Scheduling Jobs with Chain Precedence Constraints

Klaus Jansen*
Institut für Informatik und Praktische Mathematik
Universität zu Kiel
Kiel, Germany,
kj@informatik.uni-kiel.de

Roberto Solis-Oba†
Department of Computer Science
The University of Western Ontario
London, Canada,
solis@csd.uwo.ca

Abstract

The problem of scheduling jobs with precedence constraints is studied. A polynomial time approximation scheme is presented for the case when the machines are identical and the precedence graph partitions the jobs into groups of fixed size. As an interesting particular case, this algorithm is a polynomial time approximation scheme for scheduling jobs with chain or tree precedence constraints on identical machines, when each chain or tree is of constant size. A polynomial time approximation scheme is presented also for the case of uniformly related machines and chain precedence constraints when the number of machines is fixed.

1 Introduction

Scheduling problems are among the most fascinating and widely studied problems in Operations Research and Combinatorial Optimization. An important class of scheduling problems that arise in many industrial and scientific applications involves jobs that must be processed in come particular given order. These problems can be formally described as follows. A group $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of jobs needs to be processed by a given set $M = \{1, 2, \ldots, m\}$ of machines. Every machine can process only one job at any given time. Each job J_i has a length, p_i, and it must be processed without interruption on one machine. Furthermore, there is a set of constraints limiting the order in which the jobs can be processed. The problem is to schedule the jobs on the machines so that the *makespan* or completion time for the last job is minimized.

This is a fundamental problem in scheduling theory that has been extensively studied since the early works of Hu [9]. Despite its importance, and mainly due to its inherent complexity, there are very few results reporting algorithms to approximately solve the problem. In this paper we present polynomial time approximation schemes for several particular cases of this problem.

Job precedence constraints restricting the order in which jobs can be processed, are usually represented as a directed acyclic graph called a *precedence graph*. In a precedence graph each vertex represents a job and an edge from a job J_i to a job J_k means that J_i must be completed before J_k can start its processing. In this paper we are mainly interested in the case when the precedence graph consists of vertex disjoint paths, or *chains*. Every chain consists of a sequence of jobs $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(k)}$, that must be processed in the given order.

Concerning the set of machines to be used to process the jobs, we are interested in studying the problem when M consists of either identical or uniformly related machines. If M is formed by

*Author partially supported by EU projects APPOL, "Approximation and online algorithms", IST-1999-14084, and ARACNE "Approximation and randomized algorithms in communication networks", HPRN-CT-1999-00112.

†Author partially supported by the Natural Sciences and Engineering Research Council of Canada grant R3050A01.

identical machines, the time needed to process job J_i in any machine is always p_i. For the case of uniformly related machines, each machine j has a speed s_j, and so the time needed to process job J_i on machine j is p_i/s_j.

Using the standard three-field scheduling notation (see e.g. [14]) the above scheduling problem with arbitrary precedence constraints and identical machines is denoted as $P|prec|C_{\max}$. For the case of chain precedence constraints and identical machines we denote the problem as $P|chain|C_{\max}$, and on uniform machines as $Q|chain|C_{\max}$.

Du et al. [5] showed that problem $P|chain|C_{\max}$ is strongly NP-hard even with 2 machines. Kubiak et al. [12] showed that problem $Q|chain|C_{\max}$ is strongly NP-hard even if all jobs have unit processing time. Błażewicz and Liu [2] proved that the problem of scheduling unit time multiprocessor tasks with chain precedence constraints is strongly NP-hard. For the case of unrelated machines, Williamson et al. [16] showed that the problem of scheduling jobs with chain precedence constraints cannot be approximated within a factor smaller than 5/4 of the optimum unless P=NP. This result holds even if every job has unit processing time, a job can be processed by only one particular machine, and every chain in the precedence graph has at most 3 jobs. For the case of identical machines, unit processing times, and arbitrary precedence constraints, a result of Lenstra and Rinnooy Kan [15] implies that the problem cannot be approximated within a ratio smaller than 4/3 unless P=NP.

On the positive side, the list scheduling algorithm of Graham [7] is a $2 - 1/m$ approximation algorithm for the problem of scheduling jobs with arbitrary precedence constraints on identical machines. For identical machines and chain precedence constraints, Kunde [13] showed that the list scheduling algorithm has performance guarantee 5/3. For the case of unit processing time, and tree precedence constraints, the problem can be solved in polynomial time as showed by Hu [9].

For uniform machines, Chudak and Shmoys [4] and, independently, Chekuri and Bender [3] designed approximation algorithms with $O(\log m)$ performance guarantee for the case of arbitrary precedence constraints. For chain precedence constraints the algorithm in [3] has a performance guarantee of 6, and a recent algorithm by Woeginger [17] achieves performance ratio 2.

