Minimizing Weighted Flow Time

N. Bansal [1] K. Dhamdhere [2]

April 2002
CMU-CS-02-128

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We consider the problem of minimizing *weighted* flow time on a single machine in the preemptive setting. Our first result is an online algorithm which achieves a competitive ratio of k if there are k weight classes. Even for the special case of $k = 2$ this gives the first $O(1)$-competitive algorithm. Our algorithm also directly gives an $O(\log W)$ competitive algorithm when the maximum to the minimum ratio of weights is W. Our second result deals with the non-clairvoyant setting where the job sizes are unknown (but the weight of the jobs are known). In this case, we give a resource augmented algorithm. In particular, if the non-clairvoyant online algorithm is allowed a $(1 + \epsilon)$ speed-up, then it is $(1 + 1/\epsilon)$ competitive against an optimal offline, clairvoyant algorithm.

[1]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. nikhil@cs.cmu.edu
Supported by IBM Research Fellowship.
[2]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. kedar@cs.cmu.edu
This research was supported in part by NSF ITR grants CCR-0085982 and CCR-0122581.

Keywords: Weighted flow time, response time, online algorithms, job scheduling, single machine, preemption, non-clairvoyance, resource augmentation.

1 Introduction

We consider the problem of scheduling a collection of dynamically arriving jobs over time so as 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, thus it is the sum of times the job is waiting and its processing time. In the case 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 [18] is that the online algorithm that at any time schedules the job with the shortest processing remaining time (SRPT) minimizes the flow time on a single machine. However, not much is known for the weighted case. If W is the ratio of the maximum and minimum weight of the jobs, it is easy to see that SRPT is $O(W)$ competitive since it minimizes the number of jobs in the system. Chekuri *et al.* [8] recently gave the first non-trivial algorithm for minimizing the total weighted flow time. Their algorithm is semi-online[1] and achieves a competitive ratio of $O(\log^2 P)$, where P is the ratio of the maximum and minimum processing times of the jobs. However, the situation for the the weighted case is still far from satisfactory. For example, a natural question is how well can we do if there are just two weight classes (i.e. the jobs have weight 1 or w). Even for this simple case, no efficient algorithms were previously known. It turns out that the algorithm of [8] is $\Omega(\log P)$ competitive in this case.

Our main result is an online algorithm which is k competitive if the jobs belong to at most k different arbitrary weight classes. As a corollary, this yields a 2-competitive algorithm for the 2 weight case. This also gives us a $O(\log W)$ competitive algorithm, since we can simply round the weights up to a power of 2. This rounding can affect our solution by a factor of at most 2, and since we have $\log W$ classes, this yields a $2 \log W$ competitive algorithm. Our result is not directly comparable with the result of Chekuri *et al.* [8] since their bounds are in terms of P. However, our algorithm is fully online, in that it does not assume the knowledge of W or P while scheduling the jobs. Secondly, if the job weights do not vary as much as the job sizes or if there are a few number of weight classes, our algorithm gives the best known performance guarantee.

Our second result deals with the non-clairvoyant setting where the job sizes are 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 scenario, introduced by Motwani, Phillips and Torng [15], realistically models many situations, for example, in the case of Unix jobs it not clear what is the CPU requirement of the job when it arrives. As can be expected, the performance of non-clairvoyant algorithms is quite bad as compared with the optimum clairvoyant and offline adversary. Motwani *et al.* [15] show that even for the problem of minimizing unweighted flow time, every deterministic non-clairvoyant algorithm is $\Omega(n^{\frac{1}{3}})$ competitive and every randomized algorithm is $\Omega(\log n)$ competitive.

Kalyanasundaram and Pruhs [11] introduced the idea of augmenting the resources of the

[1] It is semi-online in the sense that it uses the knowledge of P while scheduling the jobs.

non-clairvoyant scheduler by increasing its speed. They consider the problem of minimizing the total unweighted flow time and give an online algorithm which is $(1+1/\epsilon)$ competitive, when provided with a $(1+\epsilon)$ times faster processor than the optimum offline clairvoyant algorithm. Resource augmentation has also been applied in clairvoyant settings. In particular, Becchetti *et al.* [4] recently gave a $(1+\frac{1}{\epsilon})$-speed, $(1+\epsilon)$-competitive algorithm for minimizing the weighted flow time in the clairvoyant setting.

In this paper, we give a $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-competitive algorithm for minimizing the total weighted response time in the non-clairvoyant setting. Thus our result can be viewed as an extension of the Kalyanasundaram–Pruhs [11] result for the weighted case, as well as an extension of Becchetti *et al.* [4] for the non-clairvoyant case. What is interesting is that our algorithm matches the performance of the algorithm of Becchetti *et al.* [4], while being *non-clairvoyant.*

Related Previous Work: We first discuss unweighted flow time. Smith showed the optimality of SRPT for unweighted flow time for a single machine [18]. Later, Schrage showed that SRPT in fact minimizes the number of jobs in the system at any time [17].

In the model of non-clairvoyant scheduling initiated by Motwani *et al.* [15], Kalyanasundaram and Pruhs [11] gave a $(1+\epsilon)$-speed, $(1+\frac{1}{\epsilon})$ competitive resource augmented algorithm for unweighted flow time. There analysis was later improved by Berman and Coulston, who showed that their algorithm is in fact a $(1+\epsilon)$-speed, $(\frac{2}{\epsilon})$ competitive (which yields better bounds than [11] if $\epsilon > 1$) [6]. In the absence of resource augmentation, Kalyanasundaram and Pruhs later gave a $O(\log n \log\log n)$ competitive randomized algorithm [12]. This was recently improved to $O(\log n)$ by Becchetti and Leonardi [2], which is the best possible competitive ratio achievable [15].

For the weighted case, Lenstra *et al.* showed that the problem is strongly $NP-hard$ [13]. Chekuri *et al.* gave the first non-trivial semi-online algorithm with an $O(\log^2 P)$ competitive ratio, where P is the ratio of the maximum to minimum job size. They also give a lower bound of 1.618 on the competitive ratio achievable by any online algorithm. Under the model of resource augmentation, Becchetti, Leonardi, Spaccamela and Pruhs give a $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$ competitive clairvoyant algorithm for minimizing weighted flow time [4]. They also consider a closely related problem, which they call the *deadline scheduling* problem and give an $O(1)$ algorithm for it. The problem involves minimizing the weight of jobs unfinished by some unknown deadline D.

In the context of approximation algorithms, Chekuri and Khanna [7] gave an $O(n^{O(\ln W \ln P/\epsilon^3)})$ time algorithm which computes a $(1+\epsilon)$ approximate solution. In fact, they show that when either W or P is polynomially bounded in n, their algorithm is a quasi-polynomial time approximation scheme. Surprisingly, the existence of a polynomial time $O(1)$ approximation algorithm is still unknown.

More work has been done for the special case when the weight of a job is the inverse of its processing time. This metric is commonly referred to as stretch or slowdown and is used widely to measure the performance of computer systems [9, 10, 14]. Muthukrishnan *et al.*

showed that SRPT is 2-competitive for minimizing total stretch in the single machine case, and 14-competitive for the case of multiple machines [16]. Subsequent improvements and extensions to their work can be found in [1, 3, 5, 8].

