Order Scheduling Models: Hardness and Algorithms

Naveen Garg[1], Amit Kumar[1], and Vinayaka Pandit[2]

[1] Indian Institute of Technology, Delhi
[2] IBM India Research Lab, Delhi

Abstract. We consider scheduling problems in which a job consists of components of different types to be processed on m machines. Each machine is capable of processing components of a single type. Different components of a job are independent and can be processed in parallel on different machines. A job is considered as completed only when all its components have been completed. We study both completion time and flowtime aspects of such problems. We show both lowerbounds and upperbounds for the completion time problem. We first show that even the unweighted completion time with single release date is MAX-SNP hard. We give an approximation algorithm based on linear programming which has an approximation ratio of 3 for weighted completion time with multiple release dates. We give online algorithms for the weighted completion time which are constant factor competitive. For the flowtime, we give only lowerbounds in both the offline and online settings. We show that it is NP-hard to approximate flowtime within $\Omega(\log m)$ in the offline setting. We show that no online algorithm for the flowtime can have a competitive ratio better than $\Omega(\sqrt{m})$.

1 Introduction

Consider the following scenario of scheduling customer orders. Each customer order consists of several components of different types. These orders are to be processed at m facilities each of which is specialized to execute components of a particular type. The order of a customer can be delivered only when all its components have been completed. In this paper, we consider scheduling problems in this setting which can be thought of as open shop scheduling with overlaps allowed between operations of a job. One may refer to the article by Chen and Hall [6] for an elaborate survey of practical applications of such a scheduling model. In their survey article, Leung et. al [11] have called this model as *Order Scheduling Model*.

We observe that order scheduling models occur in computational settings as well. Consider the following example. Large distributed computational grids are becoming very popular in solving complex scientific problems [4, 12]. The Master-Worker scheme is one popular approach to solving these massive computation problems [8]. Typically, the problems are solved using a master to coordinate

the exploration of the branch and bound tree with the help of a large number of worker resources [4]. There are also problems in which the computation is divided into many independent, data parallel components and executed on a grid [1, 12]. Once the different independent components of the problem are mapped to specific resources, the expected running time can be estimated using the data size and the history of response times. To solve a number of these problems efficiently on the grid, the scheduler must take into account parameters like average completion time and flow time. Some of the most challenging computational problems solved on grids include genome data analysis, earthquake simulators, drug design etc.

In the three field notation for scheduling problems we use 'G' in the first field to denote this model. In this paper we consider the problem of scheduling jobs in this model under different objectives. The problem of minimizing the completion time, $G||\sum C_j$ was studied by Wagneur and Sriskandarajah [19] and they proved it to be strongly NP-Complete. However, Leung et. al [11] showed an error in their proof. Recently, Roemer [17] showed the problem to be strongly NP-complete even when $m = 2$. In this paper, we show that the problem is MAX-SNP hard even when there is a single release date (for all the jobs) and the processing time of all components is one. As for approximation algorithms, Wang and Cheng [20] gave a constant factor approximation algorithm. They considered a time-indexed linear programming formulation and a heuristic based on its solution to get an approximation ratio of 5.83. We show how to exploit a different formulation by Queyranne [14] to get an approximation ratio of 2 for the case of single release date and 3 for the case of multiple release dates. To the best of our knowledge, no non-trivial online algorithm is known for these problems. We present the first constant factor competitive online algorithms for these problems.

In Section 3 we present hardness results for both offline completion time and flowtime. We first show that the problem of minimizing the sum of completion times, $G||\sum C_j$ is MAX-SNP hard even when all components have unit processing times. This makes the problem harder than the well-studied problem of minimizing weighted completion time on parallel machines with release dates $P|r_j|\sum_j w_j C_j$ for which a PTAS[2] is known. We then show that it is NP-hard to approximate the offline flowtime within $\Omega(\log m)$. The results of Queyranne [14] and Schulz [18] yield approximation algorithms for $1|prec|\sum_j w_j C_j$ and $1|r_j, prec|\ \sum_j w_j C_j$ problems with approximation ratios 2 and 3 respectively. In Section 4, we show how to exploit their ideas to get approximation ratios of 2 and 3 for the $G|\sum_j w_j C_j$ and $G|r_j|\sum_j w_j C_j$ respectively.

