# Approximation Algorithms for Average Stretch Scheduling*

*Michael A. Bender*[1] *S. Muthukrishnan*[2] *Rajmohan Rajaraman*[3]

July 22, 2003

**Abstract**

We study the basic problem of preemptive scheduling of a stream of jobs on a single processor. Consider an online stream of jobs, and let the $i$th job arrive at time $r(i)$ and have processing time $p(i)$. If $C(i)$ is the completion time of job $i$, then the flow time of $i$ is $C(i) - r(i)$ and the stretch of $i$ is the ratio of its flow time to its processing time; that is, $\frac{C(i)-r(i)}{p(i)}$. Flow time measures the time that a job is in the system regardless of the service it requests; the stretch measure relies on the intuition that a job that requires a long service time must be prepared to wait longer than jobs that require small service times.

We present improved algorithmic results for the average stretch metric in preemptive uniprocessor scheduling. Our first result is an offline polynomial-time approximation scheme (PTAS) for average stretch scheduling. This improves upon the 2-approximation achieved by the online algorithm SRPT that always schedules a job with the shortest remaining processing time. In recent work, Chekuri and Khanna [7] have presented approximation algorithms for weighted flow time, which is a more general metric than average stretch; their result also yields a PTAS for average stretch. Our second set of results considers the impact of incomplete knowledge of job sizes on the performance of online scheduling algorithms. We show that a constant-factor competitive ratio for average stretch is achievable even if the processing times (or remaining processing times) of jobs are known only to within a constant factor of accuracy.

## 1 Introduction

We consider the basic uniprocessing scheduling scenario. We have a single processor that processes jobs as they arrive online. The $i$th job arrives at time $r(i)$ and has processing time $p(i)$ that is known at the time of its arrival. We restrict our attention to scheduling with preemption; that is, jobs may be stopped before their completion and resumed later after other jobs get executed in the interim.

Traditionally, the focus of performance has been on the *flow time* (also referred to as the *response time*), which is defined as the amount of time that a given job spends in the system. That is, if $C(i)$ is the completion time of job $i$, then the flow time is $C(i) - r(i)$. Alternatively, practitioners have used *slowdown* or *stretch* to measure the effect of scheduling on an individual job (e.g., [10]. The stretch of a job is the ratio of its flow time to its processing time; that is, $\frac{C(i)-r(i)}{p(i)}$ [5]. Stretch is

[1]Department of Computer Science, SUNY at Stony Brook, NY 11794, Email: `bender@cs.sunysb.edu`.

[2]Department of Computer Science, Rutgers University, Piscataway, NJ 08854, Email: `muthu@cs.rutgers.edu`. Part of this work was done while the author was at AT&T Shannon Laboratories, Florham Park, NJ 07932.

[3]College of Computer & Information Science, Northeastern University, Boston, MA 02115, Email: `rraj@ccs.neu.edu`. Supported by NSF CAREER award NSF CCR–9983901.

*The results of this paper appeared earlier in an extended abstract [6].

a rather natural criterion: jobs that require large processing time must be prepared to wait longer than the ones that need the system for less time.

**Overview of our results.** We present improved algorithmic results for the *average stretch* metric, or, equivalently the total stretch metric[a], in preemptive uniprocessor scheduling.

- *PTAS for average stretch*: We present a polynomial time approximation scheme (PTAS) for minimizing average stretch offline. For any constant $\varepsilon > 0$, our algorithm yields an $(1+\varepsilon)$-approximation in $O(n^{\text{poly}(1/\varepsilon)})$ time.

  Our PTAS result, which appears in Section 3, improves on the approximation factor of 2 achieved by the online shortest remaining processing time algorithm (SRPT). Furthermore, there exists a constant $c > 1$ such that no online algorithm can achieve a competitive ratio of better than $c$ [17].

Our approach for developing a PTAS for average stretch is to round the job sizes to the nearest integral power of $(1+\varepsilon)$, thus dividing the jobs into groups, and then scheduling the groups from the smallest rounded job size to the largest. Scheduling one group of jobs, however, constrains the times at which other groups of jobs may be scheduled. Designing a $(1+\varepsilon)$-approximate schedule for such "constrained" scheduling problems poses a key challenge in the design of a PTAS. The heart of our result, which is presented in Section 3, is a useful characterization of $(1+\varepsilon)$-approximate schedules that reduces the size of the search space of relevant schedules.

The technique of rounding job sizes, which is used in our PTAS, is a commonly-used tool and is effective in reducing the space of schedules of interest to yield efficiently computable schedules. Rounding also has great practical significance. Traditionally, scheduling algorithms assume complete knowledge (clairvoyance) of the processing times of the jobs. In practice, however, estimating job sizes cannot be accurate in general. So a more reasonable approach is to assume that the upper and lower bounds on the job sizes are known that are correct up to a constant factor. In Section 4, we analyze the average stretch and flow performance of two natural on-line algorithms that schedule jobs on the basis of the *rounded values* of the remaining processing times, and processing times, respectively.

- *Study of the impact of rounding*: We first analyze a generalization of SRPT, referred to as $\lambda$-SRPT, which schedules in each step a job with remaining processing time within a $(1+\lambda)$-factor of the shortest remaining processing time, for some constant $\lambda > 0$. We show that while $\lambda$-SRPT is $O(1)$-competitive with respect to average stretch, it is $\Omega(\log \Delta)$-competitive with respect to average flow time, where $\Delta$ is the ratio of the largest job size to the smallest job size. We then present a suitable refinement of $\lambda$-SRPT that is $O(1)$-competitive with respect to both average stretch and average flow time. The preceding results are applicable in scheduling scenarios in which remaining processing times of jobs are approximately known at each step. A more realistic model for partial knowledge of job sizes is a relaxation of the non-clairvoyant model, in which the processing time of any job is known to within a constant factor only at the time of the release of the job. Under this model, we show that a variant of the shortest processing time algorithm (SPT) is $O(1)$-competitive with respect to average stretch.

[a]The average stretch of a given schedule is the ratio of the sum of the stretches of all the jobs in the schedule, which is the total stretch, to the number of jobs in the instance. Thus, average stretch and total stretch are equivalent, in terms of both optimization and approximation.

**Related work.** This paper focuses on the online and offline complexity of stretch scheduling. Two measures closely related to average stretch are weighted completion time and weighted flow time, each of which associate a weight $w(i)$ with each job $i$. If we set the weight of job $i$ to be the reciprocal of processing time (i.e., $1/p(i)$), then the total weighted completion time of a given schedule becomes $\sum_i C(i)/p(i)$, which equals $\sum_i (C(i) - r(i))/p(i) + \sum_i r(i)/p(i)$ (here $C(i)$ is the completion time of job $i$ in the given schedule). With the preceding weight assignment, the total weighted completion time thus equals the total stretch plus a term $(\sum_i r(i)/p(i))$, which is independent of the schedule. Thus, optimizing weighted completion time also optimizes total stretch, which is identical to optimizing average stretch. In terms of approximation, however, the weighted completion time and average stretch metrics are significantly different. Consequently, the recent PTAS for weighted completion time [1] does not yield any useful approximation for average stretch. The average weighted flow time with weights given by the reciprocal of processing times, on the other hand, is identical to the average stretch metric. The best known approximation result for weighted flow time is the recent approximation scheme of [8], which takes time superpolynomial, but subexponential, in the input size. In a subsequent study [7] performed independently of our work, it has been shown that a quasi-PTAS is achievable for weighted flow time when $\Delta$ and the ratio of the maximum weight to minimum weight are both polynomially bounded. Since [7] study the weighted flow time metric, their results are more general than ours; when applied to the special case of the average stretch metric, the results of [7] yield a PTAS, thus matching our result for average stretch.

Our models for capturing incomplete information of job sizes may be viewed as relaxations of non-clairvoyant scheduling. In non-clairvoyant scheduling, no information about job sizes is available at release time. The competitiveness of non-clairvoyant uniprocessor scheduling, with respect to the average flow metric, is studied in [11, 15]. Our model of uncertainty in job sizes is related to a general framework developed in [3, 4], which also captures the variance in job sizes by using lower and upper limits. The underlying model of job arrivals and the performance metric studied are different, however; in [3, 4], the jobs are given at the start of the computation and need to be scheduled on an asynchronous multiprocessor system to minimize makespan.

As mentioned at the outset, our study concerns the basic uniprocessor preemptive scheduling setting. More complex scheduling scenarios have been studied, including multiprocessor scheduling (e.g., [14]), broadcast scheduling (e.g., [12]), and network connection scheduling (e.g., [9]). Clearly, some of the questions we have raised are relevant in these scenarios, and deserve further attention.

## 2 Preliminaries

In this section, we present some basic definitions and notation, that are used frequently in the remainder of the paper. Let $\mathcal{I}$ be a given scheduling instance. Recall that a scheduling instance is specified by a set of jobs $\mathcal{J}$, and for each job $j \in \mathcal{J}$, a release time $r(j)$ and a processing time $p(j)$.

We restrict our attention to discrete time, and assume that the release times and processing times are all nonnegative integers. We note that any instance with rational release and processing times can be transformed to an equivalent instance with integral release and processing times through scaling; furthermore, the size of the transformed instance is polynomial in the size of the original instance. In our analyses, we frequently need to refer to time intervals containing consecutive time steps. We use the notation $[t_1, t_2]$ to refer to the set of time steps $\{t : t_1 \leq t \leq t_2\}$.

For a given schedule, the *queue* at a given time $t$ consists of all jobs that have been released at or before time $t$ and not completed by time $t$. The remaining processing time of jobs in the queue plays an important role in our analyses. We let $\rho_t(j)$ denote the remaining processing time of job $j$

at time $t$ in the given schedule. We say that a job $j$ *delays* job $j' \neq j$ *at time* $t$ in a given schedule, if $j$ is scheduled at time $t$ and $j'$ is in the queue at time $t$. We overload the definition of delay and say that a job $j$ *delays* job $j'$ if there exists any time $t$ at which $j$ delays $j'$.

# 3 PTAS for offline average stretch

In this section, we describe a polynomial-time approximation scheme (PTAS) for the total stretch metric (equivalently, average stretch) in uniprocessors. Our presentation is organized into 5 subsections. In Section 3.1, we present an overview of our algorithm, state three key lemmas, and derive the main result based on the key lemmas. In Section 3.2, we establish basic characteristics of optimal schedules. In Sections 3.3 through 3.5, we prove the three lemmas stated in Section 3.1.

## 3.1 Overview and main theorem

We begin by introducing some notions of instances and schedules that play a central role in our algorithm and its analysis. In the process of constructing a complete schedule for a given instance, we derive partial schedules in which we schedule a subset of the jobs in the instance. The remaining jobs are thus forbidden to be scheduled at the times assigned in the partial schedule. We refer to the set of remaining jobs, their release times, and the forbidden times as a *constrained instance.*

A concept commonly used in scheduling is that of *list schedules.* A list schedule is a schedule that assigns a priority order among the jobs; in each step of the schedule, of those jobs already released and not yet completed, the job with the highest priority is scheduled. It is easy to show that every optimal schedule for any constrained instance is a list schedule (Lemma 3.4 of Section 3.2). We restrict our attention to another class of schedules, which we refer to as *natural*, that is well-suited for flow and stretch metrics. We say that a schedule is natural if it satisfies the property that a job $j$ delays a job $j'$ with smaller processing time at a given time $t$ only if the remaining processing time of $j$ at time $t$ is less than that of $j'$ at time $t$. Formally, in a natural schedule, if a job $j$ delays job $j'$ at time $t$ and $p(j) > p(j')$, then $\rho_t(j) < \rho_t(j') = p(j')$. It can be shown that every optimal schedule is a natural schedule (Lemma 3.5 of Section 3.2).

We divide the jobs into groups such that the sizes of the jobs within a group differ from one another by a factor of at most $(1+\varepsilon)$, where $\varepsilon > 0$ is an arbitrary constant. Formally, for any nonnegative integer $i$, let group $i$ consist of all jobs with size at least $(1+\varepsilon)^i$ and less than $(1+\varepsilon)^{i+1}$. To motivate our algorithm and to facilitate the analysis, we introduce the notion of *rounded stretch.* The rounded stretch of a job $j$ in a given schedule is the ratio of the flow time of $j$ in the schedule to $(1+\varepsilon)^i$, where $i$ is the group to which $j$ belongs. Since the processing time of a job in group $i$ is at least $(1+\varepsilon)^i$ and at most $(1+\varepsilon)^{i+1}$, it follows that the rounded stretch of a job in a schedule is within a factor of $1+\varepsilon$ of the actual stretch of the job in the schedule. We define the *rounded cost* of a schedule to be the sum of the rounded stretch of all the jobs in the schedule. Thus, the rounded cost of a schedule is within a $(1+\varepsilon)$ factor of the actual cost of the schedule.

We henceforth adopt rounded cost as our objective function. Thus, unless otherwise stated, whenever we refer to an optimal schedule, we refer to a schedule with minimum rounded cost. A naive approach to minimizing rounded cost for a given constrained instance is to assign equal "weight" to each job within a group and schedule the jobs within the group in FIFO order. It turns out, however, that the resultant schedule can have cost twice that of the optimal. It can be shown instead that in a schedule with optimal rounded cost, the jobs within a group need to be scheduled in SRPT order. We refer to such schedules as SRPT-*friendly* schedules. In Section 3.2, we show that every optimal schedule is SRPT-friendly (Lemma 3.6).

