Algorithmica (1997) 17: 100–110

Algorithmica

Using Quadratic Programming to Solve High Multiplicity Scheduling Problems on Parallel Machines[1]

F. Granot,[2] J. Skorin-Kapov,[3] and A. Tamir[4]

Abstract. We introduce and analyze several models of scheduling n different types (groups) of jobs on m parallel machines, where in each group all jobs are identical. Our main goal is to exhibit the usefulness of quadratic programming approaches to solve these classes of high multiplicity scheduling problems, with the total weighted completion time as the minimization criterion. We develop polynomial algorithms for some models, and strongly polynomial algorithms for certain special cases. In particular, the model in which the weights are job independent, as well as the generally weighted model in which processing requirements are job independent, can be formulated as an integer convex separable quadratic cost flow problem, and therefore solved in polynomial time. When we specialize further, strongly polynomial bounds are achievable. Specifically, for the weighted model with job-independent processing requirements if we restrict the weights to be machine independent (while still assuming different machine speeds), an $O(mn + n \log n)$ algorithm is developed. If it is also assumed that all the machines have the same speed, the complexity of the algorithm can be improved to $O(m \log m + n \log n)$. These results can be extended to related unweighted models with variable processing requirements in which all the machines are available at time zero.

Key Words. Scheduling, Quadratic programming, Parallel machines.

Introduction. We consider scheduling problems in which the set of jobs can be partitioned into a relatively small number of types or groups of identical jobs, i.e., jobs having the same characteristic parameters. Such problems are called *high multiplicity* problems. The paper by Hochbaum and Shamir (1991) studied a variety of high multiplicity scheduling problems on a single machine. Strongly polynomial algorithms were presented therein for minimizing several measures of efficiency, e.g., weighted number of tardy jobs and maximum weighted tardiness.

In this paper we focus on efficient algorithms for solving high multiplicity models on parallel machines with the total weighted completion time as the minimization criterion. Our main goal is to exhibit the usefulness of quadratic programming approaches to solve these classes of high multiplicity scheduling problems.

The general problem we consider is defined as follows. There are n types of jobs,

[1] The research of Frieda Granot was partially supported by Natural Sciences and Engineering Research Council of Canada Grant 5-83998. The research of Jadranka Skorin-Kapov was partially supported by National Science Foundation Grant DDM-8909206.

[2] Faculty of Commerce and Business Administration, The University of British Columbia, Vancouver, British Columbia, Canada VGT 1Z2.

[3] W. A. Harriman School for Management and Policy, State University of New York at Stony Brook, Stony Brook, NY 11794-3775, USA.

[4] School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

Received August 25, 1994; revised February 2, 1995. Communicated by N. Megiddo.

$J_1, \ldots, J_n$, and there are n_j identical jobs of type J_j, $j = 1, \ldots, n$. We refer to n_j as the *multiplicity* of type J_j. There are m (parallel) machines, $M_1, \ldots, M_m$, to process the jobs. For $i = 1, \ldots, m$, machine M_i is released at time r_i, and it can process at most c_i jobs. (The latter bound might reflect a constraint on the total setup cost associated with switching jobs on M_i.) We denote $N = \sum_{j=1}^{n} n_j$ and $C = \sum_{i=1}^{m} c_i$, and assume that $N \leq C$. In what follows we simplify the notation and refer to a job of type j, and to machine i. Each job must be processed in its entirety by one of the m machines. All the jobs are available for processing at time zero. The processing requirement of a job of type j on machine i is p_{ij} time units. With each job of type j, $j = 1, \ldots, n$, we associate a set of nonnegative weights, w_{ij}, $i = 1, \ldots, m$. (w_{ij} represents the weight factor applied to a type j job if it is processed by machine i.) The objective is to schedule the set of jobs and sequence them on the m machines so that the total weighted completion time (flow time) is minimized. We note that the above model is NP-hard even for the simple case where there is only one job of each type ($n_j = 1$, $j = 1, \ldots, n$), there are two identical machines released at time zero, $c_1 = c_2 = n$, and $p_{ij} = w_{ij} = w_j$, for $i = 1, 2$; $j = 1, \ldots, n$. (See Garey and Johnson, 1979.) In view of the above NP-hardness result we have to limit our discussion of polynomial solvability to restricted models, where all job types have the same weight or all job types have the same processing requirement.

