Fourteen Notes on Equal-Processing-Time Scheduling

Vadim Timkovsky
South University, Royal Palm Beach, FL, United States
40 PUBLICATIONS **375** CITATIONS

4OR 2: 111–127 (2004)

DOI: 10.1007/s10288-003-0024-4

Ten notes on equal-processing-time scheduling

At the frontiers of solvability in polynomial time

Philippe Baptiste[1], Peter Brucker[2], Sigrid Knust[2] and Vadim G. Timkovsky[3]

[1] Université de Technologie, UMR CNRS, HenDiaSyC, 60205 Compiègne CX, France and École Polytechnique, France (e-mail: philippe.baptiste@polytechnique.fr)
[2] Universität Osnabrück, FB Mathematik/Informatik, 49069 Osnabrück, Germany (e-mail: {peter.brucker; sigrid.knust}@mathematik.uni-osnabrueck.de)
[3] CGI Group Inc., 1 Dundas St. W., Suite 2700, Toronto, Ontario M5G 1Z3, Canada and McMaster University, Canada (e-mail: vadim.timkovsky@cgi.com)

Received: February 2003 / Revised version: September 2003

Abstract. Equal-processing-time scheduling problems whose complexity status has been unknown are shown to be solved in polynomial time by well-known and relatively new techniques. Single-machine, parallel-machine, parallel-batch, open-shop, flow-shop and job-shop environments are touched upon.

Key words: Scheduling, computational complexity, polynomial time

AMS classification: 90B35, 68Q25

Introduction

Looking through the listing of complexity results for scheduling problems (see Brucker and Knust, web page), one can observe that the vast majority of problems whose complexity status is open involve jobs with equal-processing-time operations. Hence, the state-of-the-art in the study of the complexity of scheduling problems has reached the level at which approaches and techniques successfully working for equal-processing-time scheduling become especially important.

This paper is devoted to polynomial-time solutions to equal-processing-time scheduling problems in single-machine, parallel-machine, parallel-batch, open-shop, flow-shop (involving a robot) and job-shop environments, an area where

All correspondence to: Vadim G. Timkovsky some progress has been recently made. When the results we present here are variations of earlier results that are explicitly discussed in cited papers, we give only short sketches on how to obtain polynomial-time algorithms. Given such sketches the reader can easily restore necessary details. Among modelling techniques we use reductions to well-studied scheduling problems, transversal graphs, flows in networks, linear programming and dynamic programming. Following this list our notes appear in the sequence of (more or less) increasing their richness.

Our main goal is to extend the existing frontiers of solvability in polynomial time in accordance with the well-known classification of scheduling problems and obtain more understanding of methods and techniques that successfully lead to solutions in polynomial time. Therefore, the problems we consider in these notes might not be of a theoretical or practical interest other than studying the computational complexity of scheduling. Some of the problems seem to be rather simple (once a solution is given) or exotic (like problems with a fixed number of jobs or an unconventional criterion). Nevertheless, they deserve attention because they also represent frontiers of solvability in polynomial time in scheduling theory nowadays, and hence carry an information about starting points of efficient approaches to more important problems in the future.

Through all the notes we follow denotations and assumptions that are commonly accepted in scheduling theory (see Graham et al. 1979; Brucker 1998): C_j, w_j, r_j, d_j and D_j are the completion time, the weight, the release date, the due date and the deadline of a job J_j, $j = 1, 2, \ldots, n$, respectively; p_j is the processing time of a single-operation job that can be processed on any of the parallel machines M_k, $k = 1, 2, \ldots, m$; p_{ij} is the processing time of the ith operation, O_{ij}, $i = 1, 2, \ldots, m_j$, of a multi-operation job, which is a chain of operations in a job shop or an unordered set of m operations in an open shop; O_{ij} has to be processed on $M_{\mu_{ij}}$ in a job shop or M_i in an open shop.

The equalities $p_j = p$ and $p_{ij} = p$ specify equal processing times. The equality $n = k$ means that the number of jobs is fixed and equal to k. The job length is meant to be the total processing time of job operations.

We write $J_j \lessdot J_k$ if J_j is an immediate predecessor of J_k in accordance with specified precedence constraints for jobs. Appended integer δ in parentheses like in $\lessdot(\delta)$ or chains(δ) indicates common precedence delay δ for given precedence constraints. Thus, the start time of J_k must be at least δ time units later than the completion time of J_j if $J_j \lessdot (\delta) J_k$.

For numbers $a_1, a_2, \ldots, a_n$ we set $a_{\min} = \min_j a_j$ and $a_{\max} = \max_j a_j$. If not stated otherwise, we assume that $r_{\min} = 0$. All numerical parameters are assumed to be integers. We touch upon all classical criteria. The most general among them is $\sum f_j$, where f_j is a nondecreasing cost function which values are computable in constant time. The criterion denotation will be omitted if a criterion is immaterial.

A key observation we use to obtain polynomial-time solutions to nonpreemptive problems with equal processing times is that only a polynomial number of time points proves to be enough to be taken into consideration. Indeed, since we consider only criteria that are nondecreasing functions of job completion times, it is enough to seek for an optimal schedule among active schedules (i.e., those where a shift of a job or its part to the left makes them infeasible), start times and completion times of jobs in active nonpreemptive schedules belong to the set

$$\mathcal{T} = \{t : \exists j \in [1, n] : \exists l \in [0, nm_{\max}] : t = r_j + lp\},$$

where $m_{\max} = 1$ in the case of single-operation jobs. Thus, $\mathcal{T}$ contains at most $n(nm_{\max} + 1)$ time points.

We will also use $\mathcal{T}_{\text{pmtn}}$ and $\mathcal{T}_{\text{no-wait}}$ to denote a time space required to store preemptive schedules and no-wait schedules in shops. In comparison with $\mathcal{T}$, they depend on particularities of preemptive problems and no-wait shop problems and will be estimated further by different ways.

