From Preemptive to Non-preemptive Scheduling Using Rejections

Giorgio Lucarelli, Abhinav Srivastav, Denis Trystram

From Preemptive to Non-preemptive Scheduling using Rejections

Giorgio Lucarelli[1⋆], Abhinav Srivastav[1,2⋆⋆], and Denis Trystram[1⋆]

[1] LIG, Université Grenoble-Alpes, France
[2] Verimag, Université Grenoble-Alpes, France
{giorgio.lucarelli,abhinav.srivastav,denis.trystram}@imag.fr

Abstract. We study the classical problem of scheduling a set of independent jobs with release dates on a single machine. There exists a huge literature on the preemptive version of the problem, where the jobs can be interrupted at any moment. However, we focus here on the non-preemptive case, which is harder, but more relevant in practice. For instance, the jobs submitted to actual high performance platforms cannot be interrupted or migrated once they start their execution (due to prohibitive management overhead). We target on the minimization of the total stretch objective, defined as the ratio of the total time a job stays in the system (waiting time plus execution time), normalized by its processing time. Stretch captures the quality of service of a job and the minimum total stretch reflects the fairness between the jobs. So far, there have been only few studies about this problem, especially for the non-preemptive case. Our approach is based to the usage of the classical and efficient for the preemptive case shortest remaining processing time (SRPT) policy as a lower bound. We investigate the (offline) transformation of the SRPT schedule to a non-preemptive schedule subject to a recently introduced resource augmentation model, namely the rejection model according to which we are allowed to reject a small fraction of jobs. Specifically, we propose a $\frac{2}{\epsilon}$-approximation algorithm for the total stretch minimization problem if we allow to reject an ϵ-fraction of the jobs, for any $\epsilon > 0$. This result shows that the rejection model is more powerful than the other resource augmentations models studied in the literature, like speed augmentation or machine augmentation, for which non-polynomial or non-scalable results are known. As a byproduct, we present a $\frac{1}{\epsilon}$-approximation algorithm for the total flow-time minimization problem which also rejects at most an ϵ-fraction of jobs.

1 Introduction

In this work we are interested in the analysis of an efficient algorithm for scheduling jobs non-preemptively under the objective of minimizing the total (or average) stretch of the jobs. Stretch is the most relevant metric used in the context

⋆ This work has been partially supported by the projet Moebus (ANR-13-INFR-0001) funded by ANR.

⋆⋆ This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program "Investissement d'avenir".

of resource management in large scale parallel computing platforms. Informally, the stretch of a job is the total time it spends in the system normalized by its processing time. Thus, the average stretch over all the jobs represents a quality of service measure in terms of fairness among the jobs. The jobs whose execution requires more time are more appropriate to wait longer than short ones. Non-preemptive scheduling policies are usually considered in computing platforms since practically, interrupting jobs during their execution is not allowed. This is due to significant communication overhead and extra memory costs that are induced by such interruptions. However, from the combinatorial side, scheduling non-preemptively is harder and as a consequence, a much less studied problem.

More formally, we consider the offline problem of scheduling a set $\mathcal{J}$ of n independent jobs on a single machine. Each job $j \in \mathcal{J}$ is characterized by a *processing time* p_j and a *release date* r_j. Given a schedule $\mathcal{S}$, we denote by $\sigma_j^{\mathcal{S}}$ and $C_j^{\mathcal{S}}$ the *starting time* and *completion time*, respectively, of the job j. Then, its *flow time* is defined as $F_j^{\mathcal{S}} = C_j^{\mathcal{S}} - r_j$, that is the total time that j remains to the system. The *stretch* of j in a schedule $\mathcal{S}$ is defined as $s_j^{\mathcal{S}} = \frac{F_j^{\mathcal{S}}}{p_j}$, that is the flow time of j is normalized with respect to its processing time. When there is no ambiguity, we will simplify the above notation by dropping $\mathcal{S}$. Our objective is to create a *non-preemptive* schedule that minimizes the total stretch of all jobs in $\mathcal{J}$, i.e., $\sum_{j \in \mathcal{J}} s_j$. This problem is known as the *total (or average) stretch minimization* problem.