The notion of SRPT-friendly schedules allows us to assign an ordering among jobs *within* a group, that is among jobs that have similar processing times. At the other end, we can argue that if we have two jobs, one of which is "substantially larger" than the other, then a schedule that optimizes total stretch tends to favor the smaller job. We formalize the notion of "substantially larger" by partitioning the groups into *blocks* and *superblocks* as follows. Block $i$, for $i \geq 0$, consists of groups $ig$ through $(i+1)g-1$, where $g$ equals $\log_{1+\varepsilon}(1/\varepsilon^2)$. For simplicity, we assume throughout Section 3 that $1/\varepsilon$ is an integer. All our arguments can be easily modified to address the case where $1/\varepsilon$ is non-integral. It follows from the definition of a block that the size of any job in block $i$ is at least $1/\varepsilon^2$ times the size of any job in block $j$ for any $j < i - 1$. We further partition the blocks into *superblocks*. Superblock 0 consists of blocks 0 through $b - 1$, where $b < 1/\varepsilon^2$ is specified below. Superblock $i$, for $i > 0$, consists of blocks $b + (i-1)/\varepsilon^2$ through $b + i/\varepsilon^2 - 1$. We select $b$ such that the total number of jobs in the largest numbered blocks of all of the superblocks is at most $n\varepsilon^2$. We note that since there are $1/\varepsilon^2$ choices for $b$, one such choice exists. For any group $i$, we let $blk(i)$ (resp., $spr(i)$) denote the block (resp., superblock) to which $i$ belongs. For any superblock $s$, we let $grps(s)$ and $blks(s)$ denote the groups and blocks, respectively, in superblock $s$.

Our organization of the jobs in groups, blocks, and superblocks has the property that for any superblock $i$, jobs in every block of $i$, but for the largest numbered block, have size at most $\varepsilon^2$ times that of any job in superblock $j$ for $j > i$. Furthermore, the total number of jobs in the largest numbered blocks of all superblocks is at most $n\varepsilon^2$. In the following lemma, we make use of the preceding properties to argue that in our search for an $(1 + O(\varepsilon))$-approximate schedule, we can restrict our attention to schedules in which no job in superblock $i$ delays a job in superblock $j < i$, for any $i$; we refer to such schedules as *hierarchical schedules.*

**Lemma 3.1** *For any $\varepsilon > 0$ chosen sufficiently small, there exists a $(1 + 3\varepsilon)$-approximate natural* SRPT*-friendly list schedule that is hierarchical.*

Lemma 3.1 allows us to divide the given instance into several independent constrained instances, each of which contains jobs belonging to one superblock only. A superblock contains jobs belonging to a constant number of groups. We are able to show that since the optimal schedule for a superblock is natural and SRPT-friendly, we can divide the given instance into a sequence of constrained instances, in each of which there is exactly one job from the largest numbered group. Unfortunately, this alone does not significantly limit the number of different schedules for one of these instances. We overcome this hurdle by showing that we can restrict our space of schedules to those schedules in which a particular job (in our case, the lone job from the highest numbered group) delays at most $c$ smaller jobs, while incurring an increase in rounded cost of at most $(1 + 1/c)$, for any given positive integer $c$.

**Lemma 3.2** *Let $\mathcal{I}$ be any constrained instance. Let $m$ denote the largest group number of any job in $\mathcal{I}$. Furthermore, suppose that there is exactly one job $j$ from group $m$ in $\mathcal{I}$. Then, given any positive integer $c$, any natural list schedule for $\mathcal{I}$ can be transformed into another list schedule in which $j$ delays at most $c$ smaller jobs, while incurring an increase in rounded cost by a factor of at most $1 + 1/c$.*

Lemma 3.2 and an enumeration of schedules of interest establishes the following claim, that forms the final piece of the algorithm.

**Lemma 3.3** *For any $\delta > 0$, there exists an $n^{O(k/\delta)}$-time algorithm to determine a $(1 + \delta)^k$-approximate schedule for any constrained instance with $k$ groups, where $n$ is the number of jobs in the given instance.*

Using Lemmas 3.1 and 3.3, we now prove the main theorem of this section.

**Theorem 1** *There exists a PTAS for average stretch scheduling.*

**Proof:** Let $\mathcal{I}$ denote the given instance. Let $s$ denote the number of superblocks. Without loss of generality, we assume that the superblocks are numbered 0 through $s-1$. For $0 \leq i \leq s$, let $\mathcal{I}[i]$ denote the sub-instance of $\mathcal{I}$ consisting of jobs in superblocks 0 through $i-1$.

Our algorithm consists of iteratively going through the superblocks, from the smallest to the largest job sizes, and applying the algorithm of Lemma 3.3 to each superblock as follows. Let $\mathcal{S}_i$ denote the schedule obtained at the start of iteration $i$ (we count iterations from 0). Note that $\mathcal{S}_0$ is the empty schedule. We let $C_i$ denote the constrained instance consisting of the jobs in superblock $i$, with the forbidden times being the set of time steps in which the schedule $\mathcal{S}_i$ schedules a job. In iteration $i$, we apply the algorithm of Lemma 3.3 to the constrained instance $C_i$. Let $\mathcal{S}$ denote the schedule obtained for the constrained instance. We obtain $\mathcal{S}_{i+1}$ by merging the two schedules $\mathcal{S}_i$ and $\mathcal{S}$; that is, a job is scheduled at time $t$ in $\mathcal{S}_{i+1}$ if it is scheduled at time $t$ in exactly one of $\mathcal{S}_i$ or $\mathcal{S}$. Since the set of times when a job is scheduled in $\mathcal{S}$ is disjoint from the set of times when a job is scheduled in $\mathcal{S}_i$, $\mathcal{S}_{i+1}$ is a well-defined schedule for the jobs in superblocks 0 through $i$. If $s$ is the number of superblocks in the given instance, then $\mathcal{S}_s$ is the final schedule obtained by the algorithm.

We now analyze the approximation ratio achieved by the algorithm. We argue that the schedule $\mathcal{S}_s$ is a $(1+7\varepsilon)$-approximate schedule, for $\varepsilon > 0$ sufficiently small. This argument is in two parts. We first prove, by induction on $i$, that $\mathcal{S}_i$ is a $(1+2\varepsilon)$-approximate schedule for the instance $\mathcal{I}[i]$ among all hierarchical schedules. For the base case, we let $i = 0$, and the claim holds trivially. For the induction step, we note that any hierarchical schedule for the instance $\mathcal{I}[i]$ consists of two disjoint schedules: one for the instance $\mathcal{I}[i-1]$ and the other for the instance $C_{i-1}$. By the induction hypothesis, $\mathcal{S}_{i-1}$ is a $(1+2\varepsilon)$-approximate schedule for $\mathcal{I}[i-1]$ among all hierarchical schedules. Superblock $i-1$ consists of at most $g/\varepsilon^2$ groups where $g = \log_{1+\varepsilon}(1/\varepsilon^2) \leq 2\lg(1/\varepsilon)/\varepsilon$, for $\varepsilon$ sufficiently small. Applying Lemma 3.3 to constrained instance $C_{i-1}$ with $\delta$ equal to $\varepsilon^4/(2\lg(1/\varepsilon))$ and $k$ equal to $2\lg(1/\varepsilon)/\varepsilon^3$, we obtain that the schedule $\mathcal{S}$ obtained in iteration $i$ is a $(1+2\varepsilon)$-approximate schedule for $C_{i-1}$, as shown in the following:

$$\left(1 + \frac{\varepsilon^4}{2\lg(1/\varepsilon)}\right)^{2\lg(1/\varepsilon)/\varepsilon^3} \leq e^{\varepsilon} \leq (1+2\varepsilon),$$

for $\varepsilon > 0$ sufficiently small. Thus, the schedule $\mathcal{S}_i$, which is obtained by merging the schedules $\mathcal{S}_{i-1}$ and $\mathcal{S}$, is a $(1+2\varepsilon)$-approximate schedule for $\mathcal{I}[i]$ among all hierarchical schedules. This completes the induction step.

For the second part of the approximation ratio argument, we invoke Lemma 3.1 to obtain that there exists a hierarchical schedule whose total stretch is at most $(1+3\varepsilon)$ of the optimum rounded stretch, taken over all schedules. Thus, the total rounded stretch of schedule $\mathcal{S}_s$ is at most $(1+2\varepsilon)(1+3\varepsilon)$ times the optimal total rounded stretch. Since the total stretch is within $(1+\varepsilon)$ of the total rounded stretch, the approximation factor achieved by our algorithm is at most $(1+\varepsilon)(1+2\varepsilon)(1+3\varepsilon) \leq (1+7\varepsilon)$, for any positive constant $\varepsilon$ sufficiently small.

We now analyze the running time of the algorithm. By Lemma 3.3, the running time for iteration $i$ is a polynomial in the number of jobs in superblock $i$ with exponent $O(1/(\varepsilon^7 \lg^2(1/\varepsilon)))$, for $\varepsilon > 0$ sufficiently small. Adding over all iterations, we obtain that the total running time is $n^{O(1/(\varepsilon^7 \lg^2(1/\varepsilon)))}$, where $n$ is the total number of jobs. We thus have a PTAS for average stretch scheduling. ■

The remainder of this section is organized as follows. Section 3.2 establishes certain characteristics of optimal schedules. Sections 3.3, 3.4, and 3.5 establish Lemmas 3.1, 3.2, and Lemma 3.3, respectively.

### 3.2 Natural SRPT-friendly list schedules

The following two lemmas apply to optimal schedules with respect to both stretch and rounded stretch.

**Lemma 3.4** *For any constrained scheduling instance, every optimal schedule is a list schedule.*

**Proof:** Let $\mathcal{S}$ be a schedule that has optimal total stretch (or rounded stretch) and yet is not a list schedule. Since $\mathcal{S}$ is not a list schedule, it follows that there exist two jobs $j_1$ and $j_2$ and a time step $t$ such that $j_1$ finishes before $j_2$ and yet at time step $t$ that occurs after the release of and prior to the completion of $j_1$, $j_2$ is executed. We modify the schedule $\mathcal{S}$ to obtain a new schedule $\mathcal{S}'$ with smaller cost as follows. $\mathcal{S}'$ is the same as $\mathcal{S}$ except that $j_1$ is scheduled at time $t$ and $j_2$ is scheduled at the time step when $j_1$ completes in $\mathcal{S}$. Since $j_1$ completes earlier in $\mathcal{S}'$ than in $\mathcal{S}$ and all other jobs complete at exactly the same times, $\mathcal{S}'$ has lower cost than $\mathcal{S}$, thus yielding a contradiction. ■

**Lemma 3.5** *For any constrained scheduling instance, every optimal schedule is a natural schedule.*

**Proof:** Let $\mathcal{S}$ be a given optimal schedule that is not natural. Let $j$ be a job that delays a smaller job $j'$ executed at time $t$, and let $t$ be the earliest such time. We first derive a contradiction if $\rho_t(j) > \rho_t(j')$. Since $\mathcal{S}$ is a list schedule, $j$ completes before $j'$. Consider the time steps starting from $t$ at which either $j$ or $j'$ is executed. During these steps, $j$ is first executed and then $j'$. We swap the order to obtain a new schedule $\mathcal{S}'$. The completion time of $j'$ in $\mathcal{S}'$ is earlier than the completion time of $j$ in $\mathcal{S}$ and the completion time of $j$ in $\mathcal{S}'$ is identical to that of $j'$ in $\mathcal{S}$. Since $p(j') < p(j)$, it follows that the total cost of $\mathcal{S}'$ is less than that of $\mathcal{S}$, thus yielding a contradiction.

We next show that $\rho_t(j') < p(j')$ is also impossible. By Lemma 3.4, the optimal schedule is a list schedule. Since $t$ is the first time step that $j'$ is delayed by $j$, it follows that $j$ is not scheduled during the period $[r(j'), t-1]$. If $\rho_t(j') < p(j')$, then $j'$ is scheduled at least once during this interval, giving it higher priority over $j$ in the schedule. Since $j$ is scheduled ahead of $j'$ at time $t$, this yields a contradiction to the fact that $\mathcal{S}$ is a list schedule. ■

Consider a schedule that minimizes total rounded cost. Since the rounded cost assigns equal "weights" to all of the jobs in the same group, scheduling within a group in an optimal schedule minimizes the total flow time of the jobs subject to constraints placed by jobs outside the group. We establish the following lemma by arguing that every schedule that minimizes total flow time is an SRPT schedule.

**Lemma 3.6** *For any constrained instance, every schedule that optimizes rounded cost is* SRPT-*friendly.*

**Proof:** Let $\mathcal{I}$ be a given constrained instance and let $\mathcal{S}$ be a schedule for $\mathcal{I}$ that optimizes rounded cost and yet is not SRPT-friendly. Fix a group in which the jobs do not execute in SRPT order. Let $G$ denote the set of jobs in the group. Consider the constrained scheduling instance $\mathcal{I}'$, in which the set of jobs is $G$ and the set of allowable times is exactly the set of times during which these jobs are scheduled in $\mathcal{S}$. Since $\mathcal{S}$ optimizes rounded cost, it follows that when restricted to the constrained instance $\mathcal{I}'$, $\mathcal{S}$ optimizes total flow time. We now argue that this leads to a contradiction. Our proof resembles closely the proof of the well-known result that SRPT optimizes total flow time for arbitrary (unconstrained) scheduling instances [2].