Our online algorithm, in Section 5, uses the technique of time intervals with geometrically increasing lengths [9][5] and is 4-competitive. If we require that the computation performed by the online algorithm at each step be polynomially bounded then we obtain a 16-competitive algorithm. Hall et.al.[9] gave a general technique for obtaining a 4ρ-competitive algorithm for weighted completion time with release dates provided there exists a dual ρ-approximation algorithm for

the *maximum scheduled weight problem*. For our problem, we cannot show a dual ρ-approximation algorithm for any constant ρ. However, we can show a bicriterion (2,2) approximation algorithm for the following problem: Given a set of jobs and a deadline D, find a schedule which minimizes the weight of jobs which are not completed by time D. We show that any (α, β) bicriterion approximation algorithm for this problem suffices to give a $4\alpha\beta$-competitive online algorithm for the problem of minimizing weighted completion time; this yields the 16-competitive algorithm mentioned above.

In section 6 we consider the problem of minimizing the total flow time in the online setting. Recall that the flowtime of a job is the difference between its completion time and release date, i.e., the amount of time it spends in the system. Here we show a family of instances where no online algorithm can achieve a competitive ratio better than $O(\sqrt{m})$ against an adaptive adversary. In this instance all job-components require unit processing time and hence the lower bound on the competitive ratio applies even if preemptions are permitted. In contrast for parallel machines, with preemptions — the problem $P|pmtn, r_j|\sum_j F_j$ — a non-migratory version of the shortest remaining processing time rule is $O(\min(\log n, P))$-competitive [3] where P is the ratio of the processing time of the longest job to the processing time of the shortest job.

The model considered in this paper is somewhat similar to open-shop scheduling with the only difference that in open-shop the operations associated with a job cannot be performed simultaneously. Some of the results known for open-shop mirror the results we obtain in this paper. In particular, it is known that $O||\sum_j C_j$ is MAX-SNP hard[10]. Queyranne and Sviridenko[15] gave a $3+2\sqrt{2}$-approximation for $O|r_j|\sum_j w_j C_j$ which was later improved to 5.06 by[7] . However, we are not aware of any results on $O|r_j|\sum_j w_j C_j$ in the online setting.

2 Preliminaries

We consider the problem of scheduling jobs with components on multiple machines. We have m machines and n jobs. Each job j specifies a vector P^j of processing times – we shall call this the processing time vector of job j. For each machine i, P_i^j denotes the processing requirement of job j on machine i. Define the length of a job j as the number of machines i such that $P_i^j > 0$. Each job j also has a release date r^j. In a valid schedule, each job j must be processed without interruption for P_i^j amount of time on each machine i. Further, processing of a job on any of the machines can not begin before its release date.

Given a valid schedule $\mathcal{A}$, define $C_i^j(\mathcal{A})$ as the time at which job j finishes processing on machine i. Define the completion time $C^j(\mathcal{A})$ of job j as $\max_i C_i^j(\mathcal{A})$. Define the flow time $F^j(\mathcal{A})$ of job j as $C^j(\mathcal{A}) - r^j$. Often, the schedule $\mathcal{A}$ will be clear from the context, and so we shall just use the notations C_i^j, C^j and F^j.

We also associate a weight w^j with job j. For a set S of jobs, let weight(S) denote the total weight of jobs in S. Let W be the total weight of all the jobs. Define the weighted completion time of j in a schedule $\mathcal{A}$ as $\sum_j w^j \cdot C^j(\mathcal{A})$.

The weighted flow time is defined similarly. We would like to compute schedules which minimize the weighted completion time or weighted flow time.