The main contributions of this paper are two a polynomial time approximation schemes. One is for problem $P|chain|C_{\max}$ for the case when every chain in the precedence graph has a fixed number of jobs. This algorithm can be extended to the case of arbitrary precedences for the particular case when the precedence constraints partition the set of jobs into groups $G_1, G_2, \ldots, G_k$ of fixed size. For each group G_i the constraints restrict the order in which its jobs can be processed, but jobs from different groups can be processed in any order. Our second polynomial time approximation scheme is for problem $Q|chain|C_{\max}$ for the case when m is fixed and the chains have arbitrary length.

The basic idea behind the first algorithm is to consider every group G_i of jobs as if it was a single task (a task is formed by glueing the jobs in a group G_i together, placing one after another in an order consistent with the precedence constraints). Then we schedule the large tasks by using dynamic programming. A second dynamic program is used in conjunction with the list scheduling algorithm [7] to schedule the small tasks inside the idle gaps left by the large ones.

The approach that we use for problem $Q|chain|C_{\max}$ is similar to the previous one. We consider every chain in the precedence graph as a single preemptive task. Then, we use the algorithm of Gonzalez and Sahni [6] to find an optimal schedule for the preemptive tasks, and finally, we convert this solution to a feasible schedule for the original set of jobs.

In order for this approach to work we need first to deal with the large jobs in a manner similar as described above. We show that a schedule of length close to the optimum can be obtained even if we allow very slow machines to remain idle.

The techniques that we use can also be used to design polynomial time approximation schemes for other scheduling problems. One of them is problem $Q|chain|C_{\max}$ for the case when the chains have fixed length and the number m of machines is arbitrary, but the fastest machine is a constant factor faster than the slowest one. Another problem is the flexible open shop problem with fixed number of machines. Here, every job consists of a sequence of operations which can be processed in any order, but at most one operation from a job can be processed at any given time. The machines used to process the jobs are unrelated, so the processing time of an operation depends on the machine that processes it. Due to space limitations we do not include these algorithms here.

2 Identical Machines

Consider a set $\mathcal{J}$ of jobs with arbitrary precedence constraints, and the corresponding precedence graph G. If we ignore the directions of the edges in G, the set of precedence constraints partitions the vertices of G into connected components. The set of jobs that belong to the same connected component is called a *task*. The processing time of a task is the sum of the processing times of its jobs. Note that there are no constraints in the order in which jobs belonging to different tasks can be processed. In this section we present a polynomial time approximation scheme for the problem of scheduling the jobs $\mathcal{J}$ so as to minimize the makespan, restricted to the case when each task consists of at most a fixed number μ of jobs. We assume $\mu > 1$, as otherwise the jobs $\mathcal{J}$ would be independent.

Let C^* be the length of an optimum schedule for the jobs $\mathcal{J}$.

Let $j_{\max}$ be the processing time of the largest chain. Clearly, $j_{\max} \leq C^*$ and $\frac{1}{m}\sum_{i=1}^{n} p_i \leq C^*$. By using the list scheduling algorithm [7] on the set of tasks we can find a feasible schedule for the original set $\mathcal{J}$ of jobs of length at most $\frac{1}{m}\sum_{i=1}^{n} p_i + j_{\max}$.

Consider the interval $[\max\{\frac{1}{m}\sum_{i=1}^{n} p_i, j_{\max}\}, \frac{1}{m}\sum_{i=1}^{n} p_i + j_{\max}]$, and divide it into $\frac{1}{\varepsilon}$ subintervals of the same size, where ε is a constant value and $0 < \varepsilon < 1$. We present below an algorithm that given a value d, it either finds a feasible schedule for $\mathcal{J}$ of length at most $(1+O(\varepsilon))d$, or it determines that no feasible schedule of length at most d exists. By using this algorithm and binary search over the above intervals, it is possible to find a value d^* such that $C^* \leq d^* \leq (1 + O(\varepsilon))C^*$, by probing at most $O(\log(\frac{1}{\varepsilon}))$ intervals. Therefore, for the rest of the paper we assume that we know a value d^* that is within a fraction $O(\varepsilon)$ of the length of the optimum schedule C^*.

2.1 Partitioning the Tasks

Let α be a constant with value

$$\alpha = \frac{\varepsilon^3}{2(\mu(1+1/\varepsilon)^{\mu-2})}. \tag{1}$$

Note that $\alpha < 1$. Let $\mathcal{K}_i$, $i > 0$, be the set of tasks whose processing times lie in the interval $(\varepsilon\alpha^i d^*, \varepsilon\alpha^{i-1} d^*]$, and let P_i be the total processing time of the tasks in $\mathcal{K}_i$. Let τ be the smallest non-negative integer such that $P_\tau \leq m\varepsilon d^*$.

Lemma 2.1 *The value of τ is at most $1/\varepsilon$.*

Proof. By the definition of τ, for every value $0 < i < \tau$, $P_i > m\varepsilon d^*$. Since the total processing time of all the jobs is at most md^*, then for at most $1/\varepsilon$ groups $\mathcal{K}_i$ it might be $P_i > m\varepsilon d^*$. □

We partition the set of tasks into 3 classes: $\mathcal{L}$, $\mathcal{M}$, and $\mathcal{S}$, that we call, respectively, the *large*, *medium*, and *small* tasks, as follows. The tasks with processing time at least $\varepsilon\alpha^{\tau-1}d^*$ form the class $\mathcal{L}$, tasks in J_τ belong to the class $\mathcal{M}$, and tasks with processing times smaller than $\varepsilon\alpha^\tau d^*$ are in the class $\mathcal{S}$.

