Improved Algorithms for Stretch Scheduling

Michael A. Bender* S. Muthukrishnan† Rajmohan Rajaraman‡

Abstract

We study the basic problem of preemptive scheduling of an online stream of jobs on a single processor. The ith job arrives at time $r(i)$ and has processing time $p(i)$ that is known at the time of its arrival. If $C(i)$ is the completion time of job i, then the flow time is $C(i) - r(i)$ and stretch of a job is the ratio of its flow time to its processing time; that is, $\frac{C(i)-r(i)}{p(i)}$. Flow time considers the time a job is in the system regardless of the service it requested; the stretch measure relies on the intuition that a job that requested long service must be prepared to wait longer than the small jobs.

In this paper, we present improved algorithmic results in stretch scheduling. We first show that a simple online algorithm that takes amortized $O(1)$ time per job arrival is $O(\Delta^{1/2})$-competitive with respect to maximum stretch, where Δ is the ratio of the largest processing time to the smallest processing time. This is significantly more efficient than the best known online algorithm for this problem which takes $\Omega(n^2)$ per scheduling step (n is the number of jobs seen thus far). We next present a polynomial time approximation scheme for average stretch scheduling. The previous best polynomial-time algorithm is the shortest remaining processing time algorithm, which achieves a 2-approximation. Finally, we consider the impact of incomplete knowledge of job sizes on the average stretch performance of 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 a 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.

Until recently, little was known about stretch scheduling, even in the basic scheduling scenario described above. In this paper, we provide improved algorithmic results for stretch scheduling.

Overview of our results. Our work primarily focuses on two performance metrics: *maximum stretch* and *average stretch* (or equivalently, total stretch). Many of our performance guarantees depend on a parameter Δ of the scheduling instance, which we define as the ratio of the largest job size to the smallest job size. Without loss of generality, we normalize the sizes so that the smallest job has size 1 and the largest job has size Δ.

- *Online algorithm for maximum stretch*: We present a simple, fast online algorithm that takes $O(1)$ amortized time per job arrival and is $O(\Delta^{1/2})$-competitive with respect to maximum stretch.

  The previously best known online algorithm for maximum stretch is also $O(\Delta^{1/2})$-competitive, but the algorithm takes time $\Omega(n^2)$ per job arrival, where n is the number of jobs in the instance [5]. In particular, the algorithm maintains a history of all jobs seen thus far at any point during the execution of the online schedule, and hence is impractical. Furthermore, most recent papers that attempt to optimize the maximum stretch measure in practice have provided heuristics [1, 9, 12, 13] that do not have provable guarantees. In contrast, we are able to establish an $O(\Delta^{1/2})$ bound on the competitive ratio for a simple algorithm that rounds the job sizes into two classes, and schedules greedily. This algorithm is easy to implement; the algorithm and its analysis appear in Section 2. For comparison, the best known lower bound on the competitive ratio is $\Delta^{1/3}$ [5].
- *PTAS for average stretch*: We present a polynomial

*Department of Computer Science, SUNY at Stony Brook, NY 11794–4400, USA. Email: bender@cs.sunysb.edu. Supported in part by HRL Laboratories, the National Science Foundation, and Sandia National Laboratories.

†AT&T Shannon Laboratories, Florham Park, NJ 07932, USA. Email: muthu@research.att.com.

‡College of Computer Science, Northeastern University, Boston, MA 02115, USA. Email: rraj@ccs.neu.edu. Supported in part by NSF CAREER award NSF CCR–9983901.

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 [18]. Furthermore, there exists a constant $c > 1$ such that no online algorithm can achieve a competitive ratio of better than c [18]. In Section 3, we present the first PTAS for the average stretch problem. This result has also been independently obtained in recent work [8], which presents a quasi-polynomial approximation scheme for weighted flow time.

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 3, is a useful characterization of $(1+\varepsilon)$-approximate schedules that reduces the size of the search space of relevant schedules.

A technique that is common to both of the above results is that of rounding job sizes. Rounding of job sizes is a simple yet powerful tool that has been effectively used 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 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. 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.

Due to space constraints, we are unable to include many of the proofs in this paper. We refer the reader to the the full paper [6] for the omitted proofs.

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 subsequent work [8], 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.

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 [14, 17]. 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., [16]), broadcast scheduling (e.g., [15]), and network connection scheduling (e.g., [11]). Clearly, some of the questions we have raised are relevant in these scenarios, and deserve further attention.

