Approximation Schemes for Preemptive Weighted Flow Time

Chandra Chekuri*

Sanjeev Khanna†

ABSTRACT

We present the first approximation schemes for minimizing weighted flow time on a single machine with preemption. Our first result is an algorithm that computes a $(1+\epsilon)$-approximate solution for any instance of weighted flow time in $O(n^{O(\ln W \ln P/\epsilon^3)})$ time; here P is the ratio of maximum job processing time to minimum job processing time, and W is the ratio of maximum job weight to minimum job weight. This result directly gives a quasi-PTAS for weighted flow time when P and W are poly-bounded, and a PTAS when they are both $O(1)$. We strengthen the former result to show that in order to get a quasi-PTAS it suffices to have just one of P and W to be poly-bounded. Our result provides strong evidence to the hypothesis that the weighted flow time problem has a PTAS. We note that the problem is strongly NP-hard even when P and W are $O(1)$. We next consider two important special cases of weighted flow time, namely, when P is $O(1)$ and W is arbitrary, and when the weight of a job is inverse of its processing time referred to as the stretch metric. For both of the above special cases we obtain a $(1+\epsilon)$-approximation for any $\epsilon > 0$ by using a randomized partitioning scheme to reduce an arbitrary instance to several instances all of which have P and W bounded by a constant that depends only on ϵ.

1. INTRODUCTION

In this paper we address a fundamental optimization problem in scheduling: given jobs that arrive over time, find a schedule that minimizes the sum of their weighted flow time. The flow time of a job is the overall time it spends in the system including its waiting time and the processing time. Weights model the varying priorities that jobs might have. Weighted flow time is a natural and at the same time flexible (because of the arbitrary choice of weights) measure of performance for a system serving jobs. Special cases such as the total flow time and total stretch [13] have applications in databases, scheduling requests at web servers [8], operating systems [9] etc.

Despite substantial interest in this problem, progress on good upper bounds has been slow to come by. When no preemptions are allowed, even for the unweighted case it is provably hard to obtain a good approximation. Kellerer et al. [10] showed that single machine scheduling to minimize flow time is NP-hard to approximate within a factor better than $\Omega(n^{\frac{1}{2}-\epsilon})$. However, the approximability of the *preemptive* weighted flow time problem stayed wide open until very recently. Even for the related and somewhat more tractable problem of scheduling to minimize sum of weighted completion times, good approximation algorithms were discovered only in last several years. Scheduling to minimize (weighted) sum of completion times is equivalent to minimizing (weighted) sum of flow times in the exact case — the objective functions differ by an additive term that is independent of the schedule. The former measure is easier to approximate however and has been well explored in the last few years leading to a good understanding of the approximability of many variants. Constant factor approximations and polynomial time approximation schemes (see [7, 1] for references) are known. Unfortunately the techniques for approximating completion time do not seem to carry over to approximating flow time. For instance, linear programming relaxations [7] that provide good bounds for completion time scheduling are also valid relaxations for flow time but are either not easily amenable to analysis for flow time or have large integrality gaps.

Given the intractability of the non-preemtive problems, the focus has been on algorithms for the preemptive versions. Further, in many applications preemption is natural, desirable, and results in better overall system throughput. It is easy to show examples where the gap between non-preemptive and preemptive flow time for a given instance is quite large although this is not the case for completion time scheduling. In the unweighted case, a folklore result states that the online algorithm that schedules the job with the *shortest remaining processing time* (SRPT) gives the optimal total flow time on a single machine. Leonardi and Raz [12] analyzed SRPT for the multiple identical machine case and showed that it is $O(\min\{\log P, \log \frac{n}{m}\})$-competitive in the online setting; here P is the ratio of maximum and minimum job processing times, and n and m denote the number of jobs and machines respectively. This is also the best offline approximation algorithm known for the problem.

In contrast to the unweighted problem, the problem with weights is known to be strongly NP-hard on a single machine [11]. Until recently no non-trivial approximation algorithms

*Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974. E-mail: chekuri@research.bell-labs.com.

†Dept. of CIS, University of Pennsylvania, Philadelphia, PA 19104. E-mail: sanjeev@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.

STOC'02, May 19-21, 2002, Montreal, Quebec, Canada.

or online algorithms were known for this problem. Chekuri, Khanna, and Zhu [4], recently obtained several results. In particular they give a *semi-online* algorithm for a single machine that is $O(\log^2 P)$-competitive. The algorithm is semi-online only in that it assumes that the parameter P is known in advance. This also results in an $O(\log^2 P)$-approximation algorithm. They also give a quasi-polynomial time $(2+\epsilon)$-approximation for the case when the weights and processing times are polynomially bounded. A special case of the weighted flow time problem is the problem of minimizing total stretch. The stretch of a job is ratio of its flow time to its processing time. Hence minimizing total stretch is the same as minimizing weighted flow time where the weight of a job is the inverse of its processing time. Muthukrishnan et al. [13] showed that SRPT is 2-competitive and hence a 2-approximation algorithm for stretch on a single machine. They also showed that SRPT is $O(1)$-competitive for stretch on multiple machines. Further improvements and simplifications for minimizing stretch on multiple machines have been developed by Becchetti et al. [3] and Chekuri et al. [4].

Results: In this paper we further extend the understanding of the approximability of weighted flow time by presenting the first approximation schemes for the preemptive problem on a single machine. We present an algorithm that computes a $(1+\epsilon)$-approximate solution for any instance of weighted flow time in $O(n^{O(\ln W \ln P/\epsilon^3)})$ time; here W is the ratio of maximum job weight to minimum job weight. As immediate corollaries of this result, we get a quasi-PTAS for weighted flow time when P and W are poly-bounded, and a PTAS when they are both $O(1)$. Note that we are not assuming that P and W are integers, this is important because P and W are ratios. We then show that in fact, if either one of P or W is poly-bounded, then the other parameter can be effectively reduced to a poly-bounded range as well. Thus we get a quasi-PTAS whenever one of P or W is poly-bounded. Our result provides strong evidence that the weighted flow time problem has a PTAS. We next consider two natural restrictions of the problem. The first restriction is the case when P is $O(1)$ but there is no restriction on W, that is, the weights are arbitrary. The second restriction is the case when weight of a job is inversely proportional to its processing time, i.e. the stretch metric. For both cases we obtain a PTAS by using a partitioning scheme that reduces a given instance to instances with small P and W.

Approximating weighted flow time is a difficult problem even for very restricted cases because of the sensitivity of the metric to small variations in processing time. To put our results in perspective we note that no constant factor approximation algorithm was known even for the case when P and W are both $O(1)$. In fact the problem is strongly NP-hard even for this case. Furthermore, the natural relaxation that has a gap of at most 4/3 [14] for weighted completion time can be shown to have a gap of $\Omega(n^{1/3})$ for weighted flow time. However, our results indicate that, with appropriate care even approximation schemes can be obtained. We hope that these results and techniques will spur further work on weighted flow time and related problems leading to a better understanding of their approximability.

