Scheduling identical jobs on uniform parallel machines

M.I. Dessouky
Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign
1206 West Green Street, Urbana, Il 61801, U.S.A.

B.J. Lageweg
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

J.K. Lenstra
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Centre for Mathematics and Computer Science
P.O. Box 4070, 1009 AB Amsterdam, The Netherlands

S.L. van de Velde
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

We address the problem of scheduling n identical jobs on m uniform parallel machines to optimize scheduling criteria that are nondecreasing in the job completion times. It is well known that this can be formulated as a linear assignment problem, and subsequently solved in $O(n^3)$ time. We give a more concise formulation for minsum criteria, and show that general minmax criteria can be minimized in $O(n^2)$ time. We present faster algorithms, requiring only $O(n+m\log m)$ time for minimizing makespan and total completion time, $O(n\log n)$ time for minimizing total weighted completion time, maximum lateness, total tardiness and the weighted number of tardy jobs, and $O(n\log^2 n)$ time for maximum weighted tardiness. In the case of release dates, we propose an $O(n\log n)$ algorithm for minimizing makespan, and an $O(mn^{2m+1})$ time dynamic programming algorithm for minimizing total completion time.

Key Words & Phrases: parallel machine scheduling, uniform machines, identical jobs, matching, dynamic programming.

1. Introduction

The case of identical jobs within a batch is common in manufacturing systems, where the products (corresponding to jobs) have identical designs or processing requirements. While all units of the product require equal processing times on the same machine, individual products may be subject to different constraints. For example, jobs may be required to meet unequal due dates requested by customers, or they may be restricted by unequal release dates as a result of being released at different times from preceding operations.

The scheduling problem arising from such a situation can be described as follows. A set of independent jobs J_j ($j=1,\ldots,n$) have to be scheduled on a set of parallel machines M_i ($i=1,\ldots,m$). Each job J_j ($j=1,\ldots,n$) has one unit of uninterrupted processing requirement and has a cost function f_j, where $f_j(t)$ denotes the cost incurred if it is completed at time t. In addition, each job J_j may have a release date r_j, a due date d_j, and a weight w_j. Each machine M_i ($i=1,\ldots,m$) can process at most one job at a time, and does so at a speed s_i, giving rise to a processing time of $1/s_i$. In such a situation the machines are called *uniform.*

A *schedule* is an assignment of each job to exactly one machine and a specification of the completion time C_j of each job J_j. The objective is to minimize the scheduling cost, measured either by $f_{\max}=\max_{1\leqslant j\leqslant n} f_j(C_j)$ or by $\Sigma f_j=\Sigma_{j=1}^{n} f_j(C_j)$.

In the classification scheme of deterministic machine scheduling problems used by GRAHAM, LAWLER, LENSTRA, and RINNOOY KAN (1979), these problems are denoted by $Q|p_j=1|f_{\max}$ and $Q|p_j=1|\Sigma f_j$, respectively. In this notation, the first field specifies the machine environment; Q denotes the situation with uniform parallel machines, and 1 refers to the special case of a single machine. The second field contains the job characteristics; $p_j=1$ indicates that we have unit processing requirements, and we may also include a parameter r_j to indicate that each job has its own release date. The third field defines the objective function; this may depend on given due dates and weights of the jobs.

For the case of equal release dates, LAWLER, LENSTRA, and RINNOOY KAN (1982) point out that both problems can be formulated in $O(mn^2)$ time as linear assignment problems and solved accordingly in $O(n^3)$ time. In this paper, we derive a property that allows a more compact formulation, requiring only $O(n^2)$ time and space. As an immediate consequence, $Q|p_j=1|f_{\max}$ is solvable in $O(n^2)$ time. We give more efficient algorithms for minimizing maximum completion time (makespan), total weighted completion time, maximum lateness, total tardiness, maximum weighted tardiness, and the weighted number of tardy jobs.

In addition, we consider two problems with release dates. We give an $O(n\log n)$ time algorithm for minimizing makespan and an $O(mn^{2m+1})$ time dynamic programming algorithm for minimizing total completion time, which is polynomial for any fixed number of machines.