We further partition the set of jobs belonging to large tasks into two classes $\mathcal{B}$ and $\mathcal{T}$, that we call the *big* and *tiny* jobs, as follows. Let $T_i \in \mathcal{L}$ be a large task. Let the jobs of T_i, indexed in non-increasing order of length be, $J_{i_1}, J_{i_2}, \ldots, J_{i_\mu}$. Let κ be the smallest index such that $p_{i_\kappa} \geq \frac{1}{\varepsilon}\sum_{j=\kappa+1}^{\mu} p_{i_j}$. Note that κ always exists since by choosing $\kappa = \mu$ this inequality is satisfied since then the right hand side has value zero. Jobs $J_{i_1}, \ldots, J_{i_\kappa}$ are placed in class $\mathcal{B}$ and the other jobs are assigned to class $\mathcal{T}$.

We can show that every big job J_i has processing time $p_i = \Omega(d^*)$.

Lemma 2.2 *A big job has processing time at least $d^*\delta$, where $\delta = 2\alpha^\tau/\varepsilon$.*
Proof. Consider a large task T_i. Since the jobs of T_i are indexed non-increasingly by length, and for every big job J_{i_ℓ}, $1 \leq \ell \leq \kappa$, its processing time is $p_{i_\ell} \leq \frac{1}{\varepsilon}\sum_{j=\ell+1}^{\mu} p_{i_j}$, then the smallest processing time of the big jobs is minimized when $\kappa = \mu$. In that case $p_{i_\kappa} = p_{i_\mu} \geq \varepsilon p_{i_\ell}/(1+\frac{1}{\varepsilon})^{\mu-\ell-1}$, for all $1 \leq \ell < \mu$, as one can easily verify by induction. Since the total processing time of T_i is at least $\varepsilon\alpha^{\tau-1}d^*$, then the largest job in T_i has length $p_{i_1} \geq (\varepsilon\alpha^{\tau-1}d^*)/\mu$. Hence, $p_{i_\kappa} \geq \varepsilon p_{i_1}/(1+\frac{1}{\varepsilon})^{\mu-2} \geq (\varepsilon^2\alpha^{\tau-1}d^*)/(\mu(1+\frac{1}{\varepsilon})^{\mu-2}) = 2\alpha^\tau d^*/\varepsilon$. □

Note that by the above lemma, every big job is at most a constant factor smaller than d^*. Therefore, in any schedule for $\mathcal{J}$, every machine processes at most a constant number of big jobs. Before building a schedule for the large and small tasks, we perform a series of changes in the processing times of the jobs and we also remove some of the tasks so as to simplify the problem without significantly increasing the length of an optimum solution. First, we increase the processing time of each big job by a factor of $(1+\varepsilon)$. This increases the length of an optimum schedule for $\mathcal{J}$ by at most a factor of also $(1+\varepsilon)$.

The idea behind this enlargement of the big jobs is that then we can discard the tiny jobs $\mathcal{T}$. The reason why we can do this is as follows. Once we have computed a schedule for the enlarged big jobs, we fix their starting times and then reduce their processing times to their original values. This creates some idle times in the machines and it also creates a window for each big job inside which the big job can be shifted without affecting the feasibility of the solution. Consider any long task T_i. Let J_{i_j} be a tiny job of T_i and let J_{i_q} be the first big job from T_i that must be processed either before or after tiny job J_{i_j}. The size of the window for J_{i_q} is $\varepsilon p_{i_q} \geq \varepsilon p_{i_\kappa} \geq \sum_{j=\kappa+1}^{\mu} p_{i_j}$. Therefore, we can simply insert J_{i_j} in the window of J_{i_q} in such a way that we get a feasible schedule without further increasing the overall length of the solution.

2.2 Scheduling the Tasks

If we increased the length of every big job by a factor of $1+2\varepsilon$, this would increase the length of an optimum schedule by at most a factor of $(1+2\varepsilon)$. By Lemma 2.2 this enlargement would increase the length of each big job by at least $2\delta\varepsilon d^*$. Hence we can make the following changes without further increasing the length of the schedule:

- round up the processing time of each big job to the nearest multiple of $\delta\varepsilon d^*$, and
- consider only *restricted schedules* in which the starting and finishing time of each big job is a multiple of $\delta\varepsilon d^*$.

After this rounding, each big job J_i has processing time no smaller than its original processing time p_i, and so it is very easy to transform a restricted schedule for the modified set of big jobs into a feasible schedule for the original big jobs.

Note that after removing the medium tasks $\mathcal{M}$ from $\mathcal{J}$ we get an instance of the scheduling problem in which the large tasks have length at least $\varepsilon\alpha^{\tau-1}d^*$ and the small tasks have length smaller than $\varepsilon \alpha^\tau d^*$. This gap between the size of the large and small tasks is crucial to compute a schedule for $\mathcal{J}$ of length close to the optimum.

