Approximation schemes for flow time on multiple machines

Nikhil Bansal[1]

January 2003

CMU-CS-03-103

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We consider the problem of scheduling jobs on multiple machines with preemption with the goal of minimizing the total flow time and the maximum flow time.
For minimizing total flow time, we give an algorithm which produces a $1 + \epsilon$ approximate solution in time $n^{O(m \log n/\epsilon^2)}$. Here n is the number of jobs and m is the number of machines. More generally, even if we have unrelated machines and consider weighted flow time, our algorithm has running time $n^{O(m \log^2 n/\epsilon^3)}$, provided either P or W is poly-bounded in n. Here W (resp. P) is the ratio between the maximum to minimum job weight (resp. size).
For minimizing the maximum flow time, we give a PTAS for a constant number of unrelated machines.

[1]Computer Science Department, Carnegie Mellon University. Email: nikhil@cs.cmu.edu
This work was supported in part by NSF ITR grant CCR-0122581 and by an IBM Graduate Fellowship.

Keywords: Flow time, multiple machines, approximation schemes, preemption, unrelated machines, approximation algorithms

1 Introduction

Recently, there has been a lot of interest in scheduling problems with the goal of minimizing flow time related metrics. In this paper, we study two basic problems for multiple machines: Minimizing total flow time and minimizing maximum flow time.

The Model: Formally, we are given a collection of n jobs and m machines. Job j has a processing time p_j and a release date r_j before which it cannot be scheduled. Without loss of generality we assume that all p_j and r_j are integers. We use P to denote the ratio of the maximum to the minimum job size. For a given schedule, the flow time of a job is defined as the difference of the time at which the job completes and its release date, equivalently, it is the total time that a job spends before it completes. The total (resp. maximum) flow time refer to the sum (resp. maximum) of the flow time of all jobs.

In the unrelated machines setting, a job could possibly have different processing requirement on each machine. We use p_{jk} to denote the processing time (or size) of job j on machine k. Most of this paper (except Section 3) deals with the case when the jobs are unweighted. In the weighted case, job j has a weight w_j, and the weighted flow time of a job is simply its flow time multiplied by its weight. We will use W to denote the ratio of the maximum to minimum job weight.

An important issue while scheduling jobs on multiple machines is that of migration. A schedule is said to be *migratory* if it can move partially executed jobs from one machine to another. Migration of jobs is usually considered unattractive as it incurs a huge overhead (cost) in practice. Throughout this paper we only consider non-migratory algorithms.

New Results: Our main result is an algorithm for minimizing total flow time which produces a $1+\epsilon$ approximate solution and has running time $n^{O(m \log n/\epsilon^2)}$. Thus for a fixed number of machines, this gives a quasi-polynomial time approximation scheme.

The above result assumes that the machines are identical and the all jobs have equal weight. However, it can be easily adapted to the case when the machines are unrelated. Additionally, even if our measure is weighted flow time, but we have that either W or P is poly-bounded in n, then our approximation scheme has running time $n^{O(m \log^2 n/\epsilon^3)}$.

We do not know if the dependence on m can be made polynomial for identical machines. However, we show a weaker result that, if m is a part of the input, then total weighted flow time on identical machines is NP-hard to approximate within $o(\min\{n^{1/10}, W^{1/2}, P^{1/6}, m\})$. This holds even if both W and P are poly-bounded in n.

For maximum flow time on unrelated machines, our algorithm produces a $1+\epsilon$ approximate solution in time $O(n^{m/\epsilon})$.

Previous Work: The first non-trivial results for the problem of minimizing the total flow time were obtained by Leonardi and Raz [11]. They give an $O(\log(\min\{\frac{n}{m}, P\}))$ competitive algorithm for identical parallel machines. They also prove tight lower bounds of $\Omega(\log \frac{n}{m})$ and $\Omega(\log P)$ on the competitive ratio of any online algorithm. However, their algorithm involves migration of jobs. An equally effective algorithm in terms of competitive ratio but which does not require job migration was given by Awerbuch et al. [2]. While these algorithms are online, the guarantees provided by these algorithms are also the best offline guarantees known for the problem. Infact, obtaining a constant factor approximation or a PTAS is considered one of the major open problems in scheduling [12, 11, 2].

