Approximation Algorithms for Average Stretch Scheduling

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

July 8, 2002

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 ith 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 new algorithmic results for the average stretch metric in preemptive uniprocessor scheduling. Our first result is an offline polynomial-time approximation scheme for average stretch scheduling. This result improves on the previous best polynomial-time approximation factor of 2 achieved by the online algorithm SRPT, that always schedules a job with the shortest remaining processing time. 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 ith 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 long used *slowdown* or *stretch* to measure the effect of scheduling on an individual job. The stretch of a job is the ratio of its flow time to its processing time; that is, $\frac{C(i)-r(i)}{p(i)}$. Stretch is 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 new algorithmic results for the *average stretch* (or, equivalently, total stretch) metric in preemptive uniprocessor scheduling.

[1]Department of Computer Science, SUNY at Stony Brook, NY 11794–4400, Email: `bender@cs.sunysb.edu`.

[2]AT&T Shannon Laboratories, Florham Park, NJ 07932. Email: `muthu@research.att.com`.

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

- *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 $O(n^{\mathrm{poly}(1/\varepsilon)})$ time.

  The previous best polynomial-time approximation for average stretch was the online shortest remaining processing time algorithm (SRPT) that yields a 2-approximation [17]. Furthermore, there exists a constant $c > 1$ such that no online algorithm can achieve a competitive ratio of better than c [17]. In Section 2, we present the first PTAS for the average stretch problem.

A natural idea 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 2, 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 3, 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 λ-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 λ-SRPT is $O(1)$-competitive with respect to average stretch, it is $\Omega(\log \Delta)$-competitive with respect to average flow time, where Δ is the ratio of the largest job size to the smallest job size. We then present a suitable refinement of λ-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.

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 the reciprocal of flow time then optimizing average weighted completion time also optimizes average stretch. In terms of approximation, however, these two metrics are significantly different. Consequently, the recent PTAS for average weighted completion time [2] 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 [7], which takes time superpolynomial, but subexponential, in the input size. In a subsequent study [8] performed independently of our work, it has been shown that a quasi-PTAS is achievable for weighted flow time when Δ and the ratio of the maximum weight to minimum weight are both polynomially bounded. When applied to the special case of the average stretch metric, the techniques of [8] 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 [13, 16]. 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., [15]), broadcast scheduling (e.g., [14]), and network connection scheduling (e.g., [10]). Clearly, some of the questions we have raised are relevant in these scenarios, and deserve further attention.

2 PTAS for offline average stretch

In this section, we describe a polynomial-time approximation scheme (PTAS) for average stretch scheduling in uniprocessors. 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 assume without loss of generality that the size of the smallest job is 1.

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 in which there is a priority order among the jobs and in each step the job with the highest priority is scheduled. It is easy to show that every optimal schedule is a list schedule (Observation 1 of Section 2.1). We restrict our attention to another class of schedules that is well-suited for flow and stretch metrics. Let $\rho_t(j)$ denote the remaining processing time of job j at time t in the given schedule. We say that a schedule $\mathcal{S}$ is *natural* if it satisfies the following property: 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 (Observation 2 of Section 2.1).

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 $(1+\varepsilon)$. 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 2.1, we show that every optimal schedule is SRPT-friendly (Lemma 2.4).

We next partition the groups into *blocks*. 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 in this section 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 $b < 1/\varepsilon$ is specified below. Superblock i, for $i > 0$, consists of blocks $b+(i-1)/\varepsilon$ through $b+i/\varepsilon-1$. Thus, for any superblock i, jobs in every block of i, but for the largest numbered block, have size at most ε^2 times that of any job in superblock j for $j > i$. We select b such that the total number of jobs in the largest numbered blocks of all of the superblocks is at most $n\varepsilon$. We note that since there are $1/\varepsilon$ choices for b, one such choice exists.

Lemma 2.1 *There exists a $(1+2\varepsilon)$-approximate natural* SRPT*-friendly list schedule in which no job in superblock i delays any job in superblock j for $j < i$, for any i.*

Lemma 2.1 divides 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. The fact that the optimal schedule for a superblock is natural and SRPT-friendly enables us to 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 $1/\delta$ smaller jobs, for a δ chosen suitably small.

Lemma 2.2 *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 2.2 and an enumeration of schedules of interest establishes the following claim, that forms the final piece of the algorithm.