2. FUNDAMENTALS

Problem 1: Minimize maximum completion time $C_{\max}$

Given n independent identical jobs and m uniform parallel machines, find a schedule which minimizes the maximum job completion time, $C_{\max}=\max_{1\leqslant j\leqslant n} C_j$.

If the decision variable x_i denotes the number of jobs that is to be assigned to machine M_i, for $i=1,\ldots,m$, then the problem is to minimize

$$C_{\max} \tag{P}$$

subject to

$$x_i/s_i \leqslant C_{\max}, \qquad i=1,\ldots,m, \tag{1}$$

$$\sum_{i=1}^{m} x_i = n, \tag{2}$$

$$x_i \in \mathbb{Z}^+, \qquad i=1,\ldots,m. \tag{3}$$

Given a feasible solution to this problem, the value of $C_{\max}$ can be reduced only if for each M_i with the largest completion time, that is, with $x_i/s_i = C_{\max}$ in (1), there is another machine M_h for which $(x_h+1)/s_h < C_{\max}$. Accordingly, a sufficient condition for the optimality of a schedule is that for any two machines M_h and M_i with $x_h/s_h < x_i/s_i$, we have $(x_h+1)/s_h \geqslant x_i/s_i$.

The following procedure, which requires $O(n\log m)$ time, takes advantage of this sufficient condition. It keeps a priority queue of the m current machine completion times. (A priority queue is a data structure for an ordered set of elements; the time to insert or delete one element is proportional to the logarithm of the number of elements. See AHO, HOPCROFT, and ULLMAN (1982).) Each successive job is matched with the earliest completion time in the queue, and this time is replaced in the queue by the new completion time of the machine in question. The matching and updating of the queue is repeated until all n jobs have been scheduled. Since the queue can be initialized in $O(m\log m)$ time and updated in $O(\log m)$ time, the entire procedure runs in $O(n\log m)$ time. Note that this procedure returns the job completion times in nondecreasing order. In the remainder of this paper we refer to these completion times as $t_1,\ldots,t_n$, with $t_1 \leqslant \ldots \leqslant t_n$.

It is possible to reduce the effort to solve $Q|p_j=1|C_{\max}$ to $O(n+m\log m)$ time. The first step is to solve the linear programming relaxation of P through a procedure suggested by PALEKAR (1989), and then assigning the resulting fractional jobs appropriately. Ignoring the integrality requirement of x_i $(i=1,\ldots,m)$ in (3), an optimal allocation must satisfy

$$x_1/s_1 = x_2/s_2 = \ldots = x_m/s_m = C_{\max}. \tag{4}$$

Substituting the values of x_i from (4) in (2), we get $C_{\max} = n/\Sigma_{i=1}^{m} s_i$, and hence $x_i = ns_i/\Sigma_{i=1}^{m} s_i$. Let $\lfloor x_i \rfloor$ be the largest integer no greater than x_i, and let $n_x = \Sigma_{i=1}^{m} \lfloor x_i \rfloor$. Since the makespan given in (4) is a lower bound on the optimal makespan and the jobs are identical, we know that in each optimal schedule machine M_i $(i=1,\ldots,m)$ will accommodate at least $\lfloor x_i \rfloor$ jobs. If x_i is integral for each $i=1,\ldots,m$, then $n_x = n$ and we have found an optimal allocation. Otherwise, there are $n-n_x$ unallocated jobs, with $1 \leqslant n-n_x \leqslant m-1$; these are scheduled in $O(m\log m)$ time by making use of a priority queue in the same fashion as described above. Since scheduling the n_x jobs takes $O(n)$ time, the procedure requires $O(n+m\log m)$ time. It does not sort the job completion times, however.

Analysis of the $O(n\log m)$ time procedure for $Q|p_j=1|C_{\max}$ reveals that at no point a job will be assigned to a machine in such a manner that its completion time can be reduced by a shift to another machine. The times $t_1,...,t_n$ are the earliest possible completion times. Hence, we have the following.