Consider first the case where the weights are job independent but machine dependent, i.e., $w_{ij} = v_i$, for all job types j, $j = 1, \ldots, n$, and machines i, $i = 1, \ldots, m$. In what follows we refer to this case as the (job) *unweighted* case. The single multiplicity case, i.e., $n_j = 1$, for $j = 1, \ldots, n$ (with $r_i = 0$ and $c_i = n$ for each machine i), is solved as an assignment problem in Horn (1973) and Bruno *et al.* (1974). We prove in Section 1 that the high multiplicity case can be formulated as an integer convex separable quadratic cost flow problem, and as such it can be solved in polynomial time. Secondly, in Section 2 we consider the following *weighted* model. It is assumed that all the jobs, from each type, have the same processing requirement (normalized to one unit), but the machines have varying speeds. For $i = 1, \ldots, m$, machine i has a speed of s_i. (In the literature this model is called the *uniform* parallel machine case.) Thus, the processing time of any job on M_i is $1/s_i$ time units. For a general set of weight factors w_{ij}, again we formulate the model as an integer separable quadratic cost flow problem. If we specialize further, strongly polynomial bounds are achievable. When the weights are job and machine *decomposable*, i.e., $w_{ij} = v_i w_j$ for $j = 1, \ldots, n$, $i = 1, \ldots, m$, the problem reduces to an assignment model: There are $C = \sum_{i=1}^{m} c_i$ slots on the m machines for the $N = \sum_{j=1}^{n} n_j$ jobs. The problem then amounts to matching the earliest-finishing slots with the highest-weighted jobs. In particular, a greedy algorithm would solve the problem in $O(m + N \log N)$ time. The latter bound is not polynomial for the high multiplicity problem. However, the greedy approach can easily be modified to the latter case, and be implemented in $O(mn \log m + n \log n)$ time. We show how to improve this bound to $O(mn + n \log n)$ by using quadratic programming techniques. Finally, in Section 2.1 we consider the identical machine case ($s_i = v_i = 1$, $i = 1, \ldots, m$), and present a very efficient $O(m \log m + n \log n)$-time algorithm. We also observe the mathematical equivalence of these strongly polynomially solvable weighted models with identical processing requirements, and related unweighted models with variable processing requirements where all the machines are available at time zero.

1. Unweighted Models with Variable Processing Requirements: Integer Separable Convex Quadratic Formulations. In this section we consider the (job) unweighted case where $w_{ij} = v_i$, for all job types j, $j = 1, \ldots, n$, and machines i, $i = 1, \ldots, m$. The processing time of each one of the n_j jobs of type j on the ith machine is p_{ij} units, $i = 1, \ldots, m$; $j = 1, \ldots, n$.

We first note that the single multiplicity case can be formulated as the following linear assignment problem. The cost of assigning the jth job, $j = 1, \ldots, n$, to the kth place (from the end), $k = 1, \ldots, c_i$, on machine i, $i = 1, \ldots, m$, is $v_i(r_i + kp_{ij})$. (See Horn, 1973; Bruno *et al.*, 1974). For $r_i = 0$ and $c_i = n$, $i = 1, \ldots, m$, a bound of $O(mn + n^3)$ is reported in Lawler *et al.* (1993). Applying this approach to the high multiplicity case yields a nonpolynomial formulation. However, we show that the high multiplicity problem can be formulated as an integer convex separable quadratic cost flow problem, and as such can be solved in polynomial time.