Techniques: The basic paradigm underlying our approximation schemes is to show existence of near-optimal schedules that are "well-structured". We can then use dynamic programming to search for good schedules in this restricted space. A natural approach for generating structured schedules is to not distinguish between jobs with similar characteristics and thus consider that they can be processed in some canonical order (say, in the order of arrival). Unfortunately, the flow time metric is very sensitive to even slight changes in the job sizes. In particular, it is easy to create instances where a small relative change in the size of a single job may greatly increase the optimal flow time value. This is in contrast to the completion time metric where it is straightforward to show that small perturbations in job sizes affect the optimal value only slightly. However, we establish the following relaxation that serves as a building block for our results. There exist near-optimal schedules such that jobs with "similar" processing times and weights can be executed "almost" in the order of arrival. Specifically, we show there exists a $(1+\epsilon)$-approximate schedule such that when the ith job in a set of similar jobs is being executed, only $O(1/\epsilon)$ jobs among the first i jobs are still alive.

The above result naturally leads to a dynamic programming based approximation scheme that runs in time $O(n^{O(\ln W \ln P/\epsilon^3)})$. Several additional ideas are necessary for obtaining a PTAS for the two restrictions that we consider, namely fixed P (and unrestricted weights), and stretch. At a high-level, our approximation schemes for both these cases rely on showing that we can partition any instance into a collection of instances each of which having the property that P and W are $O(1)$ such that the interaction between the jobs in the different instances is resolved by a simple priority based rule. We assign a distinct priority to each instance, and at any time execute a job from the highest priority instance. Our partitioning scheme makes use of randomization in a crucial way to ensure that in an expected sense, weight of any job in a higher priority instance strongly dominates the weight of any job in a lower priority instance. This dominance effectively allows us to "charge" the cost of preempting jobs from the lower priority instances to jobs of higher priority.

In related work Bender, Muthukrishnan, and Rajaraman [2] have independently obtained a PTAS for the special case of minimizing stretch on a single machine. We note that their result does not have any implication for the general weighted flow time problem. Also, their algorithm for stretch is different from ours in that they do not rely on an algorithm for weighted flow time like we do.

Organization: We present our quasi-PTAS in Section 2. In Section 3 we describe our randomized partitioning scheme. We build upon these ideas to give a PTAS for the case of fixed P in Section 4 and for stretch in Section 5. In Section 6, we show a polynomially large gap for a natural relaxation for preemptive weighted flow time. Finally, we conclude with some directions for future work in Section 7.

1.1 Notation

We call a schedule *busy* if no idle time is introduced while jobs are waiting. Without any loss of generality, we restrict our attention to busy schedules. For a job x we use $p(x)$ and $w(x)$ to denote its processing time and weight respectively. For a given schedule σ the quantity $p_t^\sigma(x)$ denotes the remaining processing time of x at time t in σ. The queue of a schedule σ at time t denoted by $Q^\sigma(t)$ is the set of all the jobs that are alive in σ at time t. If σ is clear from the context we drop it from the superscripts. For simplicity of notation we will assume for the rest of the paper that $1/\epsilon$ is an integer. We denote by P the ratio of the maximum processing time to the minimum processing time and by W the ratio of the maximum weight to the minimum weight. Without loss of generality we assume that the minimum processing time is 1 and that the minimum weight is 1. This allows us assume that $p(x) \in [1, P]$

and that $w(x) \in [1, W]$. Note that we are *not* assuming that processing times or weights are integers. When working with multiple instances of the problem we use $P(I)$ and $W(I)$ to denote the quantities P and W for instance I. For a schedule σ, let $|\sigma|$ denote the value of the schedule - the sum of weighted flow times of jobs scheduled according to σ.

2. A QPTAS FOR PREEMPTIVE WEIGHTED FLOW TIME

In [4] the notion of arrival ordered schedules was used to obtain a $(2+\epsilon)$ quasi-polynomial approximation. In this paper we develop a substantial refinement to the notion of arrival ordered schedules to obtain an approximation scheme. For a subset of jobs G and a schedule σ let $S_G^\sigma(t)$ denote the subset of G that is completed in σ by time t. We start with a technical lemma that is at the heart of our schemes. The lemma allows us to simplify the space of ϵ-approximate schedules substantially. We assume w.l.o.g. (by breaking ties arbitrarily) that no two jobs are released at the the same time.

DEFINITION 2.1. *A schedule σ is k-arrival ordered with respect to a set of jobs G if at all times t the following condition is true: if $x \in G$ is the job with the largest release date in $S_G^\sigma(t)$ then the number of jobs in G that have arrived before x and are still alive at t is at most k.*

The advantage of a k-arrival ordered schedule is captured by the following lemma.

LEMMA 1. *Let σ be a k-arrival ordered schedule with respect to G. The set $S_G^\sigma(t)$ can be specified by $(k+1)O(\log |G|)$ bits. Hence the number of distinct subsets of G that are alive at any time in σ is $O(|G|)^{k+1}$.*

PROOF. We order jobs in G in increasing order of their arrival (release time). The set $S_G^\sigma(t)$ can be specified by giving the following information: ℓ, the largest index among the completed jobs, and up to k indices, each less than ℓ, that specify the jobs that arrived before ℓ but have not been completed. The definition of a k-arrival ordered schedule makes it clear that this information is sufficient. $\Box$

Let $n_G^\sigma(t)$ denote the number of jobs from a set G that are alive at time t in schedule σ. Consider a set of jobs that have the same weight and similar processing times. Our lemma below indicates that there exist near optimal schedules that are k-arrival ordered with respect to such jobs, for a small value of k.

LEMMA 2. *Let G be a subset of the jobs such that $\frac{\max_{x\in G} p(x)}{\min_{x\in G} p(x)} < (1+\epsilon)$. For any schedule σ' there exists a corresponding schedule σ that is $(1+2/\epsilon)$-arrival ordered with respect to G and such that $n_G^\sigma(t) \le (1+2\epsilon)\cdot n_G^{\sigma'}(t)$. Further, σ and σ' agree completely on the execution of all jobs not in G.*

Fix a schedule σ'. We modify the schedule σ' to create a $(1+2/\epsilon)$-arrival ordered schedule σ with the claimed property. We say that a job $x \in G$ is *critical* if there exists a time t such that $x \in Q^{\sigma'}(t)$ and $n_G^{\sigma'}(t) \le 1/\epsilon$. A job x *becomes* critical at time t if t is the earliest time when $n_G^{\sigma'}(t) \le 1/\epsilon$ and $x \in Q^{\sigma'}(t)$. Note that even though a job x may become critical at some time much later after its release, we will consider it to be a critical job from the moment it arrives in the system. Thus G is partitioned into critical and non-critical jobs. We construct the schedule σ as follows. At all times t where σ' executes a job not in G, σ mimics σ'. However, within G, σ will use a (possibly) different order for executing jobs. Consider a time t when σ' is executing a job from G.

1. If at time t, σ' is executing a critical job x, then σ executes the same job. Thus σ and σ' completely agree in the execution of critical jobs.
2. Otherwise, σ executes the earliest unfinished job from G that is not critical.