It is easy to observe that any algorithm that maintains a busy-schedule on each of the machines has minimum makespan even in the case of multiple release dates. As mentioned in the introduction, we focus on the weighted completion time and the total flowtime objective functions.

3 Hardness of Approximating Completion Time and Flow Time

3.1 Completion Time

We first show that the off-line problem of minimizing sum of completion times is MAX-SNP hard even when all release times are 0. We use the fact that vertex cover is MAX-SNP hard even on constant degree graphs [13]. Let C be a constant such that vertex cover is MAX-SNP hard on graphs where degrees of vertices are bounded by C.

An instance $\mathcal{I}$ of the vertex cover problem is given by a graph $G = (V, E)$ where the degree of any vertex in G is at most C. We map this to an instance $\mathcal{I}'$ of the problem of minimizing sum of completion times. $\mathcal{I}'$ has $|E|$ machines, one for each edge in G and $n = |V|$ jobs, one for each vertex in G. Corresponding to a vertex $v \in V$, we construct a job $j(v)$ such that $P_i^{j(v)}$ is 1 if edge i is incident on v, 0 otherwise. All jobs have weight 1 and release time 0.

Lemma 1. *There exists a constant K' such that it is NP-hard to get a K'-approximation algorithm for the minimum weighted completion time problem.*

Proof. Observe that all jobs in $\mathcal{I}'$ can be scheduled in two time steps as each edge job has two components corresponding to the vertices it is incident on. It is easy to see that the set of jobs completed in the first step correspond to an independent set. Hence, the set of jobs completed in the second step correspond to a vertex cover. Suppose $VC(G)$ is the size of the vertex cover of G that gets completed in second step. Then, the cost of such a solution is $(n - VC(G)) + 2VC(G) = n + VC(G)$. So the minimum completion time of $\mathcal{I}'$ has cost $CT^{OPT}(\mathcal{I}') = n + VC^{OPT}(G)$ where $VC^{OPT}(G)$ denotes the size of minimum vertex cover of G. Therefore,

$$2n \geq CT^{OPT}(\mathcal{I}') \geq (n + n/C) \tag{1}$$

As the degree of G is bounded by a constant, say C, we have

$$VC^{OPT}(G) \geq n/C \tag{2}$$

As mentioned before, there exists a constant $K > 1$ such that it is NP-hard to approximate the vertex cover of graphs whose degree is bounded by C within a factor of K. This, combined with equations 1 and 2 imply that there exists a constant $K' = \frac{1+K/C}{1+1/C}$ such that it is NP-hard to approximate the completion time of the transformed instances within a factor of K'.

3.2 Flow Time

Note that Lemma 1 implies that the problem of offline flowtime minimization is MAX-SNP hard even in the unweighted case. We now show that it is NP-hard to approximate the unweighted flowtime within $\Omega(\log m)$. We begin by explaining the intuitive ideas of our construction.

The flowtime of a given schedule can be written as the sum of the number of unfinished jobs at every time step. Given an instance of set cover, we construct an instance of the order scheduling problem such that, for any reasonable schedule, the set of unfinished jobs at a "deadline" can be interpreted as the set cover of the set system. Furthermore, we show that an α-approximation algorithm for the flowtime can be turned into an α-approximation algorithm for the set cover problem. We now proceed to the details of our construction.

We reduce the set cover problem to the problem of minimizing flowtime in our setting. We start with an instance of the set cover problem. Let $\mathcal{S}$ be the set system on the universe U. We now construct an instance of the scheduling problem. Corresponding to each element $e \in U$, we have a machine and for each set $S \in \mathcal{S}$, we have a job. We use S to denote both the set and its corresponding job, and e denotes an element and the corresponding machine. The job S has a component of length 1 on a machine e if $e \in S$. Let T, ϵ be such that $T > 2|\mathcal{S}|$ and $1/\epsilon > |\mathcal{S}|^2$. Let s_e denote the number of sets which contain e. On a machine e, we create $(T - s_e + 1)/\epsilon$ dummy jobs with just one component of length ϵ on e. All the jobs (including the dummy jobs) are released at time $t = 0$. After time T, we release "filler" jobs at regular intervals of ϵ. Each filler job has a component of length ϵ on each of the machines. The filler jobs are released for a very long time, say L. This completes the construction of the instance of order scheduling problem. Note that, as long as we keep L polynomially bounded in $|\mathcal{S}|$, our reduction can be done in polynomial time.