The total stretch minimization problem is a special case of the *total weighted flow-time minimization* problem where each job $j \in \mathcal{J}$ is additionally characterized by a weight w_j and the objective is to minimize $\sum_{j \in \mathcal{J}} w_j F_j$. The above problem reduces to the total stretch minimization problem if we consider that $w_j = \frac{1}{p_j}$ for each $j \in \mathcal{J}$. Another closely related problem which is also a special case of the total weighted flow-time minimization problem is the *total flow-time minimization* problem in which the weights of all jobs are equal. Although total flow-time and total stretch objectives do not have an immediate relation, the latter is generally considered to be a more difficult problem since w_j depends on the job's processing time, while in the former all the jobs have the same weight.

Based on the inapproximability results for different variants of the above problems (see for example [2, 16] and the related work below), Kalyanasundaram and Pruhs [14] and Phillips *et al.* [18] proposed to study the effect of resource augmentation, in which the algorithm is applied to a more powerful environment than the optimal one. For instance, in the *machine augmentation model* the algorithm can use more machines than the optimal solution, while in the *speed augmentation model* the algorithm can execute the jobs on faster machines comparing to the machines of the optimal schedule. More specifically, given some optimization objective (e.g., total weighted flow-time), an algorithm is said to be ℓ-machine ρ-approximation if it uses ℓm machines and it is a ρ-approximation with respect to an optimal scheduling algorithm using m machines, for some $\ell > 1$; similarly, we can define a v-speed ρ-approximation algorithm. Recently, Choudhury *et al.* [10] proposed the *rejection model*, in which the algorithm can reject a bounded fraction of the jobs (or a set of jobs whose total weight is

a bounded fraction of the total weight of all jobs), while the optimal solution should execute all the jobs of the instance. In this paper, we study the total stretch minimization problem with respect to the rejection model.

Related work. When preemptions are allowed, the well-known online Shortest Remaining Processing Time (SRPT) strategy returns the optimal solution for the total flow-time minimization problem [1] and a 2-competitive solution for the total stretch minimization problem [17]. A polynomial time approximation scheme has been also presented in [7] for the total stretch objective. On the other hand, for the total weighted flow-time minimization problem, the best known guarantee is given by a randomized algorithm which achieves an approximation ratio of $O(\log\log \Delta)$ [5], where Δ is the ratio of the largest processing time over the smallest processing time in the input instance. Furthermore, algorithms of competitive ratios $O(\log W)$ [4] and $O(\log^2 \Delta)$ [9] are known, while any algorithm should have a competitive ratio $\Omega(\min\{\sqrt{\frac{\log W}{\log\log W}}, \sqrt{\frac{\log\log \Delta}{\log\log\log \Delta}}\})$ [2], where W is the ratio of the largest weight over the smallest weight in the input instance.

In the non-preemptive context, even for the objectives of total flow-time and total stretch, the problem becomes much harder to approximate. Specifically, there is no approximation algorithm for the total flow-time minimization problem with ratio $O(n^{\frac{1}{2}-\epsilon})$, for any $\epsilon > 0$, unless $P = NP$ [15]. On the other hand, an algorithm that matches this ratio has been presented in the same paper. In the online setting, in [9] it is mentioned that any algorithm should have a competitive ratio $\Omega(n)$ even for the total flow-time objective. In [8], the greedy online Shortest Processing Time (SPT) strategy is proven to be $\frac{\Delta+1}{2}$-competitive for the total flow-time minimization problem and this ratio is the best possible for this problem. Similarly, the weighted generalization of SPT is $(\Delta+1)$-competitive for the total weighted flow-time objective and this ratio is optimal [20].

In the resource augmentation framework, an $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm is known for the total weighted flow-time minimization problem when preemptions are allowed [6]. In [11], an $O(\frac{1}{\epsilon^{12}})$-competitive algorithm has been presented for the total weighted flow-time objective which rejects an ϵ-fraction of jobs; this result holds also for parallel machines.