By performing all the above transformations, we construct an instance of the problem in which if there is a schedule of length at most d^* for the original instance, then there is a schedule of length no more than $(1+\varepsilon)(1+2\varepsilon)d^*$ for the transformed instance. On the other hand, if there is no schedule of length at most $(1+\varepsilon)(1+2\varepsilon)d^*$ for the transformed instance, then there is no schedule of length at most d^* for the original one.

Since every big job has length at least δd^*, in a schedule of length d^* there is at most a constant number of big jobs in any machine. This fact allows us to use dynamic programming to find a short feasible schedule for the big jobs. Using dynamic programming we can also find a short, but infeasible, schedule for the small tasks, as we describe now. First, we round the processing time of each job belonging to a small task up to the nearest multiple of $\varepsilon d^*/(n\mu)$. This rounding increases the length of a schedule by at most a factor of εd^*.

We divide the interval $[0, (1+\varepsilon)^2(1+2\varepsilon)d^*]$ into sub-intervals of size $\delta\varepsilon d^*$. We call these sub-intervals, *slots*. Given a schedule S for the big jobs, a *configuration* for some machine m_i is a description of the slots in which the machine is busy processing big jobs. Since there are $\eta = \lceil (1+\varepsilon)^2(1+2\varepsilon)/(\delta\varepsilon)\rceil$ slots, then there are 2^η different configurations. Note that 2^η is a constant. Let us number the configurations from 1 to 2^η. A description of the schedule S is given by a 2^η-dimensional vector M, whose i-th component indicates the number of machines with configuration i.

Every entry of vector M can have value between 0 and m, hence there are at most $(m+1)^{2^\eta}$ different vectors M, and this many possible schedules for any set of big jobs. Since the number of different schedules is polynomial in m, we can use dynamic programming to decide in polynomial time whether there is a schedule of length at most $(1+\varepsilon)^2(1+2\varepsilon)d^*$ for the big jobs.

The idea is to take every large task T_i and choose for it the starting times for its big jobs. Since we consider only restricted schedules in which the starting time of a big job is a multiple of $\delta\varepsilon d^*$, there is only a constant number of possible starting times for the jobs of T_i. We consider only starting times that yield a feasible schedule for T_i and we check that, depending on how the other tasks have been scheduled, whether there are enough free machines to process the jobs of task T_i. Since this is a rather straightforward dynamic program, we do not give more details here.

The small tasks are scheduled in those slots and machines where no big jobs are being processed. We modify, as follows, the above dynamic program so that it also determines the slots where the small tasks must be processed. Given a restricted schedule S for the large and small tasks, we define L_g, $1 \le g \le \eta$, to be the total processing time of all jobs from small tasks that must be processed in any machine during slot g. Because of the way in which we have chosen the size of the small tasks, we ensure that a job from a small task might be processed in at most two consecutive slots g and $g+1$, and so L_g can have value as large as $m_g(\delta\varepsilon d^* + \varepsilon\alpha^\tau d^*)$, where m_g is the number of machines that do not process big jobs in slot g. Without loss of generality we might assume that $m_g \le n$, and therefore, since the processing times of the jobs from small tasks are multiples of $\varepsilon d^*/(n\mu)$, there are at most $\mu(\delta+\alpha^\tau)n^2$ possible values for the load L_g.

Consider again the restricted schedule S. A load vector $L = (L_1, L_2, \ldots, L_\eta)$ can be used to describe the slots where the small tasks are scheduled in S. The total number of different load vectors L is $O(n^{2\eta})$, which is polynomial in the number of tasks.

Our dynamic program for scheduling the large tasks is modified by augmenting each configuration with a load vector L that indicates the slots where the small tasks must be scheduled. Note that the schedule produced by the dynamic program might not be feasible, since there might be conflicts among the small tasks assigned to the same slot. We use the list scheduling algorithm for finding in each slot a feasible schedule for the small tasks.

2.3 Length of the Schedule

To compute the length of the schedule produced by our algorithm, let us consider first the contribution of the medium tasks $\mathcal{M}$ to the total length of the solution. Each one of these tasks has processing time at most $\varepsilon\alpha^{\tau-1}d^*$ and the total processing time of the jobs from medium tasks is at most $m\varepsilon d^*$. The medium tasks are scheduled using the list scheduling algorithm, by considering every task as a single non-preemptive job. This schedule is appended to the solution that our algorithm computes for the remaining tasks. The length of the schedule for $\mathcal{M}$ is at most $(\varepsilon + \varepsilon\alpha^{\tau-1})d^*$.

The tasks in $\mathcal{J} \setminus \mathcal{M}$ are scheduled using our dynamic program, and the length of the solution that our algorithm produces for them is at most $(1+\varepsilon)^2(1+2\varepsilon)d^*$. As we indicated above, to transform this solution into a feasible schedule, we use the list scheduling algorithm to schedule the small tasks within each slot. Note that the schedule obtained for each slot might increase the length of the slot, and hence the total length of the solution.

