ELSEVIER

Int. J. Production Economics 46–47 (1996) 119–125

international journal of
production
economics

Parallel machine scheduling with release dates, due dates and family setup times

J.M.J. Schutten*, R.A.M. Leussink

Department of Mechanical Engineering, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

In manufacturing, there is a fundamental conflict between efficient production and delivery performance. Maximizing machine utilization by batching similar jobs may lead to poor delivery performance. Minimizing customers' dissatisfaction may lead to an inefficient use of the machines. In this paper, we consider the problem of scheduling n independent jobs with release dates, due dates, and family setup times on m parallel machines. The objective is to minimize the maximum lateness of any job. We present a branch-and-bound algorithm to solve this problem. This algorithm exploits the fact that an optimal schedule is contained in a specific subset of all feasible schedules. For lower bounding purposes, we see setup times as setup jobs with release dates, due dates and processing times. We present two lower bounds for the problem with setup jobs, one of which proceeds by allowing preemption.

Keywords: Scheduling; Parallel machines; Maximum lateness; Family setup times; Branch-and-bound

1. Introduction

In the last two decades, quality and timely delivery, next to efficiency, have become key performance indicators for manufacturing organizations (cf. [1, 2]). The term flexibility has been introduced to describe the ability to manufacture a large variety of products efficiently. On the one hand, production planners attempt to cluster jobs with similar setup characteristics for an efficient use of the machines. On the other hand, batching leads to the delay of other jobs, and as a consequence, these jobs will be completed after their delivery date. So,a trade-off must be found between an efficient use of the machines and a good due date performance.

In this paper, we consider the problem of scheduling a set $\mathcal{J}$ of n independent jobs $J, \ldots, J_n$ on m identical parallel machines $M_1, \ldots, M_m$. Each job J_j ($j = 1, \ldots, n$) must be processed on one of the machines during a given uninterrupted positive time p_j. It becomes available at its release date r_j and should be completed by its due date d_j. Each machine M_i is available from time 0 onwards and can process no more than one job at a time. Each job belongs exactly to one of the families $\mathcal{F}_1, \ldots, \mathcal{F}_F$. The index of the family that J_j belongs to is denoted by $f(j)$. If we schedule two jobs that belong to different families contiguously on the same machine, then we need a positive setup s_i in between

* Corresponding author.

SSDI 0925-5273(95)00086-0

that is completely specified by the family $\mathcal{F}_i$ the second job belongs to. We also need a setup for the first job on each machine. No setup is needed when jobs of the same family are scheduled contiguously on the same machine. This kind of setup is called *sequence independent*. No processing of jobs is possible during a setup. A machine may be set up for a particular job prior to its release date. Without loss of generality we assume that all data are integral. A schedule is called *feasible* if all conditions mentioned above are satisfied. A feasible schedule specifies for each job J_j a completion time $C_j(\sigma)$. For a given schedule, we compute the lateness of a job J_j as $L_j(\sigma) = C_j(\sigma) - d_j$. If $L_j(\sigma) \leqslant 0$, then we say that J_j is *early*; otherwise, it is *tardy*. The maximum lateness of σ is defined as $L_{max}(\sigma) = \max_{1 \leqslant j \leqslant n} L_j(\sigma)$. The problem is to find a schedule σ that has the smallest maximum lateness L^*_{max} among all feasible schedules. This problem is clearly NP-hard, even in case of one machine and no setup times [3] and in case of equal release dates [4]. In the remainder, we follow the three-field notation proposed by Graham et al. [5] to classify machine scheduling problems; our problem is accordingly denoted as $P|r_j, s_i|L_{max}$.

The presence of release dates is consistent with many MRP-controlled environments. Also, the problem $P|r_j, s_i|L_{max}$ appears as a subproblem in decomposition approaches such as (generalizations of) the shifting bottleneck approach of Adams et al. [6]. The extension of this approach to hybrid job shops, including setup times, parallel machines, machines with additional resource constraints (e.g., operators, tools) is a research project at the University of Twente. For a more elaborate discussion of this project, we refer to Meester and Zijm [7].

