Mathematical Programming 62 (1993) 461–474
North-Holland

An approximation algorithm for the generalized assignment problem

David B. Shmoys* and Éva Tardos**

School of Operations Research and Engineering, Cornell University, 232 ET&C Building, Ithaca, NY, USA

Received 10 April 1991
Revised manuscript received 14 January 1993

The generalized assignment problem can be viewed as the following problem of scheduling parallel machines with costs. Each job is to be processed by exactly one machine; processing job j on machine i requires time p_{ij} and incurs a cost of c_{ij}; each machine i is available for T_i time units, and the objective is to minimize the total cost incurred. Our main result is as follows. There is a polynomial-time algorithm that, given a value C, either proves that no feasible schedule of cost C exists, or else finds a schedule of cost at most C where each machine i is used for at most $2T_i$ time units.

We also extend this result to a variant of the problem where, instead of a fixed processing time p_{ij}, there is a range of possible processing times for each machine–job pair, and the cost linearly increases as the processing time decreases. We show that these results imply a polynomial-time 2-approximation algorithm to minimize a weighted sum of the cost and the makespan, i.e., the maximum job completion time. We also consider the objective of minimizing the mean job completion time. We show that there is a polynomial-time algorithm that, given values M and T, either proves that no schedule of mean job completion time M and makespan T exists, or else finds a schedule of mean job completion time at most M and makespan at most $2T$.

Key words: Approximation algorithms, generalized assignment problem, scheduling unrelated parallel machines.

1. Introduction

The generalized assignment problem can be viewed as the following problem of scheduling parallel machines with costs. Each of n independent jobs is to be processed by exactly one of m unrelated parallel machines; job j takes p_{ij} time units when processed by machine i, and incurs a cost c_{ij}, $i=1,\dots,m$, $j=1,\dots,n$. For notational simplicity, we shall assume that $n \geqslant m$. Given values C and T_i, $i=1,\dots,m$, we wish to decide if there exists a schedule of total cost at most C such that for each machine i, its *load*, the total processing time required for

Correspondence to: Prof. David B. Shmoys, School of Operations Research and Engineering, Cornell University, 232 ET&C Building, Ithaca, NY 14853, USA.

*Research partially supported by an NSF PYI award CCR-89-96272 with matching support from UPS, and Sun Microsystems, and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550.

**Research supported in part by a Packard Fellowship, a Sloan Fellowship, an NSF PYI award, and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550.

for the jobs assigned to it, is at most T_i, $i=1, \ldots, m$. By scaling the processing times we can assume, without loss of generality, that the machine load bounds satisfy $T_1=T_2=\cdots=T_m=T$. In other words, the generalized assignment problem is to find a schedule of minimum cost subject to the constraint that the *makespan*, the maximum machine load, is at most T. Minimizing the makespan is also an important optimization criterion, and so we shall study this problem as a bicriteria optimization problem.

Lenstra, Shmoys and Tardos [4] give a polynomial-time 2-approximation algorithm for the single criterion problem of minimizing the makespan, where a *ρ-approximation algorithm* is one that is guaranteed to produce a solution with objective function value at most ρ times the optimum. In this paper, we generalize that result to the bicriteria problem mentioned above.

Trick [8, 9] and Lin and Vitter [5] consider variants of this bicriteria problem. Lin and Vitter [5] give a polynomial-time algorithm that, given cost C, makespan T, and $\varepsilon>0$, finds a solution of cost at most $(1+\varepsilon)C$ and makespan at most $(2+1/\varepsilon)T$, if there exists a schedule of cost at most C and makespan at most T. In the variant considered by Trick [8, 9], there is an interval of possible processing times, rather than a fixed time p_{ij}, and the cost of processing job j on machine i linearly increases as the processing time decreases. Trick [9] focuses on the single criterion problem of minimizing a linear objective function that is a weighted sum of the cost and the makespan, and gives a polynomial-time 2.618-approximation algorithm.

The main result of our paper is as follows. We present a polynomial-time algorithm that, given values C and T, finds a schedule of cost at most C and makespan at most $2T$, if a schedule of cost C and makespan T exists. As a corollary, we show a similar result for the bicriteria problem of minimizing the makespan and the mean job completion time; we also give a polynomial-time 2-approximation algorithm for the variant considered by Trick.

All of the above algorithms are based on solving linear relaxations of a particular integer programming formulation, and then rounding the fractional solution to a nearby integer solution. Whereas the results of Trick [8, 9] and Lin and Vitter [5] invoke the rounding theorem of Lenstra, Shmoys and Tardos [4], the main contribution of this paper is the introduction of a new rounding technique. The technique used in [4] requires that the solution to be rounded must be a vertex of the linear relaxation. One interesting aspect of the new technique is that it does not have this restriction.

The most time-consuming part of our approximation algorithms is the solution of the linear relaxations. For our results that separately treat the two criteria, we observe that these linear programs fall into the class of fractional packing problems considered in [7], and therefore a slightly further relaxed schedule can be found by a randomized algorithm in $O(n^2 \log n)$ expected time, or deterministically, in $O(mn^2 \log n)$ time.