1 $Q|\text{pmtn}, p_j = p|\sum U_j$

Since late jobs can be arbitrarily scheduled we can consider schedules of only early jobs. A straightforward interchange argument shows that we can assume that the early jobs are those with largest due dates. In other words, if $d_1 \geq d_2 \geq \dots \geq d_n$, then $J_1, J_2, \dots, J_i$ are early jobs in an optimal schedule if and only if there is no schedule of the jobs $J_1, J_2, \dots, J_i, J_{i+1}$ which are all early. For a fixed i we can apply an $O(n \log n + mn)$ algorithm of Labetoulle et al. (1984) for $Q|\text{pmtn}, r_j, p_j = p|C_{\max}$ to the symmetrical instance with release dates $d_1, d_2, \dots, d_i$ to check whether there exists a schedule where the jobs $J_1, J_2, \dots, J_i$ are early. A binary search on i gives a solution in time $O(n \log^2 n + mn \log n)$.

2 $1|\text{pmtn}, \text{prec}, r_j, p_j = p|\sum C_j$

As we show, there is no advantage to preemptions in this problem. Thus it can be solved in polynomial time by an algorithm of Simons (1983) for $1|\text{prec}, r_j, p_j = p|\sum C_j$.

Let t be the earliest preemption time, and let J_j be the job preempted at time t, started at time S_j and completed at time C_j in a preemptive schedule. Let $\mathcal{K}$ and k denote the set and number of jobs, respectively, that are started not earlier than t and completed earlier than C_j. We can reconstruct the schedule in the time interval $[t, C_j]$ by moving all parts of J_j into the time interval $[t, S_j + p]$, so that J_j becomes non-preempted, and placing all parts of other jobs in $[t, C_j]$ without changing their order into the time interval $[S_j + p, C_j]$. The reconstruction obviously observes precedence constraints and release dates.

One can make sure that the decrease of completion time of J_j is at least kp and that the increase of total completion time of jobs in $\mathcal{K}$ is at most $k(p - t + S_j)$. Thus, the reconstruction decreases the total completion time by at least $k(t - S_j)$. Recursively applying the reconstruction procedure we obtain a nonpreemptive schedule that is not worse that the original preemptive schedule.

3 $Pm|\text{intree}, p_j = 1|\sum C_j$

Using the fact that the precedence relation is an intree, it is easy to make sure by contradiction that the number of busy machines is not increasing in time in any optimal schedule. Hence, an optimal schedule defines the time unit $[t, t + 1]$ that contains a set L with less than m independent jobs such that either $t = 0$ or the time unit $[t - 1, t]$, where $t > 0$, contains m jobs.

It is evident that L is the set of leaves of the subtree S scheduled in $[t, C_{\max}]$, and the set F of other jobs is a forest with mt jobs scheduled in $[0, t]$. Let C_j denote the completion time of J_j in an optimal schedule. Then the total completion time of the schedule can be written as

$$m(1 + 2 + \ldots + t) + \sum_{J_j \in S} C_j.$$

Note that any active schedule for S is optimal because it requires less than m machines. Once L is given, F and S can be obviously restored in linear time. The subschedule for F and the time point t can also be found in linear time by an algorithm of Hu (1961). To find a set L we need to check all $O(n^m)$ subsets with less than m jobs. Since m is fixed, we obtain a solution in polynomial time.

$Om|\text{no-wait, intree}, p_{ij} = 1|\sum C_j$ can also be solved in polynomial time by a reduction to $Pm|\text{intree}, p_j = 1|\sum C_j$ (see Brucker et al. 1993).

4 $F3, R1|r_j, t_k, p_{ij} = 1|C_{\max}$

The addition $R1$ to the machine environment field denotes a single robot in a flow shop that transports the jobs between the machines, and t_k denotes the transportation time from M_k to M_{k+1}. We assume that the transportation time t_k from M_k to M_{k+1} by the robot includes the time to load a job at M_k onto the robot and to unload it at M_{k+1}. If $t_k = 0$, then a job is available at M_{k+1} immediately after it leaves M_k, i.e., the robot is not needed. Moves of the empty robot are assumed to be done instantaneously.

Note that the robotic flow-shop problem is a special case of $J4|r_j|C_{\max}$ where the jobs have identical data with the exception of only release dates. Namely, O_{1j}, O_{3j}, O_{5j} are assigned to be processed for one unit on M_1, M_2, M_3, respectively, and O_{2j} and O_{4j} are assigned to be processed during times t_1 and t_2, respectively, on M_4 that stands for the robot. Further we denote O_{2j} and O_{4j} as a_j and b_j, respectively. Instances of the problem are of the following two types.

An instance with $t_2 = 0$ or $t_1 = 0$ is equivalent to the instance of $1|r_j|C_{\max}$ with n jobs a_j or b_j, processing times t_1 or t_2 and release dates $r_j + 1$ or $r_j + 2$, respectively. Ordering the jobs by nondecreasing release dates obviously gives an optimal schedule.

An instance with $t_1 \geq 1$ and $t_2 \geq 1$ is equivalent to the instance of $1|\text{chains}(1), r_j|C_{\max}$ with $2n$ jobs a_j and b_j with processing times t_1 and t_2, release dates $r_j + 1$

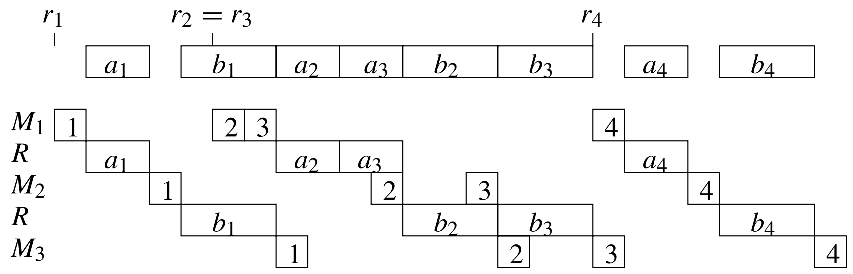


Fig. 1. Extension of a feasible single-machine schedule to a feasible robotic flow-shop schedule

and $r_j + 2 + t_1$, respectively, and unit-delay-time precedence constraints $a_j \lessdot (1)\, b_j$. It is not hard to make sure that a feasible single-machine schedule with maximum completion time ℓ for the latter instance presents a feasible schedule for the robot that can be easily extended to a feasible robotic flow-shop schedule with maximum completion time $\ell + 1$ for the former instance (see Fig. 1).