2 Maximum stretch

In this section, we provide a new online algorithm for optimizing maximum stretch. Our algorithm achieves the best known competitive ratio of $O(\Delta^{1/2})$, where Δ is the ratio of the maximum processing time to the minimum processing time. Our algorithm only performs an amortized constant amount of computation per job arrival. Thus, the total running time of the algorithm over an entire scheduling instance is linear in the number of jobs in the instance. We contrast this property with the previously known $O(\Delta^{1/2})$-competitive ADJUSTIBLE-DEADLINE-EDF algorithm of [5]. In ADJUSTIBLE-DEADLINE-EDF, every time a new job arrives, the offline optimal maximum stretch of all jobs that have arrived thus far is recomputed, deadlines are assigned according to this maximum stretch value, and then the algorithm assigns jobs according to the earliest-deadline-first rule. Thus, the computational time at each step of the algorithm is $\Omega(n^2)$, where n is the number of jobs that have arrived so far.

We first analyze a simple scheduling policy that achieves an $O(\Delta^{1/2})$-competitive ratio and performs $O(1)$ work for every job arrival, under the assumption that Δ is known. Later in this section, we describe how to implement this policy without the knowledge of Δ. The resultant scheduling algorithm achieves the same competitive ratio of $O(\Delta^{1/2})$ and performs amortized $O(\log\log\log\Delta)$ work per job arrival. It can be shown that our algorithms generate schedules that have only $O(n)$ preemptions.

A natural algorithm to optimize maximum stretch is a greedy policy in which we always execute the job in the system that has the current largest stretch. An implementation of the greedy policy needs to maintain the current stretch values of the jobs, which change at different rates depending on the processing times of the jobs. In particular, the scheduler has to determine when one job's stretch becomes larger than the stretch of the job being currently executed.

We now describe a simplified variant of greedy, GREEDY-WITH-ROUNDING, in which the stretches are approximately maintained. For each job j, we define the *pseudostretch* $\widehat{s_j(t)}$ of j at time t as follows:

$$\widehat{s_j(t)} = \begin{cases} \frac{t-r(j)}{\Delta^{1/2}}, & \text{if } 1 \le p(j) \le \Delta^{1/2}, \\ \frac{t-r(j)}{\Delta}, & \text{if } \Delta^{1/2} < p(j) \le \Delta. \end{cases}$$

At each time t, algorithm GREEDY-WITH-ROUNDING schedules the job that has the largest pseudostretch at time t. One desirable property of the algorithm is that all jobs with size at most $\Delta^{1/2}$ (henceforth referred to as *short* jobs) are scheduled first-in first-out (FIFO) and all jobs of size greater than $\Delta^{1/2}$ (henceforth referred to as *long* jobs) are scheduled FIFO.

THEOREM 1. *Algorithm* GREEDY-WITH-ROUNDING *is* $\Theta(\sqrt{\Delta})$*-competitive for max-stretch.*

In order to establish Theorem 1, we need to understand the structure of the optimal schedule OPT. Let $S^* \ge 1$ denote the stretch used by OPT. This schedule can be computed as follows: (1) To each job j, assign the deadline $d(j) = r(j) + S^* \cdot p(j)$. (2) Execute the jobs Earliest Deadline First (EDF), in which the scheduler always executes the job in the system that has the earliest deadline. For any scheduling problem in which jobs have arrival times, processing times, and deadlines, the (preemptive) EDF scheduling policy will find a feasible schedule (i.e., all jobs finish before their deadlines) if one exists. Consequently, the value of S^* can be approximated by binary searching on stretch sizes. It can be computed exactly by observing that even though there are an unbounded number of assignments for deadlines, there are only $O(n^2)$ different relative priorities of the jobs, and therefore there are only $O(n^2)$ different EDF schedules of the jobs.

In our analysis, we are going to consider a modification of OPT, PSEUDO-OPT, which we use to compare with our online algorithm. Before presenting this schedule, we introduce some additional definitions. Above we have defined the pseudostretch of a job at each time step. We define the pseudostretch $\widehat{s(j)}$ of a job j in a schedule as simply the pseudostretch of j at its completion time. We now define PSEUDO-OPT as a schedule that minimizes the maxium pseudostretch of a job. We compute PSEUDO-OPT in the same manner as OPT; as before we search over the possible values of maxstretch, assign deadlines accordingly, and determine whether the schedule is feasible. Let $\widehat{S^*}$ denote the maximum pseudostretch of PSEUDO-OPT. Let $\widehat{d}(j)$ denote the deadline of job j in PSEUDO-OPT. We say that job j is a *pseudo-predecessor* (resp., *pseudo-successor*) of job k if j has an earlier deadline than k according to PSEUDO-OPT.

