslash \kappa(t)$, and *associate* each job $j \in K$ with a job $j' \in K'$ with $p_{j't}(\kappa) < p_{jt}(\kappa)$ such that no two jobs in K are associated with the same job.
   (c) For each job $j' \in K'$ which was not associated with a job $j \in K$, **move**$(\kappa, j', c_{j'}(\kappa), t)$.
   (d) For each job $j \in K$, **remove**(κ, j, t), **move**(κ, j', t', t), and **trickle**$(\kappa', j, t + 1)$, whereas $j' \in K'$ is the job associated with j and $t' \leftarrow c_{j'}(\kappa)$.
3. Return κ and κ'.

We refer to an iteration of the loop in Step 2 simply as an *iteration*, and we use the terms iteration and time slot interchangeably. The main idea of algorithm **MRG** is to transform κ and κ' in each iteration t such that ultimately $\kappa(t) = \kappa'(t)$, whereas we iteratively adapt $\kappa(t)$ to $\kappa'(t)$. More specifically, in Step 2a, we swap jobs as often as possible to achieve this. On the other hand, Step 2c is only important if initially $|\kappa'(t)| > |\kappa(t)|$, since we can then simply adapt $\kappa(t)$ to $\kappa'(t)$ by moving jobs to time slot t. Finally, in Step 2d, we apply a more involved sequence of operations which also decreases some remaining processing times with the **remove**- and **trickle**-operations. The only problematic step with respect to the correctness of algorithm **MRG** is Step 2b, since we need to ensure that we can associate the jobs K in the described way. Recall that $\Delta^- = \text{cost}(\sigma') - \text{cost}(\kappa')$ and $\Delta^\sim = \text{cost}(\sigma^*) - \text{cost}(\kappa)$, and that $n_t = n_t(\sigma^*)$, for each time slot t.

Example. Consider the SRPT schedule σ' and the optimal schedule σ^* from Figure 1. Moreover, consider the first iteration $t = 1$ when applying algorithm **MRG** to these two schedules. We have in this iteration that initially $\kappa(t) = \{1, 3\}$ and $\kappa'(t) = \{1, 2\}$, and hence $K = \{3\}$ and $K' = \{2\}$. Since $p_{3,1}(\kappa) = 2 > p_{2,1}(\kappa) = 1$, no jobs are swapped with the **swap**-operation in Step 2a, but we associate job $j = 3$ with job $j' = 2$ in Step 2b. Consequently, in Step 2d, the remaining processing times $p_{3,1}(\kappa)$ and $p_{3,1}(\kappa')$ are decreased by 1 with operations **remove** and **trickle**, respectively, and we ultimately have due to the **move**-operation that job j' is scheduled by both schedules κ and κ' at time slot 1. We obtain that κ and κ' are now identical, and hence all following iterations do not modify these schedules.

LEMMA 3.3. *If, just before some iteration t, (1) κ and κ' define the same instance after time slot t, and (2) κ' is* SRPT*-scheduled after time slot t, then iteration t can be executed in the described way.*

Proof. We only need to show that we can associate the jobs K in the way described in Step 2b, since all other steps can then clearly be executed. By prerequisite (1), we can abbreviate $p_{jt} = p_{jt}(\kappa) = p_{jt}(\kappa')$, for each job j. On the other hand, by prerequisite (2), we have that κ' schedules as many jobs as possible at time slot t, and hence $|\kappa'(t)| \geq |\kappa(t)|$. Let now p be as defined in Step 2a. Therefore, by prerequisite (2), we initially have before Step 2a that $\{j \in \kappa(t) \mid p_{jt} < p\} \subseteq \{j \in \kappa'(t) \mid p_{jt} < p\}$. Moreover, we have after Step 2a that one of the two sets $\{j \in \kappa(t) \mid p_{jt} = p\}$ and $\{j \in \kappa'(t) \mid p_{jt} = p\}$ is contained in the other one. Consequently, by the setting of K and K' in Step 2b, we find that $p_{j't} < p_{jt}$, for each pair of jobs $j' \in K'$ and $j \in K$. On the other hand, since initially $|\kappa(t')| \geq |\kappa(t)|$, we still have that $|K'| \geq |K|$. By combining these facts, we obtain that we can associate the jobs K with some jobs in K' in the way described in Step 2b. $\square$

LEMMA 3.4. *Algorithm* **MRG** *terminates correctly, and, just before each iteration t, (1) κ and κ' are identical at each time slot $s < t$, (2) κ and κ' define the same instance after time slot t, and (3) κ' is* SRPT*-scheduled after time slot t.*

Proof. We show via induction on the iterations that the three parts of the claim hold just before each iteration t. Using Lemma 3.3 during this induction also gives the correctness of algorithm **MRG**.

Induction start. The three parts hold just before iteration $t = 1$, since κ' is initially SRPT-scheduled and $p_{j1}(\kappa) = p_{j1}(\kappa') = p_j$, for each job j.