The plan of this paper is as follows. In Section 2, we describe a branch-and-bound algorithm for the problem $P|r_j, s_i|L_{max}$. This algorithm is based on an effective algorithm for the single-machine case developed by Schutten et al. [8]. Section 3 reports some implementation aspects and our computational experiments. In Section 4 we draw some conclusions and point out future research directions.

2. The branch-and-bound algorithm

2.1. Characterization of an optimal schedule

First, note that we may restrict ourselves to *active* schedules. A schedule is active if no job can start earlier without interfering with the execution of any other job. Woerlee [9] shows that the number of active schedules of n jobs on m machines is equal to

$$n! \cdot \sum_{i=1}^{m} \frac{\binom{n-1}{i-1}}{i!}.$$

We prove that an optimal schedule is contained in a class of only $n!$ schedules. Let $\pi \neq \emptyset$ be a *permutation* of a subset of $\mathcal{J} = \{J_1, \ldots, J_n\}$. We transform this permutation into a feasible schedule for m machines by subsequently assigning the jobs of π to some machine, as follows: the next job of π is scheduled on the machine on which it is completed first. The schedule that results from π is denoted by $g(\pi)$. Note that $g(\pi)$ is a *partial* schedule if π does not contain every job of $\mathcal{J}$.

Theorem 1. *There is a permutation π of $\mathcal{J}$ for which $L_{max}(g(\pi))$ is equal to the optimal maximum lateness L^*_{max}.*

Before we prove this theorem, we introduce some additional definitions. First, we say that a partial schedule σ' *deviates* from a complete schedule σ if the jth job on machine M_i in σ' is not the jth job on machine M_i in σ, for some i and j. Let Π be the set of all permutations of all subsets of $\mathcal{J}$. Define for any schedule σ,

$$\tau(\sigma) := \max_{\pi \in \Pi} \{|\pi| \,|\, g(\pi) \text{ does not deviate from } \sigma\},$$

with $|\pi|$ the number of elements in π. Finally, let Ω be the collection of all optimal schedules.

Proof. We will prove the theorem by contradiction. Let σ^* be an optimal schedule for which

$$\tau(\sigma^*) = \max_{\sigma \in \Omega} \tau(\sigma).$$

Let π^* be such that $|\pi^*| = \tau(\sigma^*)$ and π^* does not deviate from σ^*. If the theorem does not hold, then

$\tau(\sigma^*) < n$, because otherwise $g(\pi^*)$ would be equal to σ^*. We will construct a $\pi' = \pi^* J_j$, such that:

(i) $J_j \in \mathscr{J} \setminus \pi^*$, and
(ii) $g(\pi')$ does not deviate from an optimal schedule σ'.

Such a construction clearly contradicts the maximality of $\tau(\sigma^*)$.

Construct a directed graph $D = (V, A)$ as follows. For each machine M_i $(i = 1, \ldots, m)$, D has a node $v_i \in V$. Suppose that the number of jobs on machine M_i in σ^* exceeds the number of jobs on M_i in $g(\pi^*)$. Let $J_{[i]}$ be the first job on M_i in σ^* that is not in $g(\pi^*)$ and R_i the sequence of jobs on M_i that are not in $g(\pi^*)$. Suppose $J_{[i]}$ is scheduled on machine M_q in $g(\pi^* J_{[i]})$. Then we know that $q \neq i$. Draw an arc in D from v_i to v_q. If the number of jobs on machine M_i in σ^* does not exceed the number of jobs on M_i in $g(\pi^*)$, then v_i has no outgoing arc. We can distinguish two cases:

1. There is a $v_j \in V$ with an incoming arc and no outgoing arc. Suppose $(v_i, v_j) \in A$. Then we know that in σ^* and in $g(\pi^*)$ exactly the same jobs are scheduled on M_j, because otherwise v_j would have had an outgoing arc. Also,

$$C_{[i]}(g(\pi^* J_{[i]})) \leqslant C_{[i]}(\sigma^*),$$

due to the way $J_{[i]}$ is assigned to a machine in $g(\pi^* J_{[i]})$. Reschedule the sequence R_i on M_j instead of on M_i. The resulting schedule σ' is also optimal and $g(\pi^* J_{[i]})$ does not deviate from σ'. This is a contradiction with the maximality of $\tau(\sigma^*)$.