The volume of the jobs released on each machine at time $t = 0$ is equal to $T + 1$. So, the volume of the unfinished components on any machine at T is exactly 1. For any machine e, if all the components corresponding to the set jobs are finished by T, then, there will be $1/\epsilon$ jobs left unfinished on e. Given that $1/\epsilon > |\mathcal{S}|^2$, finishing all components belonging to set jobs would result in very high penalty from T to L. So, every schedule is forced to be left with exactly one component corresponding to a set job on each machine. Thus, for every reasonable schedule, the set of unfinished tasks at time T corresponds to a set cover. Note that the filler jobs are such that, beyond T, if a schedule tries to reduce the unfinished set jobs, it ends up accumulating too many filler jobs. So, every reasonable schedule is forced to schedule filler jobs between T and L.

Let SC_{OPT} denote the size of the optimum set cover and SC_{PACK} denote the size of the set cover left unfinished at time T by any algorithm. Let I_{OPT} and I_{PACK} be the flowtime incurred by the schedules which leave the optimum set cover and a set cover of size SC_{PACK} respectively. As argued above, beyond T, every reasonable schedule is forced to schedule only filler jobs upto L. Let F_{OPT} and F_{PACK} denote the flowtime of the set cover jobs left unfinished at time L for the two schedules. Note that, $F_{OPT} \approx L \cdot SC_{OPT}$ and $F_{PACK} \approx L \cdot SC_{PACK}$.

Let FT_{OPT} and FT_{PACK} be the total flowtimes of the two schedules. We have,

$$FT_{OPT} = L + 2 \cdot L \cdot SC_{OPT} + I_{OPT} \tag{3}$$

$$FT_{PACK} = L + 2 \cdot L \cdot SC_{PACK} + I_{PACK} \tag{4}$$

Note that, we can keep L such that, $L \gg I_{OPT}$ and $L \gg I_{PACK}$. Therefore, if $FT_{OPT}/FT_{OPT} = \alpha$, then, $SC_{PACK}/SC_{OPT} \approx \alpha$. Therefore, we can turn an $o(\log m)$ approximation algorithm for flowtime into an $o(\log m)$ approximation algorithm for set cover. As it is NP-hard to approximate set cover within $\Omega(\log m)$ [16],

Theorem 1. *It is NP-hard to approximate the flowtime of order scheduling problem within $\Omega(\log m)$ where m is the number of machines.*

4 Offline Weighted Completion Time

In this section, we show how to exploit a completion time linear programming formulation by Queyranne [14] and a scheduling heuristic based on its solution (Schulz [18]) to obtain approximation ratios of 2 and 3 for the cases of single and multiple release dates respectively. The formulation by Queyranne is called the *completion time linear program* in the literature. We adapt the completion time formulation for our problem as follows:

$$\begin{array}{rcll} \min \sum_{j=n}^{n} w_j C_j & & & \\ \text{s.t.} & & & \\ C_j^m & \geq & r_j + p_j^m & \forall j \in J, m \in M \\ C_j & \geq & C_j^m & \forall j \in J, m \in M \\ \sum_{j \in S} p_j^m C_j^m & \geq & \frac{1}{2}\left[\left(\sum_{j \in S} p_j^m\right)^2 + \sum_{j \in S} (p_j^m)^2\right] & \forall S \subseteq J, m \in M \end{array}$$

In this formulation, M denotes the set of machines and J denotes the set of jobs. Further, the variable C_j^m indicates the completion time of job j on machine m, p_j^m is the processing time of j on machine m and C_j indicates the completion time of the job j. Queyranne showed a polynomial time separation oracle for the above set of constraints. So, the above program can be solved in polynomial time. The approximation ratios proved here are somewhat implicit and can be deduced from the work of Schulz [18]. We present an outline of the proof for the sake of completeness.