Induction step. Consider a fixed iteration t, and assume as induction hypothesis that the three parts hold just before iteration t. We will show that they then still hold just before iteration $t+1$. To this end, recall that the purpose of the sequence of operations in Step 2d is to ensure that job $j' \in K'$ is scheduled by κ at time slot t instead of job $j \in K$, which, in combination with Steps 2a and 2c, implies that $\kappa(t) = \kappa'(t)$ after iteration t. This shows that part (1) still holds just before iteration $t + 1$. However, to do so, we need to decrease $p_{jt}(\kappa)$ with the **remove**-operation, since this gives us some extra scheduling capacity at time slot t, which we use to schedule job j' with the **move**-operation there. Likewise, we use the **trickle**-operation to also decrease $p_{jt}(\kappa')$. This ensures that still $p_{jt}(\kappa) = p_{jt}(\kappa')$ afterwards, and hence, since $\kappa(t) = \kappa'(t)$, also $p_{jt+1}(\kappa) = p_{jt+1}(\kappa')$. Consequently, also part (2) still holds just before iteration $t + 1$. Moreover, by the definition of operation **trickle**, we have that κ' is still SRPT-scheduled after time slot $t + 1$, which finally implies that part (3) still holds just before iteration $t+1$. Combining all this proves the induction step. However, it is not clear that all prerequisites of the **trickle**-operation are satisfied. The prerequisite $r_j \le t + 1$ is clearly satisfied, and we moreover know from the induction hypothesis that κ' is SRPT-scheduled after time slot $t + 1$. Consequently, we only need to show in the next paragraph that $p_{jt+1}(\kappa') > 1$.

Note that $f_{jt}(\kappa') \ge t + 1$, since otherwise $j \in \kappa'(t)$, and in this case, we have that $j \notin K$ because of the setting of K in Step 2b. Assume now for contradiction that $p_{jt+1}(\kappa') = 1$. Since $f_{jt}(\kappa') \ge t + 1$, it holds that $p_{jt}(\kappa') = p_{jt+1}(\kappa')$, and consequently, because of part (2) of the induction hypothesis, we have that $p_{jt}(\kappa) = p_{jt}(\kappa') = 1$ as well. On the other hand, since $f_{jt}(\kappa') \ge t + 1$, $r_j \le t$, and $p_{jt}(\kappa') = 1$, part (3) of the induction hypothesis implies that all jobs scheduled by κ' at time slot t must be completed at this time slot, and therefore $p = 1$. Consequently, because of Step 2a and the setting of K in Step 2b, we have that $j \notin K$, which gives a contradiction. We conclude that $p_{jt+1}(\kappa') > 1$. □

Lemma 3.5. *Just before each iteration t, we have that $n_{t+1}(\kappa') \le n_{t+1}$.*

Proof. By part (2) of Lemma 3.4, we have that $n_t(\kappa) = n_t(\kappa')$. Consequently, by part (3), it follows that $n_{t+1}(\kappa) \ge n_{t+1}(\kappa')$. On the other hand, since initially $n_{t+1}(\kappa) = n_{t+1}$, and we clearly never increase $n_{t+1}(\kappa)$ with any of the invoked operations during algorithm **MRG**, we have that $n_{t+1}(\kappa) \le n_{t+1}$. Combining these facts completes the proof of the claim. □

Lemma 3.6. $\Delta^- - \Delta^\sim \le \mathrm{OPT}$

Proof. Recall that Δ^- and $\Delta^\sim$ increase exactly as $\mathrm{cost}(\kappa')$ and $\mathrm{cost}(\kappa)$ decrease, respectively. First, since $c_{j'}(\kappa) > t$, note that the **move**-operation in Step 2c cannot increase $\mathrm{cost}(\kappa)$, which holds for the **remove**-operation in Step 2d as well. Consequently, since we want to upper bound $\Delta^- - \Delta^\sim$, we only need to consider the **trickle**- and **move**-operation in Step 2d. Consider now a fixed iteration t and a fixed job $j \in K$ in this iteration. By Lemma 3.1, we have that $\mathrm{cost}(\kappa')$ decreases by at most $n_{t+1}(\kappa')/m+1$ due to the **trickle**-operation in Step 2d. On the other hand, since $t < t' = c_{j'}(\kappa)$, we also have that $\mathrm{cost}(\kappa)$ decreases then by at least 1 due to the **move**-operation in the same step. Consequently, since clearly $n_{t+1} \le n_t$, Lemma 3.5 implies that $\Delta^- - \Delta^\sim$ increases by at most n_t/m for job j. Thus, because $|K| \le m$ in each iteration, by summing this up for all iterations and all jobs $j \in K$ in the respective iterations, we obtain that $\Delta^- - \Delta^\sim \le \sum_{t=1}^{\infty} n_t$, which proves the claim in combination with the alternative definition of the objective function (2.1). □

Theorem 3.2. SRPT *is 2-competitive for completion time scheduling.*

Proof. It follows from parts (2) and (3) of Lemma 3.4 that ultimately $\kappa = \kappa'$, and hence, Equation (3.2) indeed holds. Therefore, the claim of the theorem follows from Lemma 3.6. □

3.3 Construction of σ from σ^*.

In this subsection, using a random variable X with $0 < X \le 1$, we randomly construct a schedule σ from σ^*, wherein we also construct two functions $A : \{1, \dots, T\} \to \{1, \dots, 2T\}$ and $B : \{1, \dots, T\} \to \{1, \dots, 2T\}$. Recall that $T = T(\sigma^*)$ and $n_s = n_s(\sigma^*)$, for each time slot s. Given a sequence of i.i.d random variables $X_1, X_2, \dots, X_T$ which are distributed as X, we first construct a function $\pi : \{1, \dots, T\} \to \{2, \dots, T + 1\}$ as follows: For each $1 \le s < T$, let $\pi(s) := i + 1$, where $s + 1 \le i \le T$ is such that

$$\frac{n_{i+1}}{n_{s+1}} < X_s \le \frac{n_i}{n_{s+1}}.$$

Moreover, define $\pi(T) := T+1$. We illustrate the setting of $\pi(s)$ in Figure 6.