For weighted flow time, the only results known are for single machine due to Chekuri et al [5]. They give an approximation scheme which has running time $n^{O(\log W \log P/\epsilon^3)}$. They also strengthen their result for the case where either P or W is poly-bounded in n. In this case, they give an approximation scheme with running time $n^{O(\log^2 n/\epsilon^3)}$. Thus our result for weighted flow time can be thought of as extending the result of [5] to multiple machines. However, in a way the result of [5] is more general than ours: Our result does not hold if both W and P are arbitrary.

We do not know of any previous work on flow time for multiple unrelated machines. However, a lot is known about the related measure of minimizing weighted *completion time*. For identical machines, a PTAS is known for minimizing weighted completion time with arbitrary m and release dates [1]. For unrelated machines, Hoogeveen et al [7] show that minimizing total unweighted *completion* time with release dates in Max SNP-Hard, if m is a part of the input. Infact, minimizing weighted completion time on unrelated machines is Max SNP-Hard even in the absence of release dates. For weighted completion time on unrelated machines and arbitrary release dates, a PTAS is known for fixed m [1], and for arbitrary m, a 2 approximation algorithm based on convex programming was given by [13].

We next consider maximum flow time. The problem is NP-Hard even for two identical machines, as minimizing makespan on multiple machines is a special case of this problem where all the release dates are 0. Bender et al. [4] show that a natural greedy algorithm, similar to Graham's algorithm for online makespan minimization, is $(3 - 2/m)$ competitive for identical parallel machines. However, the existence of a PTAS is left open [4, 12].

We do not know of any other work on maximum flow time in the presence of release dates. However, a lot has been done for the case without release dates. For identical machines, a PTAS for arbitrary m was given by Hochbaum and Shmoys [6]. For the case of unrelated machines, Lenstra et al [10] give a 2 approximation for arbitrary m, they also show that the approximation factor cannot be improved to better than $3/2$ in polynomial time. For constant m and unrelated machines, a PTAS was first given by Horowitz et al [8]. The time and space complexity were subsequently improved by various authors. The best known result is a FPTAS with a running time of $n(m/\epsilon)^{O(m)}$ [9]. In particular, the running time is linear in the number of jobs for constant m.

Finally, in terms our techniques, a major component of our algorithms is an idea due to Bansal et al [3], which gives a way to store, using quasi-polynomial space, all possible approximate profiles of jobs under SRPT.

2 Total Flow Time

In this section we consider the case when all machines are identical and all jobs are unweighted. The extensions are considered later in Section 3.

Let I be a problem instance with largest job size P. Without loss of generality we can assume that all release dates are at most nP and that all jobs finish execution by time $2nP$ (otherwise we could reduce the problem into two disjoint problems). Let *Opt* denote the optimal schedule, we also abuse notation and use *Opt* to denote the total flow time under the optimal schedule.

Lemma 1 *Given I, rounding up the job sizes and release dates to a multiple of $\epsilon P/n^2$ only increases the optimal cost of this instance by a factor of $(1+2\epsilon)$.*

Proof: Given a schedule for the original instance, rounding up the job sizes adds at most $n \cdot \epsilon P/n^2 = \epsilon P/n$ to the flow time of each job. Similarly rounding up the release dates adds at most $\epsilon P/n^2$ to the flow time of each job. Thus, the total flow time is affected by at most $2\epsilon P \leq 2\epsilon Opt$. □

Let I' denote this rounded instance. By Lemma 1 we can assume that all job sizes are integers in the range $[1, n^2/\epsilon]$. Similarly, as no release date in I is more than $2nP$, all release dates in I' are integers in the range $[1, n^3\epsilon]$. We will obtain a schedule $S(I')$ with optimum total flow time for I'. Clearly, all events (arrivals and departures) under $S(I')$ at integral times in the range $[1, 2n^3/\epsilon]$. Also it directly seen that a schedule $S(I)$ for I follows from $S(I')$ and that the total flow time under $S(I)$ is at most that under $S(I')$. Henceforth, we only consider I'.

First now divide all the jobs into $O(\frac{1}{\epsilon}\log n)$ classes. We say that a job with size p_j lies in class i, iff $(1+\epsilon)^{i-1} \leq p_j < (1+\epsilon)^i$. Consider the optimum algorithm. Let $S_j(i,t)$ denote the jobs of class i on machine j which are alive at time t. Note that the class of a job depends only on its initial processing time and hence does not change with time. It is easy to see that on each machine the jobs are processed in the order or shortest remaining processing time (SRPT). Thus we have that,

For any $j = 1, \ldots, m$ and any time t, at most one job in $S_j(i,t)$ has a remaining processing time that is not in the range $(1+\epsilon)^{i-1}$ and $(1+\epsilon)^i$.