MINIMALITY PROPERTY. *No schedule exists with completion times $t_1' \leqslant ... \leqslant t_n'$ such that $t_k' < t_k$ for any $k = 1,...,n$.*

This property has significant implications. We can solve any problem with an objective function that is non-decreasing in the job completion times by matching the jobs J_j $(j = 1,...,n)$ with the completion times t_k $(k = 1,...,n)$. We now first show how to solve the general problems $Q|p_j = 1|\Sigma f_j$ and $Q|p_j = 1|f_{\max}$; we will assume that each cost function evaluation requires unit time. Thereafter, in Section 3, we discuss objective functions that allow faster algorithms.

Problem 2: Minimize Σf_j
The general problem $Q|p_j = 1|\Sigma f_j$ can be formulated and solved as a linear assignment problem, if the f_j's $(j = 1,...,n)$ are non-decreasing in the job completion times. The generic form is as follows. Let $c_{jk} = f_j(t_k)$ denote the cost of matching job J_j with completion time t_k. Introduce assignment variables x_{jk} $(j = 1,...,n,\ k = 1,...,n)$ such that $x_{jk} = 1$ if job J_j is matched with time t_k, and $x_{jk} = 0$ otherwise. The problem is then to minimize

$$\begin{aligned} &\textstyle\sum_{j=1}^{n} x_{jk} = 1, && k = 1,...,n, \\ &\textstyle\sum_{k=1}^{n} x_{jk} = 1, && j = 1,...,n, \\ &x_{jk} \in \{0,1\}, && j = 1,...,n, \quad k = 1,...,n. \end{aligned}$$

This linear assignment problem is formulated in $O(n^2)$ time and solved in $O(n^3)$ time.

Problem 3: Minimize maximum cost $f_{\max}$
The Minimality Property justifies the application of Lawler's algorithm for $1||f_{\max}$ (LAWLER, 1973) to $Q|p_j = 1|f_{\max}$. Starting with the largest unmatched job completion time t_k $(k = n,..., 1)$, we determine a job J_h from among the set of unscheduled jobs V for which

$$f_h(t_k) = \min_{j, \in V} f_j(t_k),$$

and match J_h with completion time t_k. This algorithm runs in $O(n^2)$ time.

3. MORE EFFICIENT ALGORITHMS

There are some objective functions for which the matching can be found faster than by the methods given in the previous section.

Problem 4: Minimize total completion time ΣC_j or any other problem with

identical f_j's
Since the completion times $t_1,\dots,t_n$ are minimum and the jobs have identical cost functions, we can arbitrarily match the jobs with the completion times. Therefore, Problem 4 can be solved in $O(n+m\log m)$ time, the time required to find the set of minimum completion times.

Problem 5: Minimize total weighted completion time $\Sigma w_j C_j$
$Q|p_j=1|\Sigma w_j C_j$ is solved by arranging the jobs in order of non-increasing weights and matching them accordingly with non-decreasing job completion times. The correctness of the algorithm is easily established by the same argument that validates Smith's shortest weighted processing time rule (SMITH, 1956) for $1||\Sigma w_j C_j$: interchanging two adjacent jobs that are not scheduled in compliance with the indicated order reduces the cost of the schedule.

Problem 6: Minimize maximum lateness $L_{\max}$
Maximum lateness is defined as $L_{\max}=\max_{1\leqslant j\leqslant n}(C_j-d_j)$. The $Q|p_j=1|L_{\max}$ problem is solved by sorting the jobs in order of non-decreasing due dates, and matching them accordingly with non-decreasing completion times. This procedure is an extension of Jackson's earliest due date rule (JACKSON, 1955) for minimizing maximum lateness on a single machine, and runs in $O(n\log n)$ time. The algorithm is again justified by an interchange argument.