LEMMA 2.1. *Let S^* and $\widehat{S^*}$ denote the maximum stretch and the maximum pseudostretch, respectively, for a given instance. Then, we have $\widehat{S^*} \le S^*$. Furthermore, the maximum stretch achieved by* PSEUDO-OPT *for the instance is at most $\Delta^{1/2}S^*$.* ■

The heart of the proof of Theorem 1 is a competitive analysis of GREEDY-WITH-ROUNDING with respect to pseudomaxstretch, as stated in the following lemma. Theorem 1 then follows from Lemmas 2.1 and 2.2.

LEMMA 2.2. GREEDY-WITH-ROUNDING *is $O(1)$-competitive with respect to maximum pseudostretch.*

We begin the proof of Lemma 2.2 by first describing the circumstances that cause a job j to have a pseudostretch larger than $\widehat{S^*}$. Suppose that a job completes x units after its deadline. This happens because for at least x units of time, PSEUDO-OPT chooses to execute jobs that are successors of j, while predecessors of j are in the system. If k is one of these successor jobs then we say that job k *delays* job j.

We need not worry about the delay incurred by jobs in the large queue:

LEMMA 2.3. *A job is never delayed while it is in the large queue.* ■

We now consider the delay incurred by small jobs.

LEMMA 2.4. *If small job j is delayed by a large successor, then j is delayed only before deadline $\widehat{d}(j)$.* ■

For each small job j we identify a critical time, in which the pseudostretch in the system is not too large.

DEFINITION 2.1. *For each small job j, let the* **critical time** *τ_j denote the earliest time beyond which no (large) job delays j. That is, τ is the earliest time after which no large job is executed while j or predecessors of j are in the system.*

Lemma 2.3 and 2.4 can be used to establish the following claim.

LEMMA 2.5. *In* GREEDY-WITH-ROUNDING, *at time τ_j the pseudostretch of all jobs in the system is at most $\widehat{S^*}$.* ■

Therefore, whenever GREEDY-WITH-ROUNDING makes a scheduling decision that disagrees with PSEUDO-OPT and risks delaying job j, the system has the following structure. Before time τ_j the scheduler has executed some work on successor jobs that it should have executed after time τ_j. As a result there are some small jobs in the system that should have been executed or partially executed by time τ_j. We consider the work from these jobs that is completed before τ_j in PSEUDO-OPT but is completed after time τ_j in GREEDY-WITH-ROUNDING. These small jobs can be viewed as "being pushed forward" by the work on the longer jobs, causing them to "jut out" beyond time τ_j. At time τ_j they all have stretch at most $\widehat{S^*}$. In the next lemma we bound the total amount of work on such small jobs.

LEMMA 2.6. *Consider all (small) predecessors k of j, such that $r(k) < \tau_j$ and k is not completed by time τ. Then the total amount of work on these jobs is at most $2\Delta^{1/2}\widehat{S^*}$.* ■

Lemma 2.6 shows that there can be at most $O(\Delta^{1/2}\widehat{S^*})$ work from predecessor jobs of j, such that the work is executed before τ_j in PSEUDO-OPT and after τ_j in GREEDY-WITH-ROUNDING. This additional predecessor work in the system can only delay j by $O(\Delta^{1/2}\widehat{S^*})$ units and conseqeuntly yields a competitive ratio of $O(1)$ with respect to maximum pseudostretch.

The analysis of of GREEDY-WITH-ROUNDING is tight to within constant factors. Consider the following problem instance: Large jobs 1 and 2 arrive at time 0 and have size Δ. At time $2\Delta - \Delta^{1/2}$ a medium-sized job 3 arrives, which has size $\Delta^{1/2}$. Then between time 2Δ and 3Δ a stream of small jobs of length 1 bombard the system one job per time unit. The optimal schedule OPT would defer one of the large size Δ jobs to the end obtaining a maxstretch of 3. However, GREEDY-WITH-ROUNDING and greedy without rounding will finish both size Δ jobs by time 2Δ. The consequence of this scheduling mistake is that the $\Delta^{1/2}$-size job or some of the unit-sized jobs have a stretch of $O(\Delta^{1/2})$, which implies a competitive ratio of $O(\Delta^{1/2})$.