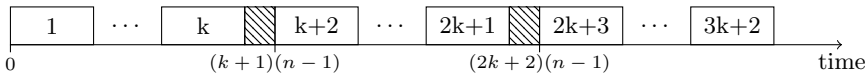
If preemptions are not allowed, a 12-speed $(2+\epsilon)$-approximation algorithm for the total flow-time objective and a 12-speed 4-approximation algorithm for the total weighted flow-time objective have been presented in [3]. In [13], a dynamic programming framework has been presented that runs in quasi-polynomial time. This framework works also for the parallel machine setting and leads to a $(1+\epsilon)$-speed and $(1+\epsilon)$-approximate solution for the total weighted flow-time minimization problem and to a $(1+\epsilon)$-speed and 1-approximate solution for the total flow-time minimization problem. In [18], an $O(\log \Delta)$-machine 1-competitive algorithm has been proposed for the total weighted flow-time objective even for parallel processors. For the unweighted version, an $O(\log n)$-machine $(1+o(1))$-competitive algorithm and an $O(\log n)$-machine $(1+o(1))$-speed 1-competitive algorithm have been proposed in the same paper. Note that the algorithms in [18] work in the online setting but they need to know the minimum and the maximum

processing times in advance. Moreover, an ℓ-machine $(1 + \Delta^{1/\ell})$-competitive algorithm designed in [12] for the total flow-time minimization problem, if Δ is known a priori to the algorithm. They also provided a lower bound which shows that their algorithm is optimal up to a constant factor for any constant ℓ.

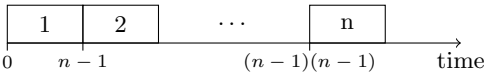
No results for the total stretch minimization problem without preemptions in the resource augmentation context are known, except from the results that derive from the more general problem of minimizing the weighted flow-time.

Contribution and organization of the paper. In this paper, we explore the relation between preemptive and non-preemptive schedules with respect to the total stretch objective subject to the rejection model. More specifically, we consider the SRPT policy for creating a preemptive schedule. In Section 2 we describe several structural properties of this schedule. Next, we show how to transform the preemptive schedule created by the SRPT policy to a non-preemptive schedule with given worst-case guarantees.

In Section 3, we use the rejection model and we give an $\frac{2}{\epsilon}$-approximation algorithm if we are permitted to delete a subset of jobs such that their total weight is an ϵ-fraction of the total weight of all jobs. Note that, the relation among the rejection model and other resource augmentation models is not clear. For example, in Fig. 1 we give an instance for which the best possible solution using rejections is worse than the best possible solution using speed-augmentation, when the same constant ϵ is selected for both models. However, our result shows



Scheduling using rejections (reject the jobs $k + 1$ and $2k + 2$)



Scheduling using speed augmentation (use processing times equal to $n - 1$)

Fig. 1. An instance of $n = 3k + 2$ jobs with equal processing times $p_j = n$, equal weights $w_j = 1$, and release dates $r_j = (j - 1)(n - 1)$, where $1 \leq j \leq n$. By setting $\epsilon = \frac{1}{n-1}$, in the rejection model we are allowed to reject at most $\epsilon n \leq 2$ jobs, while in the speed augmentation model the processing time of each job becomes $\frac{p_j}{1+\epsilon} = n - 1$. The total flow time using rejections is $3\sum_{j=1}^{k}(n + j - 1) = \frac{21}{2}k^2 + \frac{9}{2}k$, while the total flow time using speed augmentation is $n(n-1) = 9k^2 + 9k + 2$ which is better for large enough k.

the strength of the rejection model, particularly in the non-preemptive context, since the known results subject to other resource augmentation models either need quasi-polynomial time [13] or they cannot arrive arbitrarily close to the

classical model without resource augmentation [3, 18] even for the total flow-time objective. Contrarily, our result is the best possible we can expect in the rejection model.

Finally, using the same rejection strategy and analysis, we obtain an $\frac{1}{\epsilon}$-approximation algorithm if we are allowed to delete an ϵ-fraction of jobs. We conclude in Section 4. Before continuing, we give some additional notation which we use throughout the paper.

Notations. In what follows, for each job $j \in \mathcal{J}$ and schedule $\mathcal{S}$, we define the interval $[\sigma_j^{\mathcal{S}}, C_j^{\mathcal{S}}]$ to be the *active interval* of j in $\mathcal{S}$. In the case where preemptions are allowed, the active interval of j may have a length bigger than p_j. A job j is *available* at a time t if it is released but it is not yet completed, i.e., $r_j \leq t < C_j$. We call a schedule *compact* if it does not leave any idle time whenever there is a job available for execution.