2. Each $v \in V$ has either an outgoing arc or neither an incoming nor an outgoing arc. D must then contain a directed cycle K. Without loss of generality we assume that $K = v_1 v_2 \cdots v_p v_1$. Change σ^* to σ' by moving the sequences $R_1, \ldots, R_p$ as follows. Move sequence R_1 to machine M_2, R_2 to M_3, ..., R_{p-1} to M_p and R_p to M_1. With the same arguments as in case case 1, we conclude that σ' is optimal. Since $g(\pi^* J_{[1]})$ does not deviate from σ', this is, again, a contradiction with the maximality of $\tau(\sigma^*)$. □

This type of proof is valid for a broad class of parallel-machine problems, including $P|r_j, s_{ij}|\sum T_j$ and $P|r_j|C_{max}$.

2.2. The search tree

We adopt a *forward branching rule*: each node at level k of the search tree corresponds to an active permutation π consisting of k jobs $(k = 0, \ldots, n)$. A node at level k has $n - k$ descendant nodes: one for each unscheduled job. We employ an *active node search*: we branch only from one node at a time, thereby adding some unscheduled job J_j to π, which leads to the sequence πJ_j. We branch from the nodes in order of non-decreasing release date. We backtrack at level n, or if we can discard the active node.

2.3. Upper bounds

We used several constructive heuristics to generate upper bounds on L^*_{max}. Some of these heuristics are based on dispatching rules and some are based on cheapest insertion. Test results for these algorithms can be found in Section 3.2.

2.4. Lower bounds

An important issue in the computation of lower bounds on L^*_{max} is how to take into account the necessary setups. Schutten et al. [8] observe that setups can be seen as *setup jobs* with specific release and due dates and processing times. Also sufficient conditions are given that ensure when a setup job may be introduced. In our application, we also use the concept of setup jobs in the computation of lower bounds. We present lower bounds on instances of $P|r_j|L_{max}$. These instances contain all jobs of the original problem as well as derived setup jobs. The lower bounds will then also be valid for the problem $P|r_j, s_i|L_{max}$. Two kinds of lower bounds have been used: one in which we allow preemption, i.e., a job may be interrupted and resumed later on; the second lower bound is based on a lower bound for $P|r_j|L_{max}$ given by Carlier [10].

2.4.1. The preemptive lower bound

The problem $1|r_j, pmtn|L_{max}$ is easy to solve by Horn's rule [11]. This rule schedules at each moment in time the job with the smallest due date

Table 1
Data for a counter example

J_j	r_j	p_j	d_j
J_1	0	10	10
J_2	3	10	35
J_3	4	30	40
J_4	0	15	15

among all released jobs. This rule, however, does not apply to $P|r_j, pmtn|L_{max}$. To see this, consider the example from Table 1 with four jobs and two machines.

Horn's rule gives the schedule J_1J_2 on machine M_1 and J_4J_3 on machine M_2. The maximum lateness of this schedule is 5. The optimal schedule, however, is J_1J_3 on machine M_1 and J_4J_2 on machine M_2 with maximum lateness 0.

Suppose that we want to compute the preemptive lower bound in a node of the branch-and-bound tree. We are interested only whether this lower bound is at least *ub*, where *ub* is an upper bound on L^*_{max}. Suppose π is the permutation of a subset of $\mathscr{J}$ that is associated with this node. Let c_i be the completion time of the last job on machine M_i in $g(\pi)$. Without loss of generality, we assume that $c_1 \leqslant c_2 \leqslant \ldots \leqslant c_m$. Each job $J_j \in \mathscr{J}\backslash\pi$ must be completed before $\bar{d}_j := d_j + ub - 1$; otherwise, the lateness of J_j is at least *ub*, and π does not lead to an improvement of *ub*. Let T be the collection of points in time $\{r_j | J_j \in \mathscr{J}\backslash\pi\} \bigcup \{\bar{d}_j | J_j \in \mathscr{J}\backslash\pi\} \bigcup \{c_1, \ldots, c_m\}$. Define $q := 2\cdot(n - |\pi|) + m$; q is the maximum number of elements in T. Let t_i $(1 \leqslant i \leqslant q)$ be such that $t_i \in T$, $\bigcup_{i=1}^{q} t_i = T$ and $t_1 \leqslant t_2 \cdots \leqslant t_q$. We may assume that no job is interrupted at any moment in time, except, possibly, at $t_i \in T$.