Lemma 2.3 *For any $\delta > 0$, there exists an $n^{O(m/\delta)}$-time algorithm to determine a $(1+\delta)^m$-approximate schedule for any constrained instance with m groups.*

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

Proof: Our algorithm consists of sequentially going through the superblocks, from the smallest to the largest job sizes, and applying the algorithm of Lemma 2.3 to each superblock (subject to the schedule already computed prior to the consideration of the superblock). A superblock consists of at most g/ε groups where $g = \log_{1+\varepsilon}(1/\varepsilon^2) \leq 2\log_2(1/\varepsilon)/\varepsilon$. Applying Lemma 2.3 with δ equal to $\varepsilon^3/\lg(1/\varepsilon^2)$ and m equal to $2\log_2(1/\varepsilon)/\varepsilon^2$, we obtain a $(1+2\varepsilon)$-approximate schedule for each superblock in $N^{O(1/(\varepsilon^5 \log^2(1/\varepsilon)))}$ time, for $\varepsilon > 0$ sufficiently small, where N is the number of jobs in the superblock.

By Lemma 2.1, we lose another factor of $(1+2\varepsilon)$ for restricting to schedules in which no job in superblock i delays any job in superblock $j > i$, for any i. Thus, the approximation factor is $(1+2\varepsilon)^2 \leq (1+5\varepsilon)$ for any positive constant ε sufficiently small. The running time is $n^{O(1/(\varepsilon^5 \log^2(1/\varepsilon)))}$. We thus have a PTAS for average stretch scheduling. ■

The remainder of this section is organized as follows. Section 2.1 establishes certain characteristics of optimal schedules. Sections 2.2, 2.3, and 2.4 establish Lemmas 2.1, 2.2, and Lemma 2.3, respectively.

2.1 Natural SRPT-friendly list schedules.

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

Observation 1 *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 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. ∎

Observation 2 *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. 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. Since the total flow time is minimized by SRPT scheduling, we have the following lemma.

Lemma 2.4 *For any constrained instance, every optimal schedule is* SRPT*-friendly.*

Proof: Let $\mathcal{S}$ be an optimal schedule that 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 $\mathcal{I}$ which consists of jobs in G and the set of allowable times is exactly the set of times during which these jobs are scheduled in $\mathcal{S}$. Since SRPT optimizes the total flow time in any constrained scheduling problem it follows that scheduling the jobs in G in SRPT order optimizes the rounded cost in $\mathcal{I}$. The desired claim follows. ∎

2.2 Eliminating delays of small jobs by larger jobs.

In this section, we prove Lemma 2.1. 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 in I and let s denote the superblock to which group m belongs. We first 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 $s-1$ or lower.

Sweep procedure. Let t be the earliest instant at which a job j in group m delays a job in superblock $s-1$ or lower. Let t' be the earliest time step after t at which there are no jobs from superblock $s-1$

or lower in the queue. We now claim that at every time step t_1 in the interval $[t, t']$, either j or some job in superblock $s-1$ or lower is executed. The proof is by contradiction. Let t_1 be the earliest time in $[t, t']$ at which a job, say $j_1 \neq j$ belonging to superblock s is executed. If $r(j_1) \leq t$, then since $\mathcal{S}$ is a natural schedule and j is executed at time t, it follows that $\rho_t(j_1) = p(j_1)$. Since t_1 is the earliest time at which j_1 is executed, $\rho_{t_1}(j_1) = p(j_1)$. Since $t_1 < t'$, there exists a job smaller than j_1 in the queue, and yet j_1 is executed. This yields a contradiction to the fact that $\mathcal{S}$ is natural.

We next observe that since $\mathcal{S}$ is a list schedule and j delays some job in superblock $s-1$ 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 $[t, t']$. We modify the schedule during 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 $[t, t']$. In the new schedule, the rounded stretch of the jobs in J is at most that in $\mathcal{S}$, while the completion time of j is at most t'. Let k_i denote the number of jobs in J from group i. Therefore, the increase in rounded cost is at most $\sum_{j' \in J} p(j')/p(j)$. Let $\mathcal{S}'$ denote the new schedule obtained.