On the other hand, a feasible robotic flow-shop schedule contains a schedule for the robot that is isomorphic to a feasible single-machine schedule. Hence, there is an obvious one-to-one correspondence between feasible single-machine schedules and feasible robotic flow-shop schedules that changes the maximum completion time by one. Thus, to solve the robotic flow-shop problem it is sufficient to only have an algorithm solving the related instance of the single-machine problem. It is also not hard to check by an exchange argument that the following $O(n \log n)$-algorithm does that.

Without loss of generality, we assume that the lists of jobs $a_1, a_2, \dots, a_n$ and $b_1, b_2, \dots, b_n$ are ordered by nondecreasing release dates. Suppose the first $i - 1$ and $j - 1$ jobs from the lists, respectively, have been already scheduled, and let t be the completion time of the job which has been scheduled last (it can be a_{i-1} or b_{j-1}). Let C_k denote the completion time of a_k, $k = 1, 2, \dots, n$. Then we schedule a_i next as early as possible if $\max\{r_i + 1 - t, 0\} \le \max\{C_j + 1 - t, 0\}$ or b_j otherwise.

5 $J|\text{prec}, r_j, p_{ij}=p, n=k|\sum f_j$

Recall that the number of jobs in this problem is fixed. Following Middendorf and Timkovsky (1999), we reduce the problem to finding a shortest path in a transversal graph associated with the problem. Vertices of such a graph are transversals crossing precedence constraints. They present partial instances of a given instance of the problem. In this case, transversals are vectors

$$u = (v_1, v_2, \dots, v_k; t_1, t_2, \dots, t_k),$$

where $v_j \in \{0, 1, 2, \dots, m_j\}$ corresponds to the operation $O_{v_j j}$ in J_j if $v_j > 0$, and t_j is 0 if $v_j = 0$ or the completion time of $O_{v_j j}$ if $v_j > 0$. A vertex u presents the right front of a partial schedule at time $t_{\max}$ that involves v_jth operations of jobs J_j with $v_j > 0$ and does not involve jobs J_j with $v_j = 0$. Vertices

$$\begin{array}{l} (\ 0\ ,\ 0\ ,\ \ldots,\ 0\ ;\ 0\ ,0\ ,\ldots,0\ ),\\ (\ m_1\ ,\ m_2\ ,\ \ldots,\ m_k\ ;\ t_1\ ,\ t_2\ ,\ \ldots,\ t_k\ ) \end{array}$$

are the source and a sink, respectively. A pair (u', u) is an arc if and only if $\exists i \in [1, k] : \forall j \neq i :$

$$\begin{array}{llll} (1) & & v_i = v_i' + 1 & \text{and} \\ & & v_j = v_j' & \text{and} \\ & & t_j = t_j' & \text{and} \\ & & t_i \geq t_j & \text{and} \\ (2) & \mu_{v_i i} = \mu_{v_j j} \Rightarrow & t_i \geq t_j + p & \text{and} \\ (3) & v_i > 1 \quad \Rightarrow & t_i \geq t_i' + p & \text{and} \\ (4) & v_i = 1 \quad \Rightarrow & & \end{array}$$

$$\left\{ [\ J_j \lessdot J_i \Rightarrow v_j = m_j\ ] \ \text{ and } \ t_i \geq \max\{r_i, \max_{J_j \lessdot J_i} t_j\} + p \right\}.$$

The propositions indexed above mean that:

(1) the front u is at time t_i and reached from the front u' by adding O_{1i} or changing $O_{v_i-1,i}$ into $O_{v_i i}$ if $v_i > 1$,
(2) $O_{v_i i}$ does not overlap in time with other operations that require the same machine $M_{\mu_{v_i i}}$,
(3) the start time $t_i - p$ of $O_{v_i i}$ is not earlier than completion time of the preceding operation $O_{v_i' i}$,
(4) J_i starts only after all its immediate predecessors are completed and start time $t_i - p$ of J_i is not earlier than release date r_i or completion times of immediate predecessors of J_i.

Define the length of the arc (u', u) to be

$$\ell(u', u) = \begin{cases} f_i(t_i) \text{ if } v_i = m_i, \\ \ \ 0 \quad \text{otherwise.} \end{cases}$$

It is easy to make sure that source-sink paths in the graph are in a one-to-one correspondence with schedules of costs equal to the lengths of the correspondent paths. Hence, the problem reduces to searching a shortest path in the graph. Since the number of vertices is at most $m_{\max}^k \cdot |\mathcal{T}|^k$, where $|\mathcal{T}| = O(k^2 m_{\max})$, and a shortest path can be found in quadratic time, the problem can be solved in polynomial time $O(k^{4k} m_{\max}^{4k})$.

A polynomial-time solution to the no-wait counterpart of the problem and even to $J|\text{no-wait, prec}, r_j, n = k|\sum f_j$ can be obtained by exchanging "$\geq$" into "$=$" in Proposition (3) and replacing "p" by "$p_{v_i-1,i}$" in Propositions (2), (3) and (4). The number of vertices in the transversal graph in this case is at most $m_{\max}^k \cdot |\mathcal{T}_{\text{no-wait}}|^k$, where $\mathcal{T}_{\text{no-wait}}$ is the time space required for saving all active no-wait job-shop schedules.

Let us set $p_{0j} = 0$ for all $j = 1, 2, \ldots, n$. Since no-wait jobs are not interrupted in such schedules,

$$\begin{aligned}\mathcal{T}_{\text{no-wait}} = \{\, t : \exists\ l &\in [1,\ k\ ] :\\ \forall\ j &\in [1,\ k\ ] :\\ \exists\ x_j &\in [0,\ m_j\ ] :\\ \exists\ y_j &\in [0,\ m_j\ ] :\end{aligned}$$

$$\left. x_j \le y_j \ \text{ and } \ t = r_l + \sum_{j=1}^{k} \sum_{i=x_j}^{y_j} p_{ij} \right\}.$$

Thus,

$$|\mathcal{T}_{\text{no-wait}}| \le k \cdot \prod_{j=1}^{k} m_j^2 = O\left(k m_{\max}^{2k}\right).$$

Hence, a shortest path in this case can be found in time $O(k^{2k} m_{\max}^{4k^2+2k})$.