Construct a directed graph $D = (V, A)$ with:

$$V = \{s_1\} \bigcup \{v_j | J_j \in \mathscr{J}\backslash\pi\} \bigcup \{w_1, w_2, \ldots, w_{q-1}\} \bigcup \{s_2\},$$

where s_1 is the source and s_2 the sink; v_j is a node associated with the unscheduled job J_j; $w_i \in V$ is associated with the interval $[t_i, t_{i+1}]$. D has the following arcs:

- (s_1, v_j) with capacity p_j;
- (v_j, w_i) if $r_j \leqslant t_i$ and $\bar{d}_j \geqslant t_{i+1}$ with capacity $t_{i+1} - t_i$;
- (w_i, s_2) with capacity $\max\{k | 1 \leqslant k \leqslant m,\ c_k \leqslant t_i\} \cdot (t_{i+1} - t_i)$.

The number $\max\{k | 1 \leqslant k \leqslant m,\ c_k \leqslant t_i\}$ is the number of machines that are available in the interval $[t_i, t_{i+1}]$ for processing jobs from $\mathscr{J}\backslash\pi$.

In D, there exists a flow of value $\sum_{J_j \in \mathscr{J}\backslash\pi} p_j$ if and only if a preemptive schedule for the unscheduled jobs exists with maximum lateness smaller than *ub* (see, e.g., [12]). So the problem of determining whether the maximum lateness in an optimal preemptive schedule is smaller than *ub* is equivalent to finding a maximum flow in the constructed graph D. Therefore, the preemptive lower bound can be computed in $O(n^3)$ time.

2.4.2. Carlier's lower bound

Carlier [10] gives a lower bound on the optimal makespan for the parallel machine scheduling problem in which the jobs have heads and tails. Based on this lower bound, one can derive a lower bound for the problem $P|r_j|L_{max}$.

Let $\mathscr{A} \subseteq \mathscr{J}\backslash\pi$. $w := \min(m, |\mathscr{A}|)$. Suppose $J, \ldots, J_{i_w}$ are the jobs in $\mathscr{A}$ with the w smallest release dates and $J_{j_1}, \ldots, J_{j_w}$ the jobs in $\mathscr{A}$ with the w largest due dates. Then,

$$G(\mathscr{A}) := \frac{(r_{i_1} + \cdots + r_{i_w}) + \sum_{J_i \in \mathscr{A}} p_j - (d_{j_1} + \cdots + d_{j_w})}{w}$$

is a lower bound on L^*_{max}, for any $\mathscr{A} \subseteq \mathscr{J}$. We refer to this lower bound as Carlier's lower bound.

We choose to compute Carlier's lower bound for subsets $\mathscr{B}_k$ and $\mathscr{D}_k$ of $\mathscr{J}$, with $\mathscr{B}_k$ containing jobs with the k largest release dates, and $\mathscr{D}_k$ containing jobs with the k smallest due dates $(1 \leqslant k \leqslant |\mathscr{J}\backslash\pi|)$.

Of course, $\max_{J \in \mathscr{J}\backslash\pi}\{r_j + p_j - d_j\}$ is also a lower bound on the maximum lateness of jobs in $\mathscr{J}\backslash\pi$.

2.5. Dominance rules

Suppose π_1 and π_2 are any two permutations of subsets of $\mathscr{J}$. If we know some way or another that π_1 never leads to a better solution than π_2, then we say that π_1 is *dominated* by π_2. There are some easy-to-check rules to establish that

π is dominated by another. These rules are called dominance rules.