Each slot initially has length $\delta\varepsilon d^*$ and the total processing time of jobs from small tasks assigned to some slot g is $L_g \leq m_g(\delta\varepsilon d^* + \varepsilon\alpha^\tau d^*)$. Since the small tasks have length at most $\varepsilon\alpha^\tau d^*$, then the list scheduling algorithm finds a schedule of length no larger than $(\delta\varepsilon + 2\varepsilon\alpha^\tau)d^*$ for the small tasks in any slot g. This represents an increase of $2\varepsilon\alpha^\tau d^*$ in the length of a slot. Since there are $\lceil(1+\varepsilon)^2(1+2\varepsilon)/(\delta\varepsilon)\rceil$ slots, then the total increase in the length of the schedule is at most $2\varepsilon\alpha^\tau d^* \lceil(1+\varepsilon)^2(1+2\varepsilon)/(\delta\varepsilon)\rceil < [(1+\varepsilon)^2(1+2\varepsilon)(2\alpha^\tau)/\delta + 2\varepsilon\alpha^\tau]d^*$.

Adding to this the length of the schedule for the medium tasks, the total length of the solution produced by our algorithm is at most $[(1+\varepsilon)^2(1+2\varepsilon)(1+\frac{2\alpha^\tau}{\delta}) + 2\varepsilon\alpha^\tau + \varepsilon + \varepsilon\alpha^{\tau-1}]d^* \leq [(1+\varepsilon)^3(1+2\varepsilon) + \varepsilon + 3\varepsilon\alpha^{\tau-1}]d^*$.

Theorem 2.1 *The algorithm finds a solution of length at most $(1+\epsilon)C^*$ for any value $\epsilon > 0$.*

Proof. By choosing $\varepsilon < 1$ and $\epsilon = 27\varepsilon$, the total length of the solution is at most $(1+\epsilon)C^*$. □

3 Fixed Number of Uniform Machines and Chain Precedence Constraints

In this section we assume that the precedence graph G is formed by vertex disjoint paths or chains. The set of jobs that belong to the same chain forms a *task*. Let machine j have speed s_j, so the time needed to process job J_i on machine j is p_i/s_j. Scale the speeds and index the machines so that $1 = s_1 \leq s_2 \leq \cdots \leq s_m$.

In the next section we describe an algorithm for the case when the fastest machine is at most a constant factor faster than the slowest machine. Then we extend the algorithm to consider the case of arbitrary speeds.

3.1 The Fastest Machine is a Constant Factor Faster than the Slowest Machine

Let $s_m/s_1 = r$, where r is a constant. Let C^* be the makespan of an optimum schedule for the set of jobs $\mathcal{J}$. Let $D = \sum_{J_i \in \mathcal{J}} p_i$, so if all jobs are processed on the fastest machine the total length of this schedule is D/r. It is not difficult to see that $D/(mr) \leq C^* \leq D/r$. We can use binary search on the interval $[D/(mr), D/r]$ and the procedure described below to find a value d^* such that $C^* \leq d^* \leq (1+\varepsilon)C^*$. So, without loss of generality we assume that we know the value d^*.

Let $\varepsilon > 0$, $\delta > 0$, and $\beta < 1$ be constants to be determined later. We partition the jobs into groups $G_1, G_2, \ldots, G_k$, placing in group G_i all jobs J_j of length $p_j \in [d^*\beta^i, d^*\beta^{i-1})$. Then, we find the first group G_τ for which $\sum_{J_i \in G_\tau} p_i \leq \delta d^*$. Note that, as in the previous section, τ has a constant value since $\tau \leq \frac{mr}{\delta}$. Group G_τ partitions the set of jobs into 3 sets: the large jobs $\mathcal{L} = \{J_i \mid p_i \geq d^* \beta^{\tau-1}\}$, the medium jobs $\mathcal{M} = G_\tau$, and the small jobs $\mathcal{S} = \{J_i \mid p_i < d^* \beta^\tau\}$.

We round the processing times of the jobs and constrain the set of possible starting times for the large and medium jobs so that we can use enumeration and dynamic programming for constructing a good schedule for $\mathcal{J}$. To this effect, we multiply the processing time of each large and medium job by $(1+\varepsilon)$. This increases the length of an optimum schedule by at most the same factor. Then, we round the processing time of each small job up to the nearest multiple of $\varepsilon d^*/n$. This latter rounding might further increase the length of an optimum schedule by a factor of at most $(1+\varepsilon)$.

Consider a feasible schedule $\tilde{S}$ for the jobs $\mathcal{J}$ assuming these enlarged processing times. Fix the starting time of each job as determined in $\tilde{S}$. Then, we round down, as indicated below, the processing time of each large and medium job depending on the speed of the machine that is selected to process it. If large job J_i is scheduled for processing on machine j, then its processing time is rounded down to the nearest multiple of $\varepsilon d^* \beta^{\tau-1} s_j / r$. Similarly, if medium job J_k is scheduled for processing on machine j, its processing time is rounded down to the nearest multiple of $\varepsilon d^* \beta^\tau s_j / r$. Observe that this rounding does not further increase the length of an optimum schedule, and the rounded processing times are not smaller that the original processing times. Let $\tilde{p}_i$ describe the rounded processing time of job J_i.