THEOREM 2. *When* GREEDY-WITH-ROUNDING *is provided with the value of Δ it runs in linear time, executing only $O(1)$ operations per job arrival.*

The bound on the running time is established because the jobs can be maintained in two FIFO queues, one for small jobs and one for large jobs. Thus, the scheduler can make its scheduling decisions by merely examining the first job in each queue.

The analysis of GREEDY-WITH-ROUNDING and Theorem 1 still holds when Δ is unknown and must be estimated as the algorithm progresses. We assume that the algorithm is provided with the size of the smallest permissable job, that is, the scheduler knows $\min_j p(j) = 1$ Thus, as the computation progresses, the scheduler must estimate the size of the largest job.

At time t, let $p(j)$ be the size of the maximum job so far, that is $p(j) = \min_i$ such that $r(i) < t$. Then, the algoriithm determines that any job k such that $p(k) < p(j)^{1/2}$ is small and any job k such that $p(k) \geq p(j)^{1/2}$ is large. As larger jobs arrive, jobs from the small queue may migrate to the large queue as the definition of small and large changes.

It can be established that the previous lemmas still hold in this case, except for the analysis of the running time. The difficulty with the running time is that now jobs migrate from the large group to the small group and so queues may need to be divided and combined.

Our solution is to maintain a $\min(\log \Delta, n)$ different queues, which group jobs by powers of two. Queues are called large or small depending on whether they hold large jobs or small. In order for GREEDY-WITH-ROUNDING to be efficient, the algorithm must find the earliest element in all the small queues and the earliest element in the large queues. Thus, we maintain a priority queue of FIFO queues. Each FIFO queue is inserted into the priority queue, with the arrival time of the first job in the queue as the key. Each time a job is executed, the FIFO queue is removed and reinserted with a different key. Furthermore, the FIFO queues may be inserted from the large priority queue and inserted in the small. Thus, just with a simple heap, we obtain a $O(n \log\log \Delta)$ running time. If we use more efficient data structures we can obtain faster results. For example, using the priority queue of Thorup [19] which assumes the word model of computation [10], we obtain the near-linear running time of $O(n \log\log\log \Delta)$.

3 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 3.1). 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 3.1).

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 3.1, we show that every optimal schedule is SRPT-friendly (Lemma 3.4).

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 3.1. *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 3.1 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 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 $n^{O(m/\delta)}$-time algorithm to determine a $(1 + \delta)^m$-approximate schedule for any constrained instance with m groups.*

THEOREM 3. *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 3.3 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 3.3 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 3.1, 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 3.1 establishes certain characteristics of optimal schedules. Sections 3.2 and 3.3 establish Lemmas 3.1 and 3.2, respectively. Due to space constraints, we refer the reader to the full paper [6] for the proof of Lemma 3.3.

3.1 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.* ∎

OBSERVATION 2. *For any constrained scheduling instance, every optimal schedule is a natural 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 3.4. *For any constrained instance, every optimal schedule is SRPT-friendly.* ∎

3.2 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 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 3.5. *The schedule $\mathcal{S}'$ is an* SRPT*-friendly natural list schedule.* ∎

We repeat the above sweep procedure with the new schedule which, by Lemma 3.5, 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 $\sum_{i \le (b+(s-1)/\varepsilon)g} \frac{n_i}{(1+\varepsilon)^{m-i}}$, 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

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

(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 3.1.

3.3 Eliminating delays by many larger jobs.

In this section, we prove Lemma 3.2. 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, $t_1 \ge t$ 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.

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 actual remaining processing times. This algorithm, which we refer to as λ-SRPT, is analyzed in Section 4.1.

The algorithms studied in Section 4.1 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 4.2 analyzes λ-SPT, a variant of SPT, under this model.

4.1 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 $\sum_t \sum_{j \ge 0} N_t(j)$. 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 4.1. *For all times t and groups i, we have* $\sum_{j \le i} V_t(j) \le \sum_{j \le i} V'_t(j) + (1+\lambda)^{i+1}$. ■

Lemma 4.1 can be used to establish a constant-factor upper bound on the competitiveness of λ-SRPT with respect to average stretch.