Consider the optimal solution to the above linear program. Let $\bar{C}$ denote the vector of completion times C_js and $\bar{C^i}$ denote the vector of completion times C_j^is on machine i. Let A be an algorithm which schedules the jobs independently on each of the machines. Let $\bar{D^i}$ denote the vector of completion times for the jobs it achieves on machine i. Furthermore, let D_j^i denote the completion time of job j on machine i. We claim that:

Lemma 2. *If there exists a constant K such that $D_j^i \leq K \cdot C_j^i$, then, the schedule given by $\bar{D}_i s$ is an K-approximation for the $G|r_j|\sum w_j C_j$.*

Proof. Note that $C_j = \max\{C_j^i | \forall i \in \{1, \ldots, m\}\}$. The individual schedule $\bar{D^i}$ on machine i satisfies $D_j^i \leq KC_j^i$. Therefore, $D_j = \max\{D_j^i | \forall i \in \{1, \ldots, m\}\} \leq K \cdot C_j$. This implies that the schedule obtained by $\bar{D^i}$s is an K-approximation.

Schulz shows that, on a single machine, scheduling the jobs in the non-decreasing order of the completion times suggested by optimal solution to the above linear program satisfies the condition required by Lemma 2 with $K = 3$ in case of multiple release dates and $K = 2$ in case of single release date. Thus, scheduling components on a machine i in the non-decreasing order of C_j^is we get the approximation ratios stated above. At this point it is appropriate to highlight that, in the standard scheduling model, the above program and the scheduling order can be made to work even when there are precedence constraints between jobs. However, in our case, we are not able to show that the above approach can be made to work with precedence constraints.

The application of Schulz's heuristic on individual machines to get an approximation for the problem of m machines gives rise to the following question: Can algorithms for completion time minimization on single machine be used in place of Schulz? If indeed it is possible, then one could use the PTAS for $1|r_j|\sum w_j C_j$ [2] to get a PTAS for the problem and it would contradict the MAX-SNP hardness proved in Section 3. However, note that Lemma 2 is applicable to only those schedules which bound completion times of components on their corresponding machines in terms of the completion times C_j^is of the optimal solution to the above linear program. So, the heuristic by Schulz which works specifically with the output of the linear program cannot be replaced by other algorithms for completion time on single machine.

5 Online Algorithm for Minimizing Weighted Completion Time

We now consider the problem of minimizing weighted completion time in the online setting. Our approach is similar to that of Hall et.al. [9] and leads to a 4-competitive algorithm for this problem.

For $k \geq 0$, let $t_k = 2^k - 1$. We divide the time line into intervals of geometrically increasing size. For $k \geq 0$, define the interval I_k as $[t_k, t_{k+1})$.

Our algorithm produces a schedule which we denote $\mathcal{A}$. It maintains the invariant that if it processes a job in an interval I_k, then the job will finish processing in this interval. More formally, let $R^{\mathcal{A}}(t_k)$ be the set of jobs released before time t_k but not scheduled before t_k in the schedule $\mathcal{A}$. Then $\mathcal{A}$ schedules only such jobs in I_k (and finishes them in I_k).

Our algorithm can be described as follows :

For $k = 0, 1, 2, \ldots$ do
  By considering all subsets of $R^{\mathcal{A}}(t_k)$, determine the maximum weight collection of jobs that can be completed in I_k.

Schedule this set of jobs in the interval I_k so that they finish processing in this interval only.