Model: We are given a set J of n jobs that are released over time. For a job $x \in J$, the quantities $r(x)$, $|x|$ and $w(x)$ denote the release time, processing time (or size) and the weight respectively. The flow time of x is $F(x) = C(x) - r(x)$, where $C(x)$ is the completion time of job x in the schedule. Our objective is to minimize $\sum_{x \in J} w(x)F(x)$, the total weighted flow time. We consider a single machine setting where job preemptions are allowed. In the clairvoyant setting we assume that processing time of a job is known upon its arrival, while in the non-clairvoyant setting we assume that the processing time is unknown throughout the time the job is present in the system. In either setting, the weight of a job is known when it arrives.

Organization: The rest of the paper is organized as follows. In Section 2 we give the intuition for the problem and describe our algorithm. In sections 3.1 and 3.2 we analyze the 2 weight case and then for the general case, where we show that our algorithm is k-competitive. Finally, in section 4 we give a non-clairvoyant algorithm that is $(1 + 1/\epsilon)$ competitive with a $(1 + \epsilon)$ speed up.

2 The Online Algorithm

Intuition: To get a feel for the problem, we first describe some simple algorithms and describe why they perform poorly. Secondly, we show by an example that the algorithm of Chekuri *et al.* [8] has a competitive ratio of $\Omega(\log P)$ even in the case when the job weights are either 1 or 2. We then give intuition for our approach and describe our algorithm.

Let O be the optimum algorithm and A some other algorithm. Let $w_o(t)$ and $w_a(t)$ denote the total weight of jobs present in O's and A's system respectively at time t. The total weighted flow time under A is given by $\int_0^\infty w_a(t)dt$. Notice that if $w_a(t) \leq cw_o(t)$ for some c at all times t, then A is c-competitive. In this case we say that the A is locally $c - competitive$. While local c-competitiveness is sufficient for global c-competitiveness, it also turns out to be necessary [4]. The idea is that if $w_a(t) > cw_o(t)$ at some time t, then the adversary can starting giving a stream jobs of size $\epsilon \to 0$ and weight 1 every ϵ units of time. This forces both O and A to work on the new stream. By making the stream long enough, a competitive ratio of greater than c can be achieved.

Thus it is essential to keep the total weight of the jobs in our algorithm low at all times. A natural strategy to consider is greedy, which at any time instant schedules the job with the highest weight to remaining time ratio. However this is easily shown to be $\Omega(\sqrt{P})$ competitive [8]. The idea is that breaking ties adversarially, the naive greedy algorithm can get stuck with a huge job while the optimum is not. Similarly, the algorithm which simply disregards the size of the job and works on the highest weight job is easily shown to be $P-competitive$ [8]. The $O(\log^2 P)$ competitive algorithm of Chekuri *et al.* deals with this problem by striking a balance. It processes a large weight job without regard to its ratio as long as the total weight of jobs with a strictly better ratio does not get too large. However, even when the job weights are just 1 or 2, it can be shown that their algorithm is $\Omega(\log P)$ competitive[2]. For completeness, their algorithm is described in Appendix A. We now describe our example.

Consider the following scenario: At time $t = 0$ two jobs, one of size P and weight 2 and other of size $P/2$ and weight 1 are released. For i ranging from 1 to $\log P - 1$, at each time instant $P(1-2^{-i})$ a job of weight 1 and size $P/2^{i+1}$ is released. Finally, at time $t = P(1-2^{i+1})$, another job of size $P/2^{i+1}$ is released.

At time $t = 0$, their algorithm will schedule the job of weight 2 and continue executing till time P. At this stage the weight of the jobs in the $Q(t)$ will be $(\log P+1)$. The optimum algorithm on the other hand works only on jobs of weight 1 and hence finishes all jobs except the one of weight 2 by time P. Thus the ratio of weights in the queues of the algorithm to that of the optimum will be $\frac{1}{2}(\log P + 1)$ at time t. At this stage the adversary can give a stream of jobs of weight 2 and size 1 forcing both the algorithms to work on new jobs. Thus, the algorithm has a competitive ratio of $\Omega(\log P)$ even for the 2 weight case.

The reason for the bad performance of the algorithm above is that it continues to work on the high weight job even when a lot of smaller weight jobs keep accumulating. Our algorithm will be careful in this respect and will in fact be 2-competitive for the 2 weight case. We next describe our algorithm and analyze it in Sections 3.1 and 3.2.

2.1 Algorithm Description

We present an k-competitive algorithm, that we call *Balanced SRPT* for minimizing weighted flow time. We assume that the weights of the jobs are drawn from the set $\{a_1, a_2, \ldots, a_k\}$, where $a_1 < a_2 \ldots < a_k$. We will refer to these as weight classes $1, \ldots, k$. We also assume that a_i/a_{i-1} is integral for $1 < i \leq k$. [3] Let X be a set of jobs. We use $Wg(X)$ to mean the sum of the weights of the jobs in X.

We partition the sets of jobs that are alive at a specific time based on the weight class. Let $Q_j(t)$ (or just Q_j when the time t is evident from the context) denote the set of alive jobs of class j at time t.

Balanced SRPT: At all times, pick a Q_j which has maximum $Wg(Q_j(t))$. (Break ties in favor of higher weight class j). Execute the job with shortest remaining processing time from Q_j.

Informally speaking, the algorithm tries to balance the weight in various weight classes,

[2] Their algorithm rounds up the weights of incoming jobs to an integral power of $4(\log P + 1)$. Hence, it is trivially loses a $4(\log P + 1)$ factor in competitive ratio. However, our example does not use this rounding to prove $\Omega(\log P)$ competitiveness.

[3] Arbitrary weights can be rounded up to an integral power of 2 and we get the required property of integral ratios, while losing only a factor of 2 in competitive ratio.

while running SRPT algorithm within each weight class.

3 Analysis

While our algorithm is extremely simple to describe, its analysis is quite involved. We first give an analysis for the special case for only 2 weight classes. Then we give a proof sketch for general case with k weight classes (Appendix B contains the full proof for the general case).

3.1 Analysis for the 2 Weight Case

Notation: Throughout the discussion, we will denote the Balanced SRPT algorithm by B and the optimum algorithm by *Opt*. Since preemptions are allowed, we assume without loss of generality that *Opt* is work conserving, i.e. it doesn't unnecessarily idle the processor. Let's assume that the two weight classes have weights 1 and w respectively. w is assumed to be an integer.

The idea of our proof will be as follows: We will form groups of jobs in B and also of those of jobs in *Opt*. The groups in B and *Opt* will be formed differently. In particular, a group in B can contain upto twice as more weight as a group in *Opt*. We will consider "prefixes", where prefix i will consist of the sum of remaining times of jobs in the first i groups in B and *Opt*. We then show that the prefixes of B always dominate those of *Opt* (the notion of domination will be made precise below). This will allow us to prove that the total weight at any time in B is not more than 2 times than the total weight in *Opt*.