We now define the state of the algorithm $Q(t)$ at time t. For each class i and each machine j, we store at most $\frac{1}{\epsilon}+1$ numbers: the first $\frac{1}{\epsilon}$ are the remaining processing times of the $\frac{1}{\epsilon}$ jobs in $S_j(i,t)$ with the largest remaining processing time; the last entry is the sum of the remaining processing times of the rest of the jobs in $S_j(i,t)$. Notice that as each job has size at most n^2/ϵ, there are at most $(n^2/\epsilon)^{1/\epsilon} = O(n^{2/\epsilon})$ possible choices for the first $1/\epsilon$ entries and n^3/ϵ possible choices for the sum of remaining sizes. Finally, since there there are at most $O(\log n/\epsilon)$ classes and m machines, the total number of distinct states at any step is at most $n^{O(m \log n/\epsilon^2)}$.

The following lemma shows how this information helps us in estimating the number of unfinished jobs at any point of time.

Lemma 2 *We can estimate the number of jobs in queue at time t to within a factor of $(1+2\epsilon)$ using the information in $Q(t)$.*

Proof: If there are fewer than $1/\epsilon$ jobs in level i, we know their number precisely because we store their remaining processing times precisely. Let us now consider the case when there are more than $1/\epsilon$ jobs in some level i.

Suppose at first that the remaining processing times of all these jobs lies between $(1+\epsilon)^{i-1}$ and $(1+\epsilon)^i$. Then, by assuming that all the jobs have size $(1+\epsilon)^{i-1}$ and computing the number of jobs using the total remaining processing time, our estimate is off from the correct number by at most a factor of $1+\epsilon$.

Finally, as at most one job in every level i lies outside the range $(1+\epsilon)^{i-1}$ and $(1+\epsilon)^i$. Thus our estimate above could be off by another job. However, there are at least $1/\epsilon$ unfinished jobs. Thus we get an estimate within a factor of $1+2\epsilon$. $\square$

Thus the algorithm is a large dynamic program with $O(n^3/\epsilon)$ times $n^{O(m \log n/\epsilon^2)}$ states, and for each state store the value of the minimum total flow time that can be achieved if that state is reached.

Now we only need to show how to update the state of the algorithm with time. When a new job arrives, we have m choices for the machine to which this job can be assigned. Once a machine is decided, the size of the job determines the class to which it belongs. Also, it is straightforward to update the state, as either the job is added to the first $1/\epsilon$ jobs in $S_j(i,t)$, or else if it is smaller that the $1/\epsilon$ largest jobs in its class, then its size is added to the $(1/\epsilon+1)^{th}$ entry. When the algorithm works on a job in level i on machine j,

Now consider the case when there are no arrivals. Suppose at time t, the algorithm works on the state $s(j)$ on machine j. We do not know $s(j)$, but we can try out all possible $O(\log n/\epsilon)^m$ choices for the different $s(j)$ on each machine j. For a fixed choice of $s(j)$, clearly the algorithm works on the job with the least remaining processing time in the class $s(j)$. So we either need to decrement the $\frac{1}{\epsilon}+1^{\text{t}h}$ entry of $s(j)$ by one, or if there are no more than $\frac{1}{\epsilon}$ jobs, then the last non-zero entry is decremented by 1.

Thus we have shown that:

Theorem 1 *The above algorithm gives a $(1+\epsilon)$ approximation for the problem of minimizing total flow time on m identical machines and runs in time $n^{O(m \log n/\epsilon^2)}$.*

3 Extensions

We now consider the case when the machines are unrelated and our measure is weighted flow time.

Let p_j^* denote $\min_k p_{jk}$. Thus p_j^* denotes the minimum size of j among all the machines. Let $Q = \sum_j p_j^*$, and suppose that the weights are integers in the range $1, \ldots, W$. For unrelated machines, we define $P = \max_{i \neq j} p_i^*/p_j^*$. Note that if the machines are identical, this definition of P corresponds exactly to the maximum of minimum job size ratio. We will show that the problem has a QPTAS if either W or P is polynomially bounded in n.

We first consider the case when W is polynomially bounded in n. By assigning each job to the machine where it takes the least time, it is clear that nWQ is an upper bound on *Opt*. Similarly, Q is a lower bound on *Opt*. Hence we will assume that our algorithm simply ignores p_{jk} where $p_{jk} \geq 2nWQ$.

