

# Minimizing Weighted Flow Time

NIKHIL BANSAL

IBM T.J. Watson Research

and

KEDAR DHAMDHERE

Google Inc.

---

We consider the problem of minimizing the total weighted flow time on a single machine with preemptions. We give an online algorithm that is  $O(k)$  competitive for  $k$  weight classes. This implies an  $O(\log W)$  competitive algorithm, where  $W$  is the maximum to minimum ratio of weights. This algorithm also implies an  $O(\log n + \log P)$  approximation ratio for the problem, where  $P$  is ratio of the maximum to minimum job size and  $n$  is the number of jobs. We also consider the non-clairvoyant setting where the size of a job is unknown upon its arrival and becomes known to the scheduler only when the job meets its service requirement. We consider the resource augmentation model, and give a  $(1 + \epsilon)$ -speed,  $(1 + 1/\epsilon)$ -competitive online algorithm.

Categories and Subject Descriptors: F.2.2 [Nonnumerical Algorithms and Problems]: Sequencing and Scheduling

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Scheduling, Online Algorithms, Response Time, Non-Clairvoyant Scheduling

---

## 1. INTRODUCTION

We consider the classic problem of scheduling dynamically arriving jobs, on a single machine with preemptions to minimize the total weighted flow time. The flow time of a job (also known as the response time) is the total time it spends in the system, and hence is the sum of time spent waiting and its processing time. When jobs have different degrees of importance, indicated by the weight of the job, the total (average) weighted flow time is one of the simplest and natural metrics that measures the quality of service received by the jobs.

For the unweighted case, a well known result [Smith 1956] is that the online algorithm that at any time schedules the job with the shortest remaining processing time (SRPT) minimizes the total flow time on a single machine. The weighted case however, was shown to be strongly NP-hard by Lenstra et al. [Lenstra et al.

---

Author's address: N. Bansal, IBM TJ Watson Research Labs, PO Box 218, Yorktown, NY 10598. bansal@gmail.com.

K. Dhamdhere, Google Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043. kedar.dhamdhere@gmail.com.

This work was done while the authors were graduate students at Carnegie Mellon University.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 \$5.00

1977]. It is also known that preemption is critical to obtain non-trivial results for flow time. In particular for the non-preemptive setting, Kellerer et al. [Kellerer et al. 1996] showed that unless  $P=NP$ , any polynomial time algorithm must have an approximation ratio of  $\Omega(n^{1/2-\epsilon})$ . Since the flow time of a job is equal to its completion time minus its release time, any schedule that is optimal for weighted completion time is also optimal for weighted flow time. In the last few years the weighted completion problem received significant attention and this has culminated in polynomial time approximation schemes for various versions of the problem [Hall et al. 1997; Afrati et al. 1999; Chekuri et al. 1997]. However, these results do not imply good approximations for weighted flow time. The flow time metric is much more sensitive to small changes in schedule as compared with the completion time metric and hence none of the known algorithmic techniques for weighted completion time seem to give good results for weighted flow time.

The progress on weighted flow time has been relatively recent. Chekuri, Khanna and Zhu [Chekuri et al. 2001] gave the first non-trivial algorithm for minimizing the weighted flow time on a single machine. Their algorithm achieves a competitive ratio of  $O(\log^2 P)$ , where  $P$  is the ratio of the maximum to minimum job size. Strictly speaking, the algorithm of [Chekuri et al. 2001] is not completely online (the authors call it semi-online), as it requires the knowledge of  $P$  beforehand. They also show that no online algorithm can have a competitive ratio better than 1.618 for single machine, and that any randomized online algorithm is  $\Omega(\min(W^{1/2}, P^{1/2}, (\frac{n}{m})^{1/4}))$  competitive for  $m \geq 2$  machines. Here  $W$  is the ratio of the maximum to minimum job weight. This is in sharp contrast to the unweighted case, where several  $O(\min(\log P, \log n/m))$  competitive online algorithms are known [Leonardi and Raz 1997; Awerbuch et al. 1999; Avrahami and Azar 2003] for  $m$  machines. In the context of offline algorithms, the  $O(\log^2 P)$  guarantee mentioned above is also the best known approximation ratio for the single machine case. Somewhat surprisingly, Chekuri and Khanna [Chekuri and Khanna 2002] showed that much better results are possible if the algorithm is allowed quasi-polynomial time. In particular, they gave a  $(1 + \epsilon)$  approximation algorithm with running time  $O(n^{O(\ln W \ln P/\epsilon^3)})$ . This implies a quasi-polynomial time approximation scheme (QPTAS) when both  $W$  and  $P$  are polynomially bounded in  $n$ . They further refine their algorithm to obtain a QPTAS even when only one of either  $W$  or  $P$  is polynomially bounded in  $n$ . Bansal [Bansal 2005] extended the QPTAS of [Chekuri and Khanna 2002] to the case of a constant number of machines.

The special case where the weight of a job is the inverse of its size, commonly referred to as stretch or slowdown, has also been studied extensively [Bender et al. 1998; Muthukrishnan et al. 1999; Becchetti et al. 2000; Bender et al. 2002]. Stretch is a very natural measure of fairness and indicates the waiting time per unit of job size. In the online setting, it is known that SRPT is 2 competitive for minimizing stretch on single machine and is 14 competitive for multiple machines [Muthukrishnan et al. 1999]. In the offline case, there is a polynomial time approximation scheme for minimizing stretch on a single machine [Bender et al. 2002].

Weighted flow time has also been investigated extensively in the resource augmentation model. The resource augmentation model was first introduced in the context of scheduling by Kalyanasundaram and Pruhs [Kalyanasundaram and Pruhs 2000],

has been very successful in the analysis of algorithms, particularly in situations where the traditional worst case analysis is overly pessimistic. The main idea is to compare the algorithm to a more restricted optimum offline algorithm. An algorithm Alg is said to be  $s$ -speed,  $c$ -competitive (or approximate) if the worst case ratio  $\text{Alg}_s(I)/\text{Opt}_1(I)$  considered over all possible instances  $I$ , never exceeds  $c$ . Here  $\text{Alg}_s(I)$  denotes the performance of Alg on  $I$  with an  $s$ -speed processor and  $\text{Opt}_1(I)$  is the best possible offline optimum on  $I$  with a unit speed processor. We refer the reader to the survey [Pruhs et al. 2004] for a detailed discussion of this model. Phillips et al. [Phillips et al. 1997] gave the first guarantee with resource augmentation for minimizing weighted flow time on a single machine. Their algorithm was 2-speed, 1-competitive. Becchetti et al. [Becchetti et al. 2001] reduced the amount of resource augmentation required and showed that HDF (Highest Density First), the algorithm that at any time works on the job with the highest weight to size ratio, is  $(1 + \epsilon)$ -speed,  $(1 + 1/\epsilon)$ -competitive. Bansal and Pruhs [Bansal and Pruhs 2004] extended this to show that HDF has similar guarantees for all weighted  $\ell_p$  norms of flow time. For multiple machines, Chekuri et al. [Chekuri et al. 2004] showed that the algorithm that simply assigns a job randomly to a machine and does SRPT on each machine is  $(1 + \epsilon)$ -speed,  $\text{poly}(1/\epsilon)$ -competitive for  $\ell_p$  norms of unweighted flow time.