Lemma 2.5 *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 π 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 π' 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. Suppose j delays a job from group $m-k$, for some $k > 0$, at time t in $\mathcal{S}$. Since $\mathcal{S}$ is a natural schedule, it follows that $\rho_t(j)$ is less than $(1+\varepsilon)^{m-k}$. 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. For this, we need to consider the executions during the interval $[t, t']$ only. Let j' be executed at time $t_1 \in [t, t']$ in $\mathcal{S}'$. We first consider the case $j' = j$. We note that $\rho_{t_1}(j) < (1+\varepsilon)^{m-k}$ and there are no jobs in groups $m-k$ or lower that are alive at time t_1 (by construction of $\mathcal{S}'$). Furthermore, all of the jobs belonging to groups $m-1$ through $m-k+1$ are full at time t_1 (i.e., have not been executed at all until t_1). This is because they are delayed by j at time t in $\mathcal{S}$ and not executed until after t' in $\mathcal{S}$, and $\mathcal{S}$ ia a natural schedule. We now consider the second case where $j' \in J$. The only smaller jobs that j' delays in $\mathcal{S}'$ belong to J, and by construction of $\mathcal{S}'$ it follows that all the smaller jobs that j' delays are full and $\rho_{t_1}(j')$ is less than the processing time of all of these jobs. This establishes that $\mathcal{S}'$ is a natural schedule. ■

We repeat the above sweep procedure with the new schedule which, by Lemma 2.5, is an SRPT-friendly natural list schedule, and continue until every job in group m delays no job in superblock $s-1$ or lower. The increase in cost due to this transformation is at most $\sum_{i \leq (b+(s-1)/\varepsilon)g} \frac{n_i}{(1+\varepsilon)^{m-i}}$, where n_i is the number of jobs in group i. We now describe how 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 (resp., s) denote the maximum group (resp., superblock) 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 $s-1$ or lower.

2. We fix the scheduling of the jobs in group m according to $\widehat{\mathcal{S}}$. Let $\mathcal{S}'$ denote the partial schedule of $\widehat{\mathcal{S}}$ induced by the jobs in groups $m-1$ and lower and let $\mathcal{I}'$ denote the associated constrained instance. We set $\mathcal{S}$ to $\mathcal{S}'$, $\mathcal{I}$ to $\mathcal{I}'$ and repeat step 1.

We now calculate an upper bound on the increase in cost due to the above procedure. If m_i is the number of jobs in block i, then the total increase in cost is at most

$$\begin{aligned} & \sum_s \left[m_{b+s/\varepsilon-1} \left(\sum_{i>0} \frac{1}{(1+\varepsilon)^i} \right) \right. \\ & \left. + \sum_{k \geq b+(s-1)/\varepsilon}^{b+s/\varepsilon-2} m_k \left(\sum_{i>g} \frac{1}{(1+\varepsilon)^i} \right) \right] \\ \leq \; & \sum_s \left[\frac{m_{b+s/\varepsilon-1}}{\varepsilon} + \sum_{k \geq b+(s-1)/\varepsilon}^{b+s/\varepsilon-2} m_k \varepsilon \right] \leq 2n\varepsilon. \end{aligned}$$

(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 $n\varepsilon^2$.) This completes the proof of Lemma 2.1.

2.3 Eliminating delays by many larger jobs.

In this section, we prove Lemma 2.2. Let $\mathcal{S}$ be the given natural list schedule for instance I. Let m be the largest group number in $\mathcal{S}$ and let j denote the lone job from group m in $\mathcal{S}$. 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 smaller jobs in $\mathcal{S}'$.

Let t denote the earliest time step at which j delays more than c smaller jobs. Let t' denote the earliest time step after t in which there are exactly c jobs smaller than j in the queue. Let J denote the set of c jobs smaller than j that are alive 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'. Clearly, $t_1 \geq t$ since no job smaller than j can be executed prior to t and then delayed by j at step t in $\mathcal{S}$ because $\mathcal{S}$ is a natural schedule. If j' is executed at time $t_1 \in [t, t']$, then at time t_1 there are greater than c jobs in the queue. Since j' delays the remaining c jobs at time t_1 and is still alive at time t', the remaining c jobs should also be alive at time t' because $\mathcal{S}$ is a list schedule. But there are only c jobs alive at time t', thus yielding a contradiction.

We modify the schedule $\mathcal{S}$ to derive schedule $\mathcal{S}'$ as follows. During the interval $[t, t']$, 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 $[t, t']$ in $\mathcal{S}$. It is easy to see that the increase in rounded cost is at most $(t' - t)/(1 + \varepsilon)^m$. Since there are at least c jobs in groups $m - 1$ or lower at each time step in $[t, t']$, each contributing as much to the rounded stretch as j, it follows that the increase in rounded stretch is by at most a factor of $1 + 1/c$.