Since the starting times of the jobs were fixed, by shrinking the processing times of the medium and large jobs we create an idle window inside which every medium and large job can be shifted without affecting the feasibility of the schedule. This window is of size at least $\varepsilon d^* \beta^{\tau-1}/r$ for the large jobs and of size $\varepsilon d^* \beta^\tau / r$ for the medium jobs. Thus, we can shift the medium and large jobs so that their starting times are respectively multiples of $\varepsilon d^* \beta^\tau / r$ and $\varepsilon d^* \beta^{\tau-1}/r$, without increasing the length of the schedule.

3.1.1 Restricted Schedules

For the rest of the paper we consider only *restricted schedules* in which medium jobs J_i have processing times $\tilde{p}_i$ and starting times that are multiples of $\varepsilon d^* \beta^\tau / r$, and the large jobs J_j have processing times $\tilde{p}_j$ and starting times that are multiples of $\varepsilon d^* \beta^{\tau-1}/r$. By the above discussion an optimum restricted schedule has length at most $(1+3\varepsilon)d^*$.

To find a near optimum restricted schedule for the jobs $\mathcal{J}$, we first divide the interval $[0, (1+3\varepsilon)d^*]$ into sub-intervals of size $\varepsilon d^* \beta^{\tau-1}/r$, that we call *large intervals*, and intervals of size $\beta^\tau \varepsilon d^*/r$ that we call *medium intervals*. Note that once we have determined a restricted schedule and rounded the processing times of the jobs as described above, the processing time of a large job is a multiple of the length of a large interval and the processing time of a medium job is a multiple of the length of a medium interval.

The number of medium and large jobs is at most $\lceil D/\beta^\tau d^* \rceil \leq \lceil D/(\beta^\tau \frac{D}{mr}) \rceil = \lceil \frac{mr}{\beta^\tau} \rceil$, which is a constant value. Also the number of medium and of large intervals are constant values. Therefore, the number of possible restricted schedules of length at most $(1+3\varepsilon)d^*$ for the large and medium jobs is also constant. We build all these possible schedules, and for each one of them we use a dynamic program to assign small jobs to idle large intervals.

Consider a restricted schedule $\bar{S}$ for the large and medium jobs. Round the processing times of large and medium jobs as described above. For this schedule, consider a large interval I. Let $\mathcal{M}_I$ be the set of medium jobs that are completely contained in interval I and have processing time strictly smaller than the length of I.

We briefly describe the dynamic program for assigning small jobs to the large intervals of $\bar{S}$. Let v be a vector of dimension equal to the number of large intervals. Each entry of the vector stores $m+1$ pieces of information describing the load of the small jobs and medium jobs from $\mathcal{M}_I$ assigned to each large interval I, specifically: (i) the processing times of the $m-1$ largest chains formed by medium jobs from $\mathcal{M}_I$ and small jobs assigned to I, (ii) the total processing time of medium jobs from $\mathcal{M}_I$ and small jobs assigned to the interval, and (iii) the total processing time of the last chain of small and medium jobs $\mathcal{M}_I$ assigned to the interval by the dynamic program. The dynamic program fills up a table containing all possible vectors v. Because of the way in which we have discretized the processing times of the jobs, every entry of vector v can have $O(n^{m+1})$ possible values, and so the aforementioned table has size polynomial in n.

In the dynamic program we consider the jobs of every task in order, and place a small job J_i in a large interval I if the following condition (2) is satisfied. Let m_I be the number of machines available in I for processing small and medium jobs $\mathcal{M}_I$. Let $t_1 \geq t_2 \geq \cdots \geq t_{m_I-1}$ be the lengths of the largest $m_I - 1$ chains (including the chain containing job J_i) formed by medium $\mathcal{M}_I$ and small jobs placed so far in I. Let $T_j = \sum_{i=1}^{j} t_i$. Let T be the total length of small and medium jobs $\mathcal{M}_I$ placed in I. Let $s'_1 \geq s'_2 \geq \cdots \geq s'_{m_I}$ be the speeds of the machines available for processing small and medium jobs $\mathcal{M}_I$ in interval I, and let $S_j = \sum_{i=1}^{j} s'_i$. We place job J_i in interval I only if

$$\max\ \{\max_{1 \leq j < m_I}\{T_j/S_j\}, T/S_{m_I}\} \leq \varepsilon d^* \beta^{\tau-1}/r + \beta^\tau d^*. \tag{2}$$

If we think of each chain of small and medium jobs assigned to I as a single preemptive job, then condition (2) ensures, as shown in [6], that there is a preemptive schedule of length at most $\varepsilon d^* \beta^{\tau-1}/r$, the length of interval I. The second term in the right hand side on inequality (2) is due to the fact that in an optimum schedule a small job might cross the boundary of a large interval.