Approximation algorithms for special cases of the scheduling problems considered in this paper have been studied over the last twenty-five years, and for a survey of this literature, the reader is referred to [3]. Finally, we note that it is likely that our results cannot be too substantially improved upon, since Lenstra, Shmoys and Tardos [4] have shown the following result for the single criterion problem of minimizing the makespan: for any $\varepsilon<\frac{1}{2}$, no polynomial-time $(1+\varepsilon)$-approximation algorithm exists, unless P = NP.

2. The main result

We first consider the simplest version of our scheduling problem, when there is a fixed processing time p_{ij} and a cost c_{ij} associated with each machine $i=1, \ldots, m$, and each job $j=1, \ldots, n$. For any $t \geqslant T$, integer solutions to the following linear program, LP(t), are in one-to-one correspondence with schedules of cost at most C and makespan at most T.

$$\begin{aligned}
\mathrm{LP}(t)\colon\ & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \leqslant C, \\
& \sum_{i=1}^{m} x_{ij} = 1 \quad \text{for } j=1, \ldots, n, \\
& \sum_{j=1}^{n} p_{ij}x_{ij} \leqslant T \quad \text{for } i=1, \ldots, m, \\
& x_{ij} \geqslant 0 \quad \text{for } i=1, \ldots, m,\ j=1, \ldots, n, \\
& x_{ij} = 0 \quad \text{if } p_{ij} > t,\ i=1, \ldots, m,\ j=1, \ldots, n.
\end{aligned}$$

Theorem 2.1. *If* LP(t) *has a feasible solution, then there exists a schedule that has makespan at most $T+t$ and cost at most C.*

We will prove the theorem by providing an algorithm that converts a feasible solution x of LP(t) to the required schedule. We will construct a bipartite graph $B(x)=(V, W, E)$ and a value $x'(v, w)$ for each edge $(v, w) \in E$. One side of the bipartite graph consists of *job nodes*

$$W=\{w_j\colon j=1, \ldots, n\}.$$

The other side consists of *machine nodes*

$$V=\{v_{is}\colon i=1, \ldots, m,\ s=1, \ldots, k_i\},$$

where $k_i=\lceil\sum_j x_{ij}\rceil$; the k_i nodes $\{v_{is}\colon s=1, \ldots, k_i\}$ correspond to machine i, $i=1, \ldots, m$.

Edges of the graph $B(x)$ will correspond to machine–job pairs (i, j) such that $x_{ij}>0$. For each positive coordinate of x, there will be one or two corresponding edges in $B(x)$. The vector x' defined on the edges of $B(x)$ will have the property that, for each $i=1, \ldots, m$, $j=1, \ldots, n$,

$$x_{ij}=\sum_{s\colon(v_{is},\ w_j)\in E} x'(v_{is}, w_j).$$

The cost of each edge $(v_{is}, w_j) \in E$ is c_{ij}.

The graph $B(x)$ and the vector x' are constructed in the following way. To construct the edges incident to the nodes corresponding to machine i, sort the jobs in order of nonincreasing processing time p_{ij}; for simplicity of notation, assume for the moment that

$$p_{i1} \geqslant p_{i2} \geqslant \cdots \geqslant p_{in}.$$

If $\sum_{j=1}^{n} x_{ij} \leqslant 1$, then there is only one node $v_{i1} \in V$ corresponding to machine i: in this case, for each $x_{ij} > 0$, include $(v_{i1}, w_j) \in E$, and set $x'(v_{i1}, w_j) := x_{ij}$.

Otherwise, find the minimum index j_1 such that $\sum_{j=1}^{j_1} x_{ij} \geqslant 1$. Let E contain those edges (v_{i1}, w_j), $j = 1, \ldots, j_1 - 1$, for which $x_{ij} > 0$, and for each of these set $x'(v_{i1}, w_j) := x_{ij}$. Furthermore, add edge (v_{i1}, w_{j_1}) to E and set $x'(v_{i1}, w_{j_1}) := 1 - \sum_{j=1}^{j_1-1} x'(v_{i1}, w_j)$. This ensures that the sum of the components of x' for edges incident to v_{i1} is exactly 1. If $\sum_{j=1}^{j_1} x_{ij} > 1$, then a fraction of the value x_{ij_1} is still unassigned, and so create an edge (v_{i2}, w_{j_1}), and set

$$x'(v_{i2}, w_{j_1}) := x_{ij_1} - x'(v_{i1}, w_{j_1}) = \left(\sum_{j=1}^{j_1} x_{ij}\right) - 1.$$

We then proceed with jobs $j > j_1$, i.e., those with smaller processing times on machine i, and construct edges incident to v_{i2}, until a total of exactly one job is assigned to v_{i2}, and so forth. More precisely, for each $s = 2, \ldots, k_i - 1$, find the minimum index j_s such that $\sum_{j=1}^{j_s} x_{ij} \geqslant s$. Let E contain those edges (v_{is}, w_j), $j = j_{s-1} + 1, \ldots, j_s - 1$, for which $x_{ij} > 0$, and for each of these set $x'(v_{is}, w_j) := x_{ij}$. Furthermore, add edge (v_{is}, w_{j_s}) to E and set $x'(v_{is}, w_{j_s}) := 1 - \sum_{j=j_{s-1}+1}^{j_s-1} x'(v_{is}, w_j)$. If $\sum_{j=1}^{j_s} x_{ij} > s$, then also put edge $(v_{i,s+1}, w_{j_s}) \in E$, and set

$$x'(v_{i,s+1}, w_{j_s}) := x_{ij_s} - x'(v_{is}, w_{j_s}) = \left(\sum_{j=1}^{j_s} x_{ij}\right) - s.$$