*Our Results.* In this paper, we give the first truly online algorithm to minimize weighted flow time on a single machine. Our algorithm, that we call Balanced SRPT, is  $O(k)$  competitive if the job weights belong to at most  $k$  different arbitrary weight classes. This yields an  $O(1)$ -competitive algorithm if  $k = O(1)$ . Even for  $k = 2$ , no  $O(1)$  competitive algorithm was known prior to our work. Rounding the weights up to the nearest power of 2, it easily follows that our algorithm is  $O(\log W)$  competitive. While our bounds are in terms of  $W$ , we also show that Balanced SRPT can be modified to give an algorithm with an approximation ratio of  $O(\log n + \log P)$ .

Our second result deals with the non-clairvoyant setting where the size of a job is unknown throughout its execution, i.e. the size of a job becomes known only when the job finishes its service requirement and leaves the system. The non-clairvoyant setting was introduced by Motwani, Phillips and Torng [Motwani et al. 1994] and realistically models many situations. For example in the case of Unix operating system, it is usually not clear what is the CPU requirement of a job when it arrives. As can be expected, the performance of non-clairvoyant algorithms is quite bad when compared with the optimum clairvoyant and offline adversary. Motwani et al. [Motwani et al. 1994] show that any non-clairvoyant deterministic (resp. randomized) algorithm is at least  $\Omega(n^{1/3})$  (resp.  $\Omega(\log n)$ ) competitive for unweighted flow time on a single machine.

In the non-clairvoyant setting, a natural analog of the Shortest Job First or SRPT policy is the Shortest Elapsed Time First (SETF) policy, that at any time works on the job that has received the least amount of work thus far. In one of the first applications of resource augmentation, Kalyanasundaram and Pruhs [Kalyanasundaram and Pruhs 2000] showed that SETF is  $(1 + \epsilon)$ -speed,  $(1 + 1/\epsilon)$ -competitive for minimizing total flow time non-clairvoyantly on a single machine. Interestingly, Kalyanasundaram and Pruhs [Kalyanasundaram and Pruhs 2003] gave a random-

ized variant of SETF with an almost optimal competitive ratio of  $O(\log n \log \log n)$ . Becchetti and Leonardi [Becchetti and Leonardi 2004] tightened the guarantee to  $O(\log n)$ , and also gave a randomized  $O(\log n \log(n/m))$  competitive algorithm for multiple machines.

In this paper, we show that a weighted variant of SETF (that we call WSETF) is  $(1 + \epsilon)$ -speed  $(1 + \frac{1}{\epsilon})$ -competitive for minimizing the total weighted flow time in the non-clairvoyant setting. Our result can be viewed as an extension of the result of Kalyanasundaram and Pruhs [Kalyanasundaram and Pruhs 2000] to the weighted case, as well as an extension of the result of Becchetti et al. [Becchetti et al. 2001] to the non-clairvoyant setting. Since the publication of our result, several related results have been obtained. Bansal and Pruhs [Bansal and Pruhs 2004] showed that WSETF is also  $(1 + \epsilon)$ -speed,  $\text{poly}(1/\epsilon)$ -competitive for minimizing  $\ell_p$  norms of weighted flow time non-clairvoyantly. Chekuri et al. [Chekuri et al. 2004] showed that SETF coupled with random assignment is  $(1 + \epsilon)$ -speed,  $\text{poly}(1/\epsilon)$ -competitive for minimizing  $\ell_p$  norms of flow time on multiple machines. It is also known that SETF is  $(1 + \epsilon)$ -speed,  $\text{poly}(\log P/\epsilon)$ -competitive for minimizing the  $\ell_p$  norms of stretch [Bansal et al. 2003; Bansal and Pruhs 2003]. Note that unlike in the clairvoyant setting, non-clairvoyant stretch minimization is not a special case of weighted flow time, because here the weight is equal to the inverse of job size, which is also unknown.

## 2. NOTATION AND PRELIMINARIES

There are  $n$  jobs indexed  $1, \dots, n$  that are released over time. For a job  $j$  the quantities  $r(j)$ ,  $p(j)$  and  $\text{wt}(j)$  denote the release time, processing time (or size) and the weight respectively. Throughout this paper we consider the single machine setting with preemptions, where the execution of a job can be interrupted arbitrarily and the job can be resumed later from the point of interruption without any penalty. In the online setting, at any time  $t$  the algorithm is only aware of the jobs that have been released until  $t$ . In particular, at time  $t = r(j)$ , it learns  $p(j)$  and  $\text{wt}(j)$  for job  $j$ . In the non-clairvoyant model, the scheduler only learns  $\text{wt}(j)$  when  $j$  arrives and the size  $p(j)$  is not revealed. The scheduler learns  $p(j)$  when  $j$  completes its service requirement and terminates. We say that a job is alive at time  $t$ , if it has been released by time  $t$  but has not completed its service requirement. At any time  $t$ , we say that an alive job  $j$  has volume  $\text{vol}(j, t)$ , if it requires  $\text{vol}(j, t)$  amount of processing before it completes. We drop  $t$  from the notations wherever it is clear from the context. We also use the notation  $\text{vol}(X, t)$  and  $\text{wt}(X, t)$  to denote the cumulative volume and cumulative weight respectively of subsets of jobs  $X$ . The flow time of  $j$  is  $f(j) = c(j) - r(j)$ , where  $c(j)$  is the completion time of job  $j$  in the schedule. Our objective is to minimize  $\sum_j \text{wt}(j)f(j)$ , the total weighted flow time.