In B, we form groups of jobs as follows: In each weight class sort all jobs in decreasing order of remaining time. In each group we greedily put 1 job of weight w and w jobs of weight 1. Thus if there are n_1 and n_w jobs of weight 1 and weight w respectively, then for $1 \leq i \leq \min(\lfloor \frac{n_1}{w} \rfloor, n_w)$, group i contains w jobs of weight 1 and 1 job of weight w. The groups with index higher than $\min(\lceil \frac{n_1}{w} \rceil, n_w)$ contain either weight 1 or weight w jobs only.

In *Opt*, the groups are formed in a different way: In each weight class, we sort the jobs in decreasing order of remaining time. For weight 1 jobs, w jobs taken together form a group. For weight w jobs, each job forms a separate group. Thus each group in Opt has weight w, except possibly for the last group of weight 1 jobs.

Let $S_i(w)$ denote the "prefix" for a weight w job in B defined as: sum of the remaining times of all jobs in first i groups. Let $S_i(1,j)$ denote the "prefix" for a weight 1 job in B defined as: sum of the remaining times of all jobs in first $i-1$ groups plus the remaining times of first j weight 1 jobs in ith group.

Note: $S_i(1,j) \leq S_i(w)$ since $S_i(w)$ includes all the jobs that are included in $S_i(1,j)$ and some more (viz. from $j+1$ to w of weight 1 jobs and a weight w job)

For *Opt* we define prefixes as follows: First consider an arbitrary ordering of groups of weight 1 jobs and groups of weight w jobs. An ordering is *valid* if the groups of any particular weight class are in the order of decreasing remaining time. Thus, in a valid ordering if we consider only groups containing jobs of weight w then these are present in the decreasing order of remaining time, similarly for groups containing jobs of weight 1. Let O denote some valid ordering of groups. Then, given this valid ordering, $T_i(w)$ (if i^{th} group is a job of weight w) is defined as the sum of remaining times of all jobs in first i groups. On the other hand, if i^{th} group contains weight 1 jobs, then we define $T_i(1,j)$ and the sum of remaining times of all jobs from first $i-1$ groups along with j jobs from group i. Strictly speaking the T_i's depend on O, but we will drop this in the notation, since O will always be explicitly mentioned whenever T_i's are used.

Abuse of notation: we use S_i's and T_i's to mean the sum of remaining times as well as the collection of jobs whose remaining time is being added.

Lemma 1 *Consider any arbitrary valid ordering O of Opt's groups, then at all times t and for all valid orderings O the following invariant holds: $S_i(w) \geq T_i(w)$ and $S_i(1,j) \geq T_i(1,j)$ (whichever one applies depending on whether group i in the ordering O contains weight w or weight 1 jobs).*

Proof: The proof involves showing that the conditions hold through various events in the system.

1. **Neither arrival or departure:** In this case, both B and Opt are working on jobs: If B is working on a weight w job in group l, then l is the last group and only $S_l(w)$ is decreasing. Moreover, $S_l(w)$ is total work in the system, thus $S_l(w) \geq T_l(w)$. If B was working on a weight 1 job, identical argument holds.

2. **Departure:** If B finishes a job, this is trivial since all other prefixes in B remain unchanged.

   If *Opt* finishes a job: Supposed it finished a job x of weight w job, then to prove that invariant holds for any arbitrary valid order O after x is finished we consider the valid order O appended with an additional group containing x at the end. Note that this is a valid order for jobs of *Opt* just before x finishes. Since the invariant was true at that time, it will still be true.

   Similarly, if *Opt* finishes a weight 1 job x then to prove the invariant for any arbitrary valid order O after x is finished we consider the valid order O' obtained from O by appending x in the end of the last group in O containing weight 1 jobs. Note that O' is a valid order unless the last group in O containing weight 1 jobs is full (i.e. has w jobs). Invoking the invariant on O' gives the result. In the case where the last group in O containing weight 1 jobs is full, we simply define O' to be O appended with another group in the end which just contains x. Again, invoking the invariant on O' gives the result.

3. **Arrival:** Let us denote the new job by x. We will also use x to denote its size. Let's suppose we want to argue the invariant after the arrival for an valid order O.

Observation 1 *If x is included in a prefix $S_i(w)$, then $S_i(w) \geq S'_i(w)$ (where $S'_i(w)$ denotes the old prefix, just before the arrival.) Similarly, if $x \in S_i(1,j)$, then $S_i(1,j) \geq S'_i(1,j)$. This follows from the fact that we keep the jobs sorted according to decreasing time.*

Observation 2 *If $x \notin S_i(w)$, then $S_i(w)$ remains unchanged (i.e. $S_i(w) = S'_i(w)$). This is true for $S_i(1,j)$, $T_i(w)$ and $T_i(1,j)$ as well.*

From observations 1 and 2, it's evident that, in the case when $x \notin T_i(w)$ or $x \notin T_i(1,j)$, the invariant holds trivially. So, hereafter we assume that $x \in T_i(w)$ or $x \in T_i(1,j)$ (whichever one applies depending on the valid order O)

Let x denote the new job. Let O' be the old order obtained by O by deleting x and possibly shifting the jobs upwards to fill up the *hole* created in O. Note that O' was a valid order just before the arrival of x. Let us denote B's prefixes after the arrival by S_i and just before the arrival by S'_i. To denote Opt's prefixes, we use T_i for the order O after the arrival and T'_i for the order O'.

Let us first consider arrival of a job of weight w. Fix an index i. If x is included in both $S_i(w)$ and $T_i(w)$. Then, from old valid order O', we know $S'_{i-1}(w) \geq T'_{i-1}(w)$ (or $S'_{i-1}(w) \geq S'_{i-1}(1,w) \geq T'_{i-1}(1,w)$). Also, $S_i(w) \geq S'_{i-1}(w) + x$ (inequality because also include remaining times of weight 1 jobs in level i) and $T'_i(w) = T'_{i-1}(w) + x$ or $T'_i(w) = T'_{i-1}(1,w) + x$. Thus, $S_i(w) \geq T_i(w)$.

If x is included in $T_i(w)$ but not in $S_i(w)$. In this case, $T_i(w) = T'_{i-1}(w) + x$ (or $T'_{i-1}(1,w) + x$, whereas $S_i(w) \geq S'_{i-1}(w) + w_i$, where w_i is the weight of weight w job in i^{th} group (the inequality holds because we include remaining times of weight 1 jobs in level i). Now $w_i \geq x$ (as x wasn't included in $S_i(w)$ and $S'_{i-1}(w) \geq T'_{i-1}(w)$ or $S'_{i-1}(w) \geq S'_{i-1}(1,w) \geq T'_{i-1}(1,w)$). It follows that $S_i(w) \geq T_i(w)$.

Now we argue that $S_i(1,j) \geq T_i(1,j)$.

If x is included in $T_i(1,j)$, but not in $S_i(1,j)$. Then $T_i(1,j) = T'_{i-1}(1,j) + x$. And $S_i(1,j) = S'_i(1,j) \geq S'_{i-1} + w_{i-1}$ (where w_{i-1} is remaining time of weight w job in group $i-1$, again inequality holds since we also include some weight 1 jobs in going from $S'_{i-1}(1,j)$ to $S'_i(1,j)$. Combine these with $w_{i-1} \geq x$ (since x was not included in $S_i(1,j)$). Thus we get $S_i(1,j) \geq T_i(1,j)$.