Let j' be the last job assigned in this way; that is, $j' = j_{k_i-1}$. For each $j > j'$ for which $x_{ij} > 0$, create an edge (v_{ik_i}, w_j) and set $x'(v_{ik_i}, w_j) := x_{ij}$. For each machine node v_{is}, let $p_{is}^{\max}$ denote the maximum of the processing times p_{ij} corresponding to edges $(v_{is}, w_j) \in E$; let $p_{is}^{\min}$ denote the analogous minimum.

We shall use the following instance to give an example of this construction: $m = 3$; $n = m(m-1) + 1$; $p_{i1} = m$, $i = 1, \ldots, m$; $p_{ij} = 1$, $i = 1, \ldots, m$, $j = 2, \ldots, n$; $c_{ij} = 0$, $i = 1, \ldots, m$, $j = 1, \ldots, n$; $C = 0$; $T = m$. Figure 1 gives a feasible solution $X = (x_{ij})$ to LP(T), and the corresponding graph $B(x)$.

$$X = \begin{bmatrix} 1/3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

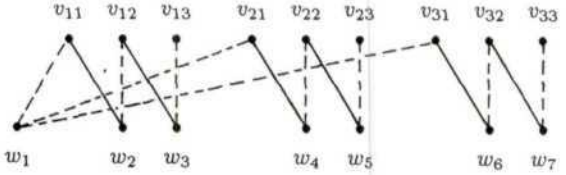

Fig. 1. Constructing $B(x)$.

A non-negative vector z on the edges of a graph is a *fractional matching* if, for each node u, the sum of the components of z corresponding to the edges incident to u is at most 1. The fractional matching *exactly matches a node u* if the corresponding sum is exactly 1. A fractional matching z is a *matching* if each component of z is 0 or 1. The following lemma summarizes some simple properties of the above construction.

Lemma 2.2. *The vector x' is a fractional matching in $B(x)$ of cost at most C. It exactly matches each node w_j, $j=1, \ldots, n$, and each node v_{is}, for each $i=1, \ldots, m$, $s=1, \ldots, k_i-1$. Finally, $p_{is}^{\min} \geqslant p_{i,s+1}^{\max}$ for each $i=1, \ldots, m$, $s=1, \ldots, k_i-1$.* □

The algorithm to construct a schedule from a feasible solution x of LP(t) is as follows.

The algorithm.

Step 1. Form the bipartite graph $B(x)$ with costs on its edges.

Step 2. Find a minimum-cost (integer) matching M that exactly matches all job nodes in $B(x)$.

Step 3. For each edge $(v_{is}, w_j) \in M$, schedule job j on machine i.

Proof of Theorem 2.1. We shall prove that the schedule produced by the algorithm satisfies the requirements of the theorem. By Lemma 2.2, x' is a fractional matching in $B(x)$ of cost at most C, which matches all job nodes exactly. This implies there exists an (integral) matching M in $B(x)$ of cost at most C that exactly matches all job nodes (see, for example, [6]). Therefore, the matching required in Step 2 exists and has cost at most C. The cost of the matching is the same as the cost of the schedule constructed. Therefore, the cost of the schedule constructed is at most C.

Next we show that the makespan of the schedule constructed is at most $T+t$. Consider the load of machine i, $i=1, \ldots, m$. There are k_i nodes corresponding to machine i in $B(x)$, and for each of these, there will be at most one job scheduled on machine i corresponding to some incident edge. Therefore, the length of the time required by machine i is at most $\sum_{s=1}^{k_i} p_{is}^{\max}$. Clearly, $p_{i1}^{\max} \leqslant t$. Lemma 2.2 implies that the sum of the remaining terms,

$$\sum_{s=2}^{k_i} p_{is}^{\max} \leqslant \sum_{s=1}^{k_i-1} p_{is}^{\min} \leqslant \sum_{s=1}^{k_i-1} \sum_{j:(v_{is}, w_j) \in E} p_{ij} x'(v_{is}, w_j)$$

$$\leqslant \sum_{s=1}^{k_i} \sum_{j:(v_{is}, w_j) \in E} p_{ij} x'(v_{is}, w_j) = \sum_{j=1}^{n} p_{ij} x_{ij} \leqslant T,$$

which proves the theorem. □

Observe that the algorithm ensures that if $x_{ij}=1$, then job j is assigned to be scheduled on machine i, since each edge incident to w_j in $B(x)$ is of the form (v_{is}, w_j) for some s. Also note that the obvious m-machine generalization of the example given in Figure 1 shows that the analysis of this algorithm is asymptotically tight.