Suppose each job  $j$  pays  $\text{wt}(j)$  units at each time unit it is alive, then it is easily seen that the total payment is equal to the total weighted flow time. In this view, the payment accrued per time unit is exactly equal to the weight of the jobs alive at time  $t$ . Thus for any algorithm  $A$  the total weighted flow time is equal to  $\sum_t \text{wt}_A(t)$ , where  $\text{wt}_A(t)$  denotes the total weight of alive jobs under  $A$  at time  $t$ . Thus to show that algorithm  $A$  is  $c$ -competitive, it suffices to show that

at any time  $\text{wt}_A(t) \leq c \cdot \text{wt}_O(t)$  for any other algorithm  $O$ . If this condition holds at all times  $t$ , we say that  $A$  is *locally*  $c$ -competitive. Interestingly, it turns out that local-competitiveness is also necessary for deterministic algorithms to achieve a good competitive ratio with respect to total weighted flow time [Becchetti et al. 2001; Chekuri et al. 2001]. The reason is that if  $\text{wt}_A(t) > c \text{wt}_O(t)$  at some time  $t$ , then the adversary can start giving a sequence of jobs, where each job has size  $\epsilon$  and weight  $\delta \gg \epsilon$  that arrives every  $\epsilon$  units of time. By setting  $\epsilon$  and  $\delta$  arbitrarily close to 0, this forces any algorithm to interrupt whatever it was doing earlier and immediately start working on this job sequence. By making the sequence long enough, the adversary can force the competitive ratio to be at least  $c - 2\delta$ .

Thus any algorithm with a non-trivial competitive ratio must keep the total weight of alive jobs low at all times, or equivalently, at any time try to get rid of as much weight as possible. A natural greedy strategy is Highest Density First (HDF), which at any time schedules the job with the largest weight to processing time ratio and (as mentioned earlier) is  $(1 + \epsilon)$ -speed,  $O(1/\epsilon)$ -competitive [Becchetti et al. 2001]. However, it is easily seen that HDF can perform very badly without resource augmentation. The problem is that it does not distinguish between a job with a high weight and correspondingly high processing time and a small weight job with a small processing time. For example, suppose two jobs arrive at time  $t = 0$ , one of weight 1 and size 1, and another of weight  $W$  and size  $W + 1$ . HDF will work on the size 1 job first, and at time  $W + 1$  will be left with the weight  $W$  job, while the optimum algorithm could have worked on weight  $W$  job first and only have the weight 1 job left unfinished at time  $W + 1$ . Hence HDF can have an arbitrary bad local (and hence global) competitive ratio. Thus a good algorithm must carefully handle both high weight jobs and high ratio jobs. The  $O(\log^2 P)$  competitive algorithm of [Chekuri et al. 2001] deals with the two problems above by striking a balance between them. It processes a large weight job without regard to its ratio as long as the total weight of jobs with a strictly better weight to remaining processing time ratio does not get too large.

### 3. ALGORITHM

We now present our algorithm Balanced SRPT. We assume that the weights of the jobs are drawn from the set  $\{w_1, w_2, \dots, w_k\}$ , where  $w_1 < w_2 < \dots < w_k$ . We will refer to these as weight classes  $1, \dots, k$ . We will assume that  $w_i/w_{i-1}$  is integral for  $1 < i \leq k$ . One way to achieve this is to round up each weight to the nearest power of 2, while losing only a factor of 2 in the competitive ratio. Let  $Q_i(t)$  denote the set of alive jobs of weight class  $i$  at time  $t$ .

**3.0.0.1 *Balanced SRPT*:** At all times, consider the weight class  $i$  that has maximum total weight  $\text{wt}(Q_i(t))$ , breaking ties in favor of the higher weight class. Among the jobs in  $Q_i(t)$  execute the one with shortest remaining processing time.

Intuitively, the algorithm tries to balance the weight in various weight classes, while giving priority to larger weights. It runs SRPT within each weight class. Note that the algorithm implicitly favors jobs with a higher weight. For example, if there are  $W$  or fewer jobs of weight 1 and one job of weight  $W$ . The algorithm will work on the  $W$  weight job first.

### 3.1 Analysis

We now prove the  $O(k)$  competitiveness of the Balanced SRPT algorithm and also show that the analysis is tight. Our proof can be viewed as an extension of the technique used by Schrage [Schrage 1968] to prove the (local) optimality of SRPT for unweighted flow time. The idea of Schrage was to consider the alive jobs under SRPT at time  $t$  ordered non-increasingly according to their remaining volume. Under this ordering, SRPT always works on the lowest ranked job. Schrage showed by an inductive argument that for any integer  $l > 0$ , the total volume of the  $l$  largest volume jobs under SRPT is no less than the volume of the  $l$  largest volume jobs under Opt. Since the total unfinished work under any two algorithms is the same (assuming that the algorithms do not idle unnecessarily when there is work to be done), this implies that at any time  $t$ , the number of jobs under SRPT must be no more than that under Opt. We use a similar technique for the analysis of Balance SRPT, but our proof is substantially harder because of the different weights involved.

We first define the required notation. We use Bal to denote Balanced SRPT and let Opt denote some fixed optimum algorithm. It convenient to view the jobs grouped according to their weight class and arranged in the non-increasing order of their remaining processing time (see Figure 1 for a pictorial view). Consider a fixed time  $t$ . The tuple  $(j, l, t)$  will refer to the job in weight class  $j$  that has the  $l^{\text{th}}$  largest remaining processing time among jobs in that class under Bal. We will drop the index  $t$  when it is clear from the context.

Suppose we are given a collection of jobs  $X$ , and no more jobs arrive. If we consider the execution of Bal on  $X$ , it defines a fixed order on jobs in  $X$ . We formalize this by defining a total order  $\preceq$  on jobs or equivalently 2-tuples of integers  $(j, l)$ . We say that  $(j, l) \preceq (j', l')$  if and only if one of the following condition holds:

- (1)  $w_j \cdot l < w_{j'} \cdot l'$
- (2)  $w_j \cdot l = w_{j'} \cdot l'$ , and  $j < j'$

It can be easily verified that  $(j', l') \preceq (j, l)$  and  $(j, l) \preceq (j', l')$  holds if and only if  $j = j'$  and  $l = l'$ , and hence  $\preceq$  defines a total order on the 2-tuples. Observe that Bal can be described in the following alternate way: At any time order the jobs according the alive jobs according to  $\preceq$  and execute the job with the highest priority.

Let  $B(j, l, t)$  denote the set of jobs alive at time  $t$  under Bal with priority no more than that of job  $(j, l)$ . Thus,  $B(j, l, t)$  consists of all jobs at  $(j', l')$  such that  $(j', l') \preceq (j, l)$ . Figure 1 below describes the sets  $B(j, l, t)$  for various of  $j$  and  $l$ . Let  $\text{vol}(B(j, l, t))$  denote the total remaining processing time of jobs in  $B(j, l, t)$ .

We state some simple properties of the sets  $B(j, l, t)$ .

**Observation 3.1** For  $h \leq j$  and any  $l$ ,  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(h, w_j/w_h \cdot l, t))$ .