Let $t$ be the earliest time instant when $\mathcal{S}$ schedules a job $j \in G$, while a job $j' \in G$ with lower remaining processing time is in the queue at time $t$ (i.e., $\rho_t(j') < \rho_t(j)$). By our assumption that $\mathcal{S}$ is not SRPT-friendly, such a time instant exists. Let $T$ denote the set of time intervals during the period $[t, \infty)$ when either job $j$ or job $j'$ is scheduled in $\mathcal{S}$. Thus, the total length of $T$ equals $\rho_t(j') + \rho_t(j)$. Consider the schedule $\mathcal{S}'$ which is identical to $\mathcal{S}$ except that during $T$, we completely schedule the remainder of $j$ (which equals $\rho_t(j')$) and then the remainder of $j'$ (which equals $\rho_t(j)$). Since $\rho_t(j') < \rho_t(j)$, it follows that the completion time of $j'$ in $\mathcal{S}'$ is less than the completion time of $j$ in $\mathcal{S}$, while the completion time of $j$ in $\mathcal{S}'$ is at most the completion time of $j'$ in $\mathcal{S}$. Since the total flow time is the sum of completion times minus the sum of release times, it follows that schedule $\mathcal{S}'$ has a smaller total flow time than $\mathcal{S}$ for instance $\mathcal{I}'$, yielding a contradiction. The desired claim follows. ■

### 3.3 Eliminating delays of small jobs by larger jobs

In this section, we prove Lemma 3.1. Given a constrained instance $I$ and a natural SRPT-friendly list schedule $\mathcal{S}$ for $I$, we derive a natural SRPT-friendly list schedule in which no job in superblock $i$ delays any job in superblock $j$ for $j < i$, while incurring a cost increase by a factor of at most $(1 + 2\varepsilon)$.

Let $m$ denote the largest group index in $I$ such that a job in group $m$ delays a job in superblock $spr(m) - 1$ or lower. (Recall that $spr(m)$ is the superblock to which group $m$ belongs.) We describe a *sweep procedure* by which we convert $\mathcal{S}$ into a new natural SRPT-friendly list schedule in which no job in group $m$ delays any job in superblock $spr(m) - 1$ or lower. The sweep procedure consists of the repeated application of a *local reordering procedure* which ensures that a particular job in group $m$ does not delay any job in superblock $spr(m) - 1$ or lower; this job is the first job in group $m$ that delays some job in superblock $spr(m) - 1$ or lower, according to schedule $\mathcal{S}$. We first describe the local reordering procedure and then the sweep procedure.

**Local reordering.** Let $t$ be the earliest instant at which a job $j$ in group $m$ delays a job in superblock $spr(m) - 1$ or lower. Let $t'$ be the earliest time step after $t$ at which there are no jobs from superblock $spr(m) - 1$ or lower in the queue. We now claim that at every time step in the interval $[t, t' - 1]$, either $j$ or some job in superblock $spr(m) - 1$ or lower is executed. The proof is by contradiction. Let $t_1$ be the earliest time in $[t, t' - 1]$ at which a job other than $j$, say $j_1$, belonging to superblock $spr(m)$ or higher is executed. We show that $\rho_{t_1}(j_1) \geq \min\{p(j_1), p(j)\}$. We consider different cases:

- $r(j_1) > t$: This implies that $j_1$ is executed for the first time at time $t_1$; that is, $\rho_{t_1}(j_1) = p(j_1)$.
- $r(j_1) \leq t$, $p(j_1) < p(j)$: Since $j$ delays $j_1$ at time $t$ and $\mathcal{S}$ is natural, it follows that $\rho_t(j_1) = p(j_1)$. This implies that $j_1$ is executed for the first time at time $t_1$; thus, $\rho_{t_1}(j_1) = p(j_1)$.
- $r(j_1) \leq t, p(j_1) \geq p(j)$: Since no job in a group higher than $m$ delays any job in superblock $spr(m) - 1$ or lower, we obtain in this case that $j_1$ is in group $m$. Since $\mathcal{S}$ is a list schedule and $j$ is scheduled at time $t$, it follows that during the interval $[r(j), t - 1]$, $j_1$ is not scheduled. If $r(j) \geq r(j_1)$, then since $\mathcal{S}$ is SRPT-friendly, it follows that $\rho_t(j_1) = \rho_{r(j)}(j_1) \geq p(j)$. On the other hand, if $r(j) \leq r(j_1)$, then $j_1$ has not been scheduled until time $t$, which implies that $\rho_t(j_1) = p(j_1) \geq p(j)$. Since $t_1$ is the first time in the interval $[t, t' - 1]$ that $j_1$ is scheduled, it follows that $\rho_{t_1}(j_1) = \rho_t(j_1) \geq p(j)$.

For each of the above (exhaustive) cases, we have shown that $\rho_{t_1}(j_1) \geq \min\{p(j_1), p(j)\}$. Since $t_1$ is in $[t, t' - 1]$, it follows from our choice of $t'$ that there exists at least one job $j_2$ from superblock $i$,

$i < spr(m)$, in the queue at time $t_1$. We thus have a job $j_1$ in superblock $spr(m)$ delaying a job $j_1$ in a superblock $i$, $i < spr(m)$, even though the remaining processing time of $j_1$ at time $t_1$ is greater than that of $j_2$. Formally, we have $\rho_{t_1}(j_2) \leq p(j_2) < \min\{p(j), p(j_1)\} \leq \rho_{t_1}(j_1)$. This contradicts our assumption that $\mathcal{S}$ is a natural schedule.

We next observe that since $\mathcal{S}$ is a list schedule and $j$ delays some job in superblock $spr(m) - 1$ or lower at time $t$, $j$ completes before time $t'$. Let $J$ denote the set of jobs other than $j$ that are executed in the interval $[t, t' - 1]$. We modify the schedule so that $j$ is given a priority lower than any job in $J$, and the jobs in $J$ are scheduled according to an optimal SRPT-friendly, natural list schedule within $[t, t']$. Let $\mathcal{S}'$ denote the new schedule obtained. We refer to this procedure as the *reordering* procedure, and write $\mathcal{S}' = R(\mathcal{S})$. We also call $J$ the set of *promoted jobs*. Note that in $\mathcal{S}'$, the total rounded stretch of the jobs in $J$ is at most that in $\mathcal{S}$, while the completion time of $j$ is at most $t'$. Therefore, the increase in rounded cost is at most $\sum_{j' \in J} p(j')/p(j)$.

**Lemma 3.7** *The schedule $\mathcal{S}'$ is an* SRPT*-friendly natural list schedule.*

**Proof:** We first argue that the schedule $\mathcal{S}'$ is a list schedule. The completion time order for $\mathcal{S}$ consists of a sequence $J_1$, followed by $j$, followed by a permutation $\pi$ of the jobs in $J$, followed by another sequence $J_2$. Instead, the completion time order for $\mathcal{S}'$ consists of $J_1$, followed by a permutation $\pi'$ of the jobs in $J$, followed by $j$, followed by another sequence $J_2$. It is easy to see that the jobs are executed in the same priority order. Therefore, $\mathcal{S}'$ is a list schedule.

We next argue that the schedule $\mathcal{S}'$ is an SRPT-friendly schedule. The only jobs that are scheduled differently in $\mathcal{S}'$ than in $\mathcal{S}$ are in the set $J \cup \{j\}$. Consider job $j$. Since $j$ is the first job in group $m$ to complete after time $t$ in $\mathcal{S}$, it follows that the processing within group $m$ is done in SRPT order in $\mathcal{S}'$. All of the jobs in $J$ are completely executed in the interval $[t, t']$ and their scheduling is SRPT-friendly by construction. Therefore $\mathcal{S}'$ is SRPT-friendly.

We finally argue that $\mathcal{S}'$ is a natural schedule. Since $\mathcal{S}$ is a natural schedule and $\mathcal{S}'$ differs from $\mathcal{S}$ only in the jobs scheduled during the interval $[t, t' - 1]$, we need to consider the executions performed under $\mathcal{S}'$ during the interval $[t, t' - 1]$ only. Let $t_1$ be any time instant in $[t, t' - 1]$. We consider two cases.

- Case 1: Job $j$ is scheduled at time $t_1$ in $\mathcal{S}'$. Consider a job $j_1$ in the queue at time $t_1$. We need to argue that if $p(j_1) < p(j)$, then $\rho_{t_1}(j) < \rho_{t_1}(j_1) = p(j_1)$. Suppose that $p(j_1) < p(j)$. By construction of $\mathcal{S}'$, $j_1$ is not from superblock $spr(m) - 1$ or lower. Since $j$ is the sole job in superblock $spr(m)$ or higher that is scheduled in $\mathcal{S}'$ during the interval $[t, t' - 1]$, $\rho_{t_1}(j_1) = \rho_t(j_1)$. Since $\mathcal{S}$ is a natural schedule and $\mathcal{S}'$ is the same as $\mathcal{S}$ outside of the interval $[t, t' - 1]$, we have $\rho_t(j_1) = p(j_1)$; this is because $j$ delays $j_1$ at time $t$ in schedule $\mathcal{S}$. Therefore, $\rho_{t_1}(j_1) = p(j_1)$, thus completing this case.
- Case 2: Job $j' \in J$ is scheduled at time $t_1$ in $\mathcal{S}'$. The only jobs of smaller processing time that $j'$ delays in $\mathcal{S}'$ belong to $J$. By the construction of $\mathcal{S}'$, the schedule restricted to the jobs in $J$ is natural, thus completing this case.

The above two cases establish that $\mathcal{S}'$ is a natural schedule, thus completing the proof of the lemma. ■

In schedule $\mathcal{S}'$, no job in group $m$ delays a job in superblock 0 through $spr(m) - 1$ during the time interval $[0, t' - 1]$. Furthermore, $\mathcal{S}'$ shares the property of $\mathcal{S}$ that no job in group $m + 1$ or higher delays any job in a lower indexed superblock.

**Sweep**. We repeat the above transformation procedure with the schedule $\mathcal{S}'$ which, by Lemma 3.7, is an SRPT-friendly natural list schedule, and continue until every job in group $m$ delays no job in superblock $spr(m) - 1$ or lower. Let $\mathcal{S} = \mathcal{S}_0, \mathcal{S}_1 = R(\mathcal{S}_0), \ldots, \mathcal{S}_k = R(\mathcal{S}_{k-1}), \ldots, \mathcal{S}_\ell$ denote the sequence of transformations in the sweep procedure, and let $J_0 = J, J_1, \ldots, J_k, \ldots, J_{\ell-1}$ denote the sets of promoted jobs in each transformation. By the definition of the reordering process, the sets $J_k$ are all disjoint. From the cost analysis of the reordering procedure, it follows that the increase in cost as a result of the transformation from $\mathcal{S}_k$ to $\mathcal{S}_{k+1}$ is at most the ratio of the sum of the processing times of the jobs in $J_k$ to $(1+\varepsilon)^m$. Therefore, the increase in cost due to the sweep procedure is at most

$$\sum_{k=0}^{\ell-1} \sum_{j' \in J_k} \frac{p(j')}{(1+\varepsilon)^m} \quad \leq \quad \sum_{s < spr(m)} \sum_{i \in grps(s)} \frac{n_i}{(1+\varepsilon)^{m-i-1}} \tag{1}$$

where $n_i$ is the number of jobs in group $i$. The inequality follows from the following observations: (a) the set $\cup_k J_k$ contains jobs from superblocks 0 through $spr(m) - 1$ only; and (b) the processing time of a job in group $i$ is at most $(1+\varepsilon)^{i+1}$.

The schedule $\mathcal{S}_\ell$ obtained as a result of the sweep procedure is a natural SRPT-friendly list schedule and has the property that no job in groups $m$ or higher delay any job in a lower indexed superblock. We now use the sweep procedure to convert a natural SRPT-friendly list schedule into another natural SRPT-friendly list schedule in which no job in superblock $i$ delays any job in superblock $j < i$, for any $i$, while incurring an increase in cost by a factor of at most $1 + 2\varepsilon$.

1. Let $m$ denote the highest indexed group such that a job in group $m$ delays a job in superblock $spr(m) - 1$ or lower in $\mathcal{S}$. We apply the sweep procedure described above to obtain a new SRPT-friendly natural list schedule $\widehat{\mathcal{S}}$ in which no job in group $m$ delays any job in superblocks $spr(m) - 1$ or lower.

2. We set $\mathcal{S}$ to $\widehat{\mathcal{S}}$ and repeat step 1.

By repeated application of Lemma 3.7, it follows that the final schedule obtained is a natural SRPT-friendly list schedule. Furthermore, it is hierarchical; that is, no job delays a job that is in a lower indexed superblock.