6 $P|\text{chains}, r_j, p_j = 1|\sum f_j$

Dror et al. (1998) establish that $P|\text{chains}, r_j, p_j = 1|L_{\max}$ can be solved in polynomial time. In this note we show that there exists a similar polynomial-time solution to minimizing $\sum f_j$ if the cost functions of chained jobs are interchangeable, i.e.,

$$J_j \lessdot J_k \ \Rightarrow\ f_j(x) + f_k(y) = f_k(x) + f_j(y) \ \text{ for all } \ x \ne y.$$

If chained jobs have identical cost functions, i.e., $J_j \lessdot J_k \Rightarrow f_j = f_k$, then they are obviously interchangeable. Thus, $P|\text{chains}, r_j, p_j = 1|\sum w_j U_j$ and $P|\text{chains}, r_j, p_j = 1|\sum w_j T_j$ can be solved in polynomial time if chained jobs have identical weights and due dates.

Without loss of generality, we assume that release dates are adjusted to precedence constraints by changing r_k to $\max\{r_j + 1, r_k\}$ for each immediate precedence $J_j \lessdot J_k$. Hence, we can assume that release dates are increasing along each chain of precedence constraints.

Let us create a network with a source S, job vertices J_j, $1 \le j \le n$, chain-time vertices (c, t) for each chain c and integer time point $t \in \mathcal{T}$, time vertices $t \in \mathcal{T}$ and a sink T. Since the number of chains is at most n and $|\mathcal{T}| = O(n^2)$, the number of vertices in the network is $O(n^3)$. The arcs in the network will be the arcs in all possible paths

$$S \to J_j \to (c, t) \to t \to T$$

under the condition that arcs $J_j \to (c, t)$ exist if and only if $r_j \leq t$ and $J_j \in c$. We assign capacity m to the arcs $t \to T$ and capacity 1 to the other arcs, cost $f_j(t+1)$ to the arcs $J_j \to (c, t)$ and cost 0 to the other arcs, supply n to S and demand n to T.

It is well known that the quantities of a minimum-cost flow running through the arcs are integer if all arcs in the network have integer capacities. Thus, the unit flow through the arc $J_j \to (c, t)$ with unit capacity models an assignment of the job J_j belonging to the chain c to be processed in the time unit $[t, t+1]$ with $r_j \leq t$. The flow of value at most m through the arc $t \to T$ with capacity m models an assignment of at most m jobs to be processed in the time unit $[t, t+1]$.

The network-flow model observes all precedence constraints with the exception of possibly those between jobs J_i and J_j that start at time $\max\{r_i, r_j\}$ or later. However, transposing such jobs (i.e., putting two jobs in each other's place in the schedule) we can obtain a schedule that observes all precedence constraints in time $O(n^2)$. Since the cost functions of chained jobs are interchangeable, the total cost remains the same after the transposition.

The time complexity of solving the problem is $O(n^9)$ because finding a minimum-cost flow in a network takes at most cubic time in the number of its vertices.

Note that $P|\text{chains}, r_j, p_j = 1|L_{\max}$ reduces to a binary search on $\Delta = D_1 - d_1 = D_2 - d_2 = \ldots = D_n - d_n$ for finding a maximal flow (of quantity n) in the same network, where the arc condition $r_j \leq t$ is replaced by $r_j \leq t < D_j$.

The related open-shop problems of minimizing $L_{\max}$ or $\sum f_j$ with interchangeable cost functions of chained jobs, where $f_j \neq C_j$, can also be solved in polynomial time since $O|\text{prec}, r_j, p_{ij} = 1|\gamma$ reduces to $P|\text{prec}, r_j, p_j = 1|\gamma$ with the same type of precedence constraints for all criteria $\gamma \neq \sum C_j$ (see Timkovsky 2003).

7 $P|\text{pmtn}, p_j = p|\sum f_j$

Brucker et al. (2003) have recently shown that preemptions are redundant in $P|\text{pmtn}, p_j = p|\sum T_j$. But $P|p_j = p|\sum T_j$ can be solved in polynomial time by a simple reduction to the assignment problem (see Brucker and Knust, web page). We show that an extension of the problem to a more general criterion, where preemptions may be advantageous, can also be solved in polynomial time.

Let f_j be convex nondecreasing functions such that they are defined on a time space $\mathcal{T}_{\text{pmtn}}$ polynomially bounded in n, and the differences $f_i - f_j$ are all monotone functions. Without loss of generality, we can assume that the jobs are in a linear order where for each pair of jobs J_i and J_j the functions $f_i - f_j$ are nondecreasing if $i < j$. In this case, as it is shown in Baptiste (2000a), such an order always exists. It can be trivially shown by the exchange argument that then there exists an optimal schedule where $i \leq j \Rightarrow C_i \leq C_j$.

Let us consider completion times C_j for all $j = 1, 2, \ldots, n$ as deadlines. It is known (Horn 1974) that a feasible schedule for the decision problem $P|\text{pmtn}, D_j|$

with deadlines C_j exits if and only if

$$\forall j : \sum_{i=1}^{n} \max\{0, p_i - \max\{0, C_i - C_j\}\} \leq mC_j \text{ and } C_j \geq p_j.$$

Under the conditions $p_j = p$ and $i \leq j \Rightarrow C_i \leq C_j$ this predicate is

$$\forall j : \sum_{i=j+1}^{n} \max\{0, p - C_i + C_j\} \leq mC_j - jp \text{ and } C_j \geq p.$$

Introducing additional variables

$$X_{ij} = \max\{0, p - C_i + C_j\}$$

for all $i = 2, 3 \ldots, n$ and $j = 1, 2, \ldots, n$ we finally obtain the problem of minimizing the *convex-separable* objective function

$$\sum_{j=1}^{n} f_j(C_j)$$

under the linear constraints

$$\begin{aligned} \forall j : \sum_{i=j+1}^{n} X_{ij} &\leq mC_j - jp \qquad \text{and} \quad C_j \geq p, \\ \forall i : \forall j : \qquad X_{ij} &\geq p - C_i + C_j \quad \text{and} \quad X_{ij} \geq 0. \end{aligned}$$