Next, rounding each p_{ij} and release date up to the next multiple of $\epsilon Q/(Wn^2)$, it is easy to see that the total flow time of each job is affected by at most $\epsilon Q/Wn$ and hence the total weighted flow time is affected by at most $\epsilon Q \leq \epsilon Opt$. Moreover, as usual, we can assume that the algorithm is always working on at least one job, thus the release date of any job is at most nQ, and hence our algorithm needs to consider only $O(n^3 W/\epsilon)$ time steps.

Next we round up the weights to powers of $1+\epsilon$. Clearly, this affects the solution by at most $1+\epsilon$ times. Finally, observe that in the optimal schedule for this rounded instance, if we consider a particular machine and restrict our attention to time intervals when jobs from a particular weight class are executed, then clearly these jobs are executed in the SRPT order.

With these observations we can directly give an algorithm based on the ideas in Section 2. For each machine and each weight class, we maintain the states under SRPT. Since there are $O(\log W/\epsilon) = O(\log n/\epsilon)$ weight classes, the number of states at each time step is bounded by $n^{O(m \log^2 n/\epsilon^3)}$. When a job arrives, there are m choices corresponding to the machines it can be assigned. If there are no arrivals, for each machine, we need to decide which weight class to work on, and with each weight class which size class to work on. Thus there $(\log n/\epsilon)^{2m}$ choices to choose from at each time step. Thus our algorithm can be implemented directly as a dynamic program of size $Wn^3/\epsilon^2 = n^{O(1)}$ times $n^{O(m \log^2 n/\epsilon^3)}$. Hence, we have a QPTAS.

We now consider the case when P is poly-bounded and W is arbitrary. As previously, *Opt* is at most nQW. Next, observing that each job has size at least Q/nP on each machine, it follows that *Opt* is also lower bounded by WQ/nP.

Our algorithm now is as follows. We only consider jobs with weights between $\epsilon W/n^2 P$ and W. Jobs with weight below $\epsilon W/n^2 P$ will be added to the schedule arbitrarily. Note that since each job has flow time at most Q, this adds at most $\epsilon WQ/nP \leq \epsilon Opt$. Now, since P is poly-bounded in n, finding a schedule for the jobs of weight between $\epsilon W/n^2 P$ and W reduces to the previous case, where W was poly-bounded. Thus the QPTAS for poly-bounded P follows.

4 Dependence on the number of machines

It is clear that both total unweighted flow time and maximum flow time on unrelated machines are Max SNP-Hard for arbitrary m [7, 10]. However, an approximation scheme with a polynomial dependence on m might be possible for identical parallel machines. We show a related but weaker negative result that, if m is a part of the input, minimizing total weighted flow time is Max SNP-hard, even when both P and W are polybounded in n.

Consider an instance of 3-Partition (SP15). This consists of a set A of $3m$ elements, an integer bound $B > 0$; for each $x \in A$ a integer size $s(x)$ s.t. $B/4 < s(x) < B/2$ and s.t. $\sum_{x \in A} s(x) = mB$. The question is whether A can be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$. 3-Partition is known to be strongly NP-Complete. In particular, it is NP-complete for $B = O(m^4)$.

Given an instance of 3-Partition, we transform it as follows. There are m machines. Each element $x \in A$ corresponds to a job, with size $s(x)$, weight m and is released at time $t = 0$. Next at each time instance $t = B + i/m^2$, for $i = 0, 1, 2, \ldots, Bm^5$, m jobs each with size $1/m^2$ and weight $1/m$ are released. Thus the total number of these small jobs is Bm^6.

If the 3-partition has a solution, we can use it to schedule the jobs arriving at $t = 0$, and then schedule the jobs of weight $1/m$ as they arrive. Now, each of the weight m job has flow time at most $3B$, and each of the Bm^6 jobs of weight $1/m$ has flow time $1/m^2$. Thus the total weighted flow time is $3m \cdot 3B \cdot m + 1/m^3 \cdot Bm^6 = O(Bm^3)$.

On the other hand if there is no 3-partition, there is at least one weight m job (call it J) unfinished by time B. Consider the situation by time $Bm^3/2$, if J is still there, it contributes at least $Bm^4/2$ to the flow time, else there are least m^2 jobs each of weight $1/m$ piled up during $Bm^3/2$ and Bm^3, contributing $O(Bm^4)$.