We now calculate an upper bound on the increase in cost due to the above procedure. Let $m^*$ denote the largest group index. Let $s^*$ denote the largest superblock index. We need to sum up the term in Equation 1 over all the groups except those that belong to superblock 0. Let $m_k$ denote the number of jobs in block $k$. We bound the total increase in cost as follows:

$$
\begin{aligned}
& \sum_{s'>0} \sum_{\ell \in grps(s')} \sum_{s<s'} \sum_{i \in grps(s)} \frac{n_i}{(1+\varepsilon)^{\ell-i-1}} \\
= & \sum_{s<s^*} \sum_{i \in grps(s)} \sum_{s'>s} \sum_{\ell \in grps(s')} \frac{n_i}{(1+\varepsilon)^{\ell-i-1}} \\
= & \sum_{s<s^*} \sum_{i \in grps(s)} \sum_{\ell=(b+s/\varepsilon^2)g}^{m^*} \frac{n_i}{(1+\varepsilon)^{\ell-i-1}} \\
\leq & \sum_{s<s^*} \sum_{i \in grps(s)} \frac{n_i}{\varepsilon(1+\varepsilon)^{(b+s/\varepsilon^2)g-i-1}} \\
\leq & \sum_{s=0}^{s^*-1} \left[ \left( \sum_{i:blk(i)=b+s/\varepsilon^2-1} \frac{n_i}{\varepsilon(1+\varepsilon)^{(b+s/\varepsilon^2)g-i-1}} \right) + \sum_{k<b+s/\varepsilon^2-1, k \in blks(s)} \left( \sum_{i:blk(i)=k} \frac{n_i}{\varepsilon(1+\varepsilon)^{g-1}} \right) \right] \\
\leq & \sum_{s=0}^{s^*-1} \left[ \left( \sum_{i:blk(i)=b+s/\varepsilon^2-1} \frac{n_i}{\varepsilon} \right) + \sum_{k<b+s/\varepsilon^2-1, k \in blks(s)} \left( \sum_{i:blk(i)=k} \frac{n_i}{\varepsilon(1+\varepsilon)^{g-1}} \right) \right] \\
\leq & \sum_{s=0}^{s^*-1} \left[ \frac{m_{b+s/\varepsilon^2-1}(1+\varepsilon)^2}{\varepsilon} + \sum_{k<b+s/\varepsilon^2-1, k \in blks(s)} (1+\varepsilon) m_k \varepsilon \right] \\
\leq & \ 2n(1+\varepsilon)\varepsilon \\
\leq & \ 3n\varepsilon,
\end{aligned}
$$

for $\varepsilon > 0$ sufficiently small. (To obtain the second line, we change the order of summations and note that summing over $s' > 0$ and $s < s'$ is identical to summing over $s < s^*$ and $s' > s$. The third line follows from the fact that the first group index in superblock $s+1$ is $(b+s/\varepsilon^2)g$. In the fourth line, we use the inequality $\sum_{i \geq 0} 1/(1+\varepsilon)^i \leq 1/\varepsilon$. In the fifth line, we separate the summation over groups in superblock $s$ to two summations, one over groups in the last block in superblock $s$, and the other over groups in the remaining blocks in superblock $s$. In the last step, we use the fact that the number of jobs in the largest numbered blocks of all of the superblocks is at most $n\varepsilon^2$.)

Since the rounded stretch of any job is at least 1, the total rounded stretch of any schedule is at least $n$. Thus, the increase in cost as a result of the transformation is at most by a factor of $(1+3\varepsilon)$. This completes the proof of Lemma 3.1.

## 3.4 Bounding the number of smaller jobs delayed by a job

In this section, we prove Lemma 3.2. Let $\mathcal{S}$ be the given natural list schedule for a given constrained instance $I$. Let $m$ be the largest group number in $\mathcal{S}$ and let $j$ denote the lone job from group $m$ in $\mathcal{S}$. Let $c$ be a given positive integer. The goal is to determine a list schedule $\mathcal{S}'$ for $I$ of cost at most $(1+1/c)$ times the cost of $\mathcal{S}$ such that $j$ delays no more than $c$ jobs in $\mathcal{S}'$ with smaller processing time than $j$.

Let $t$ denote the earliest time step at which $j$ delays more than $c$ jobs with smaller processing time. Let $t'$ denote the earliest time step after $t$ in which there are exactly $c$ jobs smaller than $j$ in the queue. Thus, at least one of the $c$ jobs that are delayed by $j$ at time $t$ complete at or before time $t'$. Since $\mathcal{S}$ is a list schedule, we obtain that $j$ completes before time $t'$. Let $J$ denote the set of $c$ jobs smaller than $j$ that are in the queue at time $t'$. We claim that none of the jobs in $J$ is scheduled until time $t'$. The proof is by contradiction. Let $j'$ be a job in $J$ that is executed at time

$t_1$ prior to $t'$. We first argue that $t_1 > t$. If $r(j') > t$, then the preceding claim is trivial; otherwise, since $\mathcal{S}$ is a natural schedule and $j$ delays $j'$ at time $t$, $\rho_t(j') = p(j')$, thus implying that $t_1 > t$. Since $j'$ is executed at time $t_1 \in (t, t')$, at time $t_1$ there are greater than $c$ jobs in the queue that are smaller than $j$. Since $j'$ is still in the queue at time $t'$, the remaining at least $c$ jobs smaller than $j$ in the queue at time $t_1$ should also be in the queue at time $t'$ because $\mathcal{S}$ is a list schedule. But there are only $c$ jobs in the queue at time $t'$ that are smaller than $j$, thus yielding a contradiction. It follows that all of the jobs that are scheduled during the interval $[t, t'-1]$ complete prior to time $t'$.

We modify the schedule $\mathcal{S}$ to derive schedule $\mathcal{S}'$ as follows. During the interval $[t, t'-1]$, we assign a priority to $j$ higher than all jobs in $J$ and lower than all other jobs. Subject to this constraint, we derive the best schedule for the remaining jobs that complete during the interval $[t, t'-1]$ in $\mathcal{S}$. First, since $\mathcal{S}'$ is obtained by merely rescheduling the processing performed during the interval $[t, t'-1]$, it follows that all of the jobs, including $j$, that are scheduled in $\mathcal{S}'$ during the interval $[t, t'-1]$ complete prior to time $t'$. Since $j$ has least priority among these jobs, the increase in rounded cost as a result of the transformation from $\mathcal{S}$ to $\mathcal{S}'$ is at most $(t'-t)/(1+\varepsilon)^m$. In schedule $\mathcal{S}$, there are at least $c$ jobs in groups $m-1$ or lower at each instant in $[t, t'-1]$. Thus, the rounded cost of $\mathcal{S}$ is at least $c(t'-t)/(1+\varepsilon)^{m-1}$. Therefore, the rounded cost of $\mathcal{S}'$ is at most $(1+1/c)$ times the rounded cost of $\mathcal{S}$.

We note that the resultant schedule obtained may not be a list schedule. We convert $\mathcal{S}'$ into a list schedule $L(\mathcal{S}')$ by assigning priorities to each job according to their completion times. In the following lemma, we argue that the rounded cost of schedule $L(\mathcal{S}')$ is at most that of $\mathcal{S}'$. We also show that the number of jobs delayed by $j$ does not increase; hence, it remains at most $c$. This completes the proof of Lemma 3.2.

**Lemma 3.8** *Let $\mathcal{S}$ be a schedule for a given constrained instance and let $L(\mathcal{S})$ denote a list schedule obtained by scheduling every job in the constrained instance in order of their completion times in $\mathcal{S}$. The rounded cost of $L(\mathcal{S})$ is at most that of $\mathcal{S}$. Furthermore, the number of jobs delayed by a job $j$ in $L(\mathcal{S})$ is at most the number of jobs delayed by $j$ in $\mathcal{S}$.*

**Proof:** Suppose the jobs complete in $\mathcal{S}$ in the order $j_1, j_2, \ldots, j_\ell$, where $\ell$ is the number of jobs in the given instance. We claim that for $1 \le i \le \ell$ and any time $t$, the total amount of time during the interval $[0, t]$ that the jobs $j_1$ through $j_i$ are scheduled in $L(\mathcal{S})$ is at least the corresponding time in $\mathcal{S}$. This is because for any $i$ and any available time instant, a job in the set $\{j_1, \ldots, j_\ell\}$ is scheduled in $L(\mathcal{S})$ at time $t$ if it has been released and not yet complete. Applying the preceding claim inductively in the order $j_1$ through $j_\ell$, we obtain that the completion time of any job $j$ in $L(\mathcal{S})$ is at most that in $\mathcal{S}$. Since the rounded stretch of a job is the ratio of the difference between the completion time and the release time to the rounded processing time, the rounded cost of the schedule $L(\mathcal{S})$ is at most that of $\mathcal{S}$. This completes the proof of the first part of the lemma.

For the second part of the lemma, it is enough to observe that the priority order among the jobs in schedule $L(\mathcal{S})$ implies that a job $j$ can never delay a job that completes earlier than $j$ in $\mathcal{S}$. Furthermore, if in $L(\mathcal{S})$ $j$ delays a job $j'$ that completes later than $j$ in $\mathcal{S}$, then the release time of $j'$ is before the completion time of $j$ in $\mathcal{S}$, implying that $j$ delays $j'$ in $\mathcal{S}$. Thus, the number of jobs that $j$ delays in $L(\mathcal{S})$ is at most the number of jobs that $j$ delays in $\mathcal{S}$. ■

### 3.5 An approximation algorithm for a constant number of groups

The final step of the algorithm is a polynomial-time approximation algorithm for any constrained scheduling problem with a constant number of groups. More precisely, we give an $n^{O(k/\delta)}$-time algorithm to determine a $(1+\delta)^k$-approximate schedule for an instance with $k$ groups, for any positive real $\delta$. Without loss of generality, we assume that $1/\delta$ is an integer.

Our algorithm is based on enumerating schedules of interest and selecting the schedule of least rounded cost. Given a schedule, the rounded cost of the schedule can be calculated in $O(n)$ time. Therefore, the algorithm can be described by specifying the schedules that are enumerated and their number. Our algorithm is recursive, and we develop an inductive proof of its correctness along with the algorithm description. We show, by induction on the number of groups, that our algorithm enumerates $O(n^{k/\delta+k})$ schedules for an instance with $k$ groups, at least one of which is $(1+\delta)^k$-approximate.

**Base case**. The base case is when $k=1$. In this case, our algorithm returns an SRPT schedule. We first argue that every SRPT schedule has the same rounded cost. We note that the multiset of remaining processing times of all jobs at any time in any two SRPT schedules is identical since any SRPT schedule decrements the remaining processing time of a job with least remaining processing time in each step. It thus follows that the multisets of completion times of the jobs in any two SRPT schedules are identical. The rounded cost is the difference of the sum of the weighted completion times and the sum of the weighted release times, where the weight is equal to the reciprocal of the rounded job size. Since all the jobs belong to the same group, they have the same weight, implying that the sum of the weighted completion times and the sum of the weighted release times are, respectively, the same for any two SRPT schedules. Thus, every SRPT schedule has the same rounded cost.

By Lemma 3.6, every optimal schedule is an SRPT-friendly schedule. Since the instance has one group only, every SRPT-friendly schedule is an SRPT schedule. Since all SRPT schedules have the same rounded cost, it follows that every SRPT schedule is optimal, thus establishing the correctness of the algorithm for this case.

**Recursive case**. Suppose we have an instance with $k$ groups. Without loss of generality, we may assume that the $k$ groups are numbered 0 through $k-1$. By Lemmas 3.4, 3.5, and 3.6, we know that there exists an SRPT-friendly natural list schedule that has optimal rounded cost. The overall structure of the recursion step is as follows. (We present the formal details below.)

1. **Division:** Divide the given constrained instance into a sequence of constrained instances $\{I_i\}$, in each of which there is exactly one job from the largest numbered group.

2. **Enumeration:** For each instance $I_i$, determine a set $C_i$ of $O(n^{1/\delta})$ constrained instances that consist of jobs from groups 0 through $k-2$, by enumerating $O(n^{1/\delta})$ different schedules for the lone job in group $k-1$.

3. **Recursion:** For each $i$ and for each constrained instance in $C_i$, we recursively determine a $(1+\delta)^{k-1}$-approximate schedule. Each schedule thus obtained determines a candidate schedule for $I_i$. We select the best schedule among the $O(n^{1/\delta})$ candidate schedules as the schedule for $I_i$. The schedule for the instance $I$ is obtained by merging the schedules obtained for $I_i$, for all $i$.

Before describing the division step, we introduce some notation and a supporting claim. Let $J$ denote the set of jobs in the largest indexed group $k-1$, and let $\ell$ denote the number of these jobs. We first determine the order in which these jobs finish in an optimal schedule. This will enable us to split the given scheduling instance $I$ into $\ell$ constrained scheduling instances $I_1, I_2, \ldots, I_\ell$ such that each instance contains exactly one job from $J$. In order to determine the order of completion of the jobs in $J$ we use the fact that the optimal schedule is natural and SRPT-friendly. The order of completion of jobs in $J$ is the same as that in which the jobs complete assuming that the jobs in $J$ are scheduled in SRPT order and *every* job in $J$ has lower priority than any job in groups 0 through $k-2$. We refer to such a schedule as a *groupwise* schedule. Let the completion order of jobs in $J$ be $j_1, j_2, \ldots, j_\ell$. For $1 \le i \le \ell$, let $t_i$ denote the completion time of job $j_i$ in the groupwise schedule. For convenience, we set $t_0 = 0$.

**Lemma 3.9** *The order of completion of jobs in $J$ in an optimal schedule is identical to that in a groupwise schedule.*

**Proof:** Let $\rho^1_t(j)$ (resp., $\rho^2_t(j)$) denote the remaining processing time of job $j$ at time $t$ under a given optimal schedule (resp., groupwise schedule). We claim that for any job $j \in J$ and any $0 \le i \le \ell$, $\rho^1_{t_i}(j) = \rho^2_{t_i}(j)$. Before proving this claim, we argue that the statement of the lemma follows from the claim. To see this, note that (a) job $j_i$ completes in the groupwise schedule at time $t_i$ and $\rho^2_{t_{i-1}}(j_i) > 0$; (b) $j_i$ completes in the optimal schedule at the earliest time $t$ at which $\rho^1_t(j_i) = 0$. According to our claim, $\rho^1_{t_{i-1}}(j_i) > 0$ while $\rho^1_{t_i}(j_i) = 0$; therefore, $j_i$ completes at some time in the interval $(t_{i-1}, t_i]$. The statement of the lemma follows.

We now prove the claim in the preceding paragraph. The proof is by induction on $i$. For the base case, we set $i = 0$. At time $t_0 = 0$, the remaining processing time of each job in the groupwise schedule is identical to that in the optimal schedule; so the desired claim holds. We now consider the induction step $i > 0$. For the induction hypothesis, we assume that $\rho^1_{t_i}(j) = \rho^2_{t_i}(j)$ for all $j \in J$. We consider two cases for the induction step. The first case is when there does not exist any time $t$ in $[t_{i-1}, t_i - 1]$ when one schedule processes a job in $J$ while the other processes a job not in $J$. In this case, the induction hypothesis directly implies the induction step.