Let then A and B be the two injective functions with $A(\{1, \dots, T\}) \cup B(\{1, \dots, T\}) = \{1, \dots, 2T\}$ and $B(s) > A(s)$, for each $1 \le s \le T$, that, for each pair $1 \le s < s' \le T$, satisfy the following three properties:

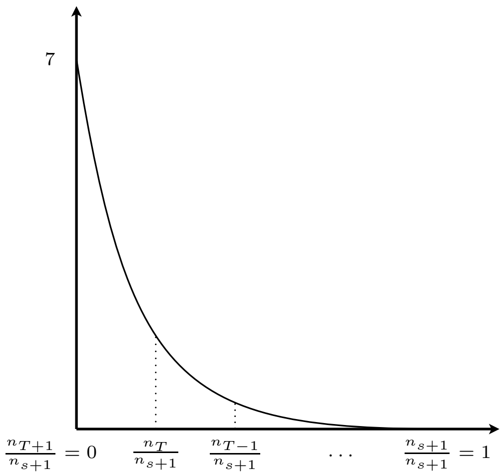


Figure 6: Assume that the random variable X has the density function $x \mapsto 7(1-x)^6$, whose graph is depicted in this figure. Then, for each $s+1 \le i \le T$, the area below this curve between n_{i+1}/n_{s+1} and n_i/n_{s+1} is the probability that $\pi(s) = i+1$.

(1) $A(s) < A(s')$,

(2) $B(s) < B(s') \iff \pi(s) < \pi(s')$,

(3) $B(s) < A(s') \iff \pi(s) \le s'$.

Combining all these properties clearly defines A and B. Let A and B also denote the images $A(\{1,\dots,T\})$ and $B(\{1,\dots,T\})$ of the functions A and B, respectively. Let now σ such that, for each $1 \le s \le T$, $\sigma(A(s)) = \sigma^*(s)$ and $\sigma(B(s)) = \emptyset$, and, for each $t > 2T$, $\sigma(t) = \emptyset$. Observe that $n_{A(s)}(\sigma) = n_s$ and $n_{B(s)}(\sigma) = n_{\pi(s)}$. Finally, define the function $C : A \to B$ as $C(t) := B(A^{\leftarrow}(t))$. We also refer to $C(t)$ as the *buffer slot* of time slot t. Recall that $\Delta^+ = \mathrm{cost}(\sigma) - \mathrm{cost}(\sigma^*)$. This construction is best illustrated with an example.

Example. Assume that σ^* is the optimal schedule from Figure 1. In this case, we have that $T = 4$, $n_1 = 7$, $n_2 = 6$, $n_3 = 4$, $n_4 = 2$, and $n_5 = 0$. Moreover, assume that $X_1 = 1/2$ and $X_2 = 2/3$. In this case, we have that $\pi(1) = \pi(2) = 4$. Note that, independent of the outcome of X_3 and X_4, we have that $\pi(3) = \pi(4) = 5$. We illustrate the resulting functions A, B, and C in Figure 7, where we also illustrate the resulting schedule σ. We have that $A = \{1,2,3,6\}$ and $B = \{4,5,7,8\}$. Note that $\Delta^+ = \mathrm{cost}(\sigma) - \mathrm{cost}(\sigma^*) = \sum_{s=1}^{4} n_{B(s)}(\sigma) = 2+2+0+0 = 4$.

Lemma 3.7. $\mathbb{E}[\Delta^+] \le \mathbb{E}[X]\,\mathrm{OPT}$

To prove Lemma 3.7, we apply the following lemma, which we also need in Section 3.5.

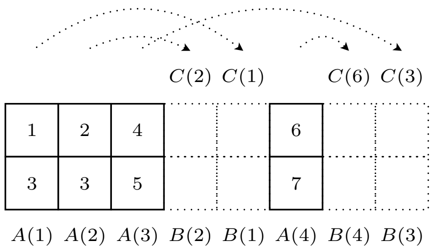


Figure 7: The schedule σ from σ^* for $X_1 = 1/2$ and $X_2 = 2/3$. The bended arcs represent the function C.

Lemma 3.8. *For each pair* $1 \le s < s' \le T+1$,

$$\mathbb{E}\left[n_{\pi(s)} \;\middle|\; \pi(s) \le s'\right] < n_{s+1}\mathbb{E}\left[X_s \;\middle|\; \frac{n_{s'}}{n_{s+1}} < X_s \le 1\right].$$

Proof. Observe that the claim of the lemma does not make sense for $s' = s+1$, since it is then impossible that $\pi(s) \le s'$. However, this case is never needed in what follows, and hence, we use this notation for the sake of simplicity.