Let $\mathcal{O}$ be the off-line schedule which minimizes the weighted completion time. Let weight$(D^{\mathcal{A}}(t_k))$ be the total weight of jobs finished by $\mathcal{A}$ by time t_k. Define weight$(D^{\mathcal{O}}(t_k))$ similarly.

Lemma 3. *weight$(D^{\mathcal{O}}(t_k))$ is at most weight$(D^{\mathcal{A}}(t_{k+1}))$.*

Proof. This is easy to see by restricting attention to the jobs in the set $R^{\mathcal{A}}(t_k)$. The jobs in $R^{\mathcal{A}}(t_k)$ which are scheduled in $\mathcal{O}$ before time t_k can also be scheduled by the online algorithm in the interval I_k (whose length is larger than t_k). □

Let W be the total weight of all jobs. Then the weighted completion time of schedule $\mathcal{O}$ is at least

$$\sum_{k\geq 1}(t_k - t_{k-1})(W - \text{weight}(D^{\mathcal{O}}(t_k))).$$

On the other hand the completion time of the schedule $\mathcal{A}$ is at most

$$\sum_{k\geq 0}(t_{k+1} - t_k)(W - \text{weight}(D^{\mathcal{A}}(t_k))).$$

Rewriting this expression we get

$$\begin{aligned}\sum_{k\geq 2}(t_{k+1} - t_k)(W - \text{weight}(D^{\mathcal{A}}(t_k))) + W + 2(W - \text{weight}(D^{\mathcal{A}}(t_1))) \\ \leq 3W + \textstyle\sum_{k\geq 1}(t_{k+2} - t_{k+1})(W - \text{weight}(D^{\mathcal{A}}(t_{k+1}))) \\ \leq 3W + \textstyle\sum_{k\geq 1} 4(t_k - t_{k-1})(W - \text{weight}(D^{\mathcal{O}}(t_k)))\end{aligned}$$

which implies that the weighted completion time of schedule $\mathcal{A}$ is at most 4 times the completion time of the best possible schedule plus an additive 3W. This implies a competitive ratio of 4 for our online algorithm.

Note however, that to determine the jobs to be scheduled in an interval the online algorithm considers all possible subsets of unfinished jobs and picks the best, leading to an exponential running time. We now describe an online algorithm which takes polynomial running time. Starting from $k = 0$, we formulate a linear program to decide which jobs to schedule in the intervals I_k for all values of k (again our schedule will maintain the invariant that a job scheduled in I_k will finish in this interval only). Each job $j \in R^{\mathcal{A}}(t_k)$ has a variable x^j associated with it which is 1 if job j is scheduled in interval I_k and 0 otherwise. The linear program is as follows

$$\min \quad \sum_j w^j(1-x^j) \tag{LP2}$$

s.t.

$$\sum_j P_i^j x^j \leq \quad 2^k - 1 \quad \text{for all } i \tag{5}$$

$$x^j \in \quad [0,1] \quad \text{for all } j$$

The objective function tries to minimize the total weight of jobs that cannot be finished in I_k. We will require that the total processing on each machine in this interval be no more that $2^k - 1$; note that this is equal to the total length of intervals I_0 to I_{k-1}. This is captured by the constraints (5). Since all jobs in $R^{\mathcal{A}}(t_k)$ which are scheduled before time t_k by $\mathcal{O}$ form a feasible solution to this linear program, the value of the optimum solution of this linear program is at most weight$(R^{\mathcal{O}}(t_k))$.

Let $\bar{x}$ be an optimum solution to this linear program. Let J be the set of jobs for which $\bar{x}^j \geq 1/2$; our algorithm schedules all the jobs in J in I_{k+1}. On any machine i, the total processing time of jobs in J is at most $2(2^k - 1) \leq 2^{k+1}$ (which is at most the length of I_{k+1}). Further, the total weight of jobs which are not scheduled is at most twice the value of the optimum solution of the linear program.