For the second case, we assume that there exists a time in $[t_{i-1}, t_i - 1]$ when one schedule processes a job in $J$ while the other processes a job not in $J$. Let $t$ be the earliest such time. The induction hypothesis implies that the total work done on jobs outside $J$ in both schedules is the same until time $t$. Since the groupwise schedule assigns lower priority to all jobs in $J$ when compared to any job not in $J$, it follows that at time $t$ the groupwise schedule processes a job not in $J$ (i.e., in one of groups 0 through $k-2$), while the optimal schedule processes a job $j \in J$. We now claim that $j$ is the first job in $J$ to complete after time $t$ in both the schedules (and hence, $j = j_i$). By the induction hypothesis, the choice of $t$ and the fact that the jobs in $J$ are processed in SRPT order by both the schedules (assuming that ties in the SRPT order are broken the same way), the remaining processing time of $j_i$ at time $t$ is the same in both schedules. Since the optimal schedule is natural, this remaining processing time is less than $(1+\varepsilon)^{k-1}$, in both schedules. This implies that $\rho_j(t)$ in both the schedules is less than the processing time of any job in $J$ that is released after time $t$. Furthermore, the remaining processing time of every job in the set $S = \{j_{i+1}, \ldots, j_\ell\}$ at time $t$ is at least $(1+\varepsilon)^{k-1}$; this is because otherwise there exists some other job $j_r$, $r > i$, that has remaining processing time less than $(1+\varepsilon)^{k-1}$ at time $r(j_i)$, implying that $j_r$ should have higher priority than $j_i$ in the SRPT order, thus yielding a contradiction. Since the jobs in $J$ are processed in SRPT order in both schedules, it follows that both the schedules assign lower priority to the jobs in the set $S = \{j_{i+1}, \ldots, j_\ell\}$, when compared with the jobs outside $S$. Thus, in both schedules, $j_i$ is the job from $J$ that will complete next. Furthermore, in both schedules, no job from the set $\{j_{i+1}, \ldots, j_\ell\}$ will be processed until the first time after $t$ when there is no job outside of $S$; this time is the same in both the schedules and equals $t_i$. Therefore, we have $\rho^1_{t_i}(j_r) = \rho^2_{t_i}(j_r) = 0$ for $r \le i$ and $\rho^1_{t_i}(j_r) = \rho^2_{t_i}(j_r) = \rho^1_t(j_r) = \rho^2_t(j_r)$, for $r > i$. This completes the proof of the induction step, and hence the claim. ■

**Division**. The order of completion of the jobs in $J$ can be used to split the optimal schedule into $\ell$ parts. The first part begins at time 0 and ends at time $t_1$. We break the given instance $I$ into two constrained instances $I_1$ and $I'$ as follows. Let the set $S_1$ consist of $j_1$ and all jobs in groups 0 through $k-2$ that arrive in the interval $[0, t_1 - 1]$. We note that all of these jobs complete both in the groupwise schedule and in the optimal schedule during the interval $[0, t_1]$. Furthermore, in both schedules, jobs in $J - \{j_1\}$ are executed in SRPT order and are given lower priority than every job in $S_1$. We set instance $I_1$ to be the set $S_1$ of jobs and their release times. We set instance $I'$ to be the remainder of the jobs with their release times subject to the forbidden times imposed by the jobs of instance $I_1$. We note that the set of time periods during which the jobs of instance $I_1$ are scheduled are independent of the particular schedule used for $I_1$; hence, the constrained instance $I'$ is well-defined.

By construction, the instances $I_1$ and $I'$ consist of disjoint sets of jobs and time periods for processing jobs. Given a schedule $\mathcal{S}_1$ for $I_1$ and a schedule $\mathcal{S}'$ for $I'$, we obtain a schedule $\mathcal{S}$ for $I$ by simply merging the two schedules. That is, a job is scheduled at time $t$ in $\mathcal{S}$ if it is scheduled at time $t$ in exactly one of $\mathcal{S}_1$ or $\mathcal{S}'$. The rounded cost of $\mathcal{S}$ is the sum of the rounded costs of $\mathcal{S}_1$ and $\mathcal{S}'$. Furthermore, the instances $I_1$ and $I'$ have been defined such that the given optimal schedule can be split into two disjoint schedules, one for $I_1$ and the other for $I'$. Therefore, an optimal schedule for $\mathcal{S}$ can be obtained by determining optimal schedules for both $I_1$ and $I'$ and then merging them. Similarly, for any $\alpha \geq 1$, an $\alpha$-approximate schedule for $I_1$ and an $\alpha$-approximate schedule for $I'$ yields an $\alpha$-approximate schedule for $I$.

The division step consists of repeating the above splitting iteratively to obtain a series of constrained instances $I_1$ through $I_\ell$ such that in any instance $I_i$, we have exactly one job from $J$ (and hence group $k-1$). By the argument in the preceding paragraph, an $\alpha$-approximate schedule for $I$ can be obtained by merging together $\alpha$-approximate schedules for each $I_i$, $1 \leq i \leq \ell$. The enumeration and recursion steps show how to obtain a $(1+\delta)^k$-approximate list schedule for any constrained instance that contains exactly one job from group $k-1$.

**Enumeration**. Consider constrained instance $I_i$. The job $j_i$ is the lone job of group $k-1$ that is in instance $I_i$. By Lemma 3.2, any natural list schedule for $I_i$ can be converted into a list schedule in which $j_i$ does not delay more than $1/\delta$ smaller jobs, while increasing the cost by a factor of at most $1+\delta$. Since there exists a natural list schedule with optimal rounded cost, it follows that there exists a list schedule with cost at most $1+\delta$ times the optimal rounded cost, in which $j$ delays no more than $1/\delta$ smaller jobs.

We now compute a $(1+\delta)^k$-approximate schedule for the instance $I_i$ by computing a schedule that is $(1+\delta)^{k-1}$-approximate among all schedules in which $j_i$ delays no more than $1/\delta$ smaller jobs. We stipulate that $j_i$ delays at most $1/\delta$ jobs in groups 0 through $k-2$. There are thus at most $\frac{1}{\delta}\binom{n}{1/\delta} = O(n^{1/\delta})$ selections for the set of jobs that may be delayed by $j_i$. Each such selection identifies a set $X$ of size at most $1/\delta$. Every list schedule for $I_i$, which assigns a priority to $j_i$ higher than any job in $X$ and lower than any other job, processes $j_i$ at exactly the same time periods. Thus, the set $X$ completely determines the time periods at which $j_i$ is processed in any schedule for $I_i$ that obeys the constraint that $j_i$ may not delay any job outside $X$. For each selection of $X$, we determine the times at which $j_i$ is processed. We then calculate two constrained instances containing jobs from groups 0 through $k-2$ only. The first instance includes jobs that do not get delayed by $j_i$ and complete before the completion of that job. The second instance consists of the jobs that get delayed and the jobs that arrive after the completion of $j_i$. All of the time steps prior to this completion can be marked as forbidden for the second instance.

For a given $\alpha$, if we obtain $\alpha$-approximate schedules for each of the two constrained instances defined above, then we can merge the two schedules to obtain a schedule for $I_i$ that is $\alpha$-approximate among all schedules in which $j_i$ does not delay any job outside of $X$. This completes the enumeration step.

**Recursion**. By the induction hypothesis we know that for any constrained instance with $n$ jobs and at most $s < k$ groups, our algorithm enumerates $O(n^{s/\delta+s})$ schedules and determines a $(1+\delta)^s$-approximate schedule. Thus, $(1+\delta)^{k-1}$-approximate schedules for all of the $O(n^{1/\delta})$ instances obtained in the enumeration step, for all $I_i$, can be computed by enumerating $n^{(k-1)/\delta+k-1}$ schedules and selecting the one with smallest cost. Thus, the total number of schedules enumerated following the recursion is $O(n^{1/\delta}) \cdot \ell \cdot O(n^{(k-1)/\delta+k-1}) = O(n^{k\delta+k})$. And the approximation factor is at most $(1+\delta)^k$. This completes the induction step and the proof of Lemma 3.3.

# 4 Rounding of job sizes

In this section, we study the impact of incomplete knowledge of job sizes on stretch and flow metrics. We first consider a natural variant of SRPT, in which jobs are scheduled according to the rounded values of their remaining processing times, rather than the remaining processing times. This class of algorithms, which we refer to as $\lambda$-SRPT, is analyzed in Section 4.1.

The algorithms studied in Section 4.1 rely on partial knowledge of the remaining processing time of each job *at each step*. A more realistic model for studying incomplete knowledge of job sizes is a relaxation of the non-clairvoyant model in which the total processing time of any job is known to within a constant factor *only at the time of the release of the job*. Section 4.2 analyzes $\lambda$-SPT, a variant of SPT, under this model.

## 4.1 Analysis of $\lambda$-SRPT

Recall that in each step, SRPT schedules a job that has the least remaining processing time. In each step of $\lambda$-SRPT, we schedule a job whose remaining processing time is within a $(1+\lambda)$ factor of that of the job with the least remaining processing time. More formally, at any step, the jobs are divided into groups as follows: a job $j$ is in group $i$ at time $t$ if $\rho_t(j) \in [(1+\lambda)^i, (1+\lambda)^{i+1})$. (Recall that $\rho_t(j)$ is the remaining processing time of $j$ at time $t$.) At any step $t$, $\lambda$-SRPT schedules a job from the smallest numbered group that is nonempty. (Note that $\lambda$-SRPT, with $\lambda \to 0$, is the same as SRPT.)

The two main results in this section concern the performance of $\lambda$-SRPT with respect to the average flow and average stretch metrics. We first show that $\lambda$-SRPT is $O(1)$-competitive with respect to average stretch, for constant $\lambda > 0$. With respect to average flow, however, we show that an adversarial mechanism of breaking ties among jobs in the same group leads to an $\Omega(\log \Delta)$-competitive ratio. (Recall that $\Delta$ is the ratio of the maximum processing time to the minimum processing time among all jobs in the given instance.) This is a surprising departure from the *true optimality* of SRPT for average flow. We finally present a specific tie-breaking mechanism and show that the resulting refinement of $\lambda$-SRPT achieves an $O(1)$ competitive ratio for average flow, and thus is simultaneously competitive for the average flow and stretch metrics.

Our analysis of $\lambda$-SRPT proceeds by comparing the state of the queue in $\lambda$-SRPT with the state of the queue in any other schedule, say $\mathcal{S}$. Let $S_t(i)$ (resp., $S'_t(i)$) denote the set of jobs in group $i$ at time $t$ in the $\lambda$-SRPT schedule (resp., $\mathcal{S}$). Let $N_t(i)$ (resp., $N'_t(i)$) denote the number of jobs in $S_t(i)$ (resp., $S'_t(i)$). For a given set of jobs, we refer to the sum of the remaining processing times of the jobs in the set at time $t$ as the *volume* of the set at time $t$. Let $V_t(i)$ (resp., $V'_t(i)$) denote the volume of jobs in $S_t(i)$ (resp., $S'_t(i)$) at time $t$. We note that the total flow of a schedule is simply the sum, over all time steps $t$, of the number of jobs in the queue at time $t$. In particular, the total flow of the $\lambda$-SRPT schedule equals $\sum_t \sum_{k \geq 0} N_t(k)$. Similar to total flow, the total stretch of a schedule can be calculated as the sum, over all time steps $t$, of the sum of the reciprocals of the processing times of the jobs in the queue at time $t$.

Before presenting the analysis in detail, we provide a brief overview. We first bound, in Lemma 4.1, the prefix sum of the group volumes in the $\lambda$-SRPT schedule in terms of the corresponding prefix sum in $\mathcal{S}$. This enables us to argue that the number of jobs in groups 0 through $i$ at any time $t$ in the $\lambda$-SRPT schedule is not much more than the corresponding number in any other schedule (Lemma 4.2). More precisely, the prefix sum of the group sizes in $\lambda$-SRPT differs from the corresponding prefix sum for any other schedule by only a constant number per group. This claim almost directly yields an upper bound on the competitive ratio of $\lambda$-SRPT with respect to average flow (see Theorem 3). For the average stretch analysis, we need to do more. By applying a simple algebraic inequality (Lemma A.1), one can show that a comparison of the prefix sums of group sizes leads to a similar comparison of the sums of the reciprocals of the remaining processing times. To establish the final result, we have to overcome two hurdles. First, stretch corresponds to the reciprocals of the processing times, not remaining processing times. Second, the preceding argument based on prefix sums does not account for the stretch contributions of a constant number of jobs per group (which are not included in the prefix sums calculated in Lemma 4.2). These hurdles are addressed in the final proof in Theorem 2.