Problem 7: Minimize the weighted number of tardy jobs $\Sigma w_j U_j$
Define U_j as the incidence of tardiness of job J_j, that is, $U_j=1$ if $C_j-d_j>0$ and $U_j=0$ otherwise. We seek a schedule that minimizes the weighted number of tardy jobs, $\Sigma_{j=1}^{n} w_j U_j$.
If all w_j's are equal, then the problem is solved in $O(n\log n)$ time through an obvious extension of Moore and Hodgson's algorithm for $1||\Sigma U_j$ (MOORE, 1968). LAWLER (1989) proposes the following algorithm for the case of general weights. Starting with the largest unmatched completion time t_k $(k=n,\dots,1)$, determine the set of unscheduled jobs V that would be in time if matched with t_k. If $V\neq\varnothing$, determine a job $J_h\in V$ for which $w_h=\max_{J_j\in V} w_j$, and match it with completion time t_k. Ultimately, we find a set of tardy jobs, and they are matched arbitrarily with the unmatched completion times. The algorithm is justified by an interchange argument and can be implemented to run in $O(n\log n)$ time.

Problem 8: Minimize total tardiness ΣT_j
The tardiness of job J_j is defined as $T_j=\max\{C_j-d_j,0\}$. It is easy to establish through an interchange argument that $Q|p_j=1|\Sigma T_j$ is solved as follows: renumber the jobs in order of non-decreasing due dates, and match them accordingly with the completion times $t_1,\dots,t_n$.
It is noteworthy that this problem can be viewed as a Gilmore-Gomory matching problem. When we define $\alpha_j=d_j$, $\beta_k=t_k$, $g(y)=1$, and $h(y)=0$, we can write the cost c_{jk} of matching job J_j with completion time t_k as

$$c_{jk} = \begin{cases} \int_{\alpha_j}^{\beta_k} g(y)dy, & \alpha_j \leqslant \beta_k, \\ \int_{\beta_k}^{\alpha_j} h(y)dy, & \beta_k < \alpha_j. \end{cases}$$

If the jobs and completion times have been indexed in order of non-decreasing values of α_j and β_k, respectively, a minimum-weight matching J_j with t_j for $j = 1,...,n$ (LAWLER, 1976).
Note that among other problems, $Q|p_j = 1|\Sigma f_j$ with $f_j = |C_j - d_j|$ can be formulated and solved as a Gilmore-Gomory matching problem. This is true only subject to the condition that the jobs must be completed at times $t_1,...,t_n$, as these cost functions are not monotone. More generally, the same matching optimizes the minsum criteria with $f_j = T_j^p$ and $f_j = |C_j - d_j|^p$ for any $p > 0$ (DESSOUKY, 1989).

Problem 9: Minimize maximum weighted tardiness $\max w_j T_j$
Recently, HOCHBAUM and SHAMIR (1989) have presented an intricate $O(n\log^2 n)$ time algorithm for $1||\max w_j T_j$, which can readily be transferred to $Q|p_j = 1|\max w_j T_j$. We propose a simpler algorithm, which is slightly less efficient in case the weights are large.
Without loss of generality we may assume that $\min_i s_i = 1$ and that all $w_j \cdot \max\{0, t_k - d_j\}$ are integral. The problem of deciding whether there exists a matching with $\max w_j T_j \leqslant K$ for a given $K \in \mathbb{N}_0$ can be answered in $O(n\log n)$ time as follows. An upper bound K on the maximum weighted tardiness induces a deadline $d_j + K/w_j$ for each J_j. Hence, the decision problem has an affirmative answer only if each job can be scheduled to meet its deadline. This is verified in $O(n\log n)$ time by solving the corresponding $Q|p_j = 1|L_{\max}$ problem.
Since $0 \leqslant \max w_j T_j \leqslant w_{\max} t_n$ for any matching, where $w_{\max} = \max_j w_j$, and $t_n \leqslant n/m + 1$, the optimal maximum weighted tardiness can be determined by binary search over the interval $[0, w_{\max}(n/m + 1)]$. Hence, the algorithm runs in $O(n\log n(\log w_{\max} + \log(n/m)))$ time.