**PROOF.** By the definition of the  $\preceq$  relation it follows directly that  $B(h, w_j/w_h \cdot l, t) \subseteq B(j, l, t)$ , for  $h \leq j$ . Thus, it follows that  $\text{vol}(B(h, w_j/w_h \cdot l, t)) \subseteq \text{vol}(B(j, l, t))$ . For example, consider the sets  $B(1, 4, t) \subset B(2, 2, t)$  in Figure 1.  $\square$

Without loss of generality we can assume that Opt does not idle and always works on some job if there is work to be done (otherwise the solution can be

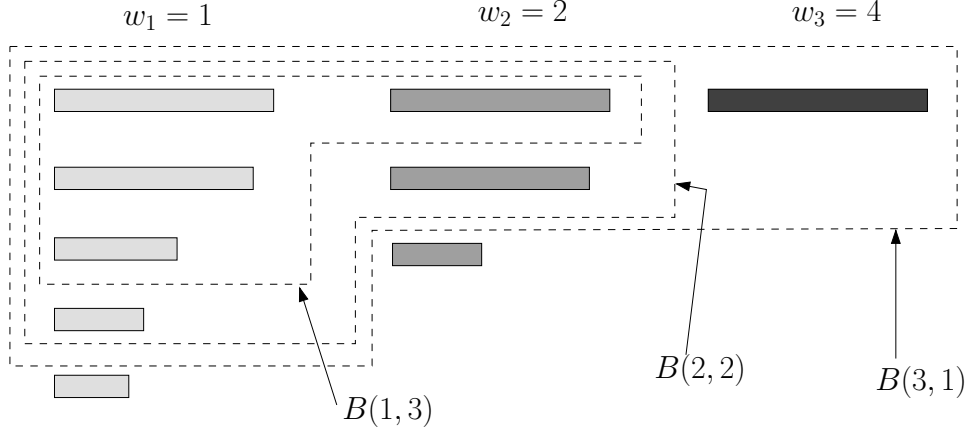


Fig. 1. An example illustrating the notation used

trivially improved). Let  $\text{vol}(t)$  denote the total amount of unfinished work at time  $t$  under Bal, which is also equal to the unfinished work under Opt. If Bal works on the job  $(j, l, t)$  at time  $t$ , then it must be that  $(j, l)$  is the highest priority job alive under Bal and hence the set  $B(j, l, t)$  contains all alive jobs in Bal at time  $t$ . We will use this observation as follows:

**Observation 3.2** *Given  $j$  and  $l$ , if Bal works on some job in  $B(j, l, t)$  at time  $t$ , then  $\text{vol}(B(j, l, t)) = \text{vol}(t)$ .*

The main proof idea will be to compare the quantity  $\text{vol}(B(j, l, t))$  with certain subsets of alive jobs under Opt. Intuitively, for any value of  $j$  and  $l$  the set  $B(j, l, t)$  consists of the lowest priority jobs. Hence these sets must contain a lot of volume. On the other hand, the total weight of jobs in  $B(j, l, t)$  is at most  $k \cdot w_j \cdot l$ . Let  $X$  be any arbitrary set of jobs alive under Opt at time  $t$  subject to the constraint that its total weight is at most  $w_j \cdot l$ . We will prove that for any such set  $X$ , the volume of  $X$  is no greater than that of  $B(j, l, t)$ , i.e.  $\text{vol}(X) \leq \text{vol}(B(j, l, t))$ . Using the fact that the total volume of jobs under Bal and Opt is identical at all times (equal to  $\text{vol}(t)$ ), this will easily imply that at any time, the total weight under Bal never exceeds more than  $k$  times that under Opt.

We now formalize this idea. Consider the set  $\text{Opt}(t)$  of alive jobs under Opt at time  $t$ . For any fixed  $j$  and  $l$ , let  $\mathcal{P}(j, l, t)$  be the collection of all subsets  $X \subseteq \text{Opt}(t)$  such that  $W(X) \leq w_j \cdot l$ . That is,  $\mathcal{P}(j, l, t)$  is the collection of all subsets of alive jobs under Opt with total weight at most  $w_j \cdot l$ . Our goal will be to show that for any set  $X$  in  $\mathcal{P}(j, l, t)$ , the volume of  $X$  never exceeds  $\text{vol}(B(j, l, t))$ . We will do this by an inductive argument over events in the system (such as arrival of a new job, completion of a job, or simply as time evolves).

The following lemma characterizes the evolution of  $B(j, l, t)$  when a new job arrives.

**Lemma 3.3** *Suppose a job  $J$  of weight  $w_i$  and size  $\text{vol}(J)$  arrives at time  $t$ , and that it is inserted in some position  $h$  based on the remaining processing times among the weight  $w_i$  jobs under Bal. Let  $t^-$  denote the time just before  $J$  arrives, then*

- (1) *For all  $j$  and  $l$ ,  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(j, l, t^-))$ . In particular,  $V(B(j, l, t))$  never decreases on the arrival of a new job.*
- (2) *For all  $j$  and for all  $l$  such that  $l > w_i/w_j$*

$$\text{vol}(B(j, l, t)) \geq \begin{cases} \text{vol}(B(j, l - w_i/w_j, t^-)) + \text{vol}(J) & \text{if } i > j \\ \text{vol}(B(i, w_j \cdot l/w_i - 1, t^-)) + \text{vol}(J) & \text{if } i \leq j \end{cases}$$

PROOF. Consider some arbitrary but fixed  $j$  and  $l$ . Let  $f_i$  be the smallest number for which  $(j, l) \preceq (i, f_i)$ . In other words, the weight  $w_i$  job  $(i, f_i - 1)$  lies in  $B(j, l, t)$  but the job  $(i, f_i)$  does not.

We first show that  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(j, l, t^-))$ . We have two cases.

- (1) If  $J \notin B(j, l, t)$ , then clearly,  $B(j, l, t) = B(j, l, t^-)$  and hence  $\text{vol}(B(j, l, t)) = \text{vol}(B(j, l, t^-))$ .
- (2) In the case when  $J \in B(j, l, t)$ , we argue as follows. Since  $J \in B(j, l, t)$  by our assumption, we must have that  $h < f_i$ . Then, the only way in which  $B(j, l, t)$  differs from  $B(j, l, t^-)$  is that  $J$  is inserted in position  $(i, h)$  and the jobs previously in positions  $(i, h), (i, h+1), \dots$  move to positions  $(i, h+1), (i, h+2), \dots$  respectively. Let  $\text{vol}(J')$  denote the remaining processing time of the job at position  $(i, f_i - 1)$  at time  $t^-$ . We have that  $\text{vol}(B(j, l, t)) = \text{vol}(B(j, l, t^-)) - \text{vol}(J') + \text{vol}(J)$ . Since  $h \leq f_i - 1$ , it must be that  $\text{vol}(J') \leq \text{vol}(J)$ , and hence  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(j, l, t^-))$ .