The dominance rules presented in Schutten et al. [8] for $1|r_j, s_i|L_{max}$ are valid for the individual machines in the problem $P|r_j, s_i|L_{max}$. The most important dominance rule states that whenever a release date of a job J_j is "too large", then πJ_j is dominated by πJ_k for some $J_k \in \mathcal{J} \setminus \pi$. "Too large" can be expressed more formally by $r_j \geq C_k(g(\pi J_k)) + s(f(k), f(j))$, with

$$s(f(k), f(j)) = \begin{cases} s_{f(j)} & \text{if } f(j) \neq f(k) \\ 0 & \text{otherwise} \end{cases}$$

If this dominance rule holds for J_j and J_k, then there is idle time before the processing of J_j in $g(\pi J_j)$. This idle time can be used for processing J_k without interfering with the processing of J_j.

Now, we give a dominance rule that is specific for parallel machine problems. Suppose that π_1 and π_2 are different permutations of the same subset of $\mathcal{J}$. If $g(\pi_1) = g(\pi_2)$, then π_1 is dominated by π_2, and vice versa. So, we may discard one of them. We can sharpen this rule slightly by saying that two schedules are equal (possibly after renumbering the machines) if they have the same maximum lateness and each machine has the same completion time and ends with a job from the same family in both schedules.

3. Implementation and computational experiments

3.1. Implementation

In Section 2.4, we presented two kinds of lower bounds: one based on a max-flow algorithm for the preemptive case and one based on Carlier's [10] lower bound. In terms of quality, the preemptive lower bound dominates Carlier's lower bound. In terms of speed, however, it is the other way around: Carlier's lower bound takes $O(n \log n)$ time and the preemptive lower bound $O(n^3)$ time. We tested three versions of our branch-and-bound algorithm: one that uses only the preemptive lower bound, one that uses only Carlier's lower bound, and one that firstly computes Carlier's lower bound and then, if necessary, the preemptive lower bound.

3.2. Computational experiments

The performance of the branch-and-bound algorithm was evaluated for instances with up to 25 jobs and 2 or 3 machines. All parameters were randomly generated from discrete uniform distributions, except for the release dates that come from a Poisson distribution. The processing times were drawn from the interval [1, 100], the number of families F from the interval $[2, \lfloor n/5 \rfloor]$, and the family indices of the jobs from $[1, F]$. Let $\bar{p}$ denote the average processing time for the jobs. In addition to n and m, there are three input parameters:

- k, defining the mean interarrival time $k \cdot \bar{p}/m$,
- t, defining the interval $[r_j + p_j, r_j + p_j + t \cdot \bar{p}]$ from which the due date of job J_j is drawn,
- s, defining the interval $[1, s \cdot \bar{p}]$ from which the setup times are drawn.

We generated instances for $n = 10, 15, 20, 25$, $m = 2, 3$, $k = 1, 1.2, 1.4$, $t = 1, 3$ and $s = 0.2, 0.6, 1$. For each combination of n, m, k, t and s, we generated 15 instances. Table 2 gives a summary of our computational results for varying values of n and k.

The parameter k more or less determines the workload on the machines: the smaller k, the higher the workload. The column '# heur' gives the number of times out of 180 that one of the heuristics found an optimal solution. The column '# Car' gives the number of times the algorithm found an optimal solution within one minute on a HP 9000/710 workstation using only Carlier's lower bound. The column '%Car' gives this number as a percentage of the 180 instances. The columns '# pmtn', '%pmtn', '# both' and '%both' give the same data, but now for the algorithm that uses only the preemptive lower bound, and for the algorithm with both lower bounds. We see that the algorithm using only Carlier's lower bound gives the best results: it solves almost all instances with up to 15 jobs. Also, the instances with a workload less than 100% ($k = 1$) are solved quite often for problems with up to 20 jobs. The reason for this is that although more nodes are searched, the time per node is much smaller in this algorithm, compared with the algorithms that use the preemptive lower bound. This can also be concluded from Table 3.