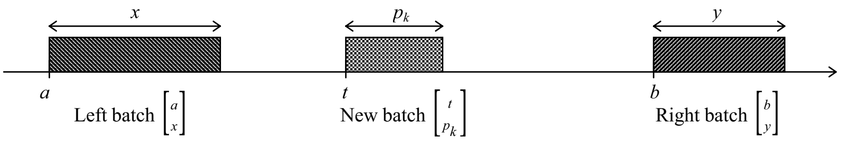
The problem reduces to a linear programming problem in time polynomial in the total number of *break points* of the functions f_j (Dantzig 1998). In our case, the number of break points of each of the n functions is at most $|\mathcal{T}_{\text{pmtn}}|$. Hence, the problem can be solved in time polynomial in $n \cdot |\mathcal{T}_{\text{pmtn}}|$.

Once the optimal completion times are known, the optimal schedule can be produced by an algorithm of Sahni (1979).

8 $1|\text{p-batch}, r_j, p_j = p|\sum f_j$

Let us consider the problem where job lengths can be different (see Brucker et al. 1998). The jobs are to be processed in parallel batches, i.e., jobs in a batch start and complete at the same time, and the length of a batch is the maximum length of its jobs. We assume that batches can contain an unlimited number of jobs.

We first prove that the problem can be solved in pseudopolynomial-time. It is easy to show by contradiction that there exists an optimal schedule where, for any batch, jobs that are shorter than the batch and released not later than the batch starts



x
p_k
y
a
Left batch $\begin{bmatrix} a \\ x \end{bmatrix}$
t
New batch $\begin{bmatrix} t \\ p_k \end{bmatrix}$
b
Right batch $\begin{bmatrix} b \\ y \end{bmatrix}$

Fig. 2. Three batches involved in the calculation of $F'_k[{a \atop x}][{b \atop y}]$

are assigned to the batch or a previous batch. Hence, if start time of a longest batch is known, then the problem can be decomposed into two subproblems that appear before and after completion time of the batch.

Let the jobs be sorted by nondecreasing lengths, i.e., $p_1 \le p_2 \le \dots \le p_n$, and let $[{a \atop x}]$ and $[{b \atop y}]$ denote two batches with start times a and b and lengths x and y, where $a < b$. We call them a left batch and a right batch, respectively.

Define a subproblem for $k = 1, 2, \dots, n$ as finding a minimum-cost schedule of jobs J_j with $j \le k$ and $a < r_j \le b$ that are assigned to the batch $[{b \atop y}]$ or a batch following $[{a \atop x}]$. Let $F_k[{a \atop x}][{b \atop y}]$ denote the minimum cost of the schedule, and let

$$G_k[{a \atop x}][{b \atop y}] = F_{k-1}[{a \atop x}][{b \atop y}] + f_k(b + y).$$

Then F_k can be expressed via G_k and F_{k-1} by the recurrence formula

$$F_k[{a \atop x}][{b \atop y}] = \begin{cases} F'_k[{a \atop x}][{b \atop y}] & \text{if } a < r_k \le b, \\ F_{k-1}[{a \atop x}][{b \atop y}] & \text{otherwise,} \end{cases}$$

where

$$F'_k[{a \atop x}][{b \atop y}] = \min \left\{ G_k[{a \atop x}][{b \atop y}], \min_{\max\{a+x, r_k\} \le t \le b - p_k} \left\{ G_k[{a \atop x}][{t \atop p_k}] + F_{k-1}[{t \atop p_k}][{b \atop y}] \right\} \right\}.$$

The interior minimum models the appearance of a new batch $[{t \atop p_k}]$ capturing J_k between the batches $[{a \atop x}]$ and $[{b \atop y}]$. $G_k[{a \atop x}][{b \atop y}]$ models the appearance of J_k in the batch $[{b \atop y}]$. (Recall that J_k is the longest job in the batch.)

Setting $F_0[{a \atop x}][{b \atop y}] = 0$ for all possible batches $[{a \atop x}]$ and $[{b \atop y}]$ we can calculate the minimum $\sum f_j$ as $F_n[{0 \atop 0}][{r_{\max} + np_{\max} \atop p_{\max}}]$.

If $H = r_{\max} + \sum p_i$ denotes the time horizon of the schedule, then the number of quintets (t, a, b, x, y) is at most H^5 and the recurrence formula should be applied for each $k = 1, 2, \dots, n$, the problem can be solved by dynamic programming in time $O(n \cdot H^5)$. In the case of equal-processing-time jobs, there are only $O(n^2)$ relevant time points and thus, we can solve $1|\text{p-batch}, r_j, p_i = p|\sum f_j$ in time $O(n^{11})$.

9 $Pm|r_j, p_j = p|\sum w_j U_j$

We describe a decomposition scheme that follows the technique used in Baptiste (1999, 2000a) for solving $1|r_j, p_j = p|\sum w_j U_j$, $Pm|r_j, p_j = p|\sum w_j C_j$, $Pm|r_j, p_j = p|\sum T_j$ and a suggestion of Chuzhoy et al. (2001).

Let us define a resource profile to be a vector $a = (a_1, a_2, \dots, a_m)$ with integer components in $\mathcal{T}$ such that $a_{\max} - a_{\min} \le p$, and the components are associated with the machines $M_1, M_2, \dots, M_m$. We assume that $a_{\min}$ is the completion time of a job, $a_i = a_{\min}$ implies that a_i is the completion time of a job or an idle time on M_i, and $a_i \ne a_{\min}$ implies that a_i is the completion time of a job that is in process on M_i at time $a_{\min}$.