In other words, σ completely agrees with σ' on critical jobs, and executes the non-critical jobs strictly in arrival order.

LEMMA 3. *At any time t, σ' has at most $2/\epsilon$ critical jobs from G.*

PROOF. Suppose that at some time t, the schedule σ' has a set S of more than $2/\epsilon$ critical jobs from G. Partition S into two sets S_1 and S_2 such that S_1 contains the jobs that have already become critical by time t and S_2 contains jobs that will become critical at some later time after t. We claim that S_1 and S_2 each must contain at most $1/\epsilon$ jobs each.

Suppose $|S_1| > 1/\epsilon$. Consider the job $x \in S_1$ that is the last job to have become critical, say at some time t'. Then every other job in S must be already present at time t', contradicting that x became critical at time t'.

Now suppose $|S_2| > 1/\epsilon$. Then consider the first job $x \in S_2$ to become critical at some time t'' after time t. Every other job in S_2 must still be alive in the system, contradicting that x becomes critical at time t''. $\Box$

Thus, in σ, at any time, there are at most $(1+2/\epsilon)$ partially executed jobs from G. We now argue that $n_G^\sigma(t) \le (1+2\epsilon)n_G^{\sigma'}(t)$. Consider a time t such that $n_G^{\sigma'}(t) \le 1/\epsilon$. Then each job in $Q^{\sigma'}(t)$ is critical. The schedule σ at t must completely agree with σ' in the execution of critical jobs and neither schedule carries any non-critical jobs at time t, hence the number of jobs is identical in both schedules. Otherwise for any time t such that $n_G^{\sigma'}(t) > 1/\epsilon$, define t_l as the latest time before t when σ' had at most $1/\epsilon$ jobs.

Let $\alpha(t)$ and $\beta(t)$ be the number of non-critical and critical jobs at time t in σ. Similarly let $\alpha'(t)$ and $\beta'(t)$ be the number of non-critical and critical jobs at time t in σ'. Then we know that $\beta(t) = \beta'(t)$. Further, at t_l, σ had no non-critical jobs and neither did σ' so $\alpha(t_l) = \alpha'(t_l) = 0$. In the interval $[t_l, t)$, σ spends exactly the same time as σ' in non-critical jobs. Moreover, σ never has more than one partially executed non-critical job. The processing times in G are within a factor of $(1+\epsilon)$, hence we can claim that $\alpha(t) \le \lceil (1+\epsilon)\alpha'(t) \rceil$. Thus if $\alpha'(t) + \beta'(t) > 1/\epsilon$ it follows from above that

$$\begin{aligned} n_G^\sigma(t) &= \alpha(t) + \beta(t) \\ &\le \lceil (1+\epsilon)\alpha'(t) \rceil + \beta'(t) \\ &\le 1 + (1+\epsilon)\alpha'(t) + \beta'(t) \\ &\le (1+2\epsilon)(\alpha'(t) + \beta'(t)) \\ &= (1+2\epsilon)n_G^{\sigma'}(t). \end{aligned}$$

The penultimate inequality is true since $\epsilon \cdot (\alpha'(t) + \beta'(t)) > \epsilon \cdot 1/\epsilon \ge 1$. This completes the proof of Lemma 2. $\Box$

We use Lemma 2 to show the existence of near-optimal schedules with good structure. Let G_{ij} be the set of jobs x such that $w(x) \in [(1+\epsilon)^i, (1+\epsilon)^{i+1})$ and $p(x) \in [(1+\epsilon)^j, (1+\epsilon)^{j+1})$.

LEMMA 4. *There exists a $(1+O(\epsilon))$-approximate schedule σ such that σ is $(1+2/\epsilon)$-arrival ordered with respect to each G_{ij}.*

PROOF. Let σ^* be an optimal schedule for the given instance. Note that the G_{ij}s partition the set of jobs. We apply Lemma 2 repeatedly to σ^* to create a schedule that is $(1+2/\epsilon)$-arrival ordered with respect to *each* G_{ij}. Let σ be the resulting schedule. Let $W^*(t)$ and $W(t)$ denote the weight of the jobs at time t in the queue's of σ^* and σ respectively. We have the following:

$$\begin{aligned} W(t) &\leq \sum_{i,j}(1+\epsilon)^{i+1} n^{\sigma}_{G_{ij}} \\ &\leq \sum_{i,j}(1+\epsilon)^{i+1}(1+2\epsilon) n^{\sigma^*}_{G_{ij}} \\ &\leq (1+2\epsilon)(1+\epsilon)\sum_{i,j}(1+\epsilon)^{i} n^{\sigma^*}_{G_{ij}} \\ &\leq (1+2\epsilon)(1+\epsilon)W^*(t). \end{aligned}$$

In the first inequality we are upper bounding the weight of jobs in G_{ij} by $(1+\epsilon)^{i+1}$ and in the third inequality we are lower bounding the weight by $(1+\epsilon)^i$. The second inequality follows from Lemma 2. □

The number of distinct sets, G_{ij}, is bounded by $O(\ln W \ln P/\epsilon^2)$. In Subsection 2.1 we show that for any partition of the jobs into ℓ sets there is an algorithm that computes an optimal schedule among schedules that are k-arrival ordered with respect to each of the ℓ sets, in time $n^{O(k\ell)}$. From this and Lemma 4 we get the following.

THEOREM 1. *Given an instance I of the problem of preemptive scheduling to minimize weighted flow time on a single machine, a $(1+\epsilon)$-approximate schedule can be computed in $n^{O(\ln W \ln P/\epsilon^3)}$ time.*

2.1 Computing k-arrival ordered schedules

Let $G_1, G_2, \ldots, G_\ell$ be a partition of the job set in to ℓ sets. We are interested in finding a schedule that is k-arrival ordered with respect to each of the sets. We describe a dynamic programming based algorithm to find an optimal schedule among such schedules that runs in time $n^{O(k\ell)}$ time.

The dynamic program maintains a table T of optimal schedule values for subsets of jobs specified by tuples from a set V. Each tuple $\bar{v} \in V$ is of the form $\langle j, Y_1, Y_2, \ldots, Y_\ell\rangle$ with the following interpretation: for $1 \leq i \leq \ell$, Y_i is a subset of G_i, and $j = \sum_{1\leq i\leq \ell}|Y_i|$ is the total number of jobs in the sets Y_i. The table entry $T[\bar{v}]$ stores the value of an optimal k-arrival ordered schedule for the job set $\cup_{i=1}^{\ell} Y_i$. We cannot of course compute table entries for all possible subsets since that would require computing values for 2^n tuples. Instead we take advantage of the fact that in a k-arrival ordered schedule the set of jobs that are completed at any time can be specified compactly. From Lemma 1, the distinct number of tuples in V is $O(n^{(k+1)\ell})$.