Before we prove the second part of the lemma, we need the following simple facts. Let  $\text{vol}(J')$  denote the remaining processing time of the job at position  $(i, f_i - 1)$  at time  $t^-$ . (We set  $\text{vol}(J') = 0$  if there is no job at position  $(i, f_i - 1)$ ). If  $i \leq j$ , we have that  $f_i = w_j \cdot l/w_i + 1$  and hence  $(i, w_j \cdot l/w_i - 1) \preceq (i, f_i - 1)$ , thus

$$\text{vol}(B(j, l, t^-)) \geq \text{vol}(B(i, l \cdot w_j/w_i, t^-)) = \text{vol}(B(i, l \cdot w_j/w_i - 1, t^-)) + \text{vol}(J'). \quad (1)$$

For  $i > j$ , we have that  $(j, \max(0, l - w_i/w_j)) \preceq (i, f_i - 1) \preceq (j, l)$ . This implies that,

$$\text{vol}(B(j, l, t^-)) \geq \text{vol}(B(j, \max(0, l - w_i/w_j), t^-)) + \text{vol}(J') \quad (2)$$

We now prove the second part of the lemma. We have two cases depending of whether  $J \in B(j, l, t)$  or not.

First, if  $J \notin B(j, l, t)$ , then it must be the case that  $\text{vol}(J) \leq \text{vol}(J')$ . For  $i \leq j$ , it follows that

$$\begin{aligned} \text{vol}(B(j, l, t)) &\geq \text{vol}(B(i, l \cdot w_j/w_i, t)) && \text{(by Observation 3.1)} \\ &\geq \text{vol}(B(i, l \cdot w_j/w_i - 1, t^-)) + \text{vol}(J') && \text{(by Equation 1)} \\ &\geq \text{vol}(B(i, l \cdot w_j/w_i - 1, t^-)) + \text{vol}(J) && \text{(as } \text{vol}(J) \leq \text{vol}(J') \text{)} \end{aligned}$$

For  $i > j$ ,

$$\begin{aligned} \text{vol}(B(j, l, t)) &\geq \text{vol}(B(j, l - w_i/w_j, t)) + \text{vol}(J') \\ &\geq \text{vol}(B(j, l - w_i/w_j, t)) + \text{vol}(J) \\ &= \text{vol}(B(j, l - w_i/w_j, t^-)) + \text{vol}(J) \end{aligned}$$

We now consider the case when  $J \in B(j, l, t)$ . If  $J \in B(j, l, t)$ , it must be that  $\text{vol}(J) \geq \text{vol}(J')$ . Thus, we have that for  $i > j$ ,

$$\begin{aligned} \text{vol}(B(j, l, t)) &\geq \text{vol}(B(j, l, t^-)) - \text{vol}(J') + \text{vol}(J) \\ &\geq \text{vol}(B(j, l - w_i/w_j, t^-)) + \text{vol}(J) \quad (\text{by Equation 2}) \end{aligned}$$

Similarly, for  $i \leq j$ ,

$$\begin{aligned} \text{vol}(B(j, l, t)) &\geq \text{vol}(B(i, l \cdot w_j/w_i, t)) \\ &= \text{vol}(B(i, l \cdot w_j/w_i, t^-)) - |J'| + \text{vol}(J) \\ &\geq \text{vol}(B(i, l \cdot w_j/w_i - 1, t^-)) + \text{vol}(J) \end{aligned}$$

□

We now show that sets  $B(j, l, t)$  have a lot of volume. In particular, for any values of  $j$  and  $l$ , the volume of  $B(j, l, t)$  is at least as large as that of any set  $X \in \mathcal{P}(j, l, t)$ .

**Lemma 3.4** *At all times  $t$ , for all  $1 \leq j \leq k$  and  $l \geq 1$ , and any subset of jobs  $X \in \mathcal{P}(j, l, t)$  the following invariant holds:*

$$\text{vol}(B(j, l, t)) \geq \text{vol}(X, t). \quad (3)$$

**PROOF.** We will prove Equation (3) by induction on the number of job arrivals. By perturbing the arrival times of jobs by an infinitesimally small amount we can assume without loss of generality that all the arrival times are distinct. Let  $0 = t_1 \leq t_2 \leq \dots \leq t_n$  denote the arrival times of the  $n$  jobs. Clearly, the invariant is true at  $t_1 = 0$ .

Suppose the invariant is true at some  $t_i$ . We first show that the invariant (3) is true at all times in the interval  $[t_i, t_{i+1})$ . For the sake of contradiction, let  $t \in [t_i, t_{i+1})$  be the first time when the invariant (3) fails to hold. Observe that since there are no new arrivals during  $(t_i, t_{i+1})$ , the quantities  $\text{vol}(B(j, l, t))$  and  $\text{vol}(X)$  are non-increasing. Since the invariant was true at time  $t_i$  and is not true at time  $t$ , it must be that  $\text{vol}(B(j, l, t))$  decreased during  $(t_i, t]$ . Let  $t' \in (t_i, t]$  be the last time when  $\text{vol}(B(j, l, t'))$  decreased. This means that at time  $t'$ , Bal was working on some job in  $B(j, l, t')$  and hence  $\text{vol}(B(j, l, t')) = \text{vol}(t')$ .

If  $t = t'$ , we know that  $\text{vol}(B(j, l, t)) = \text{vol}(t)$ , and hence  $\text{vol}(B(j, l, t)) \geq \text{vol}(X)$  for any  $X \in \mathcal{P}(j, l, t)$ . If  $t' < t$ , then by our choice of  $t'$ ,  $\text{vol}(B(j, l, t))$  did not decrease during  $(t, t']$ . As there were no arrivals during this time  $(t, t']$ , it follows that  $\text{vol}(B(j, l, t)) = \text{vol}(B(j, l, t')) = \text{vol}(t')$ . If  $\text{vol}(B(j, l, t)) < \text{vol}(X, t)$ , then  $\text{vol}(B(j, l, t')) = \text{vol}(B(j, l, t)) < \text{vol}(X, t) \leq \text{vol}(X, t')$ , contradicting the fact that  $t$  was the earliest time when the invariant (3) fails to hold.