Clearly, since our objective is to minimize the total flow time of the N jobs, all the jobs assigned to the ith machine will be processed in a nondecreasing order of their processing times p_{ij}, $j = 1, \ldots, n$.

For $i = 1, \ldots, m$, let σ_i be a permutation of $\{1, \ldots, n\}$, which arranges the n types of jobs in a nonincreasing order of their processing times. Specifically, suppose $p_{i,\sigma_i(1)} \geq p_{i,\sigma_i(2)} \geq \cdots \geq p_{i,\sigma_i(n)}$. For $i = 1, \ldots, m$; $j = 1, \ldots, n$, let x_{ij} be the number of jobs of type $\sigma_i(j)$ scheduled to be processed on the ith machine. For notational simplicity we define $\sum_{l=1}^{0} \equiv 0$. Thus, the scheduling problem minimizing the total flow time is formulated as

$$
\begin{aligned}
&\min_{x} \sum_{i=1}^{m}\sum_{j=1}^{n} r_i v_i x_{ij} + \sum_{i=1}^{m}\sum_{j=1}^{n} v_i\, p_{i,\sigma_i(j)} \sum_{k=1}^{x_{ij}} \left(\sum_{l=1}^{j-1} x_{il} + k \right) \\
&\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} \leq c_i, \qquad i = 1, \ldots, m, \\
&\qquad \sum_{\substack{(i,k) \\ i=1,\ldots,m;\sigma_i(k)=j}} x_{ik} = n_j, \qquad j = 1, \ldots, n, \\
&\qquad x_{ij} \geq 0, \text{ integral}, \qquad i = 1, \ldots, m; \quad j = 1, \ldots, n.
\end{aligned} \tag{1}
$$

Note that the first term in the objective function represents the total weighted flow time until the machines are released, while the second term represents the weighted flow time until all jobs are completed. A similar formulation, but in the context of the high multiplicity total weighted tardiness problem on a single machine, can be found in Hochbaum *et al.* (1992) and Granot and Skorin-Kapov (1993).

We show first how to replace the nonseparable objective function in (1) by an equivalent separable one. (See also Granot and Skorin-Kapov (1993) for a similar transformation applied to the minimum total weighted tardiness problem.) For each i, $i = 1, \ldots, m$, let $p_{i,\sigma_i(n+1)} = 0$ and $y_{i0} = 0$. Also, for $i = 1, \ldots, m$; $j = 1, \ldots, n$, define $y_{ij} = \sum_{k=1}^{j} x_{ik}$. Problem (1) can now be rewritten as

$$
\min_{x,y} \sum_{i=1}^{m}\sum_{j=1}^{n} v_i(r_i + \tfrac{1}{2} p_{i,\sigma_i(j)})x_{ij} + \tfrac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} v_i(p_{i,\sigma_i(j)} - p_{i,\sigma_i(j+1)})y_{ij}^2
$$

$$
\begin{aligned}
\text{s.t.}\quad & y_{in} \le c_i, && i = 1, \ldots, m,\\
& \sum_{\substack{(i,k)\\ i=1,\ldots,m;\sigma_i(k)=j}} x_{ik} = n_j, && j = 1, \ldots, n,\\
& y_{ij} = y_{i,j-1} + x_{ij}, && i = 1, \ldots, m; \quad j = 1, \ldots, n,\\
& x_{ij}, y_{ij} \ge 0, \text{ integral}, && i = 1, \ldots, m; \quad j = 1, \ldots, n.
\end{aligned} \tag{2}
$$

The objective function in (2) is clearly a convex separable quadratic function, and the linear constraints represent the conservation constraints of a flow problem with flow variables $\{x_{ij}\}$, $\{y_{ij}\}$. Thus, we can use the results in Granot and Skorin-Kapov (1993), Hochbaum and Shanthikumar (1990), Minoux (1986), and Tamir (1993), to conclude that problem (2) can be solved in polynomial time. Specifically, if we let $\bar{n} = \max\{n_j : j = 1, \ldots, n\}$ and $VP = \max\{v_i p_{ij} : i = 1, \ldots, m;\ j = 1, \ldots, n,\}$, it follows from the above references that Problem (2) can be solved in either $P_1(m, n, \log \bar{n})$ time or $P_2(m, n, \log VP)$ time, where P_1 and P_2 are polynomials.