For a tuple $\bar{v} = \langle j, Y_1, Y_2, \ldots, Y_\ell\rangle$ let $|\bar{v}|$ denote j, the number of jobs completed. Suppose we have computed $T[\bar{v}]$ for all tuples $v \in V$ with $|\bar{v}| \leq j$. Let $\bar{u} = \langle j+1, X_1, X_2, \ldots, X_\ell\rangle$ be a tuple in V. We compute $T[\bar{u}]$ as follows. Let $X = \cup_{i=1}^{\ell} X_i$. For each job $x \in X$ let σ_x be an optimal k-arrival ordered busy schedule for X where x is the last job to finish in X. Note that σ_x might not be a feasible k-arrival ordered schedule in which case we set its value to ∞. It follows that $T[\bar{u}] = \min_x |\sigma_x|$. To compute σ_x we find the optimal schedule value for the set $X \setminus \{x\}$ using the table T. If $X \setminus \{x\}$ is not a valid tuple in V we set $|\sigma_x| = \infty$. We obtain the completion time of x in σ_x by constructing a busy schedule for X in which we give x the least priority (since x is supposed to finish last among X). If x does not finish last in the schedule constructed above we make σ_x invalid by setting $|\sigma_x|$ to ∞. From the above discussion we see that $T[\bar{u}]$ can be computed from prior values in T in $O(n^2 \log n)$ time.

THEOREM 2. *An optimal k-arrival ordered schedule with respect to sets $G_1, G_2, \ldots, G_\ell$ can be computed in $n^{O(k\ell)}$ time.*

2.2 Reductions to Poly-bounded Instances

Theorem 1 yields a quasi-polynomial time approximation scheme when both P and W are poly-bounded. We show here that if either P or W is poly-bounded then we can reduce the instance, in an approximation preserving fashion, to a poly-bounded instance. Thus it is sufficient for one of P and W to be poly-bounded. We assume without loss of generality that the schedule starts at time 0 and is busy through out. Also recall that we have normalized quantities such that the smallest processing time and smallest weight are 1: this implies that P and W are lower bounds on the optimum value.

Our first reduction considers the case when P is poly-bounded. Let J_s be the subset of jobs from J whose weight is less than $W/(n^3P)$, and let J_l be the set of remaining jobs. We claim that the value of optimal schedule for J_l is a good approximation for the value of an optimal schedule for J. Let σ' be an optimal schedule for J_l. We create a schedule for J by merging σ' with an arbitrary busy schedule for J_s. In the merging we always give preference to σ'. The completion time of any job in J_l is the same in σ and σ'. The completion time of a job from J_s in σ is at most nP, hence the total contribution of J_s to σ is bounded by $nP|J_s|W/(n^3P)$ which is at most W/n. Since W is a lower bound on the optimal schedule value the contribution of J_s is negligible. By construction, the ratio of weights in J_l is at most n^3P, hence if P is poly-bounded the ratio of weights in J_l is poly-bounded as well.

Next we consider the case when W is poly-bounded. Let J_s be the set of jobs from J whose processing times are less than $P/(n^3W)$, and let J_l be the remaining jobs. Let σ_s be an arbitrary busy schedule for J_s. Let σ_l be an optimal schedule for J_l. We create a busy schedule σ for J by merging σ_s and σ_l where we always give preference to σ_s. The contribution of σ_s is at most P/n. A job in J_l can be delayed in σ because of jobs in J_s by at most $nP/(n^3W) = P/(n^2W)$. Hence the contribution of the delay to the value of σ is at most P/n. Since P is a lower bound on the optimal schedule value this is negligible. In J_l the ratio of processing times is $O(n^3W)$ which is poly-bounded if W is poly-bounded.

3. INSTANCE PARTITIONING

We develop here a basic framework that will be used in subsequent sections for obtaining a PTAS for two special cases of weighted flow time that we mentioned earlier. Informally, the idea is to partition an instance I with jobs of arbitrary weights into a collection of instances $I_1, I_2, \ldots, I_k$ such that for $1 \leq i \leq k$, $P(I_i)$ and $W(I_i)$ are $O(1)$. The interaction between these instances is limited to a simple priority rule: at any time t execute a job from I_i only if there are no jobs alive in I_j for any $j > i$. We use this rule to identify blocks of time available for executing jobs in each instance I_i, and then use the following variant of Theorem 1 to compute a near-optimal schedule for each I_i in polynomial time.

THEOREM 3. *Given an instance I of preemptive scheduling to minimize weighted flow time on a single machine, and a sequence of $O(n)$ intervals that specify the blocks of time excluded for executing jobs in I, a $(1+\epsilon)$-approximate schedule can be computed in $n^{O(\ln W \ln P/\epsilon^3)}$ time.*

As a final step, we simply merge together the schedules for $I_1, \ldots, I_k$. For this to work we need to establish the existence of near-optimal schedules of this form.

3.1 Randomized Grouping

Let $\mathcal{G} = (G_1, G_2, \ldots, G_k)$ be an ordered partition of the given job set into subsets of jobs $G_1, \ldots, G_k$. We call $G_1, \ldots, G_k$ groups. For a job x we use $g(x)$ to denote the index of the group that contains it, hence $g(x) = i$ iff $x \in G_i$. Let $q(x)$ be a real valued parameter associated with a job x. For our purposes $q(x)$ will either be $w(x)$ or $p(x)$. The partitions we are interested in are based on grouping jobs according to geometrically separated q values. Given a parameter $a > 1$ we partition jobs in to $(G_1, \ldots, G_k)$ where G_i is the set of all jobs x with $q(x) \in [a^i, a^{i+1})$. It will be advantageous for our purposes if jobs in two groups G_j and G_{j+1} differed in their $q(x)$ values by a factor of roughly a. This is clearly infeasible for arbitrary instance since two values $a^i - \epsilon$ and $a^i + \epsilon$ fall in adjacent groups but differ by a small relative factor. To get around this, we use randomization to achieve an *expected* separation. For a parameter a our expected relative separation will only be logarithmic in a but that will suffice for our purposes.

Given a parameter a we randomize the geometric grouping procedure as follows. Pick a number $r \in [1, a)$ at random according to the probability distribution with density function $f(t) = \frac{1}{t \ln a}$. A job x with value $q(x)$ is in group G_k if $q(x) \in [ra^k, ra^{k+1})$. The group of x, $g(x)$, is now a random variable that takes two values, either $\lfloor \log_a q(x) \rfloor$ or $\lfloor \log_a q(x) \rfloor + 1$. The following claim is elementary.

CLAIM 1. *The expected value of $\frac{q(x)}{ra^{g(x)}}$ is $\frac{a-1}{\ln a}$ and that of $ra^{g(x)+1}$ is $\frac{(a-1)q(x)}{\ln a}$.*

We note that the randomized geometric grouping has been used in several contexts earlier. We will use the notation $\mathcal{G}_a^r$ to denote the partition induced by geometrically increasing values of a with shift r. The parameter $q(x)$ will be clear from the context. We make one more observation that will be useful.

PROPOSITION 1. *For a fixed a the number of distinct partitions induced by choosing r randomly in the interval $[1, a)$ is at most n.*

This allows us to derandomize algorithms based on randomized geometric grouping in a simple and obvious way.