It remains to prove that the invariant remains true on a job arrival. Let  $t = t_i$  and let  $t^-$  denote the time instance just before time  $t$ . We will show that if equation (3) fails to hold at time  $t$ , then it could not be true for some set  $X \in \mathcal{P}(j, l, t^-)$  at time  $t^-$ . Let  $X$  denote the set of jobs in Opt for which  $\text{vol}(B(j, l, t)) < \text{vol}(X, t)$  and let  $J$  denote the job that arrived at time  $t$ .

If  $J \notin X$  then the weight of  $X$  is unchanged and  $X$  also lies in  $\mathcal{P}(j, l, t^-)$ , and moreover  $\text{vol}(X, t) = \text{vol}(X, t^-)$ . However, by Lemma 3.3,  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(j, l, t^-))$ . Thus if  $\text{vol}(B(j, l, t)) < \text{vol}(X, t)$ , then this would contradict that  $\text{vol}(B(j, l, t^-)) \geq \text{vol}(X, t^-)$ .

We now consider the more interesting case when  $J \in X$ . Let  $i$  denote the weight class of  $J$ . We consider two cases depending on whether  $i > j$  or  $i \leq j$ .

$i > j$  : Consider the set  $X' = X \setminus \{J\}$ . Clearly,  $X' \in \mathcal{P}(j, l - w_i/w_j, t^-)$  and  $\text{vol}(X, t) = \text{vol}(X', t^-) + \text{vol}(J)$ . By our inductive hypothesis  $\text{vol}(X', t^-) \leq \text{vol}(B(j, l - w_i/w_j, t^-))$ . By Lemma 3.3, we know that  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(j, l - w_i/w_j, t^-)) + \text{vol}(J)$ . Thus it follows that  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(j, l - w_i/w_j, t^-)) + \text{vol}(J) \geq \text{vol}(X', t^-) + \text{vol}(J) = \text{vol}(X, t)$ .

$i \leq j$  : Consider the set  $X' = X \setminus J$ . We have that  $X' \in \mathcal{P}(j, (w_j/w_i)l - 1, t^-)$  and  $\text{vol}(X, t) = \text{vol}(X', t^-) + \text{vol}(J)$ . By Lemma 3.3,  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(i, (w_j/w_i)l - 1, t^-) + \text{vol}(J)$ . Thus,  $\text{vol}(B(j, l, t)) \geq \text{vol}(B(i, w_j l/w_i - 1, t^-)) + \text{vol}(J) \geq \text{vol}(X', t^-) + \text{vol}(J) = \text{vol}(X, t)$ .

□

The main result now follows directly.

**Theorem 3.5** *The Balanced SRPT algorithm is  $O(k)$  competitive for  $k$  weight classes. In particular, if the ratio of weights  $w_i/w_{i-1}$  is integral for  $i = 2, \dots, k$ , then Balanced SRPT is exactly  $k$ -competitive.*

PROOF. Consider an arbitrary time  $t$ , and let  $X$  denote the total weight of alive jobs under Opt at that time. Clearly,  $W(X)$  is a multiple of  $w_1$ , since  $w_i/w_{i-1}$  is an integer for each  $1 < i \leq k$ . Choose  $j = 1$  and  $l = W(X)/w_1$ , and observe that  $X \in \mathcal{P}(j, l, t)$ . By Lemma 3, we have that  $\text{vol}(B(j, l, t)) \geq \text{vol}(X) = \text{vol}(t)$ . On the other hand,  $\text{vol}(B(j, l, t))$  can be at most  $\text{vol}(t)$  (since Bal does not idle unnecessarily) and hence  $\text{vol}(B(j, l, t)) = \text{vol}(t)$ . Thus  $B(j, l, t)$  contains all the jobs. The  $k$  competitiveness follows by observing that  $W(B(j, l, t)) \leq k \cdot w_j \cdot l = k \cdot W(X)$ . □

### 3.2 Analysis of Balanced SRPT is tight

It is easily seen that the competitive ratio of Bal can be as large as  $k$ . Consider the instance where  $2^{k-j}$  jobs of weight  $2^j$  arrive at time 0, for  $j = 1, \dots, k$ . The job of weight  $2^k$  has size 1, and every other job has size  $\epsilon = 2^{-k}$ . Since all weight classes have an equal amount of weight, Bal starts working on the  $2^k$  weight job until it finishes.

Consider the scenario at time  $t = (2^k - 2)\epsilon < 1$ . Bal still works on the  $2^k$  weight job, and the total weight under Bal is  $k2^k$ . Opt on the other hand can finish all the jobs of size  $\epsilon$  by time  $t$ , and have unfinished total weight of  $2^k$ . Thus Bal is locally  $k$ -competitive at time  $t$ , and the adversary can release a long continuous stream of weight  $2^k$  and size  $2^{-k}$  jobs to make it globally  $k$ -competitive.

### 3.3 A guarantee in terms of $P$ and $n$

We can modify Bal in a simple way to yield a simple approximation algorithm with guarantee in terms of the number of jobs  $n$  and the maximum to minimum job size ratio  $P$ . By suitable scaling, let us assume that minimum job size is exactly 1 and that all weights are integers. Round the weights up to the nearest power of 2. Consider only the jobs with weights between  $\frac{W}{Pn^2}$  and  $W$ . Call these jobs *heavy*.

Run the Balanced SRPT algorithm on the heavy jobs, if at some time no heavy job is present, work on any arbitrary unfinished job.

Let  $I_h$  denote the instance restricted to heavy jobs and let  $\text{Opt}(I_h)$  and  $A(I_h)$  denote the weighted flow time of some fixed optimum algorithm and our algorithm on  $I_h$  respectively. Since the weight ratio of any two jobs in  $I_h$  is at most  $Pn^2$ , by Theorem 3.5 it follows that:

$$A(I_h) \leq 4(\log n + \log P)\text{Opt}(I_h) \leq 4(\log n + \log P)\text{Opt}(I)$$

Each non-heavy job can have a flow time of at most  $nP$ . Since its weight is no more than  $W/n^2P$ , it adds at most  $W/n$  to the total weighted flow time. Since there are at most  $n$  non-heavy jobs, the total contribution of these is no more than  $W$ . Since  $\text{Opt}(I)$  is lower bounded by  $W$ , we obtain:

$$\begin{aligned} A(I) &\leq A(I_h) + W \\ &\leq 4(\log n + \log P)\text{Opt}(I) + W \\ &\leq 4(\log n + \log P)\text{Opt}(I) + \text{Opt}(I) \end{aligned}$$

#### 4. THE NON-CLAIRVOYANT CASE

We now consider the non-clairvoyant version of the weighted flow time minimization problem. In the non-clairvoyant setting, the processing time of a job becomes known only when it is completed. We give a resource augmented  $(1 + \epsilon)$ -speed  $(1 + 1/\epsilon)$ -competitive algorithm. Our algorithm is a natural weighted variant of the natural SETF (Shortest Elapsed Time First) algorithm and we call it WSETF.