Remark. The rounding technique used to obtain Theorem 2.1 can also be used for integer programs of a slightly more general form with varying right hand sides. Consider the extension of the problem of scheduling unrelated parallel machines in which there are n_j jobs of type j, $j=1, \ldots, n$, and each machine i is available for processing for only T_i time units, $i=1, \ldots, m$. Feasible schedules of cost at most C are in one-to-one correspondence with integer solutions to the following linear program for any $t_i \geqslant T_i$ for $i=1, \ldots, m$.

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \leqslant C,$$

$$\sum_{i=1}^{m} x_{ij} = n_j \quad \text{for } j=1, \ldots, n,$$

$$\sum_{j=1}^{n} p_{ij} x_{ij} \leqslant T_i \quad \text{for } i=1, \ldots, m,$$

$$x_{ij} \geqslant 0 \quad \text{for } i=1, \ldots, m,\ j=1, \ldots, n,$$

$$x_{ij} = 0 \quad \text{if } p_{ij} > t_i,\ i=1, \ldots, m,\ j=1, \ldots, n,$$

The natural extension of the proof of Theorem 2.1 shows that if this linear program has a feasible solution, then there exists a schedule of cost at most C where the load on machine i is at most $T_i + t_i$, for $i=1, \ldots, m$. The immediate extension of the construction of Theorem 2.1 is pseudopolynomial in the job multiplicities n_j: there are at least $\sum_j n_j$ machine nodes v_{is} in the bipartite graph $B(x)$. In order to find a solution in polynomial time we first schedule the integer part of the fractional schedule x; that is, we assign $\lfloor x_{ij} \rfloor$ jobs of type j to machine i for each $j=1, \ldots, n$ and $i=1, \ldots, m$, which leaves fewer than mn jobs unscheduled. We then used the fractional part of the vector x in the construction of Theorem 2.1 to schedule the remaining jobs.

Corollary 2.3. *In the problem with fixed processing times p_{ij}, $i=1, \ldots, m$, $j=1, \ldots, n$, for any given cost C and makespan T, we can find, in polynomial time, a schedule of cost C and makespan at most $2T$, if one of cost C and makespan T exists.*

Proof. If there exists a schedule of cost at most C and makespan at most T, then LP(T) must have a feasible solution. We can use any polynomial-time linear programming algorithm to find a feasible solution to this linear program. The algorithm used to prove Theorem 2.1 can be implemented to run in polynomial time, which implies the claim. $\square$

Corollary 2.4. *In the problem with fixed processing times p_{ij}, $i=1, \ldots, m$, $j=1, \ldots, n$, and non-negative costs, for any given cost C and makespan T, and for any fixed $\varepsilon > 0$, we can find a schedule of cost at most $(1+\varepsilon)C$ and makespan at most $(2+\varepsilon)T$, if a schedule of cost C and makespan T exist, using a randomized algorithm that runs in expected $O(n^2 \log n)$ time.*

Proof. Plotkin, Shmoys and Tardos [7] developed an algorithm that efficiently finds approximate solutions to a wide class of linear programming problems, known as fractional packing problems. If the costs in LP(T) are nonnegative, then this linear program is a fractional packing problem of the form considered in [7]. The techniques of [7] can be used to determine that LP(T) is infeasible, or else produce a solution that is nearly feasible, in the sense that it is feasible if the right-hand sides of the cost constraint and machine load constraints are relaxed by a factor of $1+\varepsilon$. Thus, if this algorithm produces such a fractional solution, we can then use Theorem 2.1 to find the claimed schedule.

To view the linear program LP(T) as a fractional packing problem, we partition the constraints into two categories: the m machine load constraints and the cost constraint are the packing constraints, and the remaining constraints are the job assignment constraints. The algorithms of [7] work by maintaining a solution that satisfies the latter, and iteratively moving towards a solution that also satisfies the former.

An important parameter of a fractional packing problem is its *width*, which is the maximum ratio of the right-hand side to the left-hand side of any packing constraint for any solution x that satisfies the remaining constraints. The width of the above formulation can be as high as $\sum_j \max_i c_{ij}/C$. This can be improved as follows: add constraints that set $x_{ij}=0$ if $c_{ij}>C$. As a consequence, the width is reduced to at most n. If there exists a schedule of makespan at most T and cost at most C then this modified linear program has a feasible solution. Furthermore, we can use the algorithm of Theorem 2.1 to round a feasible solution to this linear program to a schedule.

Since the width is at most n and there are $m+1$ packing constraints, the packing algorithm of [7] finds an approximate solution in $O(n \log n)$ iterations (see Theorem 2.7 of [7]). In each iteration, the algorithm first computes a dual variable corresponding to each packing constraint, which is completely determined by the current primal solution; let y denote the dual variable corresponding to the cost constraint, and let y_i correspond to the load constraint for machine i, $i=1, \ldots, m$. The algorithm then selects a job j uniformly at random, and finds the machine i on which job j may be scheduled (i.e., $p_{ij} \leqslant t$ and $c_{ij} \leqslant C$) for which $yc_{ij}+y_i p_{ij}$ is minimum. A small fraction of job j is rescheduled on this machine. Each iteration takes $O(m)$ time. Therefore, the packing algorithm terminates in $O(mn \log n)$ expected time.