If x is included in both $T_i(1,j)$ and $S_i(1,j)$. Then once again, $T_i(1,j) = T'_{i-1}(1,j) + x$. Note that, $S_i(1,j)$ includes all the jobs of weight w from $S'_{i-1}(1,j)$ and job x. Thus $S_i(1,j) \geq S_{i-1}(1,j) \geq S'_{i-1}(1,j) + x$ and the invariant follows.

Thus we have shown that the invariant continues to hold for the case when the new job has weight w.

We now discuss the arrival of a job of weight 1: Let O be an valid order of Opt's groups for which we want to argue the invariant. As previously, let O' be the old valid order just before the arrival of the new job x. That is, to obtain O' from O, delete the job x and move up all the weight 1 jobs by one place while keeping the weight w jobs at the same place (If the number of groups of weight 1 decreases on deleting x, we might also have to move up some weight w jobs to obtain O').

Fix an index i. Consider $S_i(1,j)$ and $T_i(1,j)$.

When x is included in $S_i(1,j)$, we have $S_i(1,j) = S_i'(1,j-1) + x$ (for $j > 1$) and $S_i(1,j) = S_{i-1}'(w) + x \geq S_{i-1}'(1,w) + x$ (for $j = 1$, inequality holds since there is an extra job of weight w getting counted in $S_i(1,1)$).

Similarly, if x is included in $T_i(1,j)$, then $T_i(1,j) = T_i'(1,j-1) + x$ (for $j > 1$ and when $j = 1$, $T_i(1,j) = T_{i-1}'(1,w) + x$ or $T_{i-1}'(w) + x$ (as the case may be).

If x is included in both $S_i(1,j)$ and $T_i(1,j)$. If $j > 1$, then $S_i'(1,j-1) \geq T_i'(1,j-1)$, thus giving us $S_i(1,j) = x + S_i'(1,j-1) \geq x + T_i'(1,j-1) = T_i(1,j-1)$. If $j = 1$, then $S_i'(1,1) = S_{i-1}'(w) + x \geq T_{i-1}'(w) + x = T_i(1,1)$ or $S_i'(1,1) \geq S_{i-1}'(1,w) + x \geq T_{i-1}'(1,w) + x = T_i(1,1)$ (as the case may be)

If x is not included in $S_i(1,j)$, but included in $T_i(1,j)$. Then $S_i(1,j) = S_i'(1,j) = S_i'(1,j-1) + w_{ij}$ for $j > 1$ and $S_i(1,j) = S_{i-1}'(w) + w_{ij}$ (where w_{ij} denotes remaining time of j^{th} weight 1 job in i^{th} group). Note that $w_{ij} \geq x$ since x is not included in S_{ij}. Thus, it follows that $S_i(1,j) \geq T_i(1,j)$.

Now, let us compare $S_i(w)$ and $T_i(w)$. Consider the following valid order O'': rearrange first i groups of O such that all the groups corresponding to weight w precede all the groups of weight 1. All the groups after i are kept as in O. Let's use T_i'' to denote *Opt*'s prefixes in O''. Note that $T_i''(1,w) = T_i(w)$, since rearrangement won't change the sum. Now for O'' we can argue as above that $S_i(1,w) \geq T_i''(1,w)$. And we know that $S_i(w) \geq S_i(1,w)$, which gives the desired result.

Thus we have showed that the invariant continues through all possible events and thus holds at all times t. □

We now use the lemma to prove 2-competitiveness of the Balanced SRPT algorithm.

Theorem 1 *If the jobs come from* 2 *weight classes,* Balanced SRPT *is* 2*-competitive.*

Proof: Consider the ordering of groups in *Opt* where all the groups of weight w are preceded by the groups containing weight 1 jobs. Let i denote the number of groups in *Opt*.

If there are no weight 1 jobs in *Opt*, then $T_i(w)$ equals the total work in system and the total weight of jobs contained in Opt is iw. Lemma 1 gives us that $S_i(w) \geq T_i(w)$. Since $T_i(w)$ is also the total remaining work in *Opt* (which is the same as the total remaining work in B), we get that $S_i(w) = T_i(w)$ and hence total weight in B is at most $2iw$.

If group i contains $j \geq 1$ jobs of weight 1, then since the total remaining work in *Opt* is $T_i(1,j)$, by Lemma 1 we get that $S_i(1,j)$ contains all the jobs in B. Now the total weight in *Opt* is $(i-1)w + j$ where as the total work in B is at most $2(i-1)w + j$.

Thus, overall B is locally 2-competitive, which ensures global 2-competitiveness. □

3.2 Analysis for the General Case

In the general case, we have k different weight classes $\{a_1, \ldots, a_k\}$. We will show that our algorithm Balanced SRPT is k competitive. To prove this we need to modify the analysis for the 2 weight case in a few ways. Firstly, the idea of forming separate groups of jobs for each weight class in *Opt* does not give us a strong enough invariant to prove k-competitiveness. So we extend the idea of groups so that *Opt*'s each group can contain jobs from different classes. Moreover, we allow fractional inclusion of a job in a group and we impose the condition that at most one of *Opt*'s group can have weight less than a_k. For B we form the groups in the same way as for the 2 weight case. That is, in each weight class, sort all jobs in decreasing order of remaining time and greedily put a_k/a_j jobs of class j. We then define a notion of "domination" between the prefixes and show that the groups of B dominate the groups of *Opt*. To prove this dominance we will also need a new notion of a "dummy" job (made precise below). Finally, we note that the idea of groups in the analysis for the 2 weight case was just used to give intuition for our analysis. In the following discussion we do not explicitly define the groups, but these will be implicit in the new notation and the arguments that follow.

We now give some notation for describing some useful quantities in the general case.

Define an order on 2-tuples of integers: $(j, l) \prec (j', l')$ iff $a_j \cdot l \leq a_{j'} \cdot l'$ for $j < j'$ or $a_j \cdot l < a_{j'} \cdot l'$ for $j \geq j'$. Also, we say $(j', l') \preceq (j, l)$ if $(j, l) \not\prec (j', l')$. Note that $(j', l') \preceq (j, l)$ and $(j, l) \preceq (j', l')$ holds iff $j = j'$ and $l = l'$, and that $\preceq$ defines a total order on the 2-tuples.

Intuitively, the notion $\preceq$ formalizes the order in which our algorithm B executes the jobs. If there were l jobs of class j and l' jobs of class j', then $(j, l) \prec (j', l')$ means that algorithm B will execute a job of class j' first.