The total weight of unfinished jobs at time t_{k+2} in $\mathcal{A}$ is at most twice the weight of unfinished jobs at time t_k in the schedule $\mathcal{O}$. The above analysis now extends to give a competitive ratio of 16 for this online algorithm.

6 Lower Bound for Minimizing Sum of Flow Times

In this section, we prove a lower bound of $\Omega(\sqrt{m})$ on the competitive ratio of any online algorithm for minimizing the sum of flowtimes. In fact this lower bound holds even when the processing times P_i^j are restricted to be either 0 or 1. Let the number of machines m be of the form k^2, where k is an integer. We first discuss the idea at a high level.

For each subset of k machines, we define a job which requires 1 unit of processing on these machines and 0 processing on other machines. Let J be the set of these jobs; note that $|J| = \binom{k^2}{k}$. All the jobs in J are released at time $t = 0$. Note that all the jobs in J can be scheduled by time $T_0 = \binom{k^2-1}{k-1}$. Let $T_1 = T_0 - k$. We show that at time T_1 in any online schedule, there is a set S of k machines such that the following condition is satisfied – there are $\Omega(k^2)$ jobs which have unscheduled components on at least one of the machines in S. Assuming this is true, we construct an adversary as follows. Let us number the machines in S from 1 to k. From time T_1+1 onwards, we release k jobs $j_1, \ldots, j_k$ of length one each. Job j_l is defined as: $P_l^{j_l} = 1$, and $P_i^{j_l} = 0$ if $i \neq l$. We then argue that with prior knowledge of these jobs, it is possible to schedule jobs such that, at time T_1, the number of jobs with unscheduled components on S is $O(k)$.

Lemma 4. *For the set of jobs J as defined above, in any schedule, at time T_1, there is a subset of k machines such that there are $\Omega(k^2)$ jobs which have unfinished components on these machines.*

Proof. Let $\mathcal{A}$ be an online schedule. Let q_j denote the number of unscheduled components of job $j \in J$ at time T_1. Note that, $0 \leq q_j \leq k$. Also note that, $\sum_{j \in J} q_j = k^3$. Consider a random subset N of k machines. Let U be the set of jobs which have unfinished components on at least one machine in N at time T_1. Consider a job j and let $\mathbf{Pr}_N^j$ denote the probability that $j \in U$. Note that

$$\mathbf{Pr}_N^j \;=\; 1 - \frac{\binom{k^2-q_j}{k}}{\binom{k^2}{k}} \;\geq\; 1 - \left(\frac{k^2-q_j}{k^2}\right)^k \;\geq\; 1 - \left(1 - \frac{q_j}{k^2}\right)^k \;\geq\; \frac{k \cdot q_j}{2k^2}$$

where the last inequality follows from the fact that for $xy \leq 1, (1-x)^y \leq 1 - xy/2$ (note that $q_j \cdot k/k^2 \leq 1$).

The expected size of U is given by

$$\sum_{j \in J} \mathbf{Pr}_N^j \geq \sum_{j \in J} \frac{q_j k}{2k^2} \geq \frac{k^2}{2}$$

where the last inequality follows from the fact that $\sum_{j \in J} q_j = k^3$. So, there must exists a subset of k machines with the desired property. □

Theorem 2. *There is no online algorithm for the flowtime problem with competitive ratio better than $\Omega(\sqrt{m})$.*

Proof. Let $\mathcal{A}$ be the schedule produced by an online algorithm. The theorem above implies that there is a set S of k machines such that there are $\Omega(k^2)$ jobs with unfinished components on at least one of these machines – let U be the set of such jobs. Number the machines in S from 1 to k. At each time instant from $t = T_1 + 1$, the adversary releases k jobs $j_1, \ldots, j_l$ of length 1 each such that $P_l^{j_l} = 1$ and $P_i^{j_l} = 0$ if $i \neq l$. We continue this till time $T_0 + X$. So, the schedule $\mathcal{A}$ is forced to have $\Omega(k^2)$ unfinished jobs till time $T_0 + X$. So, the weighted flowtime of $\mathcal{A}$ is at least $\sum_{j \in J-U} F^j(\mathcal{A}) + \Omega(k^2) \cdot (T_0 + X)$.