We note that the resultant schedule obtained may not be a list schedule. We convert $\mathcal{S}'$ into a list schedule by assigning priorities to each job according to their completion times. The number of larger jobs that j delays cannot increase. Furthermore, by Observation 1, the rounded cost does not increase either. This completes the proof of the desired claim.

2.4 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 $n^{O(m/\delta)}$-time algorithm to determine a $(1 + \delta)^m$-approximate schedule for an instance with m groups, for any positive real δ.

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. We show, by induction, that our algorithm enumerates $n^{m/\delta}$ schedules for an instance with m groups, at least one of which is $(1+\delta)^m$-approximate. For the induction basis, we note that only an SRPT schedule is enumerated; therefore the claim holds trivially.

We now consider the induction step. Suppose we have an instance with m groups. Without loss of generality, we may assume that the m groups are numbered 0 through $m-1$. By Lemma 2.4, we know that there exists an SRPT-friendly natural list schedule that has optimal rounded cost. We use the property of SRPT-friendly schedules to split the given instance into a sequence of constrained instances, in each of which there is exactly one job from the largest numbered group.

Consider a given constrained scheduling instance. Let m denote the largest group number in the instance, let J_m denote the set of jobs in group m, and let k denote the number of these jobs. Let j_1, $j_2, \ldots, j_k$ denote the jobs in J_m in order of their release times. 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 m constrained scheduling instances $I_1, I_2, \ldots, I_k$ such that each instance contains exactly one job from J_m. In order to determine the order of completion of the jobs in J_m we use the fact that the optimal schedule is natural and SRPT-friendly. The order of completion of jobs in J_m is the same as that in which the jobs complete assuming that the jobs in J_m are scheduled in SRPT order and *every* job in J_m has lower priority than any job in groups 0 through m. We refer to such a schedule as a *groupwise* schedule.

The proof is by induction. We compare the groupwise schedule and the optimal (natural SRPT-friendly) schedule. Let t be the first step in which the two schedules differ with respect to the execution of the jobs in J_m. Thus, at time t, the groupwise schedule executes a job in groups 0 through $m-1$, while the optimal schedule executes a job $j_i \in J_m$. We now claim that j_i is the first job to complete in both the schedules. Since the remaining processing time of j_i at time t is the same in both schedules (by the choice of t), it follows from the fact that the optimal schedule is natural that this time is less than $(1+\varepsilon)^m$. This implies that $\rho_{j_i}(t)$ in both the schedules is less than the processing time of any job j_ℓ, $\ell > i$, in J_m. Since the jobs in J_m are processed in SRPT order, it follows that j_i is the first job to complete in both schedules. Furthermore, the remaining processing time of every other job in J_m at time t is at least $(1+\varepsilon)^m$ because otherwise there exists some other job j_ℓ, $\ell < i$, that has remaining processing time less than $(1+\varepsilon)^m$ at time $r(j_i)$, implying that j_ℓ should have higher priority than j_i in the SRPT order, thus yielding a contradiction. It thus follows that after time t, none of the jobs in $J_m - \{j_i\}$ are executed in either schedule until the first time step, say t'. when there is no other job in the queue. The time step t' is the time at which j_i completes in the groupwise schedule. We now replace the partial schedule during the interval $[t, t']$ in the optimal schedule by the corresponding partial schedule in the groupwise schedule and proceed.

The order of completion of the jobs in J_m can be used to split the optimal schedule into k parts. Let j_i be the first job in J_m to complete in the groupwise schedule and let t be its completion time. The first part begins at time 0 and ends at time t. We break the given instance I into two constrained instances I_1 and I' as follows. Let the set S_1 consist of j_i and all jobs in groups 0 through $m-1$ that arrive in the interval $[0, t]$. We note that all of these jobs complete both in the groupwise schedule and in the optimal schedule during the interval $[0, t]$. Furthermore, in both schedules, jobs j_0 through j_{i-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.

On repeating the above division, we obtain a series of instances I_1 through I_k such that in any instance I_i, we have exactly one job from group m. We now show how to obtain a $(1+\delta)^m$-approximate list schedule for any such instance.

Consider instance I_i. Let j_ℓ denote the lone job of group m that is in instance I_i. By Lemma 2.2, any natural list schedule for I_i can be converted into a list schedule in which j_ℓ 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_ℓ delays no more than $1/\delta$ smaller jobs.