First, note that, for each i with $s+1 \le i \le T$,

$$\begin{aligned}(3.7)\quad \mathbb{E}\left[n_{\pi(s)} \;\middle|\; \frac{n_{i+1}}{n_{s+1}} < X_s \le \frac{n_i}{n_{s+1}}\right] &= n_{i+1} \\ &< n_{s+1}\mathbb{E}\left[X_s \;\middle|\; \frac{n_{i+1}}{n_{s+1}} < X_s \le \frac{n_i}{n_{s+1}}\right].\end{aligned}$$

Using this, we obtain that

$$\begin{aligned}\mathbb{E}\left[n_{\pi(s)} \;\middle|\; \pi(s) \le s'\right] &= \mathbb{E}\left[n_{\pi(s)} \;\middle|\; \frac{n_{s'}}{n_{s+1}} < X_s \le 1\right] \\ &= \sum_{i=s+1}^{s'-1}\Bigg[\Pr\left[\frac{n_{i+1}}{n_{s+1}} < X_s \le \frac{n_i}{n_{s+1}} \;\middle|\; \frac{n_{s'}}{n_{s+1}} < X_s \le 1\right] \\ &\quad \cdot \mathbb{E}\left[n_{\pi(s)} \;\middle|\; \frac{n_{i+1}}{n_{s+1}} < X_s \le \frac{n_i}{n_{s+1}}\right]\Bigg] \\ &< n_{s+1}\mathbb{E}\left[X_s \;\middle|\; \frac{n_{s'}}{n_{s+1}} < X_s \le 1\right],\end{aligned}$$

which proves the claim. The first line is due to the definition of π. Finally, the third line is due to Inequality (3.7). $\square$

Proof. [Lemma 3.7] We have that

$$\begin{aligned}\mathbb{E}\left[\Delta^+\right] &= \mathbb{E}\left[\sum_{s=1}^{T} n_{B(s)}(\sigma)\right] \\ &= \sum_{s=1}^{T} \mathbb{E}\left[n_{\pi(s)}\right] \\ &< \sum_{s=1}^{T} n_{s+1}\mathbb{E}\left[X\right] \\ &\leq \mathbb{E}\left[X\right]\mathrm{OPT},\end{aligned}$$

which proves the claim. The first line is due to the alternative definition of the objective function (2.1) and the fact that Δ^+ results from the insertion of the additional buffer slots B. The second line is due to the fact that $n_{B(s)}(\sigma) = n_{\pi(s)}$, for each $1 \leq s \leq T$, and linearity of expectation. The third line is due to Lemma 3.8 for $s' = T + 1$, since then $n_{s'}/n_{s+1} = 0$, and finally, the fourth line is due to the simple fact that $n_{s+1} \leq n_s$ and the alternative definition of the objective function (2.1). □

3.4 Merging σ and σ' to form κ. In this subsection, we extend algorithm **MRG** from Section 3.2 in order to merge σ and σ'. Specifically, we replace σ^* by σ in the input, loop in Step 2 over the range $1, \ldots, 2T$ instead of $1, \ldots, T$, and finally, we replace the sequence of three operations in Step 2d by the following case distinction, where additional inputs include the sets of time slots A and B, and the function $C : A \to B$ defined in Subsection 3.3.

Case (Bad) $t \in B$ **or** $C(t) > t'$**:**

remove(κ, j, t), **move**(κ, j', t', t), and **trickle**$(\kappa', j, t + 1)$.

Case (Good) $t \in A$ **and** $C(t) < t'$**:**

First, **move**$(\kappa, j', t', C(t))$. Afterwards, set $W \leftarrow \{i \in \kappa_j \mid i \geq t\}$ and $W' \leftarrow \{i \in \kappa_{j'} \mid i \geq t\}$ to the time slots when jobs j and j' are scheduled by κ after time slot t, respectively, and set $\overline{W} \leftarrow (W \cup W')\backslash(W \cap W')$ to the time slots among these time slots when exactly one of them is scheduled. Modify κ at time slots $\overline{W}$ such that j' is scheduled at these time slots before j, i.e., such that $i' \leq i$, for each pair $i' \in \kappa_{j'} \cap \overline{W}$ and $i \in \kappa_j \cap \overline{W}$. (Observe that this transformation is closely related to the transformation of σ^* for $m = 1$ given in the very beginning of this section.)

Let **MRG'** denote the resulting algorithm. We still refer to an iteration of the loop in Step 2 simply as an *iteration*. As in algorithm **MRG**, the main idea of algorithm **MRG'** is to transform κ and κ' in each iteration t such that ultimately $\kappa(t) = \kappa'(t)$. Note that if we are in the Bad Case, then we proceed exactly as in algorithm **MRG**. On the other hand, if we are in the Good Case, then we do not decrease any processing time of a job in κ or κ', but simply use the initially empty buffer slot $C(t)$ to modify κ. Observe that κ' is not modified at all in the Good Case, and that, since $p_{j't}(\kappa) < p_{jt}(\kappa)$, cost$(\kappa)$ can only decrease. Moreover, note that due to the fact that $j' \in K'$, we have that $t \in \overline{W}$, which implies that job j' is finally scheduled at time slot t by κ instead of job j. On the other hand, due to the initial **move**-operation, we also have that $C(t) \in \overline{W}$. Finally, observe that one less and one more job is scheduled at time slots t' and $C(t)$, respectively. Analogously to Lemma 3.4, the following lemma holds.

Lemma 3.9. *Algorithm* **MRG'** *terminates correctly, and ultimately $\kappa = \kappa'$. Moreover, just before each iteration t, we have that $n_{t+1}(\kappa') \leq n_{t+1}(\sigma)$.*

Proof. The claim can be proven by using the same arguments as in the proofs of Lemmas 3.3, 3.4, and 3.5. We only need to argue that they extend to the additional Good Case.