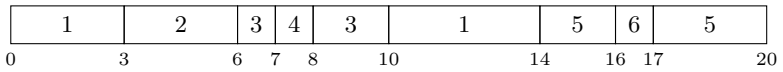
2 Structure and Properties of SRPT and an Intermediate Schedule

In this section we deal with the structure of a preemptive schedule created by the Shortest Remaining Processing Time (SRPT) policy and we give some useful properties that we will use in the following sections. According to the SRPT policy, at any time, we select to execute the available job with the shortest remaining processing time. Since the remaining processing time of the executed job $j \in \mathcal{J}$ decreases over time, its execution may be interrupted only in the case where a new job $k \in \mathcal{J}$ is released and the processing time of k is smaller than the remaining processing time of j at r_k. Hence, the SRPT policy can be seen as an event-driven algorithm in which at each time t where a job is released or completed we should take a decision about the job that we will execute at t and we always select the one with the shortest remaining processing time. In case of ties, we assume that SRPT resumes the partially executed job, if any, with the latest starting time; if all candidate jobs are not processed before, then we choose among them the job with the earliest release time.

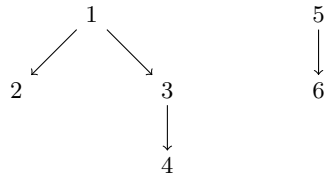
Kellerer et al. [15] observed that in the schedule produced by the SRPT policy, for any two jobs j and k, their active intervals are either completely disjoint or the one contains the other. Moreover, there is no idle time during the active interval of any job. Based on the above, the execution of the jobs in the SRPT schedule has a tree-like structure. More specifically, we can create a graph which consists of a collection $\mathcal{T}$ of out-trees and corresponds to the SRPT schedule as follows (see Fig. 2): for each job $j \in \mathcal{J}$, we create a vertex u_j. For each pair of jobs $j, k \in \mathcal{J}$, we add an arc (u_j, u_k) if and only if $[\sigma_k, C_k] \subset [\sigma_j, C_j]$ and there is no other job $i \in \mathcal{J}$ so that $[\sigma_k, C_k] \subset [\sigma_i, C_i] \subset [\sigma_j, C_j]$.

In what follows, we denote by $root(T)$ the root of each out-tree $T \in \mathcal{T}$. Intuitively, each vertex $root(T)$ corresponds to a job for which at any time t during its execution there is no other job which has been partially executed at t. We denote also by $a(j)$ the parent of the vertex that corresponds to the job

jobs	1	2	3	4	5	6
r_j	0	3	6	7	14	16
p_j	7	3	3	1	5	1



Shortest Remaining Processing Time (SRPT) schedule



Collection of out-trees

Fig. 2. A schedule created by the SRPT policy and its corresponding collection of out-trees.

$j \in \mathcal{J}$ in T. Moreover, let $T(u_j)$ be the subtree of $T \in \mathcal{T}$ rooted at a vertex u_j in T. Note that, we may refer to a job j by its corresponding vertex u_j and vice versa.

In this paper, we use the schedule created by the SRPT policy for the preemptive variant of our problem as a lower bound to the non-preemptive variant. The SRPT policy is known to be optimal [16, 19] for the problem of minimizing the sum $\sum_{j \in \mathcal{J}} F_j$ when preemptions of jobs are allowed. However, for the preemptive variant of the total stretch minimization problem, SRPT is a 2-approximation algorithm [17].

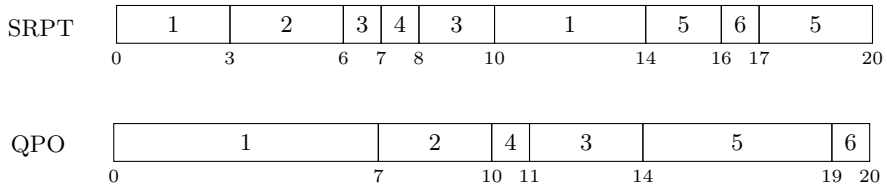


Fig. 3. Transformation from SRPT to QPO schedule

Consider now the collection of out-trees $\mathcal{T}$ obtained by an SRPT schedule and let $T(u_j)$ be the subtree rooted at any vertex u_j. We construct a non-preemptive schedule for the jobs in $T(u_j)$ as follows: during the interval $[\sigma_j, C_j]$,

we run the jobs in $T(u_j)$ starting with j and then running the remaining jobs in order of increasing SRPT completion time as shown in Figure 3. This policy has been proposed in [8] for the problem of minimizing the sum $\sum_{j\in\mathcal{J}} F_j$ and corresponds to a post order transversal of the subtree $T(u_j)$ excluding its root which is scheduled in the first position. We call the above policy as *Quasi Post Order* (QPO) and we will use it for the problem of minimizing the sum $\sum_{j\in\mathcal{J}} s_j$. The following lemma presents several observations for the QPO policy.