We now compute a $(1+\delta)^m$-approximate schedule for the instance I_i. We stipulate that j_ℓ delays at most $1/\delta$ jobs in groups 0 through $m-1$. There are thus at most $\binom{n}{1/\delta} \leq n^{1/\delta}$ choices for the jobs that are delayed by j_ℓ. Each such choice, if feasible, determines exactly the time steps at which j_ℓ is executed. Once this is determined, we are left with two sequential constrained instances containing jobs from groups 0 through $m-1$ only. The first instance includes jobs that do not get delayed by j_ℓ 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_ℓ. All of the time steps prior to this completion can be marked as forbidden for the second instance. We assume, via induction, that for any constrained instance with n jobs and at most $s < m$ groups, we can enumerate $n^{s/\delta}$ schedules and determine a $(1+\delta)^s$-approximate schedule. Thus, schedules for these two instances can be computed in time $n^{(m-1)/\delta}$. Since we perform this computation for $O(n^{1/\delta})$ different choices, we obtain that the number of schedules enumerated is at most $n^{m/\delta}$. And the approximation factor is at most $(1+\delta)^m$. This completes the proof of Lemma 2.3.

3 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 actual remaining processing times. This algorithm, which we refer to as λ-SRPT, is analyzed in Section 3.1.

The algorithms studied in Section 3.1 rely on partial knowledge about 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 3.2 analyzes λ-SPT, a variant of SPT, under this model.

3.1 Analysis of λ-SRPT.

Recall that in each step, SRPT schedules a job that has the least remaining processing time. In each step of λ-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, λ-SRPT schedules a job from the smallest numbered group that is nonempty. (Note that 0-SRPT is the same as SRPT.)

The two main results in this section concern the performance of λ-SRPT with respect to the average flow and average stretch metrics. We first show that λ-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. 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 λ-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 λ-SRPT proceeds by comparing the state of the queue in λ-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 λ-SRPT schedule (resp., $\mathcal{S}$). Let $N_t(i)$ (resp., $N'_t(i)$) denote the number of jobs in $S_t(i)$ (resp., $S'_t(i)$) and let $V_t(i)$ (resp., $V'_t(i)$) denote the total volume of jobs in $S_t(i)$ (resp., $S'_t(i)$). 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 λ-SRPT schedule equals $\sum_t \sum_{j \geq 0} N_t(j)$. 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 reciprocal of the processing times of the jobs in the queue at time t.

The following lemma relates the prefix sum of the group volumes in the λ-SRPT schedule with that in $\mathcal{S}$.

Lemma 3.1 *For all times t and groups i, we have $\sum_{j\le i} V_t(j) \le \sum_{j\le i} V_t'(j) + (1+\lambda)^{i+1}$.*

Proof: The proof is by induction on t. The induction basis is trivial. 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. Algorithm λ-SRPT executes one unit from a job residing in the smallest group. That is, if $\sum_{j\le i} V_t(j) > 0$, 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 $\sum_{j\le i} V_t(j) > 0$, then $\sum_{j\le i} V_{t+1}(j) \le \sum_{j\le i} V_t'(j) + (1+\lambda)^{i+1}$. If $\sum_{j\le i} V_t(j) = 0$ even after the addition of new jobs in the system, then it is possible that $\sum_{j\le i} V_{t+1}(j) > 0$ if a job in group $i+1$ at time t gets executed in time t and lands in group i. In this event, $\sum_{j\le i} V_{t+1}(j) \le (1+\lambda)^{i+1}$, which is at most $\sum_{j\le i} V_{t+1}'(j) + (1+\lambda)^{i+1}$. This completes the induction step and the proof of the lemma. ■

Corollary 3.1.1 *For all times t and groups i, we have*

$$\sum_{j\le i} N_t(j)(1+\lambda)^j \le \left(\sum_{j\le i} N_t'(j+1)(1+\lambda)^{j+1}\right) + (1+\lambda)^{i+1}. \tag{1}$$

Lemma 3.1 can be used to establish a constant-factor upper bound on the competitiveness of λ-SRPT with respect to average stretch.