First, Lemma 3.3 extends to the Good Case since it follows from the definition of algorithm **MRG'** that the buffer slot $C(t)$ is empty just before iteration t, and hence, we never have that $C(t) = t'$. Consequently, the case distinction in Step 2d is correct, and therefore, iteration t can be executed in the described way. Second, since Lemma 3.3 extends to the Good Case, Lemma 3.4 extends to the Good Case as well. To see this, observe that we can simply extend the induction step to the Good Case. Specifically, recall that whenever we run into the Good Case, then we ultimately have that job j' is scheduled at time slot t instead of job j, which is exactly what we need to ensure that finally $\kappa(t) = \kappa'(t)$. Moreover, we have that $p_{jt}(\kappa)$ does not change and κ' is not modified at all. Finally, since Lemma 3.4 extends to the Good Case, also Lemma 3.5 extends to the Good Case, since we also clearly never increase $n_{t+1}(\kappa)$ with any of the invoked operations during algorithm **MRG'**. This completes the proof of the lemma. □

For each iteration t, let R_t be the multiset of all considered time slots t', i.e., $t' = c_{j'}(\kappa)$ for the considered job $j \in K$ in Step 2d. Recall that a multiset may contain elements more than once. Observe that $|R_t| \leq m$, since $|K| \leq m$. Let then the multiset R'_t be the time slots $t' \in R_t$ for which we run into the Bad

Case. Finally, for each $t \in A \cup B$, let

$$\Delta_t := |R'_t|\frac{n_{t+1}(\sigma)}{m}, \text{ and define } \Delta := \sum_{t \in A\cup B} \Delta_t.$$

In the remainder of this subsection, we prove the following lemma, which states that it suffices to bound $\mathbb{E}[\Delta]$ in order to bound $\mathbb{E}[\Delta^- - \Delta^\sim]$.

Lemma 3.10. $\mathbb{E}[\Delta^- - \Delta^\sim] \le \mathbb{E}[\Delta]$

To prove Lemma 3.10, we need one preliminary lemma.

Lemma 3.11. $\Delta^\sim \ge \sum_{t\in A\cup B} |R'_t|$

Proof. We show that $\Delta^\sim$ increases by at least $|R'_t|$ in each iteration t, which is equivalent to the claim that $\text{cost}(\kappa)$ decreases by at least $|R'_t|$. Recall that we proceed as in algorithm **MRG** whenever we run into the Bad Case, and we do this $|R'_t|$ times in each iteration t. Hence, by the arguments in the proof of Lemma 3.6, we find that $\text{cost}(\kappa)$ decreases by at least $|R'_t|$ during all Bad Cases. Therefore, we only need to show that $\text{cost}(\kappa)$ does not increase whenever we run into the Good Case. However, since $C(t) < t'$, we have that the initial **move**-operation in the Good Case does not increase $\text{cost}(\kappa)$. Moreover, since $p_{j't}(\kappa) < p_{jt}(\kappa)$, we have that the modification of κ at time slots $\overline{W}$ even decreases $\text{cost}(\kappa)$, which completes the proof. ☐

Proof. [Lemma 3.10] Let Δ^-_t be the change of Δ^- in iteration t. Hence,

$$\Delta^- = \sum_{t\in A\cup B} \Delta^-_t. \tag{3.8}$$

Now consider a fixed iteration $t \in A \cup B$ and the state of κ' just before this iteration. We then have that

$$\begin{aligned} \Delta^-_t &\le |R'_t|\left(\frac{n_{t+1}(\kappa')}{m} + 1\right) \\ &\le \Delta_t + |R'_t|. \end{aligned} \tag{3.9}$$

The first line is due to Lemma 3.1 and the fact that we apply the **trickle**-operation exactly $|R'_t|$ times in iteration t, one time for every time we run into the Bad Case. Moreover, the second line is due to Lemma 3.9 and the definition of Δ_t. Therefore, we even have that

$$\begin{aligned} \Delta^- - \Delta^\sim &\le \sum_{t\in A\cup B} \left[\Delta^-_t - |R'_t|\right] \\ &\le \Delta, \end{aligned}$$

which implies the claim of the lemma. The first line is due to Equation (3.8) and Lemma 3.11, and the second line is due to Inequality (3.9) and the definition of Δ. ☐

3.5 Analysis of Δ via two potentials.

Recall that $A \cup B = \{1, \ldots, 2T\}$ are the iterations. For each pair $t, i \in A \cup B$, define $\kappa_t(i) := |\kappa(i)|$, where we consider here the state of κ just before iteration t. Using this, we define two positive potentials, called A- and B-potential, where the values of these potentials just before iteration $t \in A \cup B$ are

$$A_t := \sum_{i\in A: i\ge t} \kappa_t(i)\frac{n_i(\sigma)}{m} \text{ and } B_t := \sum_{i \in B: i\ge t} \kappa_t(i)\frac{n_i(\sigma)}{m},$$

respectively. Recall that σ is not modified during algorithm **MRG'**. Analogously, define $A_{2T+1} = B_{2T+1} := 0$ to be the respective final value of these potentials after the last iteration $2T$. Note that, for each iteration $t \in A \cup B$,

$$A_t - A_{t+1} \ge a_t \text{ and } B_t - B_{t+1} \ge b_t - c_t, \tag{3.10}$$

where

$$a_t := \chi[t\in A]|R_t|\frac{n_t(\sigma)}{m} + \sum_{t' \in R_t: t'\in A} \frac{n_{t'}(\sigma)}{m},$$