The resulting vector x has $O(n \log n)$ nonzero coordinates. Therefore, the graph $B(x)$ has at most $2n+m=O(n)$ nodes and $O(n \log n)$ edges. The minimum-cost matching that exactly matches the n job nodes can be found via n shortest path computations; using the Fredman–Tarjan implementation of Dijkstra's algorithm, the algorithm runs in $O(n^2 \log n)$ time. □

There is also a deterministic version of the algorithm of [7] that yields a running time of $O(mn^2 \log n)$.

Next consider the version of the problem where job j can be processed by machine i in t_{ij} time units, where $l_{ij} \leqslant t_{ij} \leqslant u_{ij}$ and the cost linearly increases as the processing time decreases. In this model, we are given the minimum cost c_{ij}^u and the maximum cost c_{ij}^l of assigning job j to machine i, for each $i=1, \ldots, m$, $j=1, \ldots, n$. The cost associated with processing job j on machine i in time t_{ij} is $\mu c^l_{ij}+(1-\mu)c^u_{ij}$ if the time can be written as $t_{ij}=\mu l_{ij}+(1-\mu)u_{ij}$, where $0 \leqslant \mu \leqslant 1$.

In the linear programming relaxation $\mathrm{LP}_{\mathrm{speed}}(T)$ of this problem there are two variables x^l_{ij} and x^u_{ij} associated with each machine–job pair (i,j), $i=1, \ldots, m$, $j=1, \ldots, n$. A feasible solution to the linear program directly corresponds to a feasible schedule if $x^l_{ij}+x^u_{ij}$ is integral for each machine–job pair (i,j), $i=1, \ldots, m$, $j=1, \ldots, n$. Job j is assigned to machine i if $x^u_{ij}+x^l_{ij}=1$, where the assigned time is $t_{ij}=x^u_{ij}u_{ij}+x^l_{ij}l_{ij}$ at a cost of $c^u_{ij}x^u_{ij}+c^l_{ij}x^l_{ij}$.

In the linear program $\mathrm{LP}(t)$, we forced the variable x_{ij} to zero if $p_{ij}>t$. Analogously, we want to make sure that no job is processed on a machine at a speed on which it would require more than t time units to process the whole job. To do this, we revise the upper bound of the processing times to $\hat{u}_{ij}=\min\{t, u_{ij}\}$. The revised cost $\hat{c}^u_{ij}$ associated with the revised upper bound is the cost of processing job j on machine i in time $\hat{u}_{ij}$, i.e., if $\hat{u}_{ij}=\mu u_{ij}+(1-\mu)l_{ij}$ then we set $\hat{c}^u_{ij}=\mu c^u_{ij}+(1-\mu)c^l_{ij}$. The resulting linear program $\mathrm{LP}_{\mathrm{speed}}(t)$ is as follows.

$$\begin{aligned}
\mathrm{LP}_{\mathrm{speed}}(t):\quad & \sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{c}^u_{ij}x^u_{ij}+c^l_{ij}x^l_{ij}) \leqslant C,\\
& \sum_{i=1}^{m}(x^u_{ij}+x^l_{ij})=1 \quad \text{for } j=1, \ldots, n,\\
& \sum_{j=1}^{n}(\hat{u}_{ij}x^u_{ij}+l_{ij}x^l_{ij}) \leqslant T \quad \text{for } i=1, \ldots, m,\\
& x^u_{ij}, x^l_{ij} \geqslant 0 \quad \text{for } i=1, \ldots, m,\ j=1, \ldots, n,\\
& x^u_{ij}=x^l_{ij}=0 \quad \text{if } l_{ij}>t,\ i=1, \ldots, m,\ j=1, \ldots, n.
\end{aligned}$$

Theorem 2.5. *If the linear program* $\mathrm{LP}_{\mathrm{speed}}(t)$ *has a feasible solution, then there exists a schedule with makespan at most* $T+t$ *and cost at most* C.

Proof. We will prove this theorem by constructing a feasible solution to a related linear program $\mathrm{LP}(t)$ and then applying Theorem 2.1. Consider a feasible solution x^l and x^u to $\mathrm{LP}_{\mathrm{speed}}(t)$. Define the corresponding feasible solution x, scheduling times p, and costs c as follows. Let

$$x_{ij}=x^u_{ij}+x^l_{ij};$$

that is, x_{ij} is the fraction of job j that is scheduled on machine i. For any machine–job pair (i,j) such that $x_{ij}>0$, define its processing time as

$$p_{ij}=(x^u_{ij}\hat{u}_{ij}+x^l_{ij}l_{ij})/x_{ij};$$

that is, p_{ij} is the time it would take to process all of job j on machine i at the speed used in the fractional schedule; the corresponding cost is defined to be

$$c_{ij} = (x_{ij}^u \hat{c}_{ij}^u + x_{ij}^l c_{ij}^l) / x_{ij}.$$

For machine–job pairs (i, j) such that $x_{ij} = 0$, set $p_{ij} = +\infty$ (where any value greater than $T + t$ will suffice) and $c_{ij} = 0$.