Consider some k jobs in J which require processing on all machines $1, \ldots, k-1$ in S. $\mathcal{O}$ does not schedule any component of these jobs before T_1. Further at time T_1, there are at most k jobs with unfinished components on machine k. Thus, in the schedule $\mathcal{O}$, there are at most $2 \cdot k$ jobs with unfinished components on S by time T_1 – let U' denote these jobs. Therefore, the weighted flow time of $\mathcal{O}$ is at most $\sum_{j \in J-U'} F^j(\mathcal{O}) + O(k) \cdot (T_0 + X)$. When X is very large compared to T_0, the ratio of the weighted flow time to $\mathcal{A}$ to that of $\mathcal{O}$ approaches k. This proves the theorem. □

7 Conclusion

There are many interesting open problems in the context of order scheduling model. We highlight two of them. Firstly, we are not aware of algorithmic techniques that can handle precedence constraints between different jobs. Any non-trivial approximation of even minimum makespan would be very interesting. We

were not able to use any of the standard techniques used for lower bounding the makespan in the presence precedence constraints. Secondly, it would be interesting to either get matching upperbounds or improve the lower bounds for the offline and online flowtime problem.

References

1. D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed applications. In *Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing*, 1995.
2. 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 *FOCS*, pages 32–44, 1999.
3. B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. In *ACM Symposium on Theory of Computing (STOC)*, pages 198–205, 1999.
4. N. Brixius, J. Linderoth, and J. Goux. Solving large quadratic assignment problems on computational grid. *Mathematical Programming, Series B*, 91:563–588, 2002.
5. S. Chakrabarti, C. Phillips, A. Schulz, D. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proc. of the 23rd Int. Colloquium on Automata, Languages and Programming*, pages 646–657, 1996.
6. Z. Chen and N. Hall. Supply chain scheduling: Assembly systems. Technical report, The Ohio State University, 2000.
7. R. Gandhi, M. Halldorsson, G. Kortsarz, and H. Shachnai. Improved results for data migration and open shop scheduling. In *Proc. of the 31st Int. Colloquium on Automata, Languages, and Programming*, pages 658–669, 2004.
8. J. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. Master-worker: An enabling framework for applications on the computational grids. In *Proceedings of the 9th IEEE Symposium on High Performance Distributed Computing*, pages 43–50, 2000.
9. L. Hall, A. Schulz, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: offline and online algorithms. *Mathematics of Operations Research*, 22:513–549, 1997.
10. H. Hoogeveen, P. Schuurman, and G. Woeginger. Non-approximability results for scheduling problems with minsum criteria. *Lecture Notes in Computer Science*, 1412:353–362, 1998.
11. J. Leung, H. Li, and M. Pindeo. *Multidisciplinery scheduling: Theory and Applications*, chapter Order Scheduling Models: an overview, pages 37–56. 2005.
12. J. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24:207–250, 2003.
13. C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.
14. M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:263–285, 1993.
15. M. Queyranne and M. Svirdenko. New and improved algorithms for minsum shop scheduling. In *Symposium on Discrete Algorithms*, pages 871–878, 2000.
16. R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *ACM Symposium on Theory of Computing (STOC)*, pages 475–484, 1997.

17. T. Roemer. A note on the complexity of the concurrent open shop problem. *Journal of scheduling*, 9:389–396, 2006.
18. A. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In *IPCO*, pages 301–315, 1996.
19. E. Wagneur and C. Sriskandarajah. Open shops with jobs overlap. *European Journal of Operations Research*, 71:366–378, 1993.
20. G. Wang and T. Cheng. Customer order scheduling to minimize total weighted completion time. *Omega*, 35:623–626, 2007.