Let $Q_B(t)$ denote the set of jobs that aren't finished by algorithm B by time t. We define a set of jobs $B(j, l, t)$ as follows:
$B(j, l, t) = \{x \in Q_B(t) \mid \ class(x) = j' \ \& \ x$ has l'th largest remaining time in class $j' \& (j', l') \preceq (j, l)\}$

Let $|B(j, l, t)| = \sum_{x \in B(j,l,t)} rem(x, t)$, where $rem(x, t)$ is the remaining time of job x at time t. (For clarity of notation, we will drop t from the expressions when it is clear from the context)

We want to compare the "prefixes" $|B(j, l)|$ with suitable prefixes of *Opt*.

Let $Q_{Opt}(t)$ denote the set of jobs that are not finished by *Opt* by time t. Along with these jobs, we also want to include some *"dummy"* jobs in the *Opt*'s prefixes. A *dummy* job of class j, is a job with remaining time 0. Let $\hat{Q}_{Opt}(t)$ denote the set $Q_{Opt}(t)$ along with some dummy jobs thrown in. These dummy jobs play a role only in simplifying the proofs.

Consider a permutation π of jobs in $\hat{Q}_{Opt}(t)$. Call π a *valid ordering* of $\hat{Q}_{Opt}(t)$ if jobs i & j belong to same class then $\pi(i) < \pi(j) \Rightarrow rem(\pi(i)) \geq rem(\pi(j))$. We don't impose this condition if the jobs i and j don't belong to same class. Intuitively, all the jobs from same class in $\hat{Q}_{Opt}(t)$ appear in π in descending order of their remaining times.

For a job $x \in \hat{Q}_{Opt}(t)$ and a valid ordering π of $\hat{Q}_{Opt}$, we define $Opt_\pi(x,t) = \{y \in \hat{Q}_{Opt}(t) \mid \pi(y) \leq \pi(x)\}$. In other words, $Opt_\pi(x,t)$ contains the jobs that precede x in the valid ordering π. We call $Opt_\pi(x)$ a "prefix" of Opt under the valid ordering π. Let j be the class of job x and let $l = \lceil Wg(Opt_\pi(x))/w(x) \rceil$. We will frequently use $Opt_\pi(j,l,t)$ as a synonym for $Opt_\pi(x,t)$. And $|Opt_\pi(j,l)|$ just denotes the sum of remaining times of the jobs in $Opt_\pi(j,l)$. It's important to note here that, $Opt_\pi(j,l)$ may not be defined for all pairs (j,l) unlike that for $B(j,l)$.
Finally, let $Rem(B,t) = \sum_{x \in Q_B(t)} rem(x,t)$ and $Rem(Opt,t) = \sum_{x \in Q_{Opt}(t)} rem(x,t)$.
With this notation in place, we can now state the "domination" property of prefixes as follows.

Lemma 2 *At all times t, for any valid ordering π of $\hat{Q}_{Opt}(t)$, for $1 \leq j \leq k$ and $l \geq 1$, the following holds*

$$|B(j,l)| \geq |Opt_\pi(j,l)|$$

whenever the latter is defined.

The proof of this lemma goes along the similar lines as that of lemma 1 with some modifications required to prove the stronger invariant. A complete proof is given in Appendix B. In particular, the lemma is also true for any valid ordering π of $Q_{Opt}(t)$. We use this lemma to prove our following main result.

Theorem 2 *Balanced SRPT is a k-competitive online algorithm for minimizing total weighted flow time.*

Proof: We will prove that B is locally k-competitive, which gives us the desired result.

Fix a time t. Suppose algorithm B was working on a job x at the time t. Let $j = class(x)$. For $1 \leq i \leq k$, let n_i denote the number of jobs of class i in $Q_B(t)$. In particular, x is n_jth job of class j. For $\neq j$, we know that $(i, n_i) \prec (j, n_j)$. Thus $a_j \cdot n_j \geq a_i \cdot n_i$ for all i s.t. $1 \leq i \leq k$. Hence we get,

$$Wg(B) \leq k \cdot n_j \cdot a_j \tag{1}$$

Now we derive a lower bound on $Wg(Opt)$. Consider an ordering π of Q_{Opt} in which jobs are ordering from the highest weight class to the lowest one. In particular, if y is the last job in π, then y has the lowest weight class (and smallest remaining time among it's weight class). Let $i = class(y)$, and let $l = \lceil Wg(Opt)/a_i \rceil$. Note that since i is the smallest weight class in Opt, $Wg(Opt)$ is an integral multiple of a_i and hence $l = Wg(Opt)/a_i$.

We first note that $(i,l) \succeq (j, n_j)$. This follows since, if $(i,l) \prec (j,n_j)$ then $x \notin B(j,n_j)$ and hence $B(i,l) < B(j,n_j)$, thus $Rem(B) = B(j,n_j) > B(i,l) \geq Opt_\pi(i,l) = Rem(Opt)$ and we get a contradiction. The third inequality above follows from Lemma 2 and final equality follows from our choice of π.

We can now show that $Wg(Opt) \geq n_j \cdot a_j$. Suppose $Wg(Opt) < n_j \cdot a_j$, then $a_i l < a_j n_j$. For either case, when $i < j$ or $i \geq j$, the definition of the relation $\prec$ gives us that $(i,l) \prec$

(j, n_j). Thus we have,

$$\mathrm{W}g(Opt) \geq n_j \cdot a_j \qquad (2)$$

From equations (1) and (2), local k-competitiveness of Balanced SRPT follows. $\square$

Finally, we show that the competitive ratio for the Balanced SRPT algorithm is tight. Consider following job instance, in which for $1 \leq j \leq k$, 2^{k-j} jobs of weight 2^j arrive at time 0. The job of weight 2^k has size 1 while rest of them have sizes $\epsilon = 2^{-k}$. Balanced SRPT would start working on the job of weight 2^k and continue working on it till the job finishes.

Consider the scenario at time $t = (2^k - 2)\epsilon < 1$. At t, the total weight under Balanced SRPT is $k2^k$. Where as, the optimal algorithm can finish all the jobs of size ϵ first and then work on the job of class k. In this case, the optimal algorithm has a weight of a_k. Since the algorithm is locally k competitive, we can make it globally k competitive, but giving a long stream of weight 2^k and size 2^{-k} jobs.

4 Non-clairvoyant scheduling for weighted response time

In this section, we consider the non-clairvoyant case where the processing time of a job becomes known only when it finishes. We now give a resource augmented $(1+\epsilon)$ speed $(1 + 1/\epsilon)$ competitive algorithm. We call it the *Weighted Foreground Background* (WFB) algorithm as it resembles the commonly used Foreground Background (FB) algorithm. Note that the competitive ratio is independent of the number of weight classes and also of the ratio of the jobs sizes. Moreover, it generalizes the results of [11] and [4] while still matching their bounds both in terms of resource augmentation and competitive ratio. In terms of proof techniques, our analysis builds up on the ideas in [11].

4.1 Algorithm Description

For a job J_i, let $p_i(t)$ denote the amount of work done on that job by time t. And let w_i denote the weight of the job J_i. We define a norm of job J_i as $\|J_i\|_t = \frac{p_i(t)}{w_i}$

Algorithm **WFB**: At all times, WFB splits the processor, proportional to weights of the jobs, among the jobs J_i that have the smallest norm $\|J_i\|_t$. So, if $J_1, \ldots, J_k$ are the jobs with WFB that have the smallest norm. Then the job J_j will receive $w_j/(\sum_{i=1}^k w_i)$ fraction of the processor.