3.2 Prioritized Schedules

Given a partition $\mathcal{G} = (G_1, G_2, \ldots, G_k)$ we call a schedule σ *$\mathcal{G}$-prioritized* if at any time t the job processed at t by σ is from the *highest* indexed group among alive jobs at t. If we restrict our attention to $\mathcal{G}$-prioritized schedules, the problem of scheduling the given instance can be partitioned into k separate instances $I_1, I_2, \ldots, I_k$ where the instance I_j is restricted to jobs in G_j. To get instance I_j, we compute an *arbitrary* busy schedule σ_{j+1} for jobs in $\cup_{i=j+1}^k G_i$. This partitions the time interval $[0, \infty)$ in to disjoint time intervals. In each of these intervals some job from $\cup_{i=j+1}^k G_i$ is alive and is being processed by σ_{j+1}. It is clear that in a $\mathcal{G}$-prioritized schedule, no job from G_j can be processed in the time intervals when σ_{j+1} is processing a job. The instance I_j consists of scheduling jobs in G_j subject to the constraint that no jobs are scheduled in the excluded time intervals given by σ_{j+1}. The number of excluded time intervals for any instance I_j is bounded by $n+1$.

Thus prioritized schedules allow us to partition the problem in to smaller and more manageable instances. In addition if the partition is induced by geometric rounding for some constant a, we have the advantage that each G_i has jobs with q values that differ by at most a relative factor of a. This allows us to reduce general instances with arbitrary q values to instances with fixed q values and excluded time intervals. We can now invoke Theorem 3 to compute $(1+\epsilon)$-approximate schedules in polynomial time.

4. A PTAS FOR FIXED P

In this section we give a PTAS for the case when W is arbitrary but P is $O(1)$. A slight modification to the proof of NP-hardness for the problem where P and W are arbitrary [11] shows that the problem remains strongly NP-hard for any fixed $P > 1$. In fact it is not known whether the case of $P = 1$ (that is all processing times are the same) is polynomial time solvable or not!

Our PTAS for this case is based on $\mathcal{G}_a^r$-prioritized schedules where the grouping is based on the weights of jobs. To obtain a $(1+\epsilon)$-approximation we use $a = e^{O(P/\epsilon)}$. The intuition behind our approach is as follows. Suppose the weights of jobs are "well separated", say by a factor larger than P (i.e. for any two distinct weights $w > w'$, we have $w/w' \geq P$). Since the processing times are between 1 and P, for any two jobs x and y such that $w(x) > w(y)$, we get that $w(x)/p(x) > w(y)/p(y)$. Thus, by giving preference to larger weight jobs, we give preference to jobs with larger weight to processing time ratio. It is known that in the absence of release dates this leads to an optimal schedule [15], however, when release dates are present, always giving preference to the best ratio job can lead to very poor schedules. We show that, when weights are well separated relative to P, this rule leads to schedules that are not too far from optimal. The proof is quite involved. Of course an instance need not have well separated weights. $\mathcal{G}_a^r$-prioritized schedules with $a = e^{O(P/\epsilon)}$ and r chosen randomly, allow us to assume that the weights in adjacent groups are, in the expected sense, separated by a factor of $\ln a = \Omega(P/\epsilon)$. However, jobs within the same group have to scheduled between themselves with care. We now present the technical details.

The main claim is that when r is chosen according to the distribution in Section 3.1 the expected value of an optimal $\mathcal{G}_a^r$-prioritized schedule is within a $(1+\epsilon)$ factor of the optimum. This claim is a corollary of the technical lemma below, and the rest of the section is devoted to its proof.

LEMMA 5. *Let σ^* be an optimal schedule. Then there exists a $\mathcal{G}_a^r$-prioritized schedule σ such that at any time t*

$$\sum_{x \in Q^\sigma(t)} w(x) \leq \sum_{x \in Q^*(t)} \left(w(x) + \frac{4P \cdot ra^{g(x)+1}}{a-1}\right).$$

In the above inequality, the left hand side is the sum of the weights of the jobs in the queue of σ at time t. The right hand side sums over the jobs in the queue of σ^* at time t, instead of simply the weight of each job, we charge an additional quantity to each job.

COROLLARY 1. *For $a = e^{4P/\epsilon}$ and r chosen according to the* *distribution in Section 3.1, the expected value of an optimal $\mathcal{G}_a^r$-prioritized schedule is at most* $(1+\epsilon)$OPT.

We prove the above corollary assuming Lemma 5.

PROOF. For fixed r and a, let σ_a^r be a schedule guaranteed by Lemma 5. Then it follows that

$$|\sigma_a^r| = \sum_t \sum_{x \in Q^{\sigma_a^r}(t)} w(x) \leq \sum_t \sum_{x \in Q^*(t)} (w(x) + \frac{4P \cdot ra^{g(x)+1}}{a-1}).$$

Taking expectation on both sides with respect to the distribution on r and using Claim 1 we obtain that

$$\begin{aligned} \mathbf{E}\left[|\sigma_a^r|\right] &\leq \sum_t \sum_{x \in Q^*(t)} (w(x) + \frac{4P}{\ln a} w(x)) \\ &\leq \sum_t \sum_{x \in Q^*(t)} w(x)(1+\epsilon) \\ &\leq (1+\epsilon)\text{OPT}. \end{aligned}$$

□

The proof of Lemma 5 relies on transforming an optimal schedule σ^* to a $\mathcal{G}_a^r$-prioritized schedule σ. In this process jobs from a given group G_i can get delayed because of jobs from $\cup_{j>i} G_j$. We rely on a charging scheme to account for this delay. Ideally we would like to charge at most P jobs from G_i to each job in $\cup_{j>i} G_j$. This appears to be feasible because the processing times are bounded by P. However, we are also concerned with jobs that might be in a partial state of execution whose remaining processing times could be very small. We establish the following lemma that shows that even among the partially executed jobs the rate at which an optimal schedule finishes weight is upper bounded by roughly $2W$. Let $Q_e^*(t)$ denote the set of partially executed jobs in the queue of σ^* at time t. Hence, for $t' > t$, $Q_e^*(t) \setminus Q_e^*(t')$ is the set of jobs from $Q_e^*(t)$ that have been finished in the interval $[t, t']$ by σ^*.

LEMMA 6. *Let σ^* be an optimal schedule for an instance with processing times bounded by P and weights bounded by W. Then for $t' \geq t$,*
$\sum_{x \in Q_e^*(t) \setminus Q_e^*(t')} w(x) < 2W \left\lfloor \sum_{x \in Q_e^*(t) \setminus Q_e^*(t')} p_t(x) \right\rfloor + 2W.$
The lemma holds even when the instance has excluded time intervals.