For any job  $j$  and time  $t$ , let  $p(j, t)$  denote the amount of work done on  $j$  by time  $t$ . Let  $n(j, t) = p(j, t)/\text{wt}(j)$  denote the normalized work done on  $j$  by time  $t$ . The algorithm WSETF, at any time  $t$ , works on the alive jobs  $j$  that have minimum normalized work  $n(j, t)$ . Observe that if there are many jobs that have the lowest normalized work, then WSETF works on them simultaneously while increasing their normalized work at the same rate. Thus if there are two or more jobs with least normalized work, WSETF switches back and forth between these jobs infinitely often making it completely impractical. In practice, a discrete variant of SETF known as MLF (Multi-level Feedback), that restricts the average number of preemptions per job to logarithmic is typically used. In this paper we ignore this issue and concentrate on WSETF since it is easier to analyze. We refer the reader to the survey [Pruhs et al. 2004] for more discussion on this topic.

When a new job arrives, its normalized work is zero and WSETF starts working on it immediately until its normalized work becomes equal to that of the jobs that were worked upon just before this job arrived. It then continues to increase the normalized work of these jobs at the same rate, and the process continues. The immediately implies the following observation.

**Observation 4.1** *Consider two jobs  $x$  and  $y$ . If  $n(x, t') \leq n(y, t')$  for some  $t'$ , then  $n(x, t) \leq n(y, t)$  for all times  $t$  until one of these jobs completes.*

Consider two jobs  $x$  and  $y$  such that  $x$  was released before job  $y$ . At time  $r(x)$ , when  $x$  was released, WSETF immediately starts working on it. At time  $r(y)$ , when the job  $y$  was released  $n(x, r(y)) > 0$ . Thus, by Observation 4.1 it follows

that  $n(x, t') \geq n(y, t')$  as long as both jobs are still alive. This implies the following simple but useful fact.

**Lemma 4.2** *Let  $x, y$  be two jobs such that  $r(x) \leq r(y)$ . If  $x$  and  $y$  are both alive at time  $t$  under WSETF, then  $n(x) \geq n(y)$ . Moreover,  $n(x) > n(y)$  implies that  $r(x) < r(y)$  and WSETF hasn't worked on the job  $x$  after the release of job  $y$ .*

Let us fix some optimum offline algorithm. We assume that WSETF has a  $(1 + \epsilon)$  speed processor and the optimum algorithm has a unit speed processor. Let  $U_B(t)$  and  $U_A(t)$  denote the set of alive jobs at time  $t$  under WSETF and the optimal schedule at time  $t$ . We focus our attention on bounding  $\text{wt}(U_B(t))$  and show that  $W(U_B(t)) \leq (1 + \frac{1}{\epsilon})\text{wt}(U_A(t))$  for all times  $t$ . Henceforth, we will only consider the state of the algorithms at some fixed time  $t$  and thus drop  $t$  from all the notation.

We partition the set  $U_B$  into several classes  $B_1, \dots, B_k$  based on the normalized work. Jobs in a given class  $B_l$  have the same normalized work  $N_l$ , and the classes are indexed such that  $N_1 > N_2 > \dots > N_k$ . By Lemma 4.2, it must be that all the jobs in class  $B_1$  are released before all jobs in class  $B_2$ , which are released before all jobs in class  $B_3$  and so on. Let  $b_l = \min\{r(x) | x \in B_l\}$  denote the release date of earliest arriving job in the class  $B_l$ . We will call  $b_l$  as  $l^{\text{th}}$  epoch. Observe that WSETF doesn't work on any of the jobs in the classes  $B_1, B_2, \dots, B_{l-1}$  after the epoch  $b_l$ . We partition all the jobs  $J$  in the instance based on these epochs. Let  $A_l = \{x \in J | r(x) \in [b_l, b_{l+1})\}$ . Thus  $A_l$  is the set of jobs released between epochs  $b_l$  and  $b_{l+1}$ .

We now relate the weight of the jobs in  $U_A$  to that of  $U_B$ . Let us define  $B_l^+ = B_l \cup B_{l+1} \cup \dots$  and  $A_l^+ = A_l \cup A_{l+1} \cup \dots$ . We know that the adversary finished all the jobs in  $A_l^+ - U_A$  between time  $b_l$  and  $t$ . Thus  $\sum_{x \in A_l^+ - U_A} \text{vol}(x) \leq t - b_l$ . During this time, WSETF was working on the jobs in  $A_l^+$  only. The amount of work done by WSETF is  $(1 + \epsilon)(t - b_l)$  (We can assume that WSETF did not idle at any point during this time interval, otherwise we can decompose the instance to form two independent problems). We can account for this work in two parts: work done on jobs in  $A_l^+ \cap U_A$  and work done on jobs in  $A_l^+ - U_A$ . The second quantity is bounded by sum of the sizes of jobs in  $A_l^+ - U_A$  which is at most  $t - b_l$ . Thus WSETF did at least  $\epsilon(t - b_l)$  amount of work on jobs in  $A_l^+ \cap U_A$ . Thus, we can show that:

**Lemma 4.3** *Let  $T_l^+$  denote the amount of work done by WSETF on jobs in  $A_l^+ \cap U_A$ . Then, for  $l = 1, \dots, k$*

$$T_l^+ \geq \epsilon \sum_{j=l}^k \sum_{x \in B_j - U_A} \text{wt}(x) N_j \quad (4)$$

$$T_l^+ \leq \sum_{j=l}^k \sum_{x \in A_j \cap U_A} \text{wt}(x) N_j \quad (5)$$

PROOF. As argued above,  $T_l^+ \geq \epsilon(t - b_l) \geq \epsilon \sum_{x \in A_l^+ - U_A} \text{vol}(x)$ . Restricting the

sum on the right side to jobs in  $B_l^+ - U_A$  and since  $\text{vol}(x) \geq p(x)$  it follows that

$$T_l^+ \geq \epsilon \sum_{x \in B_l^+ - U_A} \text{vol}(x) \geq \epsilon \sum_{x \in B_l^+ - U_A} p(x) \geq \epsilon \sum_{j=l}^k \sum_{x \in B_j - U_A} \text{wt}(x)N_j$$

The second inequality follows directly by noting that normalized work done by WSETF on any job in  $A_j$  can be at most  $N_j$ .  $\square$