**Lemma 4.1** *For all times $t$ and groups $i$, we have $\sum_{k\le i} V_t(k) \le \sum_{k\le i} V'_t(k) + (1+\lambda)^{i+1}$.*

**Proof:** The proof is by induction on $t$. For the induction base, we set $t=0$. Since the volume of jobs in the queue at time 0 is independent of the particular schedule, the desired claim holds trivially. We now consider the induction step $t>0$. We first note that the arrival of new jobs in the system contributes exactly the same amount to both sides of the desired inequality. We now consider the scheduling of the jobs. Algorithm $\lambda$-SRPT executes one unit from a job residing in the smallest group. That is, if $\sum_{k\le i} V_t(k) > 0$, it decreases by 1. Since at most one unit of any job may be executed in the schedule $\mathcal{S}$, it follows from the induction hypothesis that if $\sum_{k\le i} V_t(k) > 0$, then $\sum_{k\le i} V_{t+1}(k) \le \sum_{k\le i} V'_t(k) + (1+\lambda)^{i+1}$. If $\sum_{k\le i} V_t(k) = 0$ even after the addition of new jobs in the system, then it is possible that $\sum_{k\le i} V_{t+1}(k) > 0$ if a job in group $i+1$ at time $t$ gets executed in time $t$ and lands in group $i$. In this event, $\sum_{k\le i} V_{t+1}(k) \le (1+\lambda)^{i+1}$, which is at most $\sum_{k\le i} V'_{t+1}(k) + (1+\lambda)^{i+1}$. This completes the induction step and the proof of the lemma. ■

**Lemma 4.2** *For all $i$, there is a subset $T_t(i)$ of $S_t(i)$ and corresponding integer $M_t(i) = |T_t(i)|$ and volume $W_t(i)$, such that the following inequalities hold.*

$$M_t(i) \le \lceil 1+\lambda \rceil, \text{ for all } i \tag{2}$$
$$\sum_{k\le i} (N_t(k) - M_t(k)) \le (1+\lambda) \sum_{k\le i} N'_t(k) \tag{3}$$
$$\sum_{k\le i} W_t(k) \le 2(1+\lambda)^{i+1} \tag{4}$$

*Furthermore, the job that is processed by $\lambda$-SRPT at time $t$ is included in the set $\cup_{i\ge 0} T_t(i)$.*

**Proof:** We establish Equations 2 through 4 by induction on $i$. For convenience, we set $T_t(-1) = S_t(-1) = \emptyset$ and $N_t(-1) = M_t(-1) = N'_t(-1) = W_t(-1) = 0$. For the induction basis, we consider $i=-1$. The claim holds directly by the preceding settings.

We now establish the induction step. For the induction hypothesis, we assume the two equations to hold for all indices less than $i$. We now consider the equations for a given $i \ge 0$. If $S_t(i)$ is empty, then we set $T_t(i)$ to be empty, and the three equations for the induction step follow from the induction hypothesis. Otherwise, we let $T_t(i)$ be any subset of $S_t(i)$ that satisfies two conditions:

(a) $\sum_{k \le i} W_t(k)$ is at least $(1+\lambda)^{i+1}$; (b) if the group $i$ is the least numbered nonempty group and hence contains the job that will be processed by $\lambda$-SRPT, then we ensure that $T_t(0)$ contains the job. Condition (b) guarantees the last claim in the statement of the lemma. If condition (a) is not satisfied, then we set $T_t(i)$ to be $S_t(i)$.

Since each job in $T_t(i)$ has volume at least $(1+\lambda)^i$, Equation 2 holds. Since each job in $T_t(i)$ has volume less than $(1+\lambda)^{i+1}$, it follows that $\sum_{k \le i} W_t(k)$ has volume at most $2(1+\lambda)^{i+1}$, thus establishing Equation 4. For Equation 3, we consider two cases. If $T_t(i) = S_t(i)$, then the equation follows from Equation 3 of the induction hypothesis. Otherwise, we have $\sum_{k \le i} W_t(k) \ge (1+\lambda)^{i+1}$, and we derive

$$\begin{aligned}
\sum_{k \le i} (N_t(k) - M_t(k))(1+\lambda)^k &\le \left(\sum_{k \le i} V_t(k)\right) - \left(\sum_{k \le i} W_t(k)\right) \\
&\le \left(\sum_{k \le i} V'_t(k)\right) + (1+\lambda)^{i+1} - \left(\sum_{k \le i} W_t(k)\right) \\
&\le \sum_{k \le i} V'_t(k)
\end{aligned}$$

(For the first step, we note that the volume of the jobs in $S_t(k) - T_t(k)$, which equals $V_t(k) - W_t(k)$, is at least $\sum_{k \le i} (N_t(k) - M_t(k))(1+\lambda)^k$. The second step follows from Lemma 4.1.)

We thus have the following equation.

$$\sum_{k \le i} (N_t(k) - M_t(k))(1+\lambda)^k \le \sum_{k \le i} N'_t(k)(1+\lambda)^{k+1} \tag{5}$$

We invoke Lemma A.1, with $a_k = (N_t(k) - M_t(k))(1+\lambda)^k$, $b_k = N'_t(k)(1+\lambda)^{k+1}$, and $\alpha = 1/(1+\lambda)$ to obtain Equation 3. ∎

To bound the stretch contributions of the jobs in the sets $T_t(i)$, we consider the *birth groups* of jobs in the queue, which we define as follows. Let the *birth group* of a job be the group that the job resides in at the time of its release. Thus, the birth group of a job $j$ is $\lfloor \log_{1+\lambda} p(j_1) \rfloor$. (Note that the birth group is identical to the notion of group in Section 3.)

**Lemma 4.3** *There is at most one job in $\cup_{k \le i} S_t(k)$ that has birth group greater than $i$.*

**Proof:** The proof is by induction on time. For the base case, we note that the claim holds trivially at the start of the schedule. For the induction step, we consider the queue at the end of a time step $t \ge 0$. Consider the count on the number of jobs in groups 0 through $i$, for a given $i$. The jobs that are not processed do not change this count. Furthermore, any job that is released at time $t$ but not processed also does not change the count. Finally, the lone job $j$ that is processed in step $t$ changes the count only if the job moves from group $i+1$ to group $i$ as a result of the decrease in remaining processing time. In this case, $j$ is the only job in $\cup_{k \le i} S_{t+1}(k)$. This completes the proof of the desired claim. ∎

The following lemma characterizes the number of time steps that a job can be delayed by a job with a higher birth group.

**Lemma 4.4** *A job $j$ can be delayed by at most one job with higher birth group, and only at a time when the group of $j$ is identical to its birth group. Furthermore, the total amount of such delay for a job with birth group $g$ is at most $(1+\lambda)^{g+1}$.*

**Proof:** Consider a job $j$ that is released at time $t$, and arrives into its birth group $g$. By Lemma 4.3, there is at most one job (say $j'$) in groups 0 through $g$ at time $t$ that has its birth group greater than $g$. Suppose the job $j'$ exists. By the definition of $\lambda$-SRPT, no job in groups $g+1$ and higher will be executed until the completion of $j$. So the only job with birth group higher than $g$ that may delay $j$ is $j'$. Furthermore, once $j$ moves out of group $g$, it will never be delayed by $j'$. This is because at the instant $j$ moves out, either $j'$ is in group $g$ or $j'$ has already been completed; in either case, $j$ will not be delayed by $j'$ any more. The remaining processing time of $j'$ at the time of release of $j$ is at most $(1+\lambda)^{g+1}$. Therefore, the total time that $j'$ may delay $j$ is at most $(1+\lambda)^{g+1}$. ■

We are now ready to establish a constant-factor upper bound on the competitiveness of $\lambda$-SRPT with respect to average stretch.

**Theorem 2** *For any constant $\lambda > 0$, $\lambda$-SRPT is $O(1)$-competitive with respect to average stretch.*

**Proof:** Our analysis places a bound on the contribution to the total stretch by all the jobs in the queue of $\lambda$-SRPT at a given time step $t$ by comparing with the schedule that minimizes the total stretch contribution at time $t$. We note that there exists a schedule $\mathcal{S}$ such that $\mathcal{S}$ minimizes the total stretch contribution of jobs at time $t$ and there is no partially executed job in $\mathcal{S}$ at time $t$ [17]. This is because given any schedule that minimizes the total stretch contribution of jobs at time $t$ and does not satisfy the property of having no partially executed jobs at time $t$ can be converted into a schedule that satisfies the desired property by simply not processing jobs that remain incomplete at time $t$. Since the total stretch contribution at time $t$ is a function of only the processing times of the jobs in the queue at time $t$ and not the remaining processing times of the jobs in the queue at time $t$, the claim holds. In the remainder of the proof, we refer to this schedule as $\mathcal{S}$.

We first invoke Lemma 4.2 to obtain the subsets $T_t(i)$ and associated parameters $M_t(i)$ and $W_t(i)$. Consider the contribution to the total stretch by jobs in $S_t(i) - T_t(i)$, for all $i$. This contribution is at most $\sum_{i\geq 0}(N_t(i) - M_t(i))/(1+\lambda)^i$ since any job in $S_t(i) - T_t(i)$ has processing time at least $(1+\lambda)^i$. We obtain that the total stretch contribution at time $t$ in schedule $\mathcal{S}$ is at least $\sum_{i\geq 0} N'_t(i)/(1+\lambda)^{i+1}$. It thus follows from Equation 3 and Lemma A.1 that the total stretch due to jobs in $S_t(i) - T_t(i)$, taken over all $i$ and $t$, is at most $(1+\lambda)^2$ times the optimal stretch.

It remains to analyze the stretch contribution due to the jobs in $T_t(i)$, for all $i$. Let $Y$ denote the set of all of these jobs. If $Y$ is empty, then there is nothing to prove. If $|Y| = 1$, then by Lemma 4.2, the sole job in $Y$ is currently executed by $\lambda$-SRPT. Otherwise, there are at least two jobs in $Y$. Let $j_1$ denote the job in $Y$ that is being processed by $\lambda$-SRPT. Thus, $j_1$ is in the lowest numbered nonempty group. Rank the remaining jobs in $Y$ in nondecreasing order of their group number (breaking ties arbitrarily) and let $j_2$ be the first job in this list. Let $g_1$ and $g_2$ denote the current group of $j_1$ and $j_2$, respectively; thus $g_1 \leq g_2$. Let $g'_1$ and $g'_2$ denote the birth groups of $j_1$ and $j_2$, respectively. We consider two cases.

- Case 1: $g'_1 \leq g'_2$. In this case, we have $g_2 \geq g'_1$. To see this we note that if $g_2 < g'_1$, then we have two jobs $j_1$ and $j_2$ in the set $\cup_{k\leq g_2} S_t(k)$ that have larger birth groups than $g_2$, a contradiction to Lemma 4.3. It follows that the contribution to total stretch of all jobs in $Y$

at time $t$ is at most

$$\begin{aligned}
\frac{1}{p(j_1)} + \sum_{j \in Y, j \neq j_1} \frac{1}{\rho_t(j)} &\leq \frac{1}{p(j_1)} + \sum_{k \geq g_1'} \sum_{j \in T_t(k)} \frac{1}{\rho_t(j)} \\
&\leq \frac{1}{p(j_1)} + \sum_{k \geq g_1'} \sum_{j \in T_t(k)} \frac{\rho_t(j)}{(1+\lambda)^{2k}} \\
&= \frac{1}{p(j_1)} + \sum_{k \geq g_1'} \frac{W_t(k)}{(1+\lambda)^{2k}} \qquad (6)
\end{aligned}$$

(For the second step, we note that for $j \in T_t(k)$, $\rho_t(j) \geq (1+\lambda)^k$.)

- Case 2: $g_1' > g_2'$. In this case, we have a job $j_2$ being delayed by a job with a higher birth group. By Lemma 4.4, $g_2$ is the same as $g_2'$. Thus every job other than $j_1$ is in a group that is at least $g_2$. It follows that the contribution to total stretch of all jobs in $Y$ at time $t$ is at most

$$\begin{aligned}
\frac{1}{p(j_1)} + \sum_{j \in Y, j \neq j_1} \frac{1}{\rho_j(t)} &\leq \frac{1}{p(j_1)} + \sum_{k \geq g_2'} \sum_{j \in T_t(k)} \frac{1}{\rho_t(j)} \\
&\leq \frac{1}{p(j_1)} + \sum_{k \geq g_2'} \sum_{j \in T_t(k)} \frac{\rho_t(j)}{(1+\lambda)^{2k}} \\
&= \frac{1}{p(j_1)} + \sum_{k \geq g_2'} \frac{W_t(k)}{(1+\lambda)^{2k}} \qquad (7)
\end{aligned}$$

(For the second step, we note that for $j \in T_t(k)$, $\rho_t(j) \geq (1+\lambda)^k$.)

We now show that $\sum_{k \geq g} \frac{W_t(k)}{(1+\lambda)^{2k}}$ for any $g$ is at most $2(2+\lambda)/(1+\lambda)^g$. By Equation 4, we know that $\sum_{k \leq i} W_t(k) \leq 2(1+\lambda)^{i+1}$. The term $\sum_{k \geq g} \frac{W_t(k)}{(1+\lambda)^{2k}}$ is maximal when

$$W_t(g) = 2(1+\lambda)^{i+1},$$

and for $i > g$,

$$W_t(i) = 2(1+\lambda)^{i+1} - \sum_{g \leq k < i} W_t(k) = 2\lambda(1+\lambda)^i.$$

For a formal proof of the underlying claim, which relies on elementary algebraic manipulations, we refer the reader to [16, Lemma 4.2]. We thus obtain the following inequality:

$$\begin{aligned}
\sum_{k \geq g} \frac{W_t(k)}{(1+\lambda)^{2k}} &\leq \frac{2(1+\lambda)^{g+1}}{(1+\lambda)^{2g}} + \sum_{k > g} \frac{2\lambda(1+\lambda)^k}{(1+\lambda)^{2k}} \\
&= \frac{2}{(1+\lambda)^{g-1}} + \sum_{k > g} \frac{2\lambda}{(1+\lambda)^k} \\
&\leq \frac{2}{(1+\lambda)^{g-1}} + \frac{2}{(1+\lambda)^g} \\
&= \frac{2(2+\lambda)}{(1+\lambda)^g}.
\end{aligned}$$

We now substitute the above inequality in Equations 6 and 7 with $g = g_1'$ and $g = g_2'$, respectively. Since $p(j_1) \leq (1+\lambda)^{g_1'+1}$ and $p(j_2) \leq (1+\lambda)^{g_2'+1}$, we obtain that the total stretch contribution of jobs in $Y$ at time $t$ is at most the sum of two terms: (a) the reciprocal of the processing time of the job that is being processed; (b) $2(2+\lambda)(1+\lambda)$ times the reciprocal of the processing time of a job that is either being processed or is being delayed by a job with higher birth group. The total stretch contribution, over time, of the job currently being executed is at most $n$. By Lemma 4.4, the total stretch contribution of the jobs that are delayed by a job with higher birth group is at most $(1+\lambda)n$. It follows that the contribution of the jobs in $Y$ is at most $n + 2n(1+\lambda)^2(2+\lambda)$.