PROOF. We can assume that $Q_e^*(t) \setminus Q_e^*(t')$ is non-empty. Suppose the claim is false for some pair of times t and t', $t < t'$. Let $|Q_e^*(t) \setminus Q_e^*(t')| = k$ and let $J_1, J_2, \ldots, J_k$ be the jobs ordered in reverse order of arrival. This is the order in which they will finish in σ^* since they are in a preemption chain. We claim that there exist indices i and j, $1 \leq i \leq j \leq k$, such that $\sum_{i \leq \ell \leq j} w(J_\ell) \geq W$ and $\sum_{i \leq \ell \leq j} p_t(J_\ell) < 1$. This will suffice to contradict the optimality of σ^*, as follows. Since J_i is in $Q_e^*(t)$ it must have been preempted by some job x that finishes before J_i. However we know that $p(x) \geq 1$ and $w(x) \leq W$. We can modify σ^* by swapping the execution of x with that of the remaining processing times of J_i to J_j and it is clear that this will strictly improve the schedule contradicting the optimality of σ^*.

Now we prove the claim. Let $y = \sum_{x \in Q_e^*(t)} p_t(x)$. For any integer a such that $0 < a \leq \lceil y \rceil$ let $i_a = \min\{h \mid \sum_{1 \leq f \leq h} p_t(J_f) > a-1\}$ and let $j_a = \max\{h \mid \sum_{1 \leq f \leq h} p_t(J_f) < a\}$. From our assumption on the falsity of the lemma it follows that there exists an a' such that $\sum_{i_{a'} \leq \ell \leq j_{a'}} w(J_\ell) \geq 2W$. Since the weight of every job is at most W, $\sum_{i_{a'} < \ell \leq j_{a'}} w(J_\ell) \geq W$. Further, from the definition of $i_{a'}$ and $j_{a'}$, we get that $\sum_{i_{a'} < \ell \leq j_{a'}} p_t(J_\ell) < 1$, providing the indices for the contradiction above. □

We now prove Lemma 5. Let $(G_1, G_2, \ldots, G_k)$ be the partition of jobs in $\mathcal{G}_a^r$. Given an optimal schedule σ^* we create a $\mathcal{G}_a^r$-prioritized schedule σ iteratively from $i = k$ down to 1. Let $A_i = \cup_{j=i}^k G_j$ denote the set of jobs in G_i to G_k. Suppose we have already created a schedule σ_{i+1} for jobs in A_{i+1}. We create a schedule σ_i for A_i as follows. Since σ_i needs to be prioritized, the jobs in G_i can be scheduled only in the time intervals in which no job in A_{i+1} is alive. We order jobs in G_i by their completion time in σ^* and schedule them preemptively in the gaps left in σ_{i+1} where at any time t, the earliest alive job in the ordering of σ^* is executed. Note that in σ_i jobs from G_i need not finish in same order as in σ^*. It is easy to see that $\sigma = \sigma_1$ is $\mathcal{G}_a^r$-prioritized.

CLAIM 2. *At any time t in the schedule σ_i,*

$$\sum_{x \in Q^{\sigma_i}(t)} w(x) \leq \sum_{x \in Q^*(t) \cap A_i} (w(x) + 4P \sum_{j=i+1}^{g(x)} ra^j).$$

Lemma 5 follows in a straightforward fashion from the above claim by setting $i = 1$. We prove the claim by induction on i as it goes down from k to 1. From the construction of σ_k, it is not difficult to see that $Q^{\sigma_k}(t) \subseteq Q^*(t)$. This establishes the base case. Assuming that the claim is true for $i+1$, we prove it for i. Let $B_i^*(t) = Q^*(t) \cap G_i$ be the set of jobs from G_i that are alive in σ^* at t. Let $B_i(t) = Q^{\sigma_i}(t) \cap G_i$ be the set of jobs from G_i that are alive at t in σ_i. From the construction it follows that $Q^{\sigma_j}(t) \cap G_i = B_i(t)$ for all $j \leq i$. Therefore

$$\sum_{x \in Q^{\sigma_i}(t)} w(x) = \sum_{x \in Q^{\sigma_{i+1}}(t)} w(x) + \sum_{x \in B_i(t)} w(x).$$

We bound $\sum_{x \in Q^{\sigma_{i+1}}(t)} w(x)$ by the induction hypothesis for $i+1$. Hence we need to bound $\sum_{x \in B_i(t)} w(x)$. In particular we need to bound $\sum_{x \in B_i(t)} w(x) - \sum_{x \in B_i^*(t)} w(x)$. We do this by bounding the quantity $\sum_{x \in B_i(t) \setminus B_i^*(t)} w(x)$. Recall that we order the jobs in G_i by their completion times in σ^*. Without loss of generality let $J_1, J_2, \ldots, J_h$ be the ordering of jobs in G_i. Let $\ell(t)$ be the number of jobs that are finished by σ^* by t from G_i and let $f(t)$ be the index of the first job from G_i that is not completed in σ_i at time t. It follows that

$$\sum_{x \in B_i(t) \setminus B_i^*(t)} w(x) \leq \sum_{f(t) \leq j \leq \ell(t)} w(J_j).$$

We now bound the rhs in the above inequality. Let t' be the last time before t when σ_i either executes a job that is later in the order than $f(t)$ or is empty in jobs. If there is no such time then we set t' to 0. A couple of facts that are true at t'. First, there are no jobs from A_{i+1} at t' in the queue of σ_i, hence also in σ_{i+1}. This is true because σ_i is prioritized. Second, there is no job from $\{J_1, \ldots, J_{f(t)}\}$ that is in the queue of σ_i at time t' since in σ_i we give preference to lower indexed jobs from G_i. From this latter fact it also follows that t' is before the arrival time of $J_{f(t)}$. Let V denote the difference in the total time that σ_i spent and σ^* spent in the interval $[t', t)$ on jobs in A_{i+1}. We claim that $V > 0$. Suppose not. Then, in the interval $[t', t)$ σ_i processed only those jobs from $J_1, \ldots, J_{f(t)}$ that have arrived after t'. Since we process jobs in the order of σ^* we must have finished $J_{f(t)}$ by t as σ^* did. However, by the definition of $f(t)$, σ_i did not finish $J_{f(t)}$ at t — a contradiction. Since job sizes are bounded by P and at time t' there are no jobs from A_{i+1} in σ_i, we get the following.

CLAIM 3. *There are at least $\lceil V/P \rceil$ jobs from A_{i+1} alive at t in σ^* (that is in $Q^*(t)$).*

The main ingredient of our proof is the following. Let $W_i = ra^{i+1}$, the maximum weight of jobs in G_i.

CLAIM 4. $\sum_{f(t) \le j \le \ell(t)} w(J_j) \le 2W_iV + 2W_i$.