Let $\preceq$ and $\prec$ denote the coordinate order and the strict coordinate order on resource profiles, respectively, i.e.,

$$a \preceq b \;\Leftrightarrow\; \forall i \in [1, m]:\; a_i \le b_i,$$
$$a \prec b \;\Leftrightarrow\; a \preceq b \text{ and } a \ne b.$$

Let $t_{\max} = \max_{t \in \mathcal{T}} t$. For a resource profile

$$x \ne (t_{\max}, t_{\max}, \dots, t_{\max})$$

define the set of next resource profiles

$$next(x) = \{x' : \exists h \in [1, m]:$$
$$x_h = x_{\min} \text{ and } x_h + p = x'_h \text{ and } i \ne h \Rightarrow x_i = x'_i\}.$$

In what follows, we assume that jobs are ordered by nondecreasing due dates, i.e., $d_1 \le d_2 \le \dots \le d_n$. We say that a schedule of jobs is feasible on an interval of resource profiles $a \prec b$ if the jobs are scheduled between their release dates and due dates, no more than m jobs are processed simultaneously, and M_i is idle before a_i and after b_i for all $i = 1, 2, \dots, m$.

For a positive integer $k \le n$ and an interval of resource profiles $a \prec b$, let $W_k[a, b]$ denote the maximum weight of early jobs J_j with $j \le k$ and $a_{\max} - p \le r_j < b_{\min}$ in schedules that are feasible on $a \prec b$. Then the desired minimum weighted number of late jobs is

$$\sum_{j=1}^{n} w_j - W_n[(0, 0, \dots, 0), (t_{\max}, t_{\max}, \dots, t_{\max})],$$

and

$$W_k[a, b] = \begin{cases} \max\{W'_k[a, b], W_{k-1}[a, b]\} & \text{if } a_{\max} - p \le r_k < b_{\min}, \\ W_{k-1}[a, b] & \text{otherwise,} \end{cases}$$

where

$$W'_k[a, b] = \max_{\substack{r_k \le x_{\min} \le d_k - p \\ x' \in next(x) \\ a \preceq x \\ x' \preceq b}} \left\{W_{k-1}[a, x] + w_k + W_{k-1}[x', b]\right\}.$$

If $a_{\max} - p > r_k$ or $r_k \geq b_{\min}$, then J_k is not taken into account in $W_k[a, b]$, so $W_k[a, b] = W_{k-1}[a, b]$.

If $a_{\max} - p \leq r_k < b_{\min}$, then we try all possible positions of J_k by scheduling it between the resource profiles x and x' on M_h with $x_h = x_{\min} = x'_h - p$. Since J_k is early, we add weight w_k. As it is shown in Baptiste (2000a), the problem of finding $W_k[a, b]$ is a decomposition of two independent subproblems of finding $W_{k-1}[a, x]$ and $W_{k-1}[x', b]$.

Finally, we add initial conditions $W_0[a, b] = 0$ for all pairs of resource profiles a and b with $a \prec b$. Since the number of resource profiles is at most $|\mathcal{T}|^m$, where $|\mathcal{T}| = O(n^2)$, the number of triplets a, x, b is at most $(|\mathcal{T}|^m)^3 = O(n^{6m})$.

The recurrence formula should be applied for each $k = 1, 2, \dots, n$, therefore, we can calculate all weights $W_k[a, b]$ and related schedules in time $O(n^{6m+1})$ using the dynamic programming approach.

$Om|\text{no-wait}, r_j, p_{ij} = 1|\sum w_j U_j$ can also be solved in polynomial time by a reduction to $Pm|r_j, p_j = m|\sum w_j U_j$ (see Brucker et al. 1993; Kubiak et al. 1991).

10 $Pm|\text{pmtn}, p_j = p|\sum w_j U_j$

An algorithm of Lawler (1979) for $Qm|\text{pmtn}|\sum w_j U_j$ runs in pseudopolynomial time

$$\begin{aligned} &O(n^2[\textstyle\sum w_j]^2) \quad \text{if } m = 2, \\ &O(n^{3m-5}[\textstyle\sum w_j]^2) \quad \text{if } m \geq 3. \end{aligned}$$

Lawler and Martel (1989) proposed an improved algorithm for the case $m = 2$ with the time complexity $O(n^2 \sum w_j)$. Hence, the algorithms become polynomial in the case of equal weights. However, they remain pseudopolynomial for equal processing times. Here we propose another pseudopolynomial algorithm for $Pm|\text{pmtn}|\sum w_j U_j$ that turns to polynomial for equal processing times. Applying the dynamic programming approach to this problem, we use an algorithm of finding a feasible schedule for $P|\text{pmtn}, D_j|$.

The following recurrent procedure we call a *staircase algorithm* is just a variation of an algorithm of Sahni (1979). As we will see, the number of time points at which jobs are completed in a staircase schedule is polynomially bounded for equal processing times. This will ensure the polynomiality of our algorithm.

Let us introduce a dummy job J_0 with processing time $p_0 = 0$ and due date $d_0 = 0$. We assume that J_0 is scheduled at time 0 on each machine and that the jobs $J_0, J_1, \dots, J_n$ are ordered by nondecreasing due dates.

Let after scheduling $J_0, J_1, \dots, J_{j-1}$ the machines become available at times a_i, $i = 1, 2, \dots, m$, such that $a_1 \leq a_2 \leq \dots \leq a_m$. Then define an availability profile to be the vector $a = (a_1, a_2, \dots, a_m)$. Initially $a = (0, 0, \dots, 0)$.

To schedule J_j the algorithm performs a loop on $i = m, m-1, \dots, 1$ and assigns a part of J_j of length $\ell_{ij} = \max\{0, \min\{p_j, d_j - a_i\}\}$ to start at time a_i on

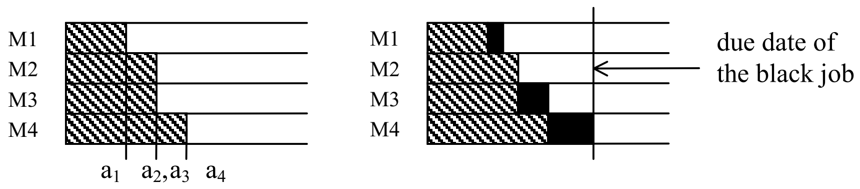


Fig. 3. Changing the availability profile when scheduling J_j (*black job*)

M_i and sets

$$a_i = a_i + \ell_{ij}, \quad d_j = d_j - \ell_{ij}, \quad p_j = p_j - \ell_{ij}.$$

If $p_j = 0$ after the loop, then J_j becomes scheduled. The resulted availability profile we denote as a^j. It is easy to see that components of a^j are nondecreasing (cf. Fig. 3).