$$b_t := \chi[t\in B]|R_t|\frac{n_t(\sigma)}{m} + \sum_{t' \in R_t: t'\in B} \frac{n_{t'}(\sigma)}{m},$$

and

$$c_t := |R_t \backslash R'_t|\frac{n_{C(t)}(\sigma)}{m}.$$

The used function χ is the indicator function, i.e., $\chi[E] := 1$ if some event E is true, and $\chi[E] := 0$ if E is false. Hence, a_t and $b_t - c_t$ are lower bounds for the change of the A- and B-potentials in iteration t, respectively. Note that $a_t, b_t, c_t \ge 0$, for each $t \in A\cup B$, and that $c_t = 0$, for each $t \in B$, since then $R'_t = R_t$. More specifically, the first parts of a_t and b_t deal with the potential changes due to the fact that we sum in both potentials with $i \ge t$, and hence, the current t will not be an index of this sum in the next iteration. The second part deals with the change due to transforming κ. Specifically, for each $t' \in R_t$, one less job is scheduled by κ at time slot t'. But on the other hand, if $t' \in R_t \backslash R'_t$, then one more job is scheduled by κ at time slot $C(t) \in B$, which motivates the definition of c_t. Note that it is possible that Inequalities (3.10) are not tight in some iteration t. This happens exactly if $|K| < m$.

Assume that all decisions in algorithm **MRG'** are implemented in a deterministic way. Specifically, assume that whenever ties are broken arbitrarily when selecting a job, we select the smallest labeled job. Therefore, the states of schedule κ during algorithm **MRG'** only depend on the i.i.d random variables $X_1, X_2, \ldots, X_T$. Observe that this holds as well for the random variables a_t, b_t, Δ_t, and c_t, $t \in A \cup B$. We need two preliminary lemmas to upper bound $\mathbb{E}[\Delta]$ with respect to OPT. The first preliminary lemma characterizes the initial values of the two potentials, and the second preliminary lemma describes their expected change 'over time'.

LEMMA 3.12. *We have that* $A_1 \leq \mathrm{OPT}$ *and* $B_1 = 0$.

Proof. The first part follows from the alternative definition of the objective function (2.1) and the fact that $\kappa_1(i) \leq m$, for each time slot $i \in A$. The second part follows from the fact that $\kappa_1(i) = 0$, for each time slot $i \in B$. $\square$

LEMMA 3.13. *For each* $1 \leq s \leq T$,

$$\mathbb{E}\left[\Delta_{A(s)} + c_{A(s)}\right] \leq \mathbb{B}[X]\,\mathbb{E}\left[a_{A(s)} + b_{A(s)}\right]$$

and

$$\mathbb{E}\left[\Delta_{B(s)}\right] \leq \mathbb{E}\left[b_{B(s)}\right].$$

Proof. Note that $R'_{B(s)} = R_{B(s)}$, for each $1 \leq s \leq T$. Consequently, the second part of the claim follows from the simple fact that $n_{B(s)+1}(\sigma) \leq n_{B(s)}(\sigma)$. Therefore, it only remains to prove the first part.

Consider a fixed $1 \leq s \leq T$, and let $t := A(s)$. Observe that the random variables $\pi(1), \pi(2), \ldots, \pi(s-1)$ define the outcome of the random multiset

$$Q_s := \{A^{\leftarrow}(t') \mid t' \in R_t \cap A\} \cup \{\pi(B^{\leftarrow}(t')) \mid t' \in R_t \cap B\},$$

since these are the variables that define the relative positions of the previously used buffer slots $B(1), B(2), \ldots, B(s-1)$. However, Q_s is independent of the random variables $\pi(s), \pi(s+1), \ldots, \pi(T)$, since the buffer slots $B(s), B(s+1), \ldots, B(T) > t$ have not been before iteration t, but they might only 'stretch' σ at the time slots $t+1, t+2, \ldots, 2T$. Specifically, we could also remove these slots again without affecting Q_s.

Consider now a fixed outcome of Q_s, say Q, and a fixed $s' \in Q$. Let t' be the element in R_t corresponding to s' due to the definition of Q_s, and define the random variable $Y_{s'}$ as

$$Y_{s'} := \chi[t' \in R'_t]\frac{n_{t+1}(\sigma)}{m} + \chi[t' \in R_t \backslash R'_t]\frac{n_{C(t)}(\sigma)}{m},$$

where χ is the indicator function. Recall that $t' \in R'_t$ if and only if we run into the Bad Case for t'. Since $t \in A$, this is equivalent to $C(t) > t'$. Recall now the three properties used to construct A and B in Subsection 3.3, and distinguish two cases:

> **Case** $t' \in A$**:** We then have that $t' = A(s')$. Consequently, since $t < t'$, we obtain that $A(s) < A(s')$, and hence, by property (1), it holds that $s < s'$. Thus, by property (3), we obtain that $C(t) = B(s) > A(s') = t'$ if and only if $\pi(s) > s'$.
>
> **Case** $t' \in B$**:** Note that if $B^{\leftarrow}(t') \geq s$, then the buffer slot t' has not been used before iteration t, and hence $t' \in R_t$ is impossible. Therefore, we have that $B^{\leftarrow}(t') < s$. Consequently, by property (2), we obtain that $t' = B(B^{\leftarrow}(t')) < B(s) = C(t)$ if and only if $s' = \pi(B^{-1}(t')) < \pi(s)$.