Lemma 1. [8] *Consider any subtree $T(u_k)$ which corresponds to a part of the schedule $\mathcal{SRPT}$ and let $\mathcal{QPO}$ be the non-preemptive schedule for the jobs on $T(u_k)$ created by applying the Quasi Post Order policy. Then,*

i) all jobs in $\mathcal{QPO}$ are executed during the interval $[\sigma_k^{\mathcal{SRPT}}, C_k^{\mathcal{SRPT}}]$ without any idle period,
ii) $\sigma_j^{\mathcal{QPO}} \geq r_j$ for each u_k in $T(u_k)$,
iii) $C_j^{\mathcal{QPO}} \leq C_j^{\mathcal{SRPT}} + p_k$ for each u_j in $T(u_k)$ with $j \neq k$, and
iv) $C_k^{\mathcal{QPO}} = C_k^{\mathcal{SRPT}} - \sum_{u_j\in T(u_k):j\neq k} p_j$.

Note that, the schedule created by the SRPT policy is a compact schedule, since it always execute a job if there is an available one. Therefore, by Lemma 1.i, the following directly holds.

Corollary 1. *The schedule created by the QPO policy is compact.*

3 The Rejection Model

In this section we consider the rejection model. More specifically, given an $\epsilon \in (0, 1)$, we are allowed to reject any subset of jobs $\mathcal{R} \subset \mathcal{J}$ whose total weight does not exceed an ϵ-fraction of the total weight of all jobs, i.e., $\sum_{j\in\mathcal{R}} w_j \leq \epsilon \sum_{j\in\mathcal{J}} w_j$. We will present our rejection policy for the more general problem of minimizing $\sum_{j\in\mathcal{J}} w_j F_j$.

Our algorithm is based on the tree-like structure of the SRPT schedule. Let us focus first on a single out-tree $T \in \mathcal{T}$. The main idea is to reject the jobs that appear in the higher levels of T (starting with its root) and run the remaining jobs using the QPO policy. The rejected jobs are, in general, long jobs which are preempted several times in the SRPT schedule and their flow time can be used as an upper bound for the flow time of the smaller jobs that are released and completed during the life interval of the longest jobs. In order to formalize this, for each job $j \in \mathcal{J}$ we introduce a charging variable x_j. In this variable we accumulate the weight of jobs whose flow time will be upper bounded by the flow time of job j in the SRPT schedule. At the end of the algorithm, this variable will be exactly equal to $\frac{1}{\epsilon} w_j$ for each rejected job $j \in \mathcal{R}$, while $x_j < \frac{1}{\epsilon} w_j$ for each non-rejected job $j \in \mathcal{J} \setminus \mathcal{R}$. In fact, for most of the non-rejected jobs this variable will be equal to zero at the end of the algorithm. Our algorithm considers the jobs in a bottom-up way and charges the weight of the current job to its ancestors in T which are closer to the root and their charging variable is

not yet full; that is the vertices to be charged are selected in a top-down way. Note that, we may charge parts of the weight of a job to more than one of its ancestors.

Algorithm 1 describes formally the above procedure. For notational convenience, we consider a fictive vertex u_0 which corresponds to a fictive job with $w_0 = 0$. We connect u_0 with the vertex $root(T)$ of each out-tree $T \in \mathcal{T}$ in such a way that u_0 becomes the parent of all of them. Let T^* be the created tree with root u_0.