PROOF. Let $Q^*_e(t')$ denote the set of preempted jobs in σ^* at time t'. Let $X = \{J_j \mid 1 \le j < f(t), J_j \text{ arrived after } t'\}$ be the set of jobs in G_i that arrived after t' but were finished by both σ^* and σ_i by time t. Let $X' = \{J_j \mid 1 \le j \le \ell(t), J_j \text{ arrived after } t'\}$ be those jobs from G_i that arrived after t' and were finished by σ^*. Therefore, $X' \setminus X$ is the set of jobs that arrived after t' but were finished by σ^* and not by σ_i by time t. Let $V' = \sum_{x \in X' \setminus X} p(x)$ be the amount of processing time that σ^* spent in excess over σ_i in the interval $[t', t)$ in jobs from G_i that arrived after t'. V is the total time that σ^* spent in excess over σ_i in the interval $[t', t)$ in jobs from G_i. It follows that σ^* spent $V'' = V - V' + p_t(J_{f(t)}) - p(J_{f(t)})$ time in processing jobs from $Y = Q^*_e(t') \cap \{J_1, \ldots, J_{\ell(t)}\}$. Note that Y is the set of preempted jobs from G_i that are alive at t' in $Q^*(t)$. Since $p_t(J_{f(t)}) - p(J_{f(t)}) < 0$ it follows that $V'' < V - V'$. Let $w_1 = \sum_{x \in X' \setminus X} w(x)$ and $w_2 = \sum_{x \in Y} w(x)$. We claim that $w_1 \le W_iV'$ since each job in $X \setminus X'$ has a processing time of at least 1 and was fully processed in the interval $[t', t)$. Invoking Lemma 6 we see that $w_2 \le 2W_iV'' + 2W_i$. Therefore we conclude that $w_1 + w_2 \le 2W_iV + 2W_i$. $\square$

Now we are ready to finish the induction step. We charge the excess weight in σ_i at time t which is at most $2W_iV + W_i$ to the $\lceil V/P \rceil$ jobs in A_{i+1} that are alive at t. We charge each job in A_{i+1} with a weight of at most $4PW_i$. We omit details of the calculations that formally prove the induction hypothesis. This finishes the proof of Claim 2 and Lemma 5.

THEOREM 4. *A $(1+\epsilon)$-approximation for the weighted flow time can be found in time $n^{O(P \ln P/\epsilon^4)}$. Hence for fixed P we obtain a PTAS.*

PROOF. From Lemma 5 we conclude that if we choose a to be $e^{4P/\epsilon}$ and pick r according the distribution in Section 3.1, then the expected value of a prioritized schedule is at most $(1+\epsilon)$ times the optimal schedule value. For the chosen r and a we compute a $(1+\epsilon)$ approximate $\mathcal{G}^r_a$-prioritized schedule. This involves finding a $(1+\epsilon)$-approximate schedule for each group G_i in the partition. However jobs in a group have max to min processing time at most P and max to min weight at most a. We use Theorem 3 to obtain the running time that we claim. $\square$

5. A PTAS FOR STRETCH

The stretch metric is a special case of weighted flow time where the weight of a job x with processing time $p(x)$ is $1/p(x)$. Note that neither the weights nor the processing times are fixed for a general instance of stretch and hence ideas from Section 4 do not apply directly. However, by taking advantage of the special structure of the stretch weight function we show that prioritized schedules with respect to weight based partitions provide good approximations.

We start with a lemma that establishes the quality of prioritized schedules as a function of the grouping parameter a. For the stretch metric it is more natural to work with the processing times rather than the weights. Let $\mathcal{G}^r_a = (G_1, G_2, \ldots, G_k)$ be the partition induced by grouping jobs based on their *processing times*. However, we want to give priority to jobs with larger weights. Since weights are inversely related to processing times, this implies that lower indexed groups in $\mathcal{G}$ have higher priority. For the rest of this section a schedule σ is $\mathcal{G}^r_a$-prioritized if at any time t, σ processes the job from the *lowest* indexed group from $\mathcal{G}$ that has alive jobs at t.

LEMMA 7. *The optimal stretch of a $\mathcal{G}^r_a$-prioritized schedule is at most* $\text{OPT} + \frac{a}{a-1}\sum_x \frac{p(x)}{ra^{g(x)+1}}$. *Choosing r according to the distribution $f(t) = \frac{1}{t \ln a}$ in $[1, a)$ the expected value of an optimal $\mathcal{G}^r_a$-prioritized schedule is at most* $\text{OPT} + \frac{n}{\ln a}$.

For the proof of the above lemma we need the following simple properties of optimal schedules for stretch.

PROPOSITION 2. *Let σ^* be any optimal schedule for total stretch on a single machine. For any two jobs x and y the following hold:*

1. *if x is completed before y in σ^* then y is not processed during the entire time x is alive in σ^*, and*
2. *if $p(x) < p(y)$ and y finishes earlier than x then $p_t(y) < p(x)$ where t is the arrival time of x.*

Fix r and a. We prove Lemma 7 by induction on k, the number of groups in the partition $\mathcal{G}$. We will prove the stronger hypothesis that there is a $\mathcal{G}^r_a$-prioritized schedule of value at most $\text{OPT} + \sum_x p(x) \sum_{i=g(x)+1}^{k} \frac{1}{ra^i}$. The claim is trivial for $k = 1$. Assuming it is true for all $j \le k$, we prove it for $k+1$. Let I be an instance with $k+1$ groups in the partition and let σ^* be an optimal schedule for I. If σ^* is prioritized with respect to $(\cup_{j=1}^{k} G_j, G_{k+1})$, by induction, the claim is true for $k+1$. Suppose not. Then there exists an earliest time t and a job $x \in G_{k+1}$ being processed at t such that x is not preempted by a job x' that arrives at t and $g(x') \le k$.

We first claim that $p_t(x) < ra^{k+1}$ for otherwise from Proposition 2, x would have been preempted at t by x'. We claim that x is the first job in G_{k+1} that finishes after t in σ^*. Suppose not. Let y be a job from G_{k+1} that finishes before x in σ^*. Then, from Proposition 2, y could not have been alive at t, hence must have arrived at or after t. If x was not preempted by x' it follows that x finishes before x', hence y finishes before x'. However, this is impossible since y is larger than x' and arrives no earlier.

Let t' be the first time when σ^* executes a job from G_{k+1} after completing x. If there is no such t' then let $t' = T$, the time at the end of the schedule.

From the discussion above, in the time interval $[t, t')$, σ^* executes either x or a job from $\cup_{i=1}^{k} G_i$. Let S be the set of jobs that are processed by σ^* in $[t, t')$ other than x. We claim that every job in S is completed by t', or in other words there is no job from S that is partially executed at t'.