The column 'n-Car' gives the mean of the number of nodes searched in the algorithm with only

Table 2
Test results for varying n and k

n	k	#heur	#car	%car	#pmtn	%pmtn	#both	%both
10	1.0	30	180	100	180	100	180	100
10	1.2	36	180	100	180	100	180	100
10	1.4	47	180	100	180	100	180	100
15	1.0	15	175	97.2	167	92.8	169	93.9
15	1.2	23	179	99.4	174	96.7	178	98.9
15	1.4	40	180	100	180	100	180	100
20	1.0	8	134	74.4	111	61.7	114	63.3
20	1.2	17	166	92.2	147	81.9	153	85.0
20	1.4	28	177	98.3	167	92.8	171	95.0
25	1.0	10	95	52.8	71	39.4	79	43.9
25	1.2	8	128	71.1	113	62.8	115	63.9
25	1.4	16	154	85.6	132	73.3	139	77.2

Table 3
Mean number of nodes investigated and mean computation time

n	k	n-Car	s-Car	n-pmtn	s-pmtn	n-both	s-both
10	1.0	811	0.2	484	0.5	484	0.3
10	1.2	481	0.1	343	0.3	343	0.2
10	1.4	242	0.0	182	0.1	182	0.1
15	1.0	14,936	2.4	3,695	5.1	4,377	3.9
15	1.2	7,314	1.0	2,789	3.0	3,674	2.9
15	1.4	3,615	0.6	2,181	2.5	2,181	1.8
20	1.0	40,822	6.1	6,272	10.0	7,610	7.6
20	1.2	36,054	5.2	6,712	7.7	8,831	7.0
20	1.4	15,685	2.2	4,633	4.5	5,642	4.3
25	1.0	52,642	6.6	6,323	7.7	11,053	9.2
25	1.2	37,391	4.4	6,464	7.6	7,401	5.9
25	1.4	34,786	4.3	6,287	6.6	8,844	7.0

Carlier's lower bound and the column 's-Car' gives the mean computation time in seconds for this algorithm. The means are taken over those instances that are solved within one minute. The columns 'n-pmtn', 's-pmtn', 'n-both' and 's-both' give the corresponding data for the algorithm with only the preemptive lower bound and the algorithm with both lower bounds.

4. Conclusions

We have presented an optimization algorithm for the problem $P|r_j, s_i|L_{max}$. In the single-machine case, a dominance rule, concerning the release date of a job being too large, was important. This dominance rule does not work as well in this problem, because the mean interarrival time in this problem

is, of course, smaller. We saw that Carlier's lower bound was more effective than the preemptive lower bound. Future research can be done on algorithms that yield possibly better upper bounds, such as algorithms based on tabu search and on simulated annealing.

References

[1] Deming, W.E., 1982. Quality, Productivity, and Competitive Position. MIT Center for Advanced Engineering Study, Cambridge, MA.

[2] Blackburn, J.D., 1991. Time Based Competition, The Next Battle Ground in American Manufacturing. Richard D. Irwin, Homewood, IL

[3] Garey, M.R. and Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco.

[4] Bruno, J. and Downey, P., 1978. Complexity of task sequencing with deadlines, set-up times and changeover costs. SIAM J. Comput., 7: 393–404.

[5] Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan. A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Ann. Discrete Math. 5: 287–326.

[6] Adams, J., Balas, E. and Zawack, D., 1988. The shifiting bottleneck procedure for job shop scheduling. Management Science, 34: 391–401.

[7] Meester, G.J. and Zijm, W.H.M., 1994. Development of a shop floor control system for hybrid job shops. Technical report LPOM-94-1, Department of Mechanical Engineering, University of Twente, Enschede.

[8] Schutten, J.M.J, van de Velde, S.L. and Zijm, W.H.M., 1993. Single-machine scheduling with release dates, due dates and family setup times. Management Science, Fothcoming.

[9] Woerlee, A.P., 1991. Decision Support Systems for Production Scheduling. PhD thesis, Erasmus University, Rotterdam.

[10] Carlier, J., 1987. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. Eur. J. Oper. Res., 29: 298–306.

[11] Horn, W.A., 1974. Some simple scheduling algorithms. Naval Res. Logist. Quar., 21: 177–185.

[12] Labetoulle, J., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.G.H., 1984. Preemptive scheduling of uniform machines subject to release dates, in Progress in Combinatorial Optimization. Academic Press, Canada, pp. 245–261.