THEOREM 4. *For any constant $\lambda > 0$, λ-SRPT is $O(1)$-competitive with respect to average stretch.*

We now present a simple lower-bound argument that λ-SRPT is $\Omega(\log \Delta)$-competitive with respect to average flow time. For concreteness, let λ be 2. At time 0, we introduce two jobs, one of size $2^{\ell+1} - 1$ and the other of size 2^ℓ, for some integer ℓ. Since both the jobs are in group ℓ, 2-SRPT may schedule the large job. When the large job enters group $\ell - 1$, we introduce a new job of size $2^{\ell-1}$. Again, since 2-SRPT does not distinguish between the two jobs, it may continue to process the first job. If we proceed in this manner, we note that in $2^{\ell+1}$ time steps, 2-SRPT completes one job and has at least $\ell - 1$ jobs in its queue, while an alternative schedule completes all but the largest job. Now, if we introduce a long sequence of say M unit-size jobs, all of which get priority in both schedules, we obtain that the total flow of 2-SRPT is $\Omega(\ell M)$, while that of the alternative schedule is $O(M)$, for M sufficiently large. Since $\ell = \Theta(\log \Delta)$, the desired lower bound on 2-SRPT holds. A tight analysis of λ-SRPT with respect to average flow is given in the full paper [6].

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 4.1.

LEMMA 4.2. *For all times t and groups i, we have*

$$\sum_{j \le i} N_t(j)(1+\lambda)^j \quad < \quad \left(\sum_{j \le i} N_t'(j+1)(1+\lambda)^{j+1} \right) \tag{4.1}$$
$$+(1+\lambda)^i.$$

■

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

$$\sum_{j \le i} N_t(j) \quad \le \quad (1+\lambda) \sum_{j \le i} N_t'(j) + \lceil 1/\lambda \rceil. \tag{4.2}$$

The proof is by induction on i. The base case follows from Equation 4.1. For the induction hypothesis, assume that Equation 4.2 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 4.2 by $\lambda(1+\lambda)^i$, add the equations together to obtain

$$\begin{aligned} &\sum_{j<k} \left((1+\lambda)^k - (1+\lambda)^j\right) N_t(j) \\ \le \quad &\sum_{j<k} \left((1+\lambda)^{k+1} - (1+\lambda)^{j+1}\right) N_t'(j) \\ &+ \lceil 1/\lambda \rceil (1+\lambda)^k. \end{aligned} \tag{4.3}$$

Adding together Equation 4.1, with index k substituted for i, and Equation 4.3, we obtain

$$(1+\lambda)^k \sum_{j \le k} N_t(j) \quad < \quad (1+\lambda)^{k+1} \sum_{j \le k} N_t'(j) \tag{4.4}$$
$$+(1+\lambda)^k(\lceil 1/\lambda \rceil + 1).$$

Dividing both sides of Equation 4.4 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 4.2.

4.2 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 4.3. *For all times t and groups i, we have $\sum_{j \le i} V_t(j) \le \sum_{j \le i} V_t'(j)$.* ■

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, $|\tilde{S}_t(i)| \ge N_t(i) - 1$.) Let $\tilde{N}_t(i)$ denote the number of jobs in $\tilde{S}_t(i)$. The following claim follows from Lemma 4.3.

LEMMA 4.4. *For all times t and groups i, we have*

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

■

From Lemma 4.4, it follows that we can map each job in $\tilde{S}_t(i)$ to distinct jobs in $\cup_{j \le i} S_t'(j)$. Since the size of each job in $\tilde{S}_t(i)$ is at least $(1+\lambda)^i$ and the size of each job in $\cup_{j \le i} S_t'(j)$ 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 $\overline{S^*} \ge n$. 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, $\log \ell \le k \le \ell$, that arrive as follows: the job of size 2^k arrives at time $(\sum_{\ell \le i < k} 2^i) - (\ell - k)$. 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. It is easy to see that the average flow achieved by 2-SPT is $\Omega(\ell M)$, while the average flow achieved by an 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. Technical report NU-CCS-01-07, Northeastern University, October 2001.

[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. Manuscript, April 2001.

[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] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer System and Sciences*, 47(3):424–436, 1993.

[11] 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.

[12] 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.

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

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

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

[16] 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.

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

[18] 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.

[19] M. Thorup. On RAM priority queues. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1996, pages 59–67.