CLAIM 5. *At t' there is no preempted job whose group is less than $k+1$.*

PROOF. If $t' = T$, the time at the end of the schedule, we are done. Let y the the job from G_{k+1} scheduled by σ^* at t'. We consider two cases based on whether y was partially executed before t or not. Suppose y was not processed before t. Then it must be that $p_{t'}(y) = p(y)$ and hence σ^* would not schedule y at t' if there was a smaller job available with group k or less. Suppose y was processed before t and the claim is false. Let $z \in Q^*(t')$ be a job with $g(z) \leq k$ and such that it has been partially executed before t'. From the choice of t there were no jobs at t from $G_1, \ldots, G_k$, therefore z arrived at t or later. Since y was alive before t it implies that z was processed while y was alive, and at t', y is processed while z is alive. By Proposition 2, this is a contradiction to the optimality of σ^*. $\square$

We modify the schedule σ^* in $[t, t')$ as follows. We find the best prioritized schedule σ' for $\mathcal{G}$ restricted to S. This step is feasible since S is completely processed in $[t, t')$. We schedule x in the gaps left by σ'. Since the maximum group index in S is k, we can apply the induction hypothesis to upper bound the value of σ'. To complete the proof we need to account for the delay in the processing of x. Whenever x is in the queue while σ' is processing a job from S we charge the weight of x, which is at most $\frac{1}{ra^{k+1}}$, to the job in S that is being executed. After the modification the schedule is prioritized in the interval $[0, t')$. The state of the schedule at t' after the modification is exactly the same as in σ^*. Further, the modification charged extra weight only to jobs in S, which are completed before t'. Therefore by continuing the process on the portion of σ^* in the interval $[t', T]$ we can change it into a prioritized schedule while charging only those jobs that finish after t'. Easy calculations verify the induction hypothesis. This finishes the proof of Lemma 7. $\square$

COROLLARY 2. *For $a = e^{1/\epsilon}$ there is a $(1+\epsilon)$-approximate $\mathcal{G}_a^r$-prioritized schedule.*

PROOF. Follows from Lemma 7 and the fact that OPT $\geq n$ for total stretch. $\square$

From the above we obtain a PTAS for stretch.

THEOREM 5. *A $(1+\epsilon)$-approximation for stretch on a single machine can be found in $n^{O(1/\epsilon^5)}$ time.*

PROOF. From Corollary 2 the expected value of prioritized schedules with $a = e^{1/\epsilon}$ is within $(1+\epsilon)$ of the optimal. Therefore, we can partition the problem in to several instances where each instance has processing times, and hence weights also, bounded by a. We find a $(1+\epsilon)$-approximation to such instances using Theorem 3 in time $n^{O(\ln^2 a/\epsilon^3)}$ time. There are at most n such instances which gives us the desired running time. $\square$

6. GAP OF A NATURAL RELAXATION

As we remarked earlier, the flow time metric is closely related to the completion time metric in the exact case. Optimum values differ by an additive term. Hence integer programming formulations and their relaxations for the two measures are essentially the same. Dyer and Wolsey [5] considered several formulations for the single machine problem in the context of non-preemptive schedules for completion time. In particular, they considered two time-indexed formulations, the weaker of which is a valid relaxation for preemptive schedules. This relaxation is based on discretizing time into unit intervals. Some of the other formulations can be shown to be equivalent to this formulation, hence we focus on this particular one. For the purposes of this section we will assume that all processing times and release dates are integers. For job x, let $r(x)$ denotes its release date. Let T be an upper bound on the time at which the schedule ends — this can be computed from the given instance trivially. Let A_{xt} be 1 if x is processed in the interval $[t, t+1]$ and 0 otherwise. Let $C(x)$ denote the completion time of x. Then the relaxation is the following:

$$\min \sum_x w(x)C(x) - \sum_x w(x)r(x)$$

subject to:

$$\begin{aligned}
\sum_{r(x) \leq t < T} A_{xt} &= p(x) && \text{forall } x && (1)\\
\sum_x A_{xt} &\leq 1 && 0 \leq t < T && (2)\\
C(x) &\geq \frac{p(x)}{2} + \frac{1}{p(x)} \sum_{t=0}^{T-1} A_{xt}(t + \frac{1}{2}) && \text{forall } x && (3)\\
A_{xt} &\in \{0, 1\}. && && (4)
\end{aligned}$$

The first constraint expresses the fact that the total processing time allocated to x in the interval $[r(x), T]$ is exactly $p(x)$. The second constraint says that at most one job is processed in each unit time interval. The third constraint lower bounds $C(x)$ when x is viewed as being continuously finished. For a clear and detailed explanation of this constraint see [5, 6]. Finally we require that only one job be processed in each unit time interval. If we remove the term $\sum_x w(x)r(x)$ from the the objective function we get a relaxation for minimizing completion time.

Note that the relaxation always yields a preemptive schedule, however, the value of the schedule according to the relaxation is not the same as the flow time of the schedule. Although the formulation is not strongly polynomial in the input, it can solved in $O(n \log n)$ time [5] even with the variables A_{xt} constrained to be binary. In fact Goemans [6] shows that the following simple greedy online algorithm yields an optimum solution to the relaxation. At any time t, among the jobs that are alive execute the job with the largest weight to processing time ratio. For completion time, Schulz and Skutella [14] showed that the integrality gap of the above relaxation is at most 4/3 and at least 8/7.

Here we show that the integrality gap of the formulation is $\Omega(n^{1/3})$ for weighted flow time. We consider the following instance. A large job with x with $p(x) = L = n^{2/3}$ and $w(x) = W = n^{1/3}$ is released at time 0. Starting at time $L-W$, n small jobs each with processing time 1 and weight 1 are released at unit intervals of time.

It is not hard to verify that the schedule which finishes all the jobs in arrival order (large job first and then the small jobs) is an optimal schedule for this instance. The value of this schedule is easily calculated to be $\Theta(n^{4/3})$.

However the above schedule is not optimal for the relaxation. An optimal schedule for the relaxation preempts the large job on arrival of the small jobs since their weight to processing time ratios are larger. In this schedule the large job finishes last after all the small jobs are completed. Again, simple calculations show that the value of the relaxation on this schedule is $O(n)$. Hence the gap is a factor of $\Omega(n^{1/3})$. Note that the flow time value of this schedule is $\Theta(n^{4/3})$ as well.

7. CONCLUSIONS

Our results are strong evidence that a PTAS exists for the problem of preemptive scheduling to minimize weighted flow time on a single machine. Resolving this is the obvious open problem. An interesting intermediate goal is to obtain a polynomial time constant factor approximation algorithm. Even though we have a quasi-polynomial time approximation scheme for the problem, the best known polynomial time approximation guarantee is only a poly-logarithmic factor [4]. Part of the difficulty is the lack of relaxations that provide good lower bounds for flow time scheduling. The multiple machine case for the weighted problem is wide open, we do not know any non-trivial algorithms even for the two-machine case.

8. REFERENCES

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

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

[3] L. Becchetti, S. Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 548–57, 2000.

[4] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for weighted flow time. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, 2001.

[5] M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates. *Discrete Applied Mathematics*, 26:255–70, 1990.

[6] M. X. Goemans. A supermodular relaxation for scheduling with release dates. In *Proceedings of IPCO*, LNCS volume 1084, 288-300, 1996.

[7] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Offline and online algorithms. *Math. of Operations Research*, 22:513–544, 1997.

[8] M. Harchol-Balter, N. Bansal, and B. Schroeder. Implementation of SRPT Scheduling in Web Servers. *Technical Report, Carnegie Mellon University*, CMU-CS-00-170, 2000.

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

[10] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 418–426, May 1996.

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

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

[13] S. Muthukrishnan, R. Rajaraman, R. Shaheen, and J. Gehrke. Online scheduling to minimize average stretch. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 433–43, 1999.

[14] A. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. To appear in *Journal of Scheduling*.

[15] W. E. Smith. Various optimizers for single-stage production. *Naval Res. Logist. Quart.*, 3:59–66, 1956.