Combining both cases shows that $t' \in R'_t$ is equivalent to $\pi(s) > s'$. Moreover, recall that $n_s = n_t(\sigma)$ and $n_{\pi(s)} = n_{C(t)}(\sigma)$. Using the first fact, it is also easy to see that also $n_{s+1} = n_{t+1}(\sigma)$. Therefore, by combining all these facts, we can rewrite $Y_{s'}$ as

$$Y_{s'} = \chi[\pi(s) > s']\frac{n_{s+1}}{m} + \chi[\pi(s) \leq s']\frac{n_{\pi(s)}}{m}.$$

Using this and the fact that the random variable $\pi(s)$ and hence X_s is independent of s', we find that

$$\begin{aligned}
&(3.11)\\
&\mathbb{E}[Y_{s'}] = \Pr[\pi(s) > s']\,\frac{n_{s+1}}{m}\\
&\quad + \Pr[\pi(s) \leq s']\,\mathbb{E}\left[\frac{n_{\pi(s)}}{m} \;\middle|\; \pi(s) \leq s'\right]\\
&< \frac{n_{s+1}}{m}\Bigg(\Pr\left[0 < X_s \leq \frac{n_{s'}}{n_{s+1}}\right]\\
&\quad + \Pr\left[\frac{n_{s'}}{n_{s+1}} < X_s \leq 1\right]\mathbb{E}\left[X_s \;\middle|\; \frac{n_{s'}}{n_{s+1}} < X_s \leq 1\right]\Bigg)\\
&\leq \frac{\mathbb{B}[X]\,n_{s+1}}{m}\left(1 + \frac{n_{s'}}{n_{s+1}}\right)\\
&\leq \mathbb{B}[X]\left(\frac{n_s}{m} + \frac{n_{s'}}{m}\right).
\end{aligned}$$

The second line is due to Lemma 3.8 and the definition of the function π. Moreover, the third line due to the definition of $\mathbb{B}[X]$, and the fourth line is due to the simple fact that $n_{s+1} \leq n_s$. Using this, we obtain that

$$\begin{aligned}
\mathbb{E}[\Delta_t + c_t \mid Q_s = Q] &= \\
&\mathbb{E}\left[\sum_{s' \in Q_s} Y_{s'} \;\middle|\; Q_s = Q\right]\\
&= \sum_{s' \in Q}\mathbb{E}[Y_{s'}]\\
&< \mathbb{B}[X]\left(|Q|\frac{n_s}{m} + \sum_{s' \in Q}\frac{n_{s'}}{m}\right)\\
&= \mathbb{B}[X]\,\mathbb{E}[a_t + b_t \mid Q_s = Q].
\end{aligned} \tag{3.12}$$

The first line is due to the definition of $Y_{s'}$, and the second line is due to linearity of expectation and the fact that $\pi(s)$ is independent of Q_s. Moreover, the third line is due to Inequality (3.11). Finally, the fourth line is due to the definitions of a_t and b_t in combination with the fact that the outcome of the sum of these two random variables is completely defined by Q_s. Using Inequality (3.12), we conclude that

$$\mathbb{E}[\Delta_t + c_t] = \mathbb{E}[\mathbb{E}[\Delta_t + c_t \mid Q_s]] < \mathbb{B}[X]\,\mathbb{E}[a_t + b_t],$$

which completes the proof of the lemma. □

LEMMA 3.14. $\mathbb{E}[\Delta] \leq \mathbb{B}[X]\,\mathrm{OPT}$

Proof. We have that

$$\begin{aligned}
\mathbb{E}[\Delta] &= \sum_{s=1}^{T} \mathbb{E}\left[\Delta_{A(s)}\right] + \sum_{s=1}^{T} \mathbb{E}\left[\Delta_{B(s)}\right] \\
&\leq \sum_{s=1}^{T} \mathbb{B}[X]\,\mathbb{E}\left[a_{A(s)} + b_{A(s)}\right] \\
&\quad - \sum_{s=1}^{T} \mathbb{E}\left[c_{A(s)}\right] + \sum_{s=1}^{T} \mathbb{E}\left[b_{B(s)}\right] \\
&\leq \mathbb{B}[X]\,\mathbb{E}\left[\sum_{t=1}^{2T} a_t\right] + \mathbb{E}\left[\sum_{t=1}^{2T} [b_t - c_t]\right] \\
&\leq \mathbb{B}[X]\,\mathbb{E}[A_1 - A_{2T+1}] + \mathbb{E}[B_1 - B_{2T+1}] \\
&\leq \mathbb{B}[X]\,\mathrm{OPT},
\end{aligned}$$

which proves the claim. The first line is due to linearity of expectation, and the second line is due to Lemma 3.13 and linearity of expectation. Moreover, the third line is due to a simple reordering using linearity of expectation, the facts that $\mathbb{B}[X] \leq 1$, and moreover $c_{B(s)} = 0$ and $a_{B(s)} \geq 0$, for each $1 \leq s \leq T$. The forth line is due to Inequalities (3.10), which characterize the change of the potentials over time. Finally, the fifth line is due to Lemma 3.12. □

Proof. [Theorem 3.1] We have that

$$\begin{aligned}
\mathrm{SRPT} - \mathrm{OPT} &= \mathbb{E}[\mathrm{SRPT} - \mathrm{OPT}] \\
&= \mathbb{E}\left[\Delta^+ + \Delta^- - \Delta^\sim\right] \\
&\leq \mathbb{E}\left[\Delta^+ + \Delta\right] \\
&\leq (\mathbb{E}[X] + \mathbb{B}[X])\mathrm{OPT}.
\end{aligned}$$