Let the optimal total stretch be $\overline{S^*}$. The total stretch achieved by $\lambda$-SRPT is at most $((1+\lambda)^2\overline{S^*} + 2n(1+\lambda)^2(2+\lambda) + n$. Since $\overline{S^*} \geq n$, we obtain an $O(1)$ competitive ratio for constant $\lambda > 0$. ■

We now show that $\lambda$-SRPT is $\Theta(\log \Delta)$-competitive with respect to average flow time.

**Theorem 3** *Let $\Delta$ be the ratio of the maximum processing time to the minimum processing time. Algorithm $\lambda$-SRPT is $O(\lambda \log_{1+\lambda} \Delta)$-competitive with respect to average flow. Furthermore, for $\lambda \leq 1$, there exists an instance for which the average flow of $\lambda$-SRPT is $\Omega(\lambda(\log_{1+\lambda}\Delta)/(1+\ln(1/\lambda)))$ times optimal.*

**Proof:** We consider the upper bound first. By Equations 2 and 3, we obtain the following inequality for all $i$ and $t$:

$$\sum_{k \leq i} N_t(k) \leq (1+\lambda)\sum_{k \leq i} N_t'(k) + \lceil \lambda + 1 \rceil (i+1). \tag{8}$$

Since the maximum number of groups is $\lceil \log_{1+\lambda} \Delta \rceil$, setting $i = \lceil \log_{1+\lambda} \Delta \rceil$ in Equation 8 yields that $\sum_{k \leq i} N_t(k)$ is at most $(1+\lambda)\sum_{k \leq i} N_t'(k)$ plus $O(\lambda \log_{1+\lambda} \Delta)$. Since $\lambda$-SRPT is work-conserving, it follows that if there is at least one job in the queue of $\lambda$-SRPT at time $t$, then there is at least one job in the queue of any other scheduling algorithm at time $t$. We therefore have $\sum_{k \leq i} N_t(k)$ is $O(\lambda \log_{1+\lambda} \Delta \sum_{k \leq i} N_t'(k))$ yielding the desired upper bound.

We now consider a lower bound for $\lambda$-SRPT. Let $\ell_i$ (resp., $h_i$) denote the lowest (resp., highest) possible size of a job in group $i$. We refer to $\ell_i$ and $h_i$ as the lower and upper bound, respectively, for group $i$. (Note that $\ell_i = \lceil (1+\lambda)^i \rceil$ and $h_i = \ell_{i+1} - 1$.) Fix nonnegative integer $i_0$. Consider an instance in which two jobs, one of size $h_{i_0}$, and another of size $\ell_{i_0}$, arrive at time 0. Let $i_1$ be chosen such that $h_{i_1}$ is the largest upper bound that is at most $h_{i_0} - \ell_{i_0}$. At time $h_{i_0} - h_{i_1}$, we introduce a job of size $\ell_{i_1}$. In general, at time $h_{i_0} - h_{i_s}$, $1 \leq s < k$, where $k$ is specified later, we introduce a job of size $\ell_{i_s}$, where $i_s$ is the largest numbered group whose upper bound $h_{i_s}$ is at most $h_{i_{s-1}} - \ell_{i_{s-1}}$. Finally, at each of the time steps $h_{i_0}, h_{i_0}+1, \ldots, h_{i_0}+m-1$, for an integer $m$ that is specified later, we introduce a unit size job.

We now consider the schedule computed by $\lambda$-SRPT for the above instance. Since $\lambda$-SRPT does not differentiate among jobs in the same group, it may start the job of size $h_{i_0}$ ahead of the job of size $\ell_{i_0}$ at time 0. Thus, at time $h_{i_0} - h_{i_1}$, exactly $h_{i_1}$ units of the job are remaining. At this time, a new job of size $\ell_{i_1}$ arrives. Again $\lambda$-SRPT may give preference to the larger job and continue the execution of the job with remaining processing time $h_{i_1}$. At time $h_{i_0} - h_{i_2}$, exactly $h_{i_2}$ time units are left, at which time a new job of size $\ell_{i_2}$ is introduced in the system. As the schedule continues, we note that there is a possible execution of $\lambda$-SRPT such that at time $h_{i_0}$, the initial job of size $h_{i_0}$ is completed; however, jobs of sizes $\ell_{i_0}, \ell_{i_1}, \ldots,$ are present and have not been processed at all. At this time, since a sequence of $m$ unit size jobs arrive, we obtain that the total flow of the schedule obtained is at least $h_{i_0} + \sum_{0 \leq s < k} \ell_{i_s} + mk$.

On the other hand, we can construct a schedule in which there are at most two jobs in the queue at any time. In particular, suppose we execute the jobs of size $\ell_{i_0}$, $\ell_{i_1}$, $\ldots$, $\ell_{i_{k-1}}$, in order, ahead of the job of size $h_{i_0}$. We find that our construction ensures that all the jobs of size $\ell_{i_0}, \ldots, \ell_{i_{s-1}}$ have been completed prior to the arrival of the job of size $\ell_{i_s}$. More significantly, when the $m$ jobs of unit size are being executed, only the largest job (of size $h_{i_0}$) is waiting in the system. The total flow for the schedule thus obtained is at most $h_{i_0} + 2(\sum_{0 \le s < k} \ell_{i_s} + m)$. We now set $i_0$, $k$, and $m$ such that $\ell_{i_{k-1}} = 1$, and $m = \Omega(h_{i_0} + \sum_{0 \le s < k} \ell_{i_s})$. We then obtain that the competitive ratio of $\lambda$-SRPT is $\Omega(k)$. If $\Delta$ is the ratio of the largest and smallest processing times, then $i_0 \ge (\log_{1+\lambda} \Delta) - 1$. In general, we have $i_s \ge i_{s-1} - \log_{1+\lambda}(1/\lambda) - 1$. Thus, we obtain

$$k \ge \frac{\log_{1+\lambda} \Delta}{1 + \log_{1+\lambda}(1/\lambda)} \ge \frac{\lambda \log_{1+\lambda} \Delta)}{2\ln(1+\lambda)(1 + \log_{1+\lambda}(1/\lambda))} \ge \Omega\left(\frac{\lambda \log_{1+\lambda} \Delta}{1 + \ln(1/\lambda)}\right).$$

(In the third step, we use the inequality $e^{\lambda/2} \le 1 + \lambda$ for $\lambda \le 1$. For the fourth step, we note that since $\lambda \le 1$, $2\ln(1+\lambda) \le 2 = O(1)$.) This completes the proof of the lower bound. ■

**A refinement of $\lambda$-SRPT.** We now show that a refinement of $\lambda$-SRPT achieves a constant factor competitive ratio for average flow as well; our analysis assumes that $\lambda$ is a positive integer. As $\lambda$-SRPT is presently defined, the algorithm does not differentiate among jobs in the same group. Since there is uncertainty in the remaining processing times, certainly we cannot use the remaining processing times. Nevertheless, we do know the processing times of the jobs up to a $(1+\lambda)$ factor; we make use of this information in our tie-breaking mechanism. Let $\tilde{S}_t(i)$ denote the set of jobs in $S_t(i)$ that have birth group $i$.

Consider a refinement of $\lambda$-SRPT which executes at each step a highest-priority job from the smallest numbered group that is nonempty, where the priority within a group is assigned as follows: the jobs in $\tilde{S}_t(i)$ have higher priority than those in $S_t(i) - \tilde{S}_t(i)$; within $\tilde{S}_t(i)$, a partially executed job is given the highest priority. We now analyze the refined $\lambda$-SRPT algorithm. We begin by noting that by the definition of the algorithm there is at most one job in $S_t(i) - \tilde{S}_t(i)$. We define a quantity $w_t(i)$, which measures the amount of work performed on the (lone) job in $S_t(i) - \tilde{S}_t(i)$ while it is in group $i$, if such a job exists. More precisely, if $S_t(i) - \tilde{S}_t(i) = \{j\}$, we set $w_t(i)$ to be $(1+\lambda)^{i+1} - 1 - \rho_j(t)$; otherwise, we set $w_t(i)$ to be $\lambda(1+\lambda)^i$.

Our analysis for refined $\lambda$-SRPT follows the same approach outlined earlier for $\lambda$-SRPT. The following lemma is a variant of Lemma 4.1.

**Lemma 4.5** *For all times $t$ and groups $i$, we have $w_t(i) + \sum_{k \le i} V_t(k) < \sum_{k \le i} V'_t(k) + (1+\lambda)^{i+1}$.*

**Proof:** The proof is by induction on $t$. The induction basis is trivial since $w_t(i) < (1+\lambda)^{i+1}$ and $V_t(k) = V'_t(k)$ for all $k$. Consider step $t$. We first note that the arrival of new jobs in the system contribute exactly the same amount to both the sides of the desired inequality. We now consider the scheduling of the jobs. The algorithm executes one unit from a job residing in the smallest nonempty group. Let $S_t(\ell)$ be the smallest nonempty group. We consider three cases. For $i > \ell$, the volume of the jobs in $\cup_{k \le i} S_t(k)$ decreases by 1, and $w_t(i)$ does not change. Thus, the left hand side (LHS) of the desired inequality decreases by 1. Moreover, the right hand side (RHS) decreases by at most 1. Therefore, the induction step holds in this case.

For $i = \ell$, if $\tilde{S}_t(i)$ is nonempty, then a job in $\tilde{S}_t(i)$ is executed; therefore, $V_t(i)$ decreases by 1 and $w_t(i)$ does not change, thus yielding the induction step since the LHS of the desired inequality decreases by 1, while the RHS can at most decrease by 1. If $\tilde{S}_t(i)$ is empty, then $w_{t+1}(i) + V_{t+1}(i)$ exactly equals $(1+\lambda)^{i+1} - 1$, thus establishing the inequality.

We finally consider the case $i < \ell$. If $i < \ell - 1$, then the inequality trivially holds since the volume of jobs in groups 0 through $i$ is 0 at time $t+1$. If $i = \ell - 1$, we only need to consider the case when a job moves from group $\ell$ to $\ell - 1$ at the end of step $t$. In this case, we have $w_{t+1}(\ell - 1) = 0$ and

$V_t(\ell - 1) < (1+\lambda)^\ell$, thus again establishing the desired inequality. This completes the induction step and the proof of the lemma. ■

**Corollary 4.5.1** *For all times $t$ and groups $i$, we have*

$$\sum_{k \le i} N_t(k)(1+\lambda)^k \quad < \quad \left( \sum_{k \le i} N_t'(k)(1+\lambda)^{k+1} \right) + (1+\lambda)^i \tag{9}$$

**Proof:** If there is no job in the queue of $\mathcal{S}$ at time $t$, then the same holds true for refined $\lambda$-SRPT since the latter is a work-conserving schedule, and the desired claim trivially holds. In the remainder we assume that there is at least one job in the queue of $\mathcal{S}$ at time $t$. Fix $i$ and $t$. We consider two cases. If $S_t(i) - \tilde{S}_t(i)$ is empty, then $w_t(i)$ equals $\lambda(1+\lambda)^i$, and Equation 9 follows from Lemma 4.5 by noting that each job in $S_t(i)$ (resp., $S_t'(i)$) has volume at least $(1+\lambda)^i$ and less than $(1+\lambda)^{i+1}$. We now consider the case when $S_t(i) - \tilde{S}_t(i)$ is nonempty and equals the singleton set $\{p\}$. In this case, we note that the sum of $w_t(i)$ and the volume of $S_t(i) - \tilde{S}_t(i)$ equals $(1+\lambda)^{i+1} - 1$. Thus, we have:

$$\begin{aligned}
\sum_{k \le i} N_t(k)(1+\lambda)^k &= N_t(i)(1+\lambda)^i + \sum_{k<i} N_t(k)(1+\lambda)^k \\
&= (1+\lambda)^i + (N_t(i) - 1)(1+\lambda)^i + \sum_{k<i} N_t(k)(1+\lambda)^k \\
&\le (1+\lambda)^i + \left( \sum_{k \le i} V_t(k) \right) - \rho_t(p) \\
&= (1+\lambda)^i + \left( \sum_{k \le i} V_t(k) \right) + w_t(i) - (1+\lambda)^{i+1} + 1 \\
&< (1+\lambda)^i + \left( \sum_{k \le i} V_t'(k) \right) + 1 \\
&\le (1+\lambda)^i + \left( \sum_{k \le i} N_t'(k)((1+\lambda)^{k+1} - 1) \right) + 1 \\
&\le (1+\lambda)^i + \sum_{k \le i} N_t'(k)(1+\lambda)^{k+1}.
\end{aligned}$$

(In the third step, we use the fact that every job in $S_t(k)$ has remaining processing time at least $(1+\lambda)^k$. In the fourth step, we use the equality $w_t(i) = (1+\lambda)^{i+1} - 1 - \rho_t(k)$. In the fifth step, we use Lemma 4.5. In the sixth step, we use the fact that every job in $S_t(k)$ has remaining processing time at most $(1+\lambda)^k$. Finally, in the last step, we invoke the condition that there is at least one job in $\mathcal{S}$.) ■

We now show that refined $\lambda$-SRPT is $O(1)$-competitive with respect to average flow. For all times $t$ and groups $i$, we establish

$$\sum_{k \le i} N_t(k) \quad \le \quad (1+\lambda) \sum_{k \le i} N_t'(k) + \lceil 1/\lambda \rceil. \tag{10}$$

The proof is by induction on $i$. The base case follows from Equation 9. For the induction hypothesis, assume that Equation 10 holds for all indices less than $\ell$, $\ell > 0$. We now establish the claim for index $\ell$. The proof is similar to the calculation in Lemma A.1. For $0 \le i < \ell$, we multiply both sides of Equation 10 by $\lambda(1+\lambda)^i$, add the equations together to obtain

$$\sum_{k<\ell}\left((1+\lambda)^\ell-(1+\lambda)^k\right)N_t(k) \quad\le\quad \sum_{k<\ell}\left((1+\lambda)^{\ell+1}-(1+\lambda)^{k+1}\right)N'_t(k)+\lceil 1/\lambda\rceil(1+\lambda)^\ell \tag{11}$$

Adding together Equation 9, with index $\ell$ substituted for $i$, and Equation 11, we obtain

$$(1+\lambda)\sum_{k\le\ell}N_t(k) \quad<\quad (1+\lambda)^{\ell+1}\sum_{k\le\ell}N'_t(k)+(1+\lambda)^\ell(\lceil 1/\lambda\rceil+1). \tag{12}$$

Dividing both sides of Equation 12 by $(1+\lambda)^\ell$ and noting that the LHS is an integer yields the desired inequality for the induction step. The $O(1)$ bound on the competitive ratio of $\lambda$-SRPT directly follows from Equation 10.