Note that, for all jobs J_i which WFB executes, the norm increases at the same rate and thus stays same. Moreover, if at some point of time t', we have $\|J_i\|_{t'} \geq \|J_j\|_{t'}$, then for all times t in future (i.e. $t \geq t'$), while both jobs are alive, we have $\|J_i\|_t \geq \|J_j\|_t$.

4.2 Analysis

For an algorithm A with a speed s processor, let $W_A(t, s)$ denote the sum of weights of the jobs that have arrived before time t, but have not been finished by time t.

The analysis of WFB algorithm hinges on proving the following lemma:

Lemma 3 *At any point t in time, we have the following relation:*

$$W_{WFB}(t, 1+\epsilon) \leq (1 + \frac{1}{\epsilon})W_{Opt}(t, 1)$$

In other words, $(1+\epsilon)$-speed WFB is locally $(1 + 1/\epsilon)$-competitive w.r.t. an adversary with a unit speed processor.

The lemma will be proved in argument that follows.

Let's fix a time t. Let V denote the set of jobs which WFB has in its queue but the adversary has finished by time t. And let U denote the set of jobs which the adversary has in its queue at time t. Let $Wg(V) = \sum_{i \in V} w_i$ and $Wg(U) = \sum_{j \in U} w_j$.

We want to bound $Wg(V)$ in terms of $Wg(U)$.

We say that a job J_i can *immediately borrow* from a job J_j ($J_i \hookleftarrow J_j$) if WFB ran J_j at some time t' satisfying $r_i \leq t' \leq c_i$. We define the *borrow* relation as the transitive closure of immediately borrow. (We also denote this by $J_i \leftarrow J_j$)

Lemma 4 *If $J_i \hookleftarrow J_j$, then we have $\|J_i\|_t \geq \|J_j\|_t$.*

Proof: It suffices to prove the assertion for immediately borrow relation. So assume that J_i immediately borrows from J_j. From the property of WFB, $\|J_i\|_{t'} \geq \|J_j\|_{t'}$ must have been true at some time t'. Thus from the observation we made above, the lemma follows. □

We partition the set V into $V_1, V_2, \ldots, V_k$ such that $J_i, J_j \in V_{k'} \iff \|J_i\|_t = \|J_j\|_t$. We also require that, if $J_i \in V_{i'}$, $J_j \in V_{j'}$ and $i' < j'$ iff $\|J_i\|_t > \|J_j\|_t$. Let $\|V_i\|$ denote $\|J_i\|$, where $J_i \in V_i$.

Intuitively, all jobs with same norm are in the same partition. Also, V_1 has the jobs with largest norm, V_2 has jobs with second largest norm, and so on.

Let $U' = \{J_j \in U \mid \exists i \in V \ s.t. \ J_i$ borrows from $J_j\}$. These constitute the jobs among U, on which WFB worked while it had some jobs from V still in queue.

We note that, WFB has a faster processor with speed $(1 + \frac{1}{\epsilon})$. And while adversary finished all jobs in the set V by time t, WFB spent at least $\sum_{i \in V} \epsilon \cdot \|J_i\|_t$ amount of work on jobs in U. In fact, the definition U' is such that, it contains all the jobs from U on which WFB worked when there was some job from V in WFB's queue.

Let us define $U'_i = \{J_j \in U' \mid \|V_i\|_t \geq \|J_j\|_t > \|V_{i+1}\|_t\}$ (for notational convenience, assume that $\|V_{k+1}\|_t = 0$). By virtue of lemma 4, we know that $\|V_1\|_t \geq \|J_j\|_t$, for all jobs $J_j \in U'$. Thus, $\bigcup_i U'_i$ is indeed U'.

Moreover, note that for $J_i \in V_i$ and $J_j \in U'_j$, J_i borrows from J_j is possible only if $i \leq j$.

Let t_i was the time when a job from V_i arrived first. Note that $t_1 < t_2 < \ldots < t_k$.

Consider an arrival time t_m. Since the time t_m, *Opt* has finished all jobs from V that arrived after time t_m. So the amount of work done by *Opt* $\geq \sum_{j=m}^{k} \sum_{i \in V_j} w_i \|J_i\|_t$. So the amount of work done by $(1+\epsilon)$-WFB $\geq \sum_{j=m}^{k} \sum_{i \in V_j} (1+\epsilon) w_i \|J_i\|_t$. WFB hasn't finished any of the jobs from V. So work done by WFB on the jobs from $\bigcup_{m \leq j \leq k} V_j$ is $\leq \sum_{j=m}^{k} \sum_{i \in V_j} w_i \|J_i\|_t$. Thus WFB has worked at least $\sum_{j=m}^{k} \sum_{i \in V_j} \epsilon w_i \|J_i\|_t$ on jobs in $\bigcup_{m \leq j \leq k} U'_j$ after time t_m. Call this work T_m.

For $1 \leq m \leq k$ we have

$$T_m \geq = \sum_{j=1}^{k} \sum_{i \in V_j} \epsilon w_i \|V_j\|_t \tag{3}$$

For a job $l \in U'_j$, work done on l, by time t is $\|J_l\|_t \leq \|V_j\|_t$, by the definition of U'_j. Thus summing up over jobs in $\bigcup_{m \leq j \leq k} U'_j$, we get

$$T_m \leq \sum_{j=m}^{k} \sum_{l \in U'_j} w_l \|V_j\|_t \tag{4}$$

We now use following inequality in conjunction with 3 and 4 to derive desired result.

Lemma 5 *Let $y_1 \geq y_2 \geq \ldots \geq y_k > 0$. Let $x_i, z_i \geq 0$ for $1 \leq m \leq k$ and $\sum_{i=m}^{k} x_i y_i \geq \sum_{i=m}^{k} z_i y_i$. Then $\sum_{i=1}^{k} x_i \geq \sum_{i=1}^{k} z_i$.*

Proof: Let $X_m = \sum_{i=m}^{k} x_i y_i$ and $Z_m = \sum_{i=m} z_i y_i$. We know, $X_m \geq Z_m$. For notational convenience let $y_0 = \infty$. Then,

$$\sum_{i=1}^{k} x_i = \sum_{i=1}^{k} \left(\frac{1}{y_{i-1}} - \frac{1}{y_i}\right) X_m \geq \sum_{i=1}^{k} \left(\frac{1}{y_{i-1}} - \frac{1}{y_i}\right) Z_m = \sum_{i=1}^{k} z_i$$

(Each $(1/y_{i-1} - 1/y_i) \geq 0$ since $y_i \leq y_{i-1}$ for $1 \leq i \leq k$.) □

Now, we set $x_j = \sum_{i \in U'_j} w_i$, $z_j = \sum_{i \in V_j} \epsilon w_i$ and $y_j = \|V_j\|_t$. The lemma 5 tells us that

$$Wg(U) \geq \sum_{j=1}^{k} \sum_{i \in U'_j} w_i \geq \epsilon \sum_{j=1}^{k} \sum_{i \in V_j} w_i = \epsilon Wg(V)$$

Proof:(Lemma 3) Let $U_B(t)$ denote the set of jobs that WFB has at time t and $U_A(t)$ denote the set of jobs with Opt. Then from the preceding discussion, $Wg(U_B - U_A) \leq 1/\epsilon Wg(U_A)$. In other words, we have proved $W_{WFB}(t, 1+\epsilon) \leq (1+1/\epsilon)W_{Opt}(t, 1)$. $\square$

Thus we have proved the following theorem.