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

Proof Sketch: It can be shown from Equation 4 that there is a subset $T_t(i)$ of $S_t(i)$ and corresponding integer $m_t(i) = |T_t(i)|$ and volume $W_t(i)$, for all i, such that the following inequalities hold.

$$\sum_{j\le i} N_t(i) - m_t(i) \quad \le \quad (1+\lambda)\sum_{j\le i} N_t'(i) \tag{2}$$

$$\sum_{j\le i} W_t(i) \quad \le \quad (1+\lambda)^{i+1} \tag{3}$$

We first consider the contribution to the total stretch by jobs in $S_t(i) - T_t(i)$, for all i. This contribution is at most $\sum_{i\ge 0}(N_t(i) - m_t(i))/(1+\lambda)^i$ since any job in $S_t(i) - T_t(i)$ has processing time at least $(1+\lambda)^i$. We further claim 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. Letting $\mathcal{S} = \mathcal{S}'$, we obtain the total stretch contribution at time t in schedule $\mathcal{S}$ is at least $\sum_{i\ge 0} N_t'(i)/(1+\lambda)^{i+1}$. It thus follows from Equation 2 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. We subdivide these jobs into two subsets: one set X consisting of jobs that lie in a group that is least as high as the lowest numbered nonempty group in schedule $\mathcal{S}$, and the other set Y consisting of the remaining jobs. Using Equation 3, it can be shown that the total stretch contribution of the jobs in the X is at most $1+\lambda$ times that the total contribution in $\mathcal{S}$. We now consider the jobs in Y, assuming Y is nonempty. If $|Y| = 1$, then the sole job in Y is the one with the least remaining processing time and is currently executed by λ-SRPT. Otherwise, there are at least two jobs in Y. Let j_1 and j_2 be the jobs with the smallest and the second smallest remaining processing time in X. If $p(j_1) \leq p(j_2)$, then we can argue that the total contribution of the jobs in Y is at most $1+\lambda$ times the contribution of j_1. Otherwise, we can argue that the total contribution of the jobs in Y is at most the contribution of j_1 plus $(1+\lambda)$ times the stretch of j_2. In the latter case, j_2 is also the job with the smallest processing time. Since the total stretch contribution, over time, of the smallest job in the system and the job currently being executed, is at most $2n$, it follows that the contribution of the jobs in Y is at most $2n(1+\lambda)$.

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

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

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

Proof: We consider the upper bound first. We show that for all i and all t, we have the following inequality

$$\sum_{j \leq i} N_t(j) \leq (1+\lambda)\sum_{j \leq i} N'_t(j) + \lambda i + 1. \tag{4}$$

Since the maximum number of groups is $\lceil \log_{1+\lambda} \Delta \rceil$, setting $i = \lceil \log_{1+\lambda} \Delta \rceil$ in Equation 4 yields that $\sum_{j \leq i} N_t(j)$ is at most $(1+\lambda)\sum_{j \leq i} N'_t(j)$ plus $O(\lambda \log_{1+\lambda} \Delta)$. Since λ-SRPT is work-conserving, it follows that if there is at least one job in the queue of λ-SRPT at time t, there is at least one job in the queue of any other scheduling algorithm at time t. We therefore have $\sum_{j \leq i} N_t(j)$ is $O(\lambda \log_{1+\lambda} \Delta \sum_{j \leq i} N'_t(j))$ yielding the desired upper bound.

We derive Equation 4 by multiplying both sides of Equation 1 by $\lambda(1-\lambda)^{k-1-i}$, and then adding over i in $[0,k]$. We obtain the following inequality.

$$\sum_{i \leq k} (1+\lambda)^k N_t(i) \quad \leq \quad (1+\lambda)\left(\sum_{i \leq k} (1+\lambda)^k N'_t(i)\right) + (1+\lambda)^k (k\lambda + 1).$$

Dividing both sides by $(1+\lambda)^k$ yields Equation 4.

We now consider a lower bound for λ-SRPT. Let ℓ_i (resp., h_i) denote the lowest (resp., highest) possible size of a job in group i. We refer to ℓ_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 ℓ_{i_0}, arrive at time 0. Let h_{i_1} is the largest upper bound that is at least $h_{i_0} - \ell_{i_0}$. At time $h_{i_0} - h_{i_1}$, we introduce a job of size ℓ_{i_1}. In general, at time $h_{i_0} - h_{i_j}$, $1 \leq j < k$, where k is specified later, we introduce a job of size ℓ_{i_j}, where i_j is the largest numbered group whose upper bound h_{i_j} is at most $h_{i_{j-1}} - \ell_{i_{j-1}}$. 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 λ-SRPT for the above instance. Since λ-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 ℓ_{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 ℓ_{i_1} arrives. Again λ-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 ℓ_{i_2} is introduced in the system. As the schedule continues, we note that there is a possible execution of λ-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 $h_{i_0} + \sum_{0 \le j < k} \ell_{i_k} + mk$.