### 4.2 Analysis of $\lambda$-SPT

We now consider a different model for uncertainty in job sizes. In this model, when a job $j$ arrives the processing time $p(j)$ of the job is not known. Instead, what is known is the number $i$ such that $(1+\lambda)^i \le p(j) < (1+\lambda)^{i+1}$; as in Section 4.1, we refer to $i$ as the *birth group* of job $j$. In this section, we show that the following simple algorithm, $\lambda$-SPT, achieves an $O(1)$ competitive ratio with respect to average stretch: In each step, $\lambda$-SPT executes one unit of work on a job that has the smallest numbered birth group; ties are broken in favor of partially executed jobs.

The analysis of $\lambda$-SPT is similar to that of $\lambda$-SRPT. Let $S_t(i)$ (resp., $S'_t(i)$) denote the set of jobs in the queue at time $t$ that have birth group $i$. Let $N_t(i)$ (resp., $N'_t(i)$) denote the number of jobs with birth group $i$ at time $t$ in the $\lambda$-SPT schedule (resp., $\mathcal{S}$). Let $V_t(i)$ (resp., $V'_t(i)$) denote the total volume of jobs with birth group $i$ at time $t$ in the $\lambda$-SPT schedule (resp., $\mathcal{S}$). The following lemma shows that the prefix sum of birth group volumes in $\lambda$-SPT is at most that in any other schedule at any time.

**Lemma 4.6** *For all times $t$ and groups $i$, we have $\sum_{k\le i}V_t(k) \le \sum_{k\le i}V'_t(k)$.*

**Proof:** The proof follows from an easy induction on time $t$. The induction base, for $t=0$, is trivial. At any time $t$, the arrival of new jobs increases both the LHS and the RHS of the desired inequality by the same amounts. Since $\lambda$-SPT executes a job that has the smallest birth group and the birth group of a job never changes, the induction step follows from the induction hypothesis. ■

Since we give preference to partially executed jobs within a birth group and the birth group of a job never changes, it follows that in each birth group $i$, there is at most one partially executed job. Let $\tilde{S}_t(i)$ denote the subset of jobs in $S_t(i)$ that have not yet been executed; hence their remaining processing times equal their processing times. (Thus, $|\tilde{S}_t(i)| \ge N_t(i)-1$.) Let $\tilde{N}_t(i)$ denote the number of jobs in $\tilde{S}_t(i)$. The following claim, which bounds the stretch contribution of the jobs in $\sum_k \tilde{S}_t(k)$, follows from Lemma 4.6 and Lemma A.1.

**Lemma 4.7** *For all times and groups $i$, we have*

$$\sum_{k\le i}\frac{\tilde{N}_t(k)}{(1+\lambda)^k} \le \sum_{k\le i}\frac{N'_t(k)}{(1+\lambda)^{k-1}} \tag{13}$$

**Proof:** The set $\tilde{S}_t(k)$ consists of jobs with birth group $k$ that have been released but not processed at all until time $t$. Thus, the remaining processing time of each of these jobs is at least $(1+\lambda)^k$. Therefore, we obtain

$$\begin{aligned}\sum_{k\leq i}\tilde{N}_t(k)(1+\lambda)^k &\leq \sum_{k\leq i}V_t(k)\\ &\leq V'_t(k)\\ &\leq \sum_{k\leq i}N'_t(k)(1+\lambda)^{k+1}\end{aligned}$$

(The first step holds since $\tilde{N}_t(k) \leq |S_t(k)|$ and each job in $S_t(k)$ has processing time at least $(1+\lambda)^k$. The second step follows from Lemma 4.6. The last step holds since $N'_t(k) \leq |S'_t(k)|$ and each job in $S'_t(k)$ has processing time at most $(1+\lambda)^{k+1}$.)

We now invoke Lemma A.1 with $a_k = \tilde{N}_t(k)(1+\lambda)^k$, $b_k = N'_t(k)(1+\lambda)^{k+1}$, and $\alpha = 1/(1+\lambda)^2$ to obtain the desired claim. ■

We now analyze the stretch contribution of the jobs in $\lambda$-SPT's queue at time $t$. Our comparison is with a schedule $\mathcal{S}$ that minimizes the total stretch contribution of all jobs in the queue of $\mathcal{S}$ at time $t$. As we have argued earlier (in the analysis of $\lambda$-SRPT), we can assume without loss of generality that none of the jobs in the queue of $\mathcal{S}$ at time $t$ have been processed since their release. It follows from Lemma 4.7 that the stretch contribution of jobs in $\cup_{k\geq 0}\tilde{S}_t(k)$ is at most $(1+\lambda)^2$ times the stretch contribution of the jobs in $\mathcal{S}$ at time $t$. (In the preceding argument, we have used the facts that the size of each job in $\tilde{S}_t(i)$ is at least $(1+\lambda)^i$ while the size of each job in $S'_t(i)$ is at most $(1+\lambda)^{i+1}$.) Thus, added over time, the stretch contribution of the jobs in $\cup_{k\geq 0}\tilde{S}_t(k)$, is at most $(1+\lambda)^2\overline{S^*}$.

We now consider the stretch contribution of the partially executed jobs in $\lambda$-SPT at time $t$. We note that this contribution is at most $(1+\lambda)/\lambda$ times that of the partially executed job in the smallest numbered nonempty group; this job is the one that is executed at time $t$. Since the stretch contribution of the job that is being processed, when added over all times, is at most $n$, it follows that the total contribution of partially executed jobs is at most $(1+1/\lambda)n$. We thus obtain that the total stretch of $\lambda$-SPT is at most $(1+\lambda)^2\overline{S^*} + (1+1/\lambda)n$, which is $O(\overline{S^*})$ for constant $\lambda > 0$, since $\overline{S^*} \geq n$. For $\lambda = 1$, the total stretch is at most $4\overline{S^*} + 2n$, and thus has a competitive ratio of at most 6. The minimum bound on the competitive ratio is achieved when $\lambda = 0.565$; for this value of $\lambda$, the competitive ratio is at most 5.22.

While $\lambda$-SPT is near-optimal with respect to average stretch, its competitive ratio with respect to average flow is $\Omega(\log\Delta)$, as exemplified by the following instance. For concreteness, we fix $\lambda = 1$, and $\ell$ to be power of 2. Consider a sequence of $\ell - \log\ell + 1$ jobs of size $2^k$, $\log\ell \leq k \leq \ell$, that arrive as follows: the job of size $2^k$ arrives at time $(\sum_{k<i\leq\ell}2^i) - (\ell - k)$. Finally, $\ell - 1$ time units after the arrival of the job of size $\ell$, a sequence of a large number, $M$, of unit-size jobs arrive one after another at consecutive time steps. From the definition of 1-SPT, it follows that when the job of size $2^k$ arrives, the algorithm will preempt the job that is currently being executed and begin processing the job of size $2^k$. When the sequence of unit-size jobs start arriving, the queue of 1-SPT consists of $\ell - \log\ell + 1$ unfinished jobs, each having one unit of remaining processing time left. Each of the $\ell - \log\ell + 1$ jobs is made to wait until the entire sequence of unit-size jobs is completed. Consequently, the average flow achieved by 1-SPT is $\Omega(\ell M + 2^\ell)$. On the other hand, suppose we schedule the jobs in the following priority order: the first $\ell - \log\ell$ jobs in order of their release times, then the $M$ unit-sized jobs in order of their release times, and finally the job of size $\ell$. The total flow of the preceding schedule is $O(M + 2^\ell)$. By setting $M \geq 2^\ell$ and noting that $\ell = \Theta(\log\Delta)$, we establish the claimed lower bound on the competitive ratio of 1-SPT.

The primary reason for the failure of $\lambda$-SPT to perform well with respect to average flow is that a job with a large processing time and very small remaining processing time may be given lower priority than a job with shorter processing time that has just been released. Since the information about processing times is only accurate up to a factor of $1+\lambda$, the algorithm does not have a good estimate on the remaining processing time of the jobs being partially executed. In fact, the range for the estimate could be a constant fraction of the processing time. To see this, we consider $\lambda = 1$; when $2^i - 1$ units of a job with birth group $i$ is executed, the remaining processing time could be anywhere in the range $[1, 2^i]$.

In recent work [13], an interesting refinement of $\lambda$-SPT has been shown to achieve an $O(1)$ competitive ratio with respect to average flow. In this refinement, the algorithm tends to schedule jobs in the smallest nonempty birth group, yet maintains the constraint that the number of partially scheduled jobs is within a constant fraction of the total number of jobs in the queue.

## References


[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for scheduling to minimize average completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–43, October 1999.

[2] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[3] H. Bast. Dynamic scheduling with incomplete information. In *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 182–191, 1998.

[4] H. Bast. On scheduling parallel tasks at twilight. *Theory of Computing Systems*, 33:489–563, 2000.

[5] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, January 1998.

[6] M. Bender, S. Muthukrishnan, and R. Rajaraman. Improved algorithms for stretch scheduling. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 762–771, January 2002.

[7] C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 297–305, 2002.

[8] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 84–93, 2001.

[9] A. Goel, M. Henzinger, S. Plotkin, and E. Tardos. Scheduling data transfers in a network and the set scheduling problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 189–197, Atlanta, Georgia, May 1999.

[10] M. Harchol-Balter, M. Crovella, and C. Murta. Task assignment in a distributed server. *Journal of Parallel and Distributed Computing*, 59:204–228, 1999.

[11] B. Kalyanasundaram and K. Pruhs. Minimizing flow time nonclairvoyantly. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 345–352, 1997.

[12] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. In *Proceedings of the Annual European Symposium on Algorithms*, pages 290–301, 2000.

[13] T. Leighton, January 2003. Personal communication.

[14] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 110–119, May 1997.

[15] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130:17–47, 1994.

[16] S. Muthukrishnan and R. Rajaraman. An adversarial model for distributed dynamic load balancing. *Journal of Interconnection Networks*, 3:35–47, 2002.

[17] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. Gehrke. Scheduling to minimize average stretch. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 433–442, October 1999.

# A An algebraic inequality

**Lemma A.1** *Let $a_i, b_i$, $0 \le i < n$, denote two sequences of reals that satisfy the following inequality for $0 \le i < n$.*

$$\sum_{0 \le k \le i} a_k \le \sum_{0 \le k \le i} b_k \tag{14}$$

*Then, for any positive real $\alpha < 1$, we have, for $0 \le i < n$,*

$$\sum_{0 \le k \le i} a_k \alpha^k \le \sum_{0 \le k \le i} b_k \alpha^k \tag{15}$$

**Proof:** The proof is by induction on $i$. For the base case, we let $i = 0$. For this case, Equation 15 follows from Equation 14. For the induction step, we consider index $\ell > 0$. We now invoke the induction hypothesis, multiply Equation 15 by $(1-\alpha)\alpha^{-i}$, for each $i < \ell$, and add the resulting inequalities together to obtain

$$\sum_{0 \le i < \ell} (1-\alpha)\alpha^{-i} \sum_{0 \le k \le i} a_k \alpha^k \quad \le \quad \sum_{0 \le i < \ell} (1-\alpha)\alpha^{-i} \sum_{0 \le k \le i} b_k \alpha^k.$$

Rearranging the order of the summation on both sides of the inequality, we obtain

$$\sum_{0 \le k < \ell} (1-\alpha) a_k \alpha^k \sum_{k \le i < \ell} \alpha^{-i} \quad \le \quad \sum_{0 \le k < \ell} (1-\alpha) b_k \alpha^k \sum_{k \le i < \ell} \alpha^{-i},$$

leading to

$$\sum_{k < \ell} a_k(\alpha^{k-\ell} - 1) \quad \le \quad \sum_{k < \ell} b_k(\alpha^{k-\ell} - 1) \tag{16}$$

Adding Equation 14, with $i$ replaced by $\ell$, and Equation 16 and multiplying by $\alpha^\ell$ yields the desired inequality for the induction step. ■