Observe that $x_{ij} > 0$ implies that $p_{ij} \leqslant t$, $l_{ij} \leqslant p_{ij} \leqslant u_{ij}$, and c_{ij} is the cost of assigning job j to machine i for time p_{ij}. Notice that x is a solution to the linear program LP(t) defined by T, C, and p_{ij} and c_{ij} for $i = 1, \ldots, m$, $j = 1, \ldots, n$. Therefore, Theorem 2.1 implies that the claimed schedule exists. □

Notice that the algorithm ensures that integral assignments (i.e., pairs (i, j) with $x_{ij}^l + x_{ij}^u$ integral) are used in the schedule constructed.

Corollary 2.6. *In the problem with variable processing times, for any given cost C and makespan T, we can find, in polynomial time, a schedule of cost C and makespan at most 2T, if one of cost C and makespan T exists.*

Proof. If there exists a schedule of cost at most C and makespan at most T, then LP$_{\text{speed}}(T)$ must have a feasible solution. We can use any polynomial-time linear programming algorithm to find a feasible solution to this linear program. By applying Theorem 2.5, we obtain the corollary. □

Corollary 2.7. *In the problem with variable processing times p_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$, and non-negative costs, for any given cost C and makespan T, and for any fixed $\varepsilon > 0$, we can find a schedule of cost at most $(1 + \varepsilon)C$ and makespan at most $(2 + \varepsilon)T$, if a schedule of cost C and makespan T exists, using a randomized algorithm that runs in expected $O(n^2 \log n)$ time.*

Proof. This proof of this result relies on techniques from [7] in a way analogous to the proof of Corollary 2.4. To make the width of the corresponding packing problem small, we must further restrict the allowed speeds for the assignments. We increase the lower bounds l_{ij} to a modified lower bound $\hat{l}_{ij}$, if necessary, to ensure that the corresponding cost $\hat{c}_{ij}^l$ is at most C. □

3. Mean job completion time and makespan

In addition to the makespan of the schedule, another important objective is to minimize the mean job completion time M. In this section we consider the bicriteria problem of minimizing the makespan and the mean job completion time with fixed processing times.

Horn [2] and Bruno, Coffmann and Sethi [1] showed that a schedule with minimum mean job completion time can be found by reducing the problem to the minimum-cost bipartite matching problem. The bipartite graph formed by the reduction is quite similar to the graph $B(x)$ used in the proof of Theorem 2.1. There is a node w_j corresponding to each job $j=1, \ldots, n$, and there are n nodes v_{is}, $s=1, \ldots, n$, corresponding to each machine i, $i=1, \ldots, m$; there is an edge between every job node and machine node, and the cost of the edge (v_{is}, w_j) is $s \cdot p_{ij}$. Schedules correspond to matchings in this bipartite graph: a job node w_j matched to machine node v_{is} is interpreted as job j being scheduled sth-to-last on machine i. If job j is the sth-to-last job on machine i, then it contributes p_{ij} to the completion time of s jobs; hence, there is a one-to-one correspondence between schedules of minimum mean job completion time and matchings that exactly match all job nodes of minimum cost.

Observe that in an optimal schedule, the jobs processed on each machine i, $i=1, \ldots, m$, are sequenced in order of nondecreasing processing time. Thus, in the construction used in Theorem 2.1, we can similarly view v_{is} as the sth-to-last position on machine i.

Theorem 3.1. *In the problem with fixed processing times p_{ij}, $i=1, \ldots, m$, $j=1, \ldots, n$, for any given mean job completion time M and makespan T, we can find, in polynomial time, a schedule of mean job completion time M and makespan at most $2T$, if one of mean job completion time at most M and makespan at most T exists.*

Proof. We shall give a polynomial-time algorithm analogous to the one used in the proof of Theorem 2.1. If there exists a schedule with mean job completion time at most M and makespan at most T, then the following linear program has a feasible solution by setting $x_{ij}^s = 1$ if job j is scheduled as the sth-to-last job on machine i, and setting $x_{ij}^s = 0$ otherwise.

$$\mathrm{LP}_{\mathrm{mean}}: \quad \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s=1}^{n} s p_{ij} x_{ij}^s \leqslant nM,$$

$$\sum_{i=1}^{m} \sum_{s=1}^{n} x_{ij}^s = 1 \quad \text{for } j=1, \ldots, n,$$

$$\sum_{j=1}^{n} x_{ij}^s \leqslant 1 \quad \text{for } i=1, \ldots, m, \ s=1, \ldots, n,$$

$$\sum_{j=1}^{n} \sum_{s=1}^{n} p_{ij} x_{ij}^s \leqslant T \quad \text{for } i=1, \ldots, m,$$

$$x_{ij}^s \geqslant 0 \quad \text{for } i=1, \ldots, m, \ s=1, \ldots, n, \ j=1, \ldots, n,$$

$$x_{ij}^s = 0 \quad \text{if } p_{ij} > T, \ i=1, \ldots, m, \ s=1, \ldots, n, \ j=1, \ldots, n.$$

If there exists a schedule of mean job completion time at most M and makespan at most T, then we can use any polynomial-time linear programming algorithm to find a feasible solution x_{ij}^s, $i=1, \ldots, m$, $j=1, \ldots, n$, $s=1, \ldots, n$, to this linear program. Consider a vector x with coordinates $x_{ij} = \sum_{s=1}^{n} x_{ij}^s$, $i=1, \ldots, m$, $j=1, \ldots, n$. Notice that x satisfies all constraints of the linear program LP(T), except for the cost constraint. Use the construction of the proof of Theorem 2.1 to construct the graph $B(x)$, but let the cost of the edge $(v_{is}, w_j) \in E$ be $s \cdot p_{ij}$. The vector x' is a fractional matching of $B(x)$ that matches all job nodes exactly.