Theorem 3 *WFB is a $(1+\epsilon)$-speed $(1+1/\epsilon)$-competitive non-clairvoyant algorithm for minimizing weighted flow time.*

5 Open problems

Regarding the weighted flow time, it will be very interesting to obtain a constant competitive online algorithm or to provide a better lower bound, since the best known lower bound is only 1.618. The existence of a constant factor approximation algorithm is also open for the offline case.

Another natural problem is to minimize the weighted flow time in the non-clairvoyant without speedup. The randomized algorithms of [12] and [2] for the unweighted case can be shown to achieve a competitive ratio of $O(W \log n \log\log n)$ and $O(W \log n)$ for the weighted case. We believe that it should be possible to obtain an $O(\log W \log n)$ competitive randomized algorithm using our techniques.

Finally, in the non-clairvoyant weighted response time case, it might be possible to prove a better competitive ratio than $(1+1/\epsilon)$, as done by [6] for the unweighted case.

References

[1] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. In *ACM Symposium on Theory of Computing*, pages 198–205, 1999.

[2] L. Becchetti and S. Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *ACM Symposium on Theory of Computing (STOC)*, pages 94–103, 2001.

[3] L. Becchetti, S. Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *Symposium on Discrete Algorithms*, pages 548–557, 2000.

[4] L. Becchetti, S. Leonardi, A. M. Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. In *RANDOM-APPROX*, pages 36–47, 2001.

[5] M. Bender, S. Muthukrishnan, and R. Rajaraman. Improved algorithms for stretch scheduling. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[6] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. *Nordic Journal of Computing*, 6(2):181–193, 1999.

[7] C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. In *ACM Symposium on Theory of Computing (STOC)*, 2002.

[8] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for weighted flow time. In *ACM Symposium on Theory of Computing (STOC)*, 2001.

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

[10] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley, New York, 1991.

[11] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[12] Bala Kalyanasundaram and Kirk Pruhs. Minimizing flow time nonclairvoyantly. In *IEEE Symposium on Foundations of Computer Science*, pages 345–352, 1997.

[13] J. Lenstra, A. Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[14] M. Mehta and D. J. DeWitt. Dynamic memory allocation for multiple-query workloads. In *International Conference on Very Large Data Bases*, pages 354–367, 1993.

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

[16] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. Gehrke. Online scheduling to minimize average stretch. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 433–442, 1999.

[17] L. Schrage. A proof of the optimality of the shortest processing remaining time discipline. *Operations Research*, 16:678–690, 1968.

[18] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

Appendix A

The $O(\log^2 P)$ algorithm of Chekuri *et al.* **[8]:**
Let the set of jobs have weights drawn from the set $\{w_1, w_2, \ldots, w_{max}\}$. Assume $w_1 = 1$ and $w_i = (4(\log P + 1))^{i-1}$. For a job x, let $p_t(x)$ denote the remaining time of x at time t, and let $w(x)$ denote the weight of x. If $w(x) = w_i$, we say that the *weight index* of x is i. The *class* of a job is $\lfloor \log(p_t(x)/w(x)) \rfloor$. A job x belongs to the set $Q_{p,q}(t)$ if the weight index of x satisfies p and the class of x at time t satisfies q. $W_{p,q}(t)$ denotes the total weight of jobs in $Q_{p,q}(t)$. Their algorithm works as follows.

At each discrete time step t, choose a job in $Q(t)$ to execute by the following rules:

1. $i \leftarrow$ largest weight index in $Q(t)$.

2. $k \leftarrow$ smallest class in job of weight w_i in $Q(t)$.

3. If $W_{<i,<k(t)} \leq w_i$, schedule a job of weight w_i and class k (a job from $Q_{i,k}(t)$).

4. Else $i \leftarrow$ largest weight index in $Q(t)$ strictly less than i. Go to step 2.

Appendix B

Proof:(Of lemma 2) We want to prove that, at all times t, for any ordering π of $\hat{Q}_{Opt}(t)$, for $1 \leq j \leq k$ and $l \geq 1$, the following holds

$$|B(j,l)| \geq |Opt_\pi(j,l)|$$

whenever the latter is defined. Note that $Opt_\pi(j,l)$ is defined only if $\exists y \in \hat{Q}_{Opt}(t)$ such that $Opt_\pi(y) = Opt_\pi(j,l)$

We prove the inequalities by induction over time. We know that the invariant holds when there are no jobs present just before time $t = 0$. Now we need to verify that, it still continues to hold at various events. We analyze following possible cases:

(1) **Neither arrival nor departure** If in an interval $[t_1, t_2)$ there are no arrivals and neither algorithm B nor Opt finishes any job. Let x be the job on which B is working. We observe that $|B(j,l,t_2)| < |B(j,l,t_1)|$ iff $x \in B(j,l,t_1)$. For $t \in [t_1,t_2)$, $x \in B(j,l,t)$ means that $\forall x' \in Q_B(t), x' \preceq x \Rightarrow x' \in B(j,l,t)$. Thus $|B(j,l,t)| = Rem(B,t)$. Thus, we can say that, $|B(j,l,t_2)| = \min(|B(j,l,t_1)|, Rem(B,t_2))$.

For a fixed ordering π of $\hat{Q}_{Opt}(t_1)$ and a job $y \in \hat{Q}_{Opt}(t_1)$ we have $|Opt_\pi(y,t_2)| \leq |Opt_\pi(y,t_1)|$. Moreover, $|Opt_\pi(y,t_2)| \leq Rem(Opt,t_2)$. Therefore, we have $|Opt_\pi(j,l,t_2)| \leq \min(|Opt_\pi(j,l,t_1)|\ ,\ Rem(Opt,t_2))$.

Putting the two together, it follows that $|B(j,l,t)| \geq |Opt_\pi(j,l,t)|$, for $t \in [t_1,t_2)$.

(2) **Departure** At time t, algorithm B finishes a job. Let $t' < t$. Observe that, $\forall x \in Q_B(t'), rem(x,t')$ is left continuous. Hence $|B(j,l,t')|$ is left continuous. If Opt hasn't finished a job at t, $|Opt_\pi(j,l,t)|$ is clearly left continuous. The inequality then follows.