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 the particular choice of i_j, ensures that all the jobs of size $\ell_{i_0}, \ldots, \ell_{i_{j-1}}$ have been completed prior to the arrival of the job of size ℓ_{i_j}. 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 $h_{i_0} + 2(\sum_{0 \le j < k} \ell_{i_k} + m)$. We now set i_0, k, and m such that $\ell_{i_{k-1}} = 1$, and $m = \Omega(h_{i_0} + \sum_{0 \le j < k} \ell_{i_k})$. We then obtain that the competitive ratio of λ-SRPT is $\Omega(k)$. If Δ is the ratio of the largest and smallest processing times, then $i_0 \ge \log_{1+\lambda} \Delta - 1$. In general, we have $i_j \ge i_{j-1} - \log_{1+\lambda}(1/\lambda) - 1$, thus yielding $k \ge \lfloor \log_{(1+\lambda)/\lambda} \Delta \rfloor$, and the desired lower bound. ■

We now show that a refinement of λ-SRPT achieves a constant factor competitive ratio for average flow as well; our analysis assumes that λ is a positive integer. As λ-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 actual remaining processing times. Nevertheless, we do know the total 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 actual processing time in $[(1+\lambda)^i, (1+\lambda)^{i+1})$.

Consider a refinement of λ-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 λ-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 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 $\lambda(1+\lambda)^i$. The following lemma is a variant of Lemma 3.1.

Lemma 3.2 *For all times t and groups i, we have $w_t(i) + \sum_{j \le i} V_t(i) < \sum_{j \le i} V'_t(i) + (1+\lambda)^{i+1}$.*

Proof: The proof is by induction on t. The induction basis is trivial. 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 $\cup_{j \le i} S_t(j)$ 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 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 ℓ 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 3.2.1 *For all times t and groups i, we have*

$$\sum_{j \le i} N_t(j)(1+\lambda)^j \quad < \quad \left(\sum_{j \le i} N'_t(j+1)(1+\lambda)^{j+1}\right) + (1+\lambda)^i. \tag{5}$$

Proof: 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 5 follows from Lemma 3.2 by noting that each job in $S_t(i)$ (resp., $S'_t(i)$) has volume at least $(1+\lambda)^i$ and less than $(1+\lambda)^{i+1}$. In the case where $S_t(i) - \tilde{S}_t(i)$ is nonempty, we note that the sum of $w_t(i)$ and the volume of $S_t(i) - \tilde{S}_t(i)$ equals $(1+\lambda)^{i+1}$, and the desired inequality again follows from Lemma 3.2. ■

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

$$\sum_{j \le i} N_t(j) \quad \le \quad (1+\lambda) \sum_{j \le i} N'_t(j) + \lceil 1/\lambda \rceil. \tag{6}$$

The proof is by induction on i. The base case follows from Equation 5. For the induction hypothesis, assume that Equation 6 holds for all indices less than k, $k > 0$. We now establish the claim for index k. For $0 \le i < k$, we multiply both sides of Equation 6 by $\lambda(1+\lambda)^i$, add the equations together to obtain

$$\begin{aligned} & \sum_{j<k} \left((1+\lambda)^k - (1+\lambda)^j\right) N_t(j) \\ \le \; & \sum_{j<k} \left((1+\lambda)^{k+1} - (1+\lambda)^{j+1}\right) N'_t(j) \\ & + \lceil 1/\lambda \rceil (1+\lambda)^k. \end{aligned} \tag{7}$$

Adding together Equation 5, with index k substituted for i, and Equation 7, we obtain

$$(1+\lambda)^k \sum_{j \le k} N_t(j) \quad < \quad (1+\lambda)^{k+1} \sum_{j \le k} N'_t(j) + (1+\lambda)^k (\lceil 1/\lambda \rceil + 1). \tag{8}$$

Dividing both sides of Equation 8 by $(1+\lambda)^k$ 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 λ-SRPT directly follows from Equation 6.

3.2 Analysis of λ-SPT.

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

The analysis of λ-SPT is similar to that of λ-SRPT. Let $N_t(i)$ (resp., $N'_t(i)$) denote the number of jobs in group i at time t in the λ-SPT schedule (resp., $\mathcal{S}$). Let $V_t(i)$ (resp., $V'_t(i)$) denote the total volume of jobs in group i at time t in the λ-SPT schedule (resp., $\mathcal{S}$).