We will argue that the total cost of x' is at most nM. If a set of jobs are (integrally) assigned to be processed on machine i, then the mean job completion time on this machine is minimized by scheduling these jobs in order of nondecreasing processing time. The following fractional analogue will suffice to bound the cost of x': each job j has a fraction x_{ij} assigned to machine i, and the sum of these fractions, $\sum_{j=1}^{n} x_{ij}$, is at most k_i; we view machine i as having k_i slots to process jobs, and each fraction x_{ij} is to be subdivided among these k_i job slots; each of the k_i job slots can be assigned fractions that sum to at most one: if a fraction y of job j is assigned to the sth-to-last job slot, it incurs a cost of $y \cdot s \cdot p_{ij}$; the minimum cost assignment can be found by working from the last job slot to the first, scheduling the fractions x_{ij} in nonincreasing order of p_{ij}, and filling the sth-to-last slot to capacity before starting the $(s+1)$th-to-last slot. A simple perturbation argument shows that this procedure constructs an optimal assignment. This procedure is exactly the one used to construct x' from x. However, since the solution x_{ij}^{*} is a feasible assignment of cost at most nM, the total cost of x' is at most nM.

Since x' has cost at most nM, there exists a matching in $B(x)$ that exactly matches all job nodes and has cost at most nM, and such a matching can be found in polynomial time. Schedule job j as the sth-to-last job on machine i if the job node w_j is matched to machine node v_{is} in the matching. The sum of the job completion times for this schedule is at most nM, and therefore the schedule has mean job completion time at most M. The proof of Theorem 2.1 implies that the makespan of the schedule is at most $2T$. □

By applying Theorem 3.1. to the minimum mean job completion time M^*, we get the following: if T is the minimum makespan among all schedules with mean job completion time M^*, then we can efficiently find a schedule with mean job completion time M^* and makespan at most $2T$. A much stronger result would state that it is possible to find a schedule with mean job completion time M^* and makespan at most $2T^*$, where T^* denotes the minimum makespan. This result is known for the special case when the machines are identical: a schedule of minimum mean job completion time can be found by sequencing the jobs in nondecreasing processing time order, and iteratively scheduling the next job on the machine on which it would finish earliest; the same algorithm is known to be a 2-approximation algorithm for the minimum makespan problem. We leave the following question as an interesting open problem: is the makespan of any minimum mean completion time schedule at most $2T^*$?

4. Minimizing a combined objective function

In this section, we return to the setting with variable processing times, and consider the problem of minimizing the objective function consisting of a weighted sum of the makespan and the operating cost,

$$\mu T + \sum_{ij} (c_{ij}^{l} x_{ij}^{l} + c_{ij}^{u} x_{ij}^{u}),$$

for some parameter $\mu > 0$, where the desired makespan is no longer a part of the input. We assume that the operating costs

$$c^l_{ij} \geqslant 0 \text{ and } c^u_{ij} \geqslant 0 \quad \text{for } i=1, \ldots, m,\ j=1, \ldots, n.$$

We shall also assume that the lower and upper bounds on the processing times, l_{ij} and u_{ij}, respectively, are integral, for each machine–job pair (i, j). Let P denote the maximum of the upper bounds on the processing times. Trick [9] gave a ρ-approximation algorithm for this problem, where ρ is roughly 2.618. Here we use Theorem 2.5 to give a 2-approximation algorithm.

For each value $t > 0$, consider the following linear program $\text{LP}_{\text{opt}}(t)$, where $\hat{c}^u_{ij}$, as in the previous section, is the cost of scheduling job j on machine i so that it is processed in $\hat{u}_{ij} = \min\{t, u_{ij}\}$ time units.

$$f(t) = \min \mu T + \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{c}^u_{ij} x^u_{ij} + c^l_{ij} x^l_{ij})$$

$$\begin{aligned} \text{subject to} \quad & \sum_{i=1}^{m} (x^u_{ij} + x^l_{ij}) = 1 \quad \text{for } j=1, \ldots, n, \\ & \sum_{j=1}^{n} (\hat{u}_{ij} x^u_{ij} + l_{ij} x^l_{ij}) \leqslant T \quad \text{for } i=1, \ldots, m, \\ & x^u_{ij}, x^l_{ij} \geqslant 0 \quad \text{for } i=1, \ldots, m,\ j=1, \ldots, n, \\ & x^u_{ij} = x^l_{ij} = 0 \quad \text{if } l_{ij} > t,\ i=1, \ldots, m,\ j=1, \ldots, n. \end{aligned}$$

To find a schedule with objective function value at most twice optimal, we will perform a bisection search on the range of possible makespan values, and maintain the following invariant: all schedules with objective function value less than half the objective function value of the best schedule found thus far must have makespan within the current range. The number of iterations of this bisection search can be bounded by using the following lemma of Trick [9], which follows from a simple perturbation argument.