Since the processing times of small jobs are small compared to the length of a large interval, then the above preemptive schedule can be transformed into a feasible schedule for the small and medium jobs with only a slight increase in the length of a large interval as shown in the next section.

Lemma 3.1 *The dynamic program produces a solution of length at most $(1+3\varepsilon)d^*$.*

Proof. Consider an optimum schedule S^* for the jobs $\mathcal{J}$ and round the lengths of the jobs and their starting times as described above. The length of the schedule increases by at most a factor of $(1+3\varepsilon)$. It is not difficult to see that the load of small and medium jobs placed in any large interval I must satisfy condition (2). □

The solution obtained by the dynamic program might not be feasible since there may be conflicts among the small and medium jobs inside each large interval. We find a feasible schedule for each large interval as follows.

3.1.2 Scheduling Jobs Inside a Large Interval

Consider a large interval I. Inside this interval we consider only those machines that, after scheduling the medium and large jobs, have some idle time inside I. Each small and medium job $\mathcal{M}_I$ is assigned its original processing time. There are two cases that we need to consider.

Case 1: the total load of the medium jobs completely or fractionally assigned to I is at most $\frac{\delta}{\varepsilon} L_I$ where $L_I = \varepsilon d^* \beta^{\tau-1}/r$ is the length of a large interval and $\delta = \frac{1}{2}\varepsilon\beta$. Because of the way in which the starting times and processing times of medium jobs have been rounded, if part of a medium job is assigned to interval I then at least a fraction $\varepsilon d^* \beta^\tau / r$ of the job should be inside the interval. Therefore, the above condition implies that no medium jobs are assigned to I.

We use the algorithm of Gonzalez and Sahni [6] to find a feasible schedule for the small jobs in each large interval. To use this algorithm we treat each task as if it was a single preemptive job. The algorithm schedules all the preemptive jobs inside the interval, but it might produce up to $2(m-1)$ preemptions. We change the preemptive jobs back to tasks and identify the jobs that are preempted. For a small job J_i that receives fractional assignments we replace one of its fractional assignments by the whole job. This might increase the size of a large interval by at most $2(m-1)\beta^\tau d^*$.

Since there are $\lceil r(1+3\varepsilon)/(\varepsilon\beta^{\tau-1})\rceil$ large intervals, scheduling the small jobs inside the large intervals might increase the overall length of the schedule by at most $2(m-1)\beta^\tau d^* \lceil r(1+3\varepsilon)/(\varepsilon\beta^{\tau-1})\rceil = 2(m-1)r\beta d^*(1+3\varepsilon)/\varepsilon$. Choosing $\beta = \varepsilon^2/(4(m-1)r(1+3\varepsilon))$, the increase is at most εd^*.

Case 2: the load of the medium jobs in I is larger than $\frac{\delta}{\varepsilon}L_I$. Note that since by definition the total processing time of all the medium jobs is at most δd^*, then this case can happen in at most $(\delta d^*)/(\frac{\delta}{\varepsilon}L_I) = \varepsilon\frac{d^*}{L_I}$ intervals, or in other words, this condition can happen in only an ε fraction of the large intervals.

We split I in 3 sub-intervals. In the first subinterval we place those medium jobs that start processing before I and finish processing inside I. In the third interval we place the medium jobs that start processing inside I but finish processing after I. In the middle sub-interval we schedule the small and medium jobs in $\mathcal{M}_I$ using Gonzalez and Sahni's algorithm as described above.

The schedule produced by this algorithm increases the length of interval I by at most $(2 + 2(m-1)r)L_I$, since the first and third subintervals each have length at most L_I, and rounding the preemptive schedule of a job in the middle subinterval increases the length of the interval by at most rL_I. Summing over all intervals where this case happens, we get a total increase in the length of the schedule of $(2+2(m-1)r)\varepsilon d^*$.

Adding the increases in all large intervals (cases 1 and 2), we get an overall increase in the length of the solution of at most $(3+2(m-1)r)\varepsilon d^*$.

Theorem 3.1 *The algorithm produces a solution of length at most* $(1+\epsilon)C^*$*, for any value* $\epsilon > 0$.

Proof. Choose ε so that $\epsilon = (7+2(m-1)r)\varepsilon$. □

3.2 Machines with Arbitrary Speeds

To deal with the case of machines with arbitrary speeds, we find the largest speed s_ℓ, $\ell \geq 1$, such that $s_\ell \geq \frac{1}{\varepsilon}\sum_{i=1}^{\ell-1} s_i$. Note that if $\ell = 1$ then we can use the algorithm of the previous section because then $s_m \leq \frac{1}{\varepsilon}\sum_{i=1}^{m-1} s_i \leq \frac{1}{\varepsilon}\left[(\frac{1}{\varepsilon}+1)\sum_{i=1}^{m-2} s_i\right] \leq \frac{1}{\varepsilon}\left(\frac{1}{\varepsilon}+1\right)^{m-2} s_1$, and therefore, the ratio s_m/s_1 has constant value. So let us assume that $\ell > 1$.