Lemma 3.3 *For all times t and groups i, we have $\sum_{j \le i} V_t(j) \le \sum_{j \le i} V'_t(j)$.* ■

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 λ-SPT executes a job that has the smallest group, the induction step follows from the induction hypothesis. ■

In each group i, there is at most one partially executed job. Let $\tilde{S}_t(i)$ denote the subset of complete jobs in $S_t(i)$. (Thus, $|\tilde{S}_t(i)| \geq N_t(i) - 1$.) Let $\tilde{N}_t(i)$ denote the number of jobs in $\tilde{S}_t(i)$. The following claim follows from Lemma 3.3.

Lemma 3.4 *For all times t and groups i, we have*

$$\sum_{j \leq i} \tilde{N}_t(j) \leq (1 + \lambda) \sum_{j \leq i} N'_t(j + 1).$$

■

From Lemma 3.4, it follows that we can map each job in $\tilde{S}_t(i)$ to distinct jobs in $\cup_{j \leq i} S'_t(j)$. Since the size of each job in $\tilde{S}_t(i)$ is at least $(1 + \lambda)^i$ and the size of each job in $\cup_{j \leq i} S'_t(j)$ is at most $(1 + \lambda)^{i+1}$, it follows that the stretch contribution of jobs in $\cup S'_t(j)$ is at most $(1 + \lambda)^2$ times that of jobs in $\mathcal{S}$ at time t, which is at most $\overline{S^*}$. We now consider the stretch contribution of the partially executed jobs in λ-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 active jobs, over all time t, is at most n, it follows that the total contribution of partially executed jobs, over all times, is at most $(1 + 1/\lambda)n$. We thus obtain that the total stretch of λ-SPT is at most $(1 + \lambda)^2 \overline{S^*} + (1 + 1/\lambda)n$, which is $O(\overline{S^*})$ for constant $\lambda > 0$. For $\lambda = 1$, the total stretch is at most $4\overline{S^*} + 2n$, and thus has a competitive ratio of at most 6 since $\overline{S^*} \geq n$. The minimum bound on the competitive ratio is achieved when $\lambda = 0.565$; for this value of λ, the competitive ratio is at most 5.22.

While λ-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 $\lambda = 1$. Consider a sequence of $\ell - \log \ell$ jobs of size 2^k, $\log \ell \leq k \leq \ell$, that arrive as follows: the job of size 2^k arrives at time $(\sum_{\ell \leq i < k} 2^i) - (\ell - k)$. Finally, $\ell - 1$ time units after the arrival of the job of size ℓ, a sequence of a large number, say M, of unit-size jobs arrive one after another at consecutive time steps. From the definition of 2-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 2-SPT consists of $\ell - \log \ell$ unfinished jobs, each having one unit of remaining processing time left. Each of the $\ell - \log \ell$ jobs is made to wait until the entire sequence of unit-size jobs is completed. Consecutively, the average flow achieved by 2-SPT is $\Omega(\ell M)$, while the average flow achieved by the optimal schedule is $O(M)$. Since $\ell = \Theta(\log \Delta)$, the claimed lower bound on the competitive ratio of 2-SPT follows.

The primary reason for the failure of λ-SPT to perform well with respect to average flow is that almost complete jobs that may have a large total processing time are abandoned in favor of jobs that have shorter processing time. 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 actual processing time. To see this, we consider $\lambda = 1$; when $2^i - 1$ units of a job of group i is executed, the remaining processing time could be anywhere in the range $[1, 2^i)$.

We conjecture that an $O(1)$ competitive ratio with respect to average flow is achieved by a particular refinement of λ-SPT, in which the algorithm tends to schedule jobs in the smallest 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] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. *Proceedings of MOBICOM*, 1998, pages 43-54.

[2] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Sartinutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999, pages 32-44.

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

[4] H. Bast. On scheduling parallel tasks at twilight. *Theory of Computing Systems*, 33(5/6):489–563, 2000.

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

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

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

[8] C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002.

[9] M. Crovella, R. Frangioso, and M. Harchol-Balter. Connection scheduling in web servers. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1999.

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

[11] M. Harchol-Balter, M. Crovella, and C. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204-228, 1999.

[12] N. Joshi, S. Kadaba, S. Patel, and G. Sundaram. Downlink scheduling in CDMA data networks. *Proceedings of MOBICOM*, 2000, pages 179-190.

[13] B. Kalyanasundaram and K. Pruhs. Minimizing flow time nonclairvoyantly, *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, 1997.

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

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

[16] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

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