4. RELEASE DATES

Suppose now that each job J_j $(j = 1,...,n)$ becomes available at a given release date $r_j \geqslant 0$. This makes it impossible to specify a set of earliest completion times in advance and to invoke some procedure that matches jobs with completion times. Nonetheless, we give an $O(n\log n)$ time procedure for minimizing makespan, and an $O(mn^{2m+1})$ time dynamic programming algorithm for minimizing makespan, and an $O(mn^{2m+1})$ time dynamic programming algorithm for minimizing total completion time.

Problem 10: Minimize makespan $C_{\max}$ subject to release dates
For minimizing makespan, LAGEWEG, LAWLER, LENSTRA, and RINNOOY KAN (1982) observe that $Q|r_j, p_j = 1|C_{\max}$ is solvable in polynomial time due to the

symmetry between this problem and $Q|p_j=1|L_{\max}$. A more explicit description of this idea is a follows.
Imagine a tentative deadline $\bar{d}$ at which all jobs have to be finished. We can identify a set of latest start times for the jobs in order to meet this deadline $\bar{d}$. Since $t_1,\dots,t_n$ obtained in the forward computation for $Q|p_j=1|C_{\max}$ are the *earliest* job completion times, $\bar{d}-t_n,\dots,\bar{d}-t_1$ must be the *latest* start times of the jobs in the $Q|r_j,p_j=1|C_{\max}$ problem. The procedure is as follows: match the jobs in order of non-decreasing release dates with nondecreasing latest start times. If we initially choose $\bar{d}=t_n$, which is an evident lower bound on the optimal makespan, then the entire schedule needs to be delayed by $\Delta=\max_{1\leqslant j\leqslant n}(r_j-\bar{d}+t_{n-j+1})$ in order not to violate any of the job release dates. The resulting schedule is optimal, and the maximum job completion time is $C_{\max}=t_n+\Delta$. The procedure is easily validated through an interchange argument. The running time is $O(n\log n)$.

Problem 11: Minimizing total completion time ΣC_j subject to release dates
We propose a dynamic programming algorithm that requires $O(mn^{2m+1})$ time, which is polynomial for any fixed number of machines. Without loss of generality we may assume that all release dates r_j and processing times $p_i=1/s_i$ are integral. We may also assume, without loss of optimality, that each machine processes its jobs in order of non-decreasing release dates. We now renumber the jobs in order of non-decreasing release dates. The algorithm below will assign the jobs successively to the machines in order of increasing indices.
Moreover, each job may be assumed to start as early as possible on the machine it is assigned to. This implies that job J_j on machine M_i will start either on r_j or on r_h+kp_i with $h<j$ and $k\geqslant 1$. For the completion time of J_j on M_i we therefore have to consider only a limited number of possible values. These values are contained in the set $Q_i=\{r_h+kp_i|h=1,\dots,n,\ k=1,\dots,n+1-h\}$; note that this set contains $O(n^2)$ points in time, for each machine. If machine M_i is available up to time q, the latest admissible completion time of any job on M_i is $l_i(q)=\max\{0,\{t\in Q_i|t\leqslant q\}\}$. If M_i is idle at time q, the latest admissible completion time prior to q is $l_i(q-1)$. If some job J_i is completed on M_i at time $q\in Q_i$, then it contributes q to the total completion time; furthermore, its predecessor must be finished by $l_i(q-p_i)$.
We can now set up the recursion. Let $F_j(q_1,\dots,q_m)$ denote the optimal total completion time for $J_1,\dots,J_j$ subject to the condition that M_i is available up to time q_i, for $i=1,\dots,m$, $j=1,\dots,n$. If J_j is scheduled on M_i, then either M_i is idle at time q_i, or J_j finishes at time q_i. Therefore,

$$F_j(q_1,\dots,q_m)=$$
$$\min_{1\leqslant i\leqslant m:q_i\geqslant r_j+p_i}\left\{\begin{array}{l} q_i+F_{j-1}(q_1,\dots,q_{i-1},l_i(q_i-p_i),q_i+1,\dots,q_m) \\ F_j(q_1,\dots,q_{i-1,}l_i(q_i-1),q_i-1),q_i+1,\dots,q_m),\text{with } l_i(q_i-1)\geqslant r_j+p_i \end{array}\right\}$$

for all $q_i\in Q_i\cup\{0\}$ and for $j=1,\dots,n$. We initialize $F_0(q_1,\dots,q_m)=0$ for all