Algorithm 1

1: Create a preemptive schedule $\mathcal{SRPT}$ and the corresponding out-tree T^*
2: Initialization: $\mathcal{R} \leftarrow \emptyset$, $x_j \leftarrow w_j$ for each $j \in \mathcal{J}$, $x_0 \leftarrow 0$
3: **for** each vertex u_j of T^* with $x_j = w_j$ in post-order traversal **do**
4: **while** $x_j \neq 0$ **and** $x_{a(j)} < \frac{1}{\epsilon} w_{a(j)}$ **do**
5: Let u_k be a vertex in the path between u_0 and u_j such that $x_{a(k)} = \frac{1}{\epsilon} w_{a(k)}$ and $x_k < \frac{1}{\epsilon} w_k$
6: Let $y \leftarrow \min\{x_j, \frac{1}{\epsilon} w_k - x_k\}$
7: $x_j \leftarrow x_j - y$ and $x_k \leftarrow x_k + y$
8: **for** each job $j \in \mathcal{J}$ **do**
9: **if** $x_j = \frac{1}{\epsilon} w_j$ **then**
10: Reject j, i.e., $\mathcal{R} \leftarrow \mathcal{R} \cup \{j\}$
11: **return** S: the non-preemptive schedule for the jobs in $\mathcal{J} \setminus \mathcal{R}$ using QPO

Note that, the for-loop in Lines 3-7 of Algorithm 1 is not executed for all jobs. In fact, it is not applied to the jobs that will be rejected as well as to some children of them for which at the end of the algorithm it holds that $w_j < x_j < \frac{1}{\epsilon} w_j$. The weight of these jobs is charged to themselves. Moreover, the while-loop in Lines 4-7 of Algorithm 1 terminates either if the whole weight of j is charged to its ancestors or if the parent of u_j is already fully charged, i.e., $x_{a(j)} = \frac{1}{\epsilon} w_{a(j)}$.

Theorem 1. *For the schedule $\mathcal{S}$ created by Algorithm 1 it holds that*

(i) $\sum_{j \in \mathcal{J} \setminus \mathcal{R}} w_j F_j^{\mathcal{S}} \leq \frac{1}{\epsilon} \sum_{j \in \mathcal{J}} w_j F_j^{\mathcal{SRPT}}$, *and*

(ii) $\sum_{j \in \mathcal{R}} w_j \leq \epsilon \sum_{j \in \mathcal{J}} w_j$.

Proof. Consider first any vertex u_k such that $k \in \mathcal{J} \setminus \mathcal{R}$ and $a(k) \in \mathcal{R}$. By the execution of the algorithm, all the jobs corresponding to vertices in the path from u_0 to $a(k)$ do not appear in $\mathcal{S}$. Hence, k starts in $\mathcal{S}$ at the same time as in $\mathcal{SRPT}$, i.e., $\sigma_k^{\mathcal{S}} = \sigma_k^{\mathcal{SRPT}}$. Thus, by Lemma 1, the jobs that correspond to the vertices of the subtree $T^*(u_k)$ are scheduled in $\mathcal{S}$ during the interval $[\sigma_k^{\mathcal{SRPT}}, C_k^{\mathcal{SRPT}}]$. In other words, for any job j in $T^*(u_k)$ it holds that $C_j^{\mathcal{S}} \leq C_k^{\mathcal{SRPT}}$, while by the construction of T^* we have that $\sigma_k^{\mathcal{SRPT}} < r_j$. Assume now that the weight of j is charged by Algorithm 1 to the jobs $j_1, j_2, \ldots, j_{q_j}$, where q_j is the number of

these jobs. Let w_j^i be the weight of j charged to $j_i \in \{j_1, j_2, \ldots, j_{q_j}\}$; note that $w_j = \sum_{i=1}^{q_j} w_j^i$. By the definition of the algorithm, each $j_i \in \{j_1, j_2, \ldots, j_{q_j}\}$ is an ancestor of both k and j in T^* (one of them may coincides with k). Therefore, by the definition of T^*, it holds that $\sigma_{j_i}^{\mathcal{SRPT}} < r_j < C_j^{\mathcal{S}} \leq C_{j_i}^{\mathcal{SRPT}}$, for each $j_i \in \{j_1, j_2, \ldots, j_{q_j}\}$. Then, we have

$$\sum_{j \in \mathcal{J} \setminus \mathcal{R}} w_j F_j^{\mathcal{S}} \leq \sum_{j \in \mathcal{J} \setminus \mathcal{R}} \sum_{i=1}^{q_j} w_j^i F_{j_i}^{\mathcal{SRPT}} \leq \sum_{j \in \mathcal{J}} x_j F_j^{\mathcal{SRPT}} \leq \sum_{j \in \mathcal{J}} \frac{1}{\epsilon} w_j F_j^{\mathcal{SRPT}}$$

where the second inequality holds by regrouping the flow time of all appearances of the same job, and the last one by the fact that Algorithm 1 charges at each job j at most $(1 + \frac{1}{\epsilon})w_j$. Finally, since the weight of each job is charged exactly once (probably to more than one other jobs) we have $\sum_{j \in \mathcal{J}} w_j \geq \frac{1}{\epsilon} \sum_{j \in \mathcal{R}} w_j$ and the theorem holds. □