Let  $W_j$  denote  $\sum_{x \in A_j \cap U_A} \text{wt}(x)$  and  $W'_j$  denote  $\sum_{x \in B_j - U_A} \text{wt}(x)$ . Using equations (4) and (5) we get:

$$\sum_{j=l}^k W_j N_j \geq \epsilon \left( \sum_{j=l}^k W'_j N_j \right) \quad \text{for } l = 1, 2, \dots, k \quad (6)$$

Let us define  $N_0 = \infty$  for convenience, and let  $S_l$  denote the left side term  $\sum_{j=l}^k W_j N_j$ . Then,

$$\text{wt}(U_A) = \sum_{j=1}^k W_j = \sum_{j=1}^k (S_j - S_{j+1})/N_j = \sum_{j=1}^k \left( \frac{1}{N_j} - \frac{1}{N_{j-1}} \right) S_j.$$

Similarly, setting  $S'_l = \sum_{j=l}^k W'_j N_j$  we obtain

$$\text{wt}(U_B - U_A) = \sum_{j=1}^k W'_j = \sum_{j=1}^k (S'_j - S'_{j+1})/N_j = \sum_{j=1}^k \left( \frac{1}{N_j} - \frac{1}{N_{j-1}} \right) S'_j.$$

By equation (6)  $S_j \geq \epsilon S'_j$  for each  $j$  and since  $(\frac{1}{N_j} - \frac{1}{N_{j-1}}) > 0$  for each  $j$ , it follows that  $W(U_A) \geq \epsilon W(U_B - U_A)$  and hence  $W(B) \leq (1 + 1/\epsilon)W(A)$ . This implies the desired theorem.

**Theorem 4.4** *WSETF is a  $(1 + \epsilon)$ -speed  $(1 + 1/\epsilon)$ -competitive non-clairvoyant algorithm for minimizing weighted flow time.*

#### REFERENCES

- AFRATI, F., BAMPIS, E., CHEKURI, C., KARGER, D., KENYON, C., KHANNA, S., MILLIS, I., QUEYRANNE, M., SKUTELLA, M., STEIN, C., AND SVIRIDENKO, M. 1999. Approximation schemes for minimizing average weighted completion time with release dates. In *IEEE Symposium on Foundations of Computer Science*. 32–43.
- AVRAHAMI, N. AND AZAR, Y. 2003. Minimizing total flow time and completion time with immediate dispatching. In *Proceedings of 15<sup>th</sup> SPAA*. 11–18.
- AWERBUCH, B., AZAR, Y., LEONARDI, S., AND REGEV, O. 1999. Minimizing the flow time without migration. In *ACM Symposium on Theory of Computing*. 198–205.
- BANSAL, N. 2005. Minimizing flow time on a constant number of machines with preemption. *Oper. Res. Lett.* 33, 267–273.
- BANSAL, N., DHAMDHARE, K., KONEMANN, J., AND SINHA, A. 2003. Non-clairvoyant scheduling for minimizing mean slowdown. In *Symposium on Theoretical Aspects of Computer Science (STACS)*. 260–270.
- BANSAL, N. AND PRUHS, K. 2003. Server scheduling in the  $l_p$  norm: A rising tide lifts all boats. In *ACM Symposium on Theory of Computing (STOC)*. 242–250.
- BANSAL, N. AND PRUHS, K. 2004. Server scheduling in the weighted  $l_p$  norm. In *Latin American Symposium on Theoretical Informatics LATIN*. 434–443.

- BECCHETTI, L. AND LEONARDI, S. 2004. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *Journal of ACM* 51, 517–539.
- BECCHETTI, L., LEONARDI, S., AND MUTHUKRISHNAN, S. 2000. Scheduling to minimize average stretch without migration. In *Symposium on Discrete Algorithms*. 548–557.
- BECCHETTI, L., LEONARDI, S., SPACCAMELA, A. M., AND PRUHS, K. 2001. Online weighted flow time and deadline scheduling. In *RANDOM-APPROX*. 36–47.
- BENDER, M., CHAKRABARTI, S., AND MUTHUKRISHNAN, S. 1998. Flow and stretch metrics for scheduling continuous job streams. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 270–279.
- BENDER, M., MUTHUKRISHNAN, S., AND RAJARAMAN, R. 2002. Improved algorithms for stretch scheduling. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms*.
- CHEKURI, C., GOEL, A., KHANNA, S., AND KUMAR, A. 2004. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *ACM Symposium on Theory of Computing (STOC)*. 363–372.
- CHEKURI, C. AND KHANNA, S. 2002. Approximation schemes for preemptive weighted flow time. In *ACM Symposium on Theory of Computing (STOC)*. 297–305.
- CHEKURI, C., KHANNA, S., AND ZHU, A. 2001. Algorithms for weighted flow time. In *ACM Symposium on Theory of Computing (STOC)*. 84–93.
- CHEKURI, C., MOTWANI, R., NATARAJAN, B., AND STEIN, C. 1997. Approximation techniques for average completion time scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 609–618.
- HALL, L. A., SCHULZ, A. S., SHMOYS, D. B., AND WEIN, J. 1997. Scheduling to minimize average completion time: Offline and online algorithms. *Math. of Operations Research* 22, 513–544.
- KALYANASUNDARAM, B. AND PRUHS, K. 2000. Speed is as powerful as clairvoyance. *Journal of the ACM* 47, 4, 617–643.
- KALYANASUNDARAM, B. AND PRUHS, K. 2003. Minimizing flow time nonclairvoyantly. *Journal of ACM* 50, 551–567.
- KELLERER, H., TAUTENHAHN, T., AND WOEGINGER, G. J. 1996. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *ACM Symposium on Theory of Computing (STOC)*. 418–426.
- LENSTRA, J., KAN, A., AND BRUCKER, P. 1977. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1, 343–362.
- LEONARDI, S. AND RAZ, D. 1997. Approximating total flow time on parallel machines. In *ACM Symposium on Theory of Computing (STOC)*. 110–119.
- MOTWANI, R., PHILLIPS, S., AND TORNG, E. 1994. Nonclairvoyant scheduling. *Theoretical Computer Science* 130, 1, 17–47.
- MUTHUKRISHNAN, S., RAJARAMAN, R., SHAHEEN, A., AND GEHRKE, J. 1999. Online scheduling to minimize average stretch. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 433–442.
- PHILLIPS, C. A., STEIN, C., TORNG, E., AND WEIN, J. 1997. Optimal time-critical scheduling via resource augmentation. In *ACM Symposium on Theory of Computing (STOC)*. 140–149.
- PRUHS, K., SGALL, J., AND TORNG, E. 2004. Online scheduling. Chapter 35, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press.
- SCHRAGE, L. 1968. A proof of the optimality of the shortest processing remaining time discipline. *Operations Research* 16, 678–690.
- SMITH, W. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* 3, 59–66.