$q_i \in Q_i \cup \{0\}$, and undefined values are taken to be infinity. The optimal total completion time is given by

$$F_n(q_1^{\max},...,q_m^{\max}),$$

where $q_i^{\max} = \max\{q \in Q_i\}$, and the corresponding schedule can be identified by backtracing.

The complexity of the algorithm is determined as follows. During the recursion, we need the values $l_i(q_i - p_i)$ and $l_i(q_i - 1)$, for $q_i \in Q_i$, $i = 1,...,m$. We compute these values in a preprocessing phase. For each i $(i = 1,...,m)$, the elements of the set Q_i are sorted in non-decreasing order, which takes $O(n^2 \log n)$ time. By running through the sorted set, we then compute and store the values $l_i(q_i - p_i)$ for all $q_i \in Q_i$ in $O(n^2)$ time altogether; at the same time, we also determine $l_i(q_i - 1)$, which is nothing but the predecessor of q_i in the sorted set. Hence, the preprocessing phase requires $O(mn^2 \log n)$ time overall. After this preprocessing phase, each value $l_i(q_i - p_i)$ and $l_i(q_i - 1)$ can be found in constant time, and the computation of $F_j(q_1,...,q_m)$ for a given job index j and a given vector $(q_1,...,q_m)$ requires only $O(m)$ time. Since each $|Q_i| = O(n^2)$, we have to consider $O(n^{2m})$ vectors $(q_1,...,q_m)$ for each j, with $j = 1,...,n$. Hence, the entire procedure runs in $O(mn^{2m+1})$ time for $m \geqslant 2$.

REFERENCES

AHO, A.V. J.E. HOPCROFT, and J.D. ULMAN (1982), *Data Structures and Algorithms*, Addison-Wesley, Reading, Massachusetts.

DESSOUKY, M.I. (1989), Bipartite weighted matching to minimize the distance norm, Report ORL 89-003, Operations Research Laboratory, University of Illinois at Urbana-Champaign.

GRAHAM, R.L. E.L. LAWLER, J.K. LENSTRA, and A.H.G. RINNOOY KAN (1979), Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics 5*, 287-326.

HOCHBAUM, D.S. and R. SHAMIR (1989), An $O(n\log^2 n)$ algorithm for the maximum weighted tardiness problem, *Information Processing Letters 31*, 215-219.

JACKSON, J.R. (1955), Scheduling a production line to minimize maximum tardiness, Research Report nr. 43, Management Science Research Project, UCLA.

LAGEWEG, B.J. E.L. LAWLER, J.K. LENSTRA, and A.H.G. RINNOOY KAN (1982), Computer aided complexity classification of deterministic scheduling problems, Report BW 138, Centre for Mathematics and Computer Science, Amsterdam.

LAWLER, E.L. (1973), Optimal sequencing of a single machine subject to precedence constraints, *Management Science 19*, 544-546.

LAWLER, E.L. (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York.

LAWLER, E.L. (1989), Private communication.

LAWLER, E.L. J.K. LENSTRA, and A.H.G. RINNOOY KAN (1982), Recent developments in deterministic sequencing and scheduling, in: *Deterministic and Stochastic Scheduling,* M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinnooy Kan (eds) Reidel, Dordrecht, 35-73.

MOORE, J.M. (1968), An *n* job, one machine sequencing algorithm for minimizing the number of late jobs, *Management Science 15,* 102-109.

PALEKAR, U.S. (1989), Private communication.

SMITH, W.E. (1956), Various optimizers for single-stage production, *Naval Research Logistics Quarterly 3,* 59-66.

Received May 1989, revised March 1990.