Lemma 4.1. *Among all schedules with minimum objective function value, consider one with minimum makespan; the makespan of this schedule is integral.* □

Since the makespan of any plausible schedule is at most nP, it follows that we can initialize the search by setting the interval to $[0, nP]$. The core of the bisection search is given by the following lemma.

Lemma 4.2. *For each value $t > 0$, one can, in polynomial time, find a schedule with objective function value $\bar{f}$, and conclude that one of the following holds:* (i) *each schedule with objective function value less than $\frac{1}{2}\bar{f}$ has makespan less than t, or* (ii) *each schedule with objective function value less than $\frac{1}{2}\bar{f}$ has makespan greater than t.*

Proof. The algorithm works as follows. First find an optimal solution x to $\text{LP}_{\text{opt}}(t)$ and compute $f(t)$. Let C and T denote the cost and makespan of this fractional solution; that is,

$f(t) = C + \mu T$. Since x is a feasible solution to $\mathrm{LP}_{\mathrm{speed}}(t)$ for these values C and T, we can apply Theorem 2.5 to obtain a schedule. The objective function value of this schedule is at most $C + \mu(T+t) = f(t) + \mu t$.

We consider two cases: $f(t) \leqslant \mu t$ or $f(t) \geqslant \mu t$. Suppose that $f(t) \leqslant \mu t$. The schedule obtained has objective function value $\bar{f} \leqslant f(t) + \mu t \leqslant 2\mu t$. Any schedule with objective function value less than $\frac{1}{2}\bar{f} \leqslant \mu t$ must clearly have makespan below t. Hence, we can conclude that alternative (i) holds. Suppose instead that $f(t) \geqslant \mu t$. We will show that each schedule with makespan at most t has objective function value at least $\frac{1}{2}\bar{f}$. The schedule constructed has objective function value $\bar{f} \leqslant f(t) + \mu t \leqslant 2f(t)$. Since $f(t)$ is the optimal value of $\mathrm{LP}_{\mathrm{opt}}(t)$, each integral schedule with makespan at most t must have objective function value at least $f(t) \geqslant \frac{1}{2}\bar{f}$. Hence, we can conclude that alternative (ii) applies. □

Observe that if $f(t) = \mu t$, then the algorithm can halt, since (i) and (ii) together imply that there does not exist a schedule with objective function value less than $\frac{1}{2}\bar{f}$. By combining Lemma 4.1 and Lemma 4.2, we obtain the following theorem.

Theorem 4.3. *For the problem of minimizing a weighted sum of the cost and the makespan in scheduling with variable speeds, there exists a 2-approximation algorithm, which needs to solve the linear program* $\mathrm{LP}_{\mathrm{opt}}(t)$ *at most* $\log nP$ *times.* □

Note added in proof

The question at the end of Section 3 was answered by G. Rote, who provided an instance for which all minimum mean completion time schedules have more than twice the optimal makespan. Based on this construction, C. Hurkens showed that the performance guarantee is, in fact, no better than $\log n / \log\log n$.

Acknowledgments

We would like to thank Leslie Hall for prompting this research. Her discovery of an error in our earlier, trivial, but fallacious proof of Corollary 2.3 led us to find this more interesting, correct solution.

References

[1] J.L. Bruno, E.G. Coffmann Jr. and R. Sethi, "Scheduling independent tasks to reduce mean finishing time," *Communications of the ACM* 17 (1974) 382–387.

[2] W.A. Horn, "Minimizing average flow time with parallel machines," *Operations Research* 21 (1973) 846–847.

[3] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, "Sequencing and scheduling: algorithms and complexity," in: A.H.G. Rinnooy Kan and P. Zipkin, eds., *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory* (North-Holland, Amsterdam, 1993) pp. 445–522.

[4] J.K. Lenstra, D.B. Shmoys and É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical Programming* 46 (1990) 259–271.

[5] J.-H. Lin and J.S. Vitter, "ε-approximations with minimum packing constraint violation," in: *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing* (1992) pp. 771–782.

[6] L. Lovász and M. Plummer, *Matching Theory* (Akademiai Kiado, Budapest, and North-Holland, Amsterdam, 1986).

[7] S.A. Plotkin, D.B. Shmoys and É. Tardos, "Fast approximation algorithms for fractional packing and covering problems" Technical Report 999, School of Operations Research and Industrial Engineering, Cornell University (Ithaca, NY, 1992).

[8] M.A. Trick, "Scheduling multiple variable-speed machines," in: *Proceedings of the 1st Conference on Integer Programming and Combinatorial Optimization* (1990) pp. 485–494.

[9] M.A. Trick, "Scheduling multiple variable-speed machines," unpublished manuscript (1991).