As $B = O(m^4)$ and $P \leq Bm^2 = O(m^6)$ and $W = m^2$ and $n = Bm^6 = O(n^{10})$, W and P are polybounded in the number of jobs, and we have an inapproximability factor of $\Omega(\min\{n^{1/10}, W^{1/2}, P^{1/6}, m\})$.

5 Maximum Flow Time

In this section we consider minimizing the maximum flow time on m unrelated machines. We give an algorithm that runs in time $O(n^{m/\epsilon})$ and produces a $(1+\epsilon)$ approximation. We begin with some easy observations.

1. As first come first served (FCFS) is optimum for minimizing the maximum flow time on a single machine. It follows that for every machine, the jobs assigned to that machine are executed in FCFS order.

2. As in Section 3, let p_j^* denote $\min_k p_{jk}$ and $Q = \sum_j p_j^*$. By assigning each job to the machine where it takes the least time, it is clear that Q is an upper bound on *Opt*.

Furthermore, we can assume that all the release dates are at most Q and all jobs finish execution by time $2Q$ (else, the problem can be reduced to two smaller problems). Finally, since there are n jobs, there is some job whose size on every machine is at least Q/n. Thus Q/n is a lower bound on Opt.

3. Let $\delta > 0$ be an arbitrary real number. Note that rounding up the release date of each job to the next multiple of δ increases the maximum response by at most δ. Similarly, rounding up each p_{ij} to the next multiple of δ and increase the maximum flow time by at most $n\delta$.

We now describe the algorithm. If some $p_{jk} > 2Q$ for some j, k, we just set $p_{jk} = \infty$. Choose $\delta = \epsilon Q/4n^2$. Round each p_{jk} and r_j up to the next multiple of δ. Time increases in multiples of δ, and all events take place only at multiples of δ. Thus, we need to consider only $O(n^2)$ time steps.

The algorithm maintains a state of the $< val, t, w_1, w_2, \ldots, w_m >$ where t denotes the time, w_j represents the total work present on each machine j, $1 \leq j \leq m$ at time t, and val is minimum value of the maximum flow time that can be achieved at time t and for the particular values of w_i.

It is trivial to update the state of the algorithm, at each time step or event. If there is an arrival J_i at time t and we assign it to machine j algorithm updates the sate as $w_j = w_j + p_{ij}$ and $val = \max\{val, w_j + p_{ij}\}$. If no arrival takes place, the algorithm simply decrements w_i for each non-zero w_i.

Thus the algorithm is a dynamic program of size $O(n^3/\epsilon)$ times $n^{O(m/\epsilon)}$ and updating an entry for a state requires $O(m)$ time. Thus we have that

Theorem 2 *The maximum flow time on multiple unrelated machines and arbitrary release dates can be approximated to within a factor of* $1 + \epsilon$ *in time* $n^{O(m/\epsilon)}$.

6 Open Problems

Obtaining a constant approximation or a PTAS for total unweighted flow time is a major open problem. Also, we do not know if the dependence in m can be made polynomial (or sub-exponential) for the total unweighted flow time problem or even for the maximum flow time problem. Settling this would be very interesting.

A PTAS for weighted flow time on a constant number of machines would be a significant breakthrough. An interesting intermediate goal might be to extend the result of Chekuri et al [5] to multiple machines and arbitrary P and W (i.e. obtain an approximation scheme with running time $O(n^{poly(\log W, \log P, 1/\epsilon, m)})$ when both W and P are arbitrary).

7 Acknowledgements

The author would like to thank Avrim Blum for very useful discussions and ideas.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Millis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *IEEE Symposium on Foundations of Computer Science*, pages 32–43, 1999.

[2] 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.

[3] N. Bansal, A. Blum, S. Chawla, and K. Dhamdhere. Scheduling for flow time with admission control, manuscript, 2002.

[4] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–279, 1998.

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

[6] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[7] J. A. Hoogeveen, P. Schuurman, and G. J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. In *Integer Programming and Combinatorial Optimization*, volume LNCS 1412, pages 353–366. Springer, 1998.

[8] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM (JACM)*, 23(2):317–327, 1976.

[9] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *ACM symposium on Theory of computing*, pages 408–417, 1999.

[10] J. Lenstra, D. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[11] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *ACM Symposium on Theory of Computing (STOC)*, pages 110–119, 1997.

[12] P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2:203 – 213, 1999.

[13] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM*, 48(2):206–242, 2001.