If $p_j > 0$ after the loop, then J_j cannot be early, and the algorithm terminates. Using induction on the number of jobs it can be shown that the algorithm terminates if and only if there is no feasible schedule. The proof, closely related to a result of Sahni (1979), can be found in Baptiste (2000b).

Let $W_j(a)$ denote the minimum weighted number of late jobs among J_j, J_{j+1}, $\ldots$, J_n for schedules with the availability profile a. Then

$$W_j(a) = \begin{cases} \min\{W_{j+1}(a) + w_j,\, W_{j+1}(a^j)\} & \text{if the algorithm schedules } J_j, \\ W_{j+1}(a) + w_j & \text{otherwise.} \end{cases}$$

Setting $W_{n+1}(a) = 0$ for all availability profiles a, which number is at most $|\mathcal{T}_{\text{pmtn}}|^m$, we can solve the problem by the dynamic programming approach in time $O(n \cdot |\mathcal{T}_{\text{pmtn}}|^m)$.

To evaluate $|\mathcal{T}_{\text{pmtn}}|$ for the case of equal processing times, let us count how many different availability profiles the staircase algorithm can produce. Obviously, components of availability profiles can be computed as completion times of non-preemptive parts of jobs.

So, let us find the number x_i of all possible completion times on the machine M_i in staircase schedules. Then we have

$$|\mathcal{T}_{\text{pmtn}}| \le x_1 + x_2 + \ldots + x_m.$$

If M_i runs a part of J_j in a staircase schedule, then C_{ij} will denote the completion time of the part.

Obviously, x_m does not exceed the number of generated due dates (the number of original due dates times the number of jobs), i.e., $x_m = O(n^2)$.

Now let $i < m$. Notice that for any job J_j in a staircase schedule there exists integer $v_j \le m$ such that $M_1, M_2, \ldots, M_{v_j-1}$ do not run J_j but $M_{v_j}, M_{v_j+1}, \ldots, M_m$ run J_j in succession (with possible zero processing time on several last machines).

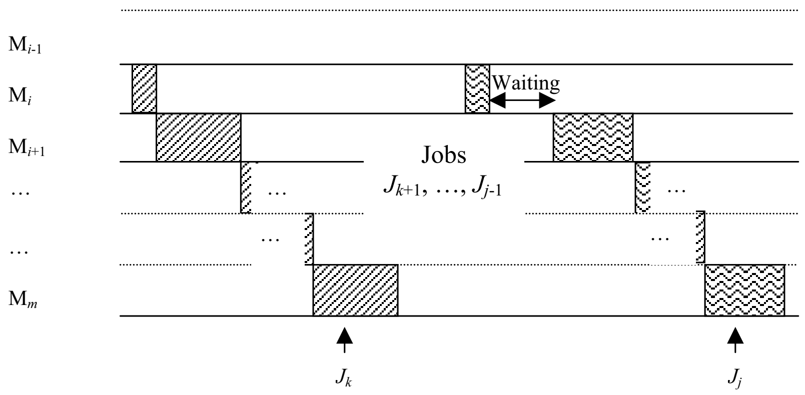


Fig. 4. Scheduling jobs between J_j and J_k

Herein, if J_j passes M_i and M_{i+1} without waiting, then C_{ij} is also the completion time of another job on M_{i+1}. If J_j waits between M_i and M_{i+1}, then $M_1, M_2, \ldots, M_{i-1}$ do not run J_j.

Let C_{ik}, where $k < j$, be the latest completion time earlier than C_{ij} such that $C_{ik} = C_{i+1,k-1}$. Given this definition, the jobs $J_{k+1}, \ldots, J_j$ wait between M_i and M_{i+1} and thus are not run through $M_1, M_2, \ldots, M_{i-1}$ (see Fig. 4).

There are no idle machines in a staircase schedule, therefore the jobs $J_{k+1}, \ldots, J_j$ are processed in the time intervals

$$\underbrace{[C_{ik}, C_{ij}]}_{\text{machine } M_i}, \underbrace{[C_{i+1,k}, C_{i+1,j}]}_{\text{machine } M_{i+1}}, \ldots, \underbrace{[C_{mk}, C_{mj}]}_{\text{machine } M_m}.$$

In other words,

$$C_{ik} = C_{i+1,k-1} \text{ and } \sum_{u=i}^{m}(C_{uj} - C_{uk}) = p(j-k).$$

Extracting C_{ij} with $i < m$ gives

$$C_{ij} = p(j-k) + C_{i+1,k-1} - \sum_{u=i+1}^{m}(C_{uj} - C_{uk}).$$

From this expression we have

$$x_i \le n x_{i+1} \cdot \prod_{u=i+1}^{m} x_u^2.$$

Since $x_{u+1} \le x_u$,

$$\prod_{u=i+1}^{m} x_u^2 \le x_{i+1}^{2m-2i} < x_{i+1}^{2m}$$

and hence

$$x_i < n x_{i+1}^{2m+1}.$$

Unrolling this inequality gives

$$x_1 < n^1 x_2^{(2m+1)^1} < n^2 x_3^{(2m+1)^2} < \ \ldots \ < n^{m-1} x_m^{(2m+1)^{m-1}}.$$

Recall that $x_m = O(n^2)$, therefore $x_1 < O(n^{m-1} n^{2(2m+1)^{m-1}})$. Taking into consideration that $x_1 + x_2 + \cdots + x_m \leq m x_1$, we eventually obtain

$$|\mathcal{T}_{\text{pmtn}}| < O\left(m n^{m-1+2(2m+1)^{m-1}}\right).$$

This implies that the problem can be solved by dynamic programming in polynomial time

$$O\left(m^m n^{1+m^2-m+2m(2m+1)^{m-1}}\right).$$

Concluding remarks

Although we did not find an extension of the technique described in Note 3 to unit-time flow shops, we conjecture that

$$Fm|\text{intree}, p_{ij}=1|\sum C_j$$

can also be solved in polynomial time.