In the first line, we simply take the expected value of a constant value, which we can interpret as a random variable with constant outcome. We know from Lemma 3.9 that ultimately $\kappa = \kappa'$, and hence, Equation (3.3) indeed holds. The second line follows from this fact. The third line is due to Lemma 3.10. Moreover, the fourth line is due to linearity of expectation and Lemmas 3.7 and 3.14. Using this, we obtain that $\mathrm{SRPT}/\mathrm{OPT} \leq 1 + \mathbb{E}[X] + \mathbb{B}[X]$, which proves the claim. □

4 Conclusions

In this paper, we gave the first proof that the competitive ratio of SRPT for completion time scheduling is bounded by a constant smaller than 2 for any number of processors. Specifically, we gave the bound 1.86. The main idea was to randomly insert buffer slots. This idea can be extended such that in addition to inserting a buffer slot for each original time slot, we also insert a buffer slot for each buffer slot in the same way, and so on. If we do so, then the standard transformation of the geometric series gives $\mathbb{E}[\Delta^+] \leq \frac{\mathbb{E}[X]}{1-\mathbb{E}[X]}\mathrm{OPT}$ instead of $\mathbb{E}[\Delta^+] \leq \mathbb{E}[X]\,\mathrm{OPT}$. However, we conjecture that this could help to significantly improve the bound on the competitive ratio of SRPT, but might be challenging technically. Another way to improve this result is to find a better random variable X, or even one that minimizes $\mathbb{B}[X]$. The random variable we used, whose density function is illustrated in Figure 6, was found by heuristic search. We conjecture that an improvement obtained in this way will be marginal.

5 Acknowledgment

The authors would like to thank anonymous referees for their helpful comments, which significantly improved the exposition of this paper.

References

[1] AFRATI, F. N., BAMPIS, E., CHEKURI, C., KARGER, D. R., KENYON, C., KHANNA, S., MILIS, I., QUEYRANNE, M., SKUTELLA, M., STEIN, C., AND SVIRIDENKO, M. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS'99)* (1999), pp. 32–44.

[2] ANDERSON, E. J., AND POTTS, C. N. On-line scheduling of a single machine to minimize total weighted completion time. *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA '02)* (2002), 548 – 557.

[3] CORREA, J., AND WAGNER, M. R. Lp-based online scheduling: From single to parallel machines. *Proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization (IPCO '05)* (2005), 196 – 206.

[4] DU, J., LEUNG, J. Y.-I., AND YOUNG, G. H. Minimizing mean flow time with release time constraint. *Theoretical Computer Science 75*, 3 (1990), 347–355.

[5] GRAHAM, R. L., LAWLER, E. L., LENSTRA, J. K., AND RINNOOY KAN, A. H. G. Optimization and approximation in deterministic sequencing and scheduling theory: a survey. *Annals of Discrete Mathematics 5* (1979), 287 – 326.

[6] Hall, L. A., Schulz, A. S., Shmoys, D. B., and Wein, J. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research 22* (1997), 513 – 544.

[7] Hoogeveen, J. A., and Vestjens, A. P. A. Optimal on-line algorithms for single-machine scheduling. *Proceedings of the 5th Conference on Integer Programming and Combinatorial Optimization (IPCO '96)* (1996), 404 – 414.

[8] Labetoulle, J., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. Preemptive scheduling of uniform machines subject to release dates. In *Progress in Combinatorial Optimization* (1986), Academic Press, Inc., pp. 245 – 261.

[9] Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1 (1977), 343 – 362.

[10] Liu, P., and Lu, X. On-line scheduling of parallel machines to minimize total completion times. *Computers and Operations Research 36*, 9 (2009), 2647 – 2652.

[11] Lu, X., Sitters, R., and Stougie, L. A class of on-line scheduling algorithms to minimize total completion time. *Operations Research Letters 31*, 3 (2003), 232–236.

[12] Megow, N., and Schulz, A. S. On-line scheduling to minimize average completion time revisited. *Operations Research Letters*, 32 (2004), 485 – 490.

[13] Phillips, C., Stein, C., and Wein, J. Scheduling Jobs That Arrive Over Time. In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS '95)* (1995), vol. 955 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 86 – 97.

[14] Phillips, C. A., Stein, C., and Wein, J. Minimizing average completion time in the presence of release dates. *Mathematical Programming 82* (1998), 199 – 223.

[15] Pruhs, K., Torng, E., and Sgall, J. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis, ed. Joseph Y.-T. Leung.* CRC Press, 2004, ch. Online scheduling, pp. 15–1 – 15–41.

[16] Queyranne, M. Structure of a simple scheduling polyhedron. *Mathematical Programming 58* (1993), 263 – 285.

[17] Schrage, L. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16 (1968), 199 – 223.

[18] Schulz, A. S., and Skutella, M. The power of α-points in preemptive machine scheduling. *Journal of Scheduling*, 5 (2002), 121 – 133.

[19] Sitters, R. *Complexity and approximation in routing and scheduling.* PhD thesis, Eindhoven University of Technology, 2004.

[20] Stein, C., 2008. personal communication.

[21] Vestjens, A. P. A. *On-line machine scheduling.* PhD thesis, Eindhoven University of Technology, Netherlands, 1997.