Since SRPT creates an optimal preemptive schedule for the problem of minimizing $\sum_{j \in \mathcal{J}} F_j$ on a single machine and an optimal preemptive schedule is a lower bound for a non-preemptive one the following theorem holds.

Theorem 2. *Algorithm 1 is a $\frac{1}{\epsilon}$-approximation algorithm for the single-machine total flow-time minimization problem without preemptions if we are allowed to reject an ϵ-fraction of the jobs.*

By combining Theorem 1 and the fact that SRPT is a 2-approximation algorithm for the preemptive variant of the total stretch minimization problem [17], the following theorem holds.

Theorem 3. *Algorithm 1 is a $\frac{2}{\epsilon}$-approximation algorithm for the single-machine total stretch minimization problem without preemptions if we are allowed to reject a set of jobs whose total weight is no more than an ϵ-fraction of the total weight of all jobs.*

4 Concluding Remarks

We studied the effects of applying resource augmentation in the transformation of a preemptive schedule to a non-preemptive one for the problem of minimizing total stretch on a single machine. Specifically, we show the power of the rejection model for scheduling without preemptions comparing with other resource augmentation models, by presenting an algorithm which has a performance arbitrarily close to optimal. Note that, SRPT is a 14-competitive algorithm for minimizing total stretch on parallel machines when preemptions and migrations are allowed [17]. So, an interesting question is to explore the general idea of this paper about transforming preemptive to non-preemptive schedules subject to the rejection model on parallel machines based on the above result.

References

1. K. R. Baker. *Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.
2. N. Bansal and H.-L. Chan. Weighted flow time does not admit $o(1)$-competitive algorithms. In *SODA*, pages 1238–1244, 2009.
3. N. Bansal, H.-L. Chan, R. Khandekar, K. Pruhs, C. Stein, and B. Schieber. Non-preemptive min-sum scheduling with resource augmentation. In *FOCS*, pages 614–624, 2007.
4. N. Bansal and K. Dhamdhere. Minimizing weighted flow time. *ACM Transactions on Algorithms*, 3(4), 2007.
5. N. Bansal and K. Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43:1684–1698, 2014.
6. L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4:339–352, 2006.
7. M. A. Bender, S. Muthukrishnan, and R. Rajaraman. Approximation algorithms for average stretch scheduling. *Journal of Scheduling*, 7:195–222, 2004.
8. D. P. Bunde. SPT is optimally competitive for uniprocessor flow. *Information Processing Letters*, 90:233–238, 2004.
9. C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *STOC*, pages 84–93, 2001.
10. A. R. Choudhury, S. Das, N. Garg, and A. Kumar. Rejecting jobs to minimize load and maximum flow-time. In *SODA*, pages 1114–1133, 2015.
11. A. R. Choudhury, S. Das, and A. Kumar. Minimizing weighted lp-norm of flow-time in the rejection model. In *FSTTCS*, pages 25–37, 2015.
12. L. Epstein and R. van Stee. Optimal on-line flow time with resource augmentation. *Discrete Applied Mathematics*, 154:611–621, 2006.
13. S. Im, S. Li, B. Moseley, and E. Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines [extended abstract]. In *SODA*, pages 1070–1086, 2015.
14. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47:617–643, 2000.
15. H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, 28:1155–1166, 1999.
16. S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, 73:875–891, 2007.
17. S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. E. Gehrke. Online scheduling to minimize average stretch. *SIAM Journal on Computing*, 34:433–452, 2005.
18. C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32:163–200, 2002.
19. L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687–690, 1968.
20. J. Tao and T. Liu. WSPT's competitive performance for minimizing the total weighted flow time: From single to parallel machines. *Mathematical Problems in Engineering*, 10.1155/2013/343287, 2013.