Since the mass reduction of unit-time open-shop problems to preemptive equal-processing-time problems on identical parallel machines does not work for the total completion time criterion only (cf. Note 6),

$$Om|\text{chains}, r_j, p_{ij}=1|\sum C_j$$

remains one of the most intriguing problems with the open complexity status even if $m = 3$. The special case with $m = 2$ and even its extension to outree-like precedence constraints (see Lushchakova 2001) or the special case without precedence constraints (see Tautenhahn and Woeginger 1997) can be solved in polynomail time.

Preemptions in $Pm|\text{pmtn, intree}, p_j = 1|\sum w_j U_j$ prove to be redundant in Brucker et al. (2003). This implies that

$$Pm|\text{pmtn, chains}, p_j=1|\sum U_j$$

is strongly NP-hard because $1|\text{chains}, p_j = 1|\sum U_j$ is strongly NP-hard (see Lenstra and Rinnooy Kan 1980).

In this paper, we also throw some light on open questions in the complexity of scheduling with arbitrary processing times. Notes 10 and 8 present pseudopolynomial-time solutions to

$$Pm|\text{pmtn}|\sum w_jU_j, \quad 1|\text{p-batch}, r_j|\sum w_jU_j \text{ and } 1|\text{p-batch}, r_j|\sum w_jT_j.$$

This means that the NP-hard problems (see Karp 1972; Brucker et al. 1998)

$$1|\text{pmtn}|\sum w_jU_j, \quad 1|\text{p-batch}|\sum w_jU_j \text{ and } 1|\text{p-batch}|\sum w_jT_j,$$

which have been open for the strong NP-hardness, are not strongly NP-hard indeed (unless P=NP).

On the other hand, although it is known that

$$P|\text{pmtn}|\sum U_j \text{ and } 1||\sum w_jU_j$$

are NP-hard (Lawler 1983; Karp 1972), it is still open whether

$$P|\text{pmtn}|\sum U_j \text{ or } Pm|r_j|\sum w_jU_j$$

can be solved in pseudopolynomial time. We recall in conclusion that the complexity statuses of

$$P2|\text{pmtn}|\sum T_j, \quad Pm|\text{pmtn}|\sum T_j \text{ and } P|\text{pmtn}|\sum T_j$$

remain open.

Acknowledgements. We would like to thank the anonymous referees for their careful reading of the original version of the paper and meaningful comments that helped us to see our omissions.

References

Baptiste Ph (1999) Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine when processing times are equal. *Journal of Scheduling* 2: 245–252

Baptiste Ph (2000a) Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics* 103: 21–32

Baptiste Ph (2000b) Preemptive scheduling of identical machines. Technical report, Université de Technologie de Compiègne, Compiègne

Brucker P (1998) *Scheduling algorithms*. Springer, Berlin Heidelberg New York

Brucker P, Gladky A, Hoogeveen H, Kovalyov M, Potts C, Tautenhahn T, van de Velde S (1998) Scheduling a batching machine. *Journal of Scheduling* 1: 31–54

Brucker P, Heitmann S, Hurink J (2003) How useful are preemptive schedules? *Operations Reserach Letters* 31: 129–136

Brucker P, Jurisch B, Jurisch M (1993) Open shop problems with unit time operations. *Zeitschrift für Operations Research* 37: 59–73

Brucker P, Knust S (web page) *Complexity results for scheduling problems*. http://www.mathematik.uni-osnabrueck.de/research/OR/class/

Chuzhoy J, Ostrovsky R, Rabani Y (2001) Approximation algorithms for the job interval selection problem and related scheduling problems. Proc. 42nd annual symposium on foundations of computer science. Las Vegas, Nevada

Dantzig GB (1998) *Linear programming and extensions*, 11th ed. Princeton University Press, Princeton, NJ

Dror M, Kubiak W, Dell'Olmo P (1998) 'Strong'-'weak' chain constrained scheduling. *Ricerca Operativa* 27: 35–49

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5: 287–326

Horn W (1974) Some simple scheduling problems. *Naval Research Logistics Quarterly* 21: 177–185

Hu TC (1961) Parallel sequencing and assembly line problems. *Operations Research* 9: 841–848

Karp RM (1972) Reducibility among combinatorial problems. In: *Complexity of Computer Computations*. Plenum Press, New York, pp 85–103

Kubiak W, Sriskandarajah C, Zaras K (1991) A note on the complexity of openshop scheduling problems. *INFOR* 29: 284–294

Labetoulle J, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1984) Preemptive scheduling of uniform machines subject to release dates. In: *Progress in combinatorial optimization*. Academic Press, New York, pp 245–261

Lawler EL (1979) Preemptive scheduling of uniform parallel machines to minimize the weighted number of late jobs. Report BW 106, Centre for Mathematics and Computer Science, Amsterdam

Lawler EL (1983) Recent results in the theory of machine scheduling. In: *Mathematical programming: the state of the art*. Springer, Berlin Heidelberg New York, pp 202–234

Lawler EL, Martel CU (1989) Preemptive scheduling of two uniform machines to minimize the number of late jobs. *Operations Research* 37: 314–318

Lenstra JK, Rinnooy Kan AHG (1980) Complexity results for scheduling chains on a single machine. *European Journal of Operational Research* 4: 270–275

Lushchakova I (2001) Two machine preemptive scheduling problem with release dates, equal processing times and precedence constraints. Technical report, Belorussian State University, Minsk

Middendorf M, Timkovsky VG (1999) Transversal graphs for partially ordered sets: sequencing, merging and scheduling problems. *Journal of Combinatorial Optimization* 3: 417–435

Sahni S (1979) Preemptive scheduling with due dates. *Operations Research* 27: 925–934

Simons B (1983) Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal on Computing* 12: 294–299

Tautenhahn T, Woeginger GJ (1997) Minimizing the total completion time in a unit-time open shop with release dates. *Operations Research Letters* 20: 207–212

Timkovsky VG (2003) Identical parallel machines vs. unit-time shops and preemptions vs. chains in scheduling complexity. *European Journal of Operational Research* 149: 355–376