If at time t, Opt finishes a job y. Let $t' < t, t' \to t$. Consider $\hat{Q}_{Opt}(t') = \hat{Q}_{Opt}(t) \bigcup \{y\} - \{$a dummy job of same class that of $y\}$. (If there is no dummy job with same class as that of y, then $\hat{Q}_{Opt}(t') = \hat{Q}_{Opt}(t) \bigcup \{y\}$. Let π' be the ordering of $\hat{Q}_{Opt}(t')$ obtained by appending y at the end of π. (If there were any dummy jobs in π in the same class as that of y, then y takes the place of the first such dummy job). Since $rem(y',t')$ is left continuous for every $y' \in \hat{Q}_{Opt}(t)$. Therefore, we have $|Opt_{\pi'}(y',t')| = |Opt_\pi(y',t)|$, in limit. (If there was a dummy job in $\hat{Q}_{Opt}(t) - \hat{Q}_{Opt}(t')$, then we also have $|Opt_{\pi'}(y,t')| = |Opt_{\pi}(\text{dummy},t)|$ in limit). This proves the left-continuity of all $|Opt_{\pi}(j,l,t)|$. If algorithm B hasn't finished a job at time t. $|B(j,l,t)|$ is also clearly left continuous since each of $rem(x,t)$ is left continuous for $x \in B(j,l,t)$.

(3) **Arrival** The most troublesome case is the arrival of a new job. In this case, neither $|B(j,l,t)|$ nor $|Opt_{\pi}(j,l,t)|$ will be left-continuous, in general.

Let x denote the newly arrived job. Let $|x|$ denote its size. And let $c = class(x)$. Let $t' < t, t' \to t$. So that by $|B(j,l,t')|$ etc, we mean the left limits. Since we arrange the jobs in decreasing order of remaining times within a class, we have the following:

Observation 3 *If $x \in B(j,l,t)$, then $|B(j,l,t)| \geq |B(j,l,t')|$. And if $x \notin B(j,l,t)$, then $|B(j,l,t)| = |B(j,l,t')|$.*

Using this observation, we can handle the case when $x \notin Opt_{\pi}(j,l,t)$. Consider $\hat{Q}_{Opt}(t') = \hat{Q}_{Opt}(t) \bigcup \{\text{a dummy job of class } c\} - \{x\}$. And let π' be an ordering on $\hat{Q}_{Opt}(t')$ s.t. $\forall i \geq 1$, $class(\pi(i')) = class(\pi(i))$. [4] Then, $x \notin Opt_{\pi}(j,l,t)$ means that $|Opt_{\pi}(j,l,t)| = |Opt_{\pi'}(j,l,t')|$. And hence using the inequality at time t' for the ordering π', we get $|B(j,l,t)| \geq |Opt_{\pi}(j,l,t)|$.

Therefore, we only need to consider the case when $x \in Opt_{\pi}(j,l,t)$. We split this in three different cases ($j > c$, $j = c$ and $j < c$). For each of these sub-cases, we use the inequality at time t' with a different ordering on $\hat{Q}_{Opt}(t')$.

Observation 4 *If $B(j,l,t)$ contains atleast one more job x' of class $c = class(x)$ than $B(j',l',t')$, then $|B(j,l,t)| \geq |B(j',l',t')| + |x|$.*

Proof: If $x \notin B(j,l,t)$, then $rem(x',t) \geq |x|$, thus giving us required result. Otherwise if $x \in B(j,l,t)$, then $B(j,l,t') = B(j,l,t') \bigcup \{x\} - \{x''\}$ for some job x'' s.t. $class(x'') = class(x)$ and $rem(x'')$ is the smallest among all the jobs of same class in $B(j,l,t')$. Thus $rem(x',t) \geq rem(x'',t)$ and the result follows. □

case (a) $j = c$. Let y be the job such that $Opt_{\pi}(y) = Opt_{\pi}(j,l,t)$. First, consider a different ordering π' of $\hat{Q}_(t)$, which reorders the jobs that precede y in the ordering π, so that in π', they are are in decreasing order of weight class (and within each class, in decreasing order of remaining times). Let z be the job in π' just preceding y. Let $Opt_{\pi'}(z,t) = Opt_{\pi'}(j',l',t)$. Then, $Wg(Opt_{\pi'}(z,t)) = Wg(Opt_{\pi'}(y,t)) - a_j$.

Now, if $j' \leq j$, it's easy to see that $(j',l') \preceq (j,l-1) \prec (j,l)$. On the other hand, if $j' > j$, then all the jobs preceding y have weight integral multiple of a_j. Hence $Wg(Opt_{\pi'}(z,t))$ is an integral multiple of a_j. Thus it follows that, $(j',l') \prec (j,l)$. It means that $B(j,l,t)$ contains atleast one more job of class j than $B(j',l',t')$. And $|B(j,l,t)| \geq |B(j',l',t')| + |x|$ follows from observation 4.

[4] Note that, specifying class of jobs at each place in π' defines it uniquely, as we require the jobs within each class to be in descending order of remaining time.

With this knowledge, we are now ready to use the inequality at time t'. Let $\hat{Q}_{Opt}(t') = \hat{Q}_{Opt}(t) \bigcup \{$a dummy job of class $c\} - \{x\}$. And let σ be an ordering of $\hat{Q}_{Opt}(t')$ s.t. $\forall i \geq 1$, $class(\sigma(i)) = class(\pi'(i))$. It's easy to see that $|Opt_\pi(y,t) = |Opt_{\pi'}(y,t)| = |Opt_\sigma(j',l',t')| + |x| \leq |B(j',l',t')| + |x| \leq |B(j,l,t)|$.

case (b) $j < c$. Let y be the job such that $Opt_\pi(y,t) = Opt_\pi(j,l,t)$. We also have $x \in Opt_\pi(y,t)$. x is not same as y. Now consider $\hat{Q}_{Opt}(t') = \hat{Q}_{Opt}(t) - \{x\}$. Let σ be an ordering of $\hat{Q}_{Opt}(t')$ obtained by just deleting x from π. Therefore, we have $|Opt_\sigma(y,t') + |x| = |Opt_\pi(y,t)|$. Let $Opt_\sigma(y,t') = Opt_\sigma(j,l',t')$. Then, we have $(j,l') \prec (c, \lfloor (a_j l)/a_c \rfloor) \prec (j,l)$. This means that, $B(j,l,t)$ contains atleast one extra job of weight c than $B(j,l',t)$. By applying, observation 4 we get the desired result.

case (c) $j > c$. Let y be the job such that $Opt_\pi(y,t) = Opt_\pi(j,l,t)$. Consider an ordering π' of $\hat{Q}_{Opt}(t)$ which reorders the jobs in $Opt_\pi(y,t)$ s.t. jobs of class c in $Opt_\pi(y,t)$ are all preceded by all other jobs. π' retains the order of the jobs not in $Opt_\pi(y,t)$ same as in π. Let z be the job of class c that ends up last among the jobs from $Opt_\pi(y,t)$ in ordering π'.

Clearly, $|Opt_\pi(y,t)| = |Opt_{\pi'}(z,t)| = |Opt_{\pi'}(c,l',t)|$ for some l'. From case (a), we get $|B(c,l',t)| \geq |Opt_{\pi'}(c,l',t)|$. And we can easily prove that $(c,l') \prec (j,l)$, whereby we get $|B(j,l,t)| \geq |B(c,l',t)| \geq Opt_\pi(j,l,t)|$.

Thus in all possible cases we have shown that the inequality holds.

$\square$