Let S^* be an optimum schedule for the jobs $\mathcal{J}$. We show that it is possible to move all jobs to the fastest $m-\ell+1$ machines (so that the slowest $\ell-1$ machines remain idle) in such a way that the length of the schedule is increased by at most a factor of $(1+\varepsilon)$.

The idea is to scan S^* starting at time zero and moving towards the end of the schedule. Every time that we find an job J_i in one of the slowest $\ell-1$ machines, we find an job J_j belonging to the same task as J_i such that either:

- J_j is the first job that must be processed after J_i in one of the fastest $m-\ell+1$ machines, or
- J_j is the last job that must be processed before J_i in one of the fastest $m-\ell+1$ machines.

If job J_j does not exist, then the whole task to which J_i belongs is processed in S^* on the slowest $\ell-1$ machines. In such a case we schedule this task on the fastest machine m at the end of the schedule. Otherwise, if job J_j exists, then all jobs that must be processed between J_j and J_i are scheduled contiguous to J_j in the same machine where J_j is processed in S^*. Note that some jobs in other machines might need to be shifted to the right to get a feasible schedule.

This process is repeated until all jobs scheduled by S^* in the first $\ell - 1$ machines are moved to one of the fastest machines. Let C^* be the length of schedule S^*. The total increase produced by the above algorithm in the length of the schedule is at most $C^* \frac{s_1}{s_\ell} + \cdots + C^* \frac{s_{\ell-1}}{s_\ell} \leq C^* \frac{1}{s_\ell} \sum_{i=1}^{\ell-1} s_i \leq C^* \varepsilon$.

Finally, we use the algorithm described in the previous section to schedule the jobs in the fastest $m - \ell + 1$ machines. We can do this since by the definition of ℓ, $s_m \leq \frac{1}{\varepsilon}(1 + \frac{1}{\varepsilon})^{m-\ell-1}$, and therefore, the ratio s_m / s_ℓ has constant value. By the above result this schedule is at most $(1 + \epsilon)$ times larger than the optimum, for any value $\epsilon > 0$.

Theorem 3.2 *This algorithm finds a schedule with makespan at most $(1 + \epsilon)C^*$ for any $\epsilon > 0$.*

References

[1] A.K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, Scheduling Multiprocessor Tasks on Dedicated Processors, *Algorithmica* to appear. A preliminary version appeared in the *Proceedings of the 5th Annual European Symposium on Algorithms*, LNCS 1284, pp. 1–12.

[2] J. Błażewicz and Z. Liu, Scheduling multiprocessor tasks with chain constraints, *European Journal on Operations Research* 94, 1996, pp. 231–241.

[3] C. Chekuri and M.A. Bender, An efficient approximation algorithm for minimizing makespan on uniformly related machines, IPCO 1999, LNCS 1412, pp. 383–393.

[4] F.A. Chudak and D.B. Shmoys, Approximation algorithms for precedence constrained scheduling problems on parallel machines that run at different speeds, *Journal of Algorithms* 30, 1999, pp. 323–343.

[5] J. Du, J.Y.T. Leung, and G.H. Young, Scheduling chain-structured tasks to minimize makespan and mean flow time, *Information and Computation* 92, 1991, pp. 219–236.

[6] T. Gonzalez and S. Sahni, *Preemptive scheduling of uniform processor systems*, Journal of the ACM, 25 (1978), pp. 92–101.

[7] R.L. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal* 45, 1966, pp. 1563–1581.

[8] D.S. Hochbaum and D.B. Shmoys, Using dual approximation algorithms for scheduling problems: practical and theoretical results, *Journal of the ACM* 23, 1976, pp. 317–327.

[9] T.C. Hu, Parallel sequencing and assembly line problems, *Operations Research*, 9, 1961, pp. 841–848.

[10] K. Jansen, M. Sviridenko, and R. Solis-Oba, Makespan minimization in job shops: a polynomial time approximation scheme, STOC 1999, pp. 394–399.

[11] K. Jansen and M. Sviridenko, Polynomial time approximation schemes for the multiprocessor open and flow shop scheduling problems, STACS 2000, LNCS 1770, pp. 455-565.

[12] W. Kubiak, B. Penz, and D. Trystram, Scheduling chains on uniform processors with communication delays, to appear in *Journal of Scheduling.*

[13] M. Kunde, Nonpreemptive LP-scheduling on homogeneous multiprocessor systems, *SIAM Journal on Computing* 10, 1981, pp. 151–173.

[14] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Sequencing and scheduling: Algorithms and complexity. In: S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.) *Logistics of Production and Inventory*, Handbooks in Operations Research and Management Science 4, North Holland, Amsterdam, pp. 445–522.

[15] J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity of scheduling under precedence constraints, *Operations Research* 26, pp. 22–35.

[16] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevastianov, and D.B. Shmoys, Short shop schedules, *Operations Research* 45, 1997, pp. 288–294.

[17] G. Woeginger, A comment on scheduling uniform machines under chain-type precedence constraints, *Operations Research Letters*, 26 (3), 2000, pp. 107–109.