2. Weighted Models with Identical Processing Requirements. We first note that the single multiplicity case of the weighted model, i.e., $p_{ij} = 1/s_i$ and general weights w_{ij}, can be formulated as the following linear assignment problem. The cost of assigning the jth job, $j = 1, \ldots, n$, to the kth place, $k = 1, \ldots, c_i$, on machine i, $i = 1, \ldots, m$, is $(r_i + k(1/s_i))w_{ij}$.

We next consider the high multiplicity version and apply the transformation used in Section 1 to show that this model can also be solved in polynomial time. Since our objective is to minimize the total weighted flow time of the N jobs, all the jobs assigned to the ith machine will be processed in a nonincreasing order of their weights w_{ij}, $j = 1, \ldots, n$. For $i = 1, \ldots, m$, let σ_i be a permutation of $\{1, \ldots, n\}$, which arranges the n types of jobs in a nonincreasing order of their weights. Specifically, suppose $w_{i,\sigma_i(1)} \ge w_{i,\sigma_i(2)} \ge \cdots \ge w_{i,\sigma_i(n)}$. For $i = 1, \ldots, m;\ j = 1, \ldots, n$, let x_{ij} be the number of jobs of type $\sigma_i(j)$ scheduled to be processed on the ith machine. Thus, the scheduling problem minimizing the total weighted flow time is formulated as

$$
\begin{aligned}
\min_x\ & \sum_{i=1}^{m}\sum_{j=1}^{n} r_i w_{i,\sigma_i(j)} x_{ij} + \sum_{i=1}^{m}\sum_{j=1}^{n} \frac{1}{s_i} w_{i,\sigma_i(j)} \sum_{k=1}^{x_{ij}} \left(\sum_{l=1}^{j-1} x_{il} + k \right)\\
\text{s.t.}\quad & \sum_{j=1}^{n} x_{ij} \le c_i, && i = 1, \ldots, m,\\
& \sum_{\substack{(i,k)\\ i=1,\ldots,m;\sigma_i(k)=j}} x_{ik} = n_j, && j = 1, \ldots, n,\\
& x_{ij} \ge 0, \text{ integral}, && i = 1, \ldots, m; \quad j = 1, \ldots, n.
\end{aligned} \tag{3}
$$

With the exception of a slight variation in the linear term of the objective, this formulation is mathematically equivalent to formulation (1) in Section 1. Therefore, the analysis in Section 1 is applicable to this model as well.

REMARK 1. We can assume without loss of generality that the inequality of the constraint $\sum_{j=1}^{n} x_{ij} \le c_i$ in (3) can be replaced by equality, since we can always introduce an additional type of job, say $n + 1$, where $w_{i,n+1} = 0$.

REMARK 2. Comparing formulations (1) and (3) we note that both are mathematically equivalent when the release times are equal to zero.

The above bounds for the high multiplicity model are polynomial but not strongly so. However, when this model is specialized further, strongly polynomial bounds are achievable. When the weights are job and machine decomposable, i.e., $w_{ij} = v_i w_j$, the problem amounts to matching the earliest-finishing slots with the highest-weighted jobs. We have developed a very simple greedy-type algorithm which executes this matching in $O(mn \log m + n \log n)$ time (see Granot *et al.* 1994). Here we prefer to present a solution approach based on the above quadratic programming formulation, and reduce the complexity of $O(mn + n \log n)$ time.

We assume that $w_1 \geq w_2 \geq \cdots \geq w_n$. Referring to (3) we note that in this case σ_i will be the identity permutation for each i, $i = 1, \ldots, m$. Let $w_{n+1} \equiv 0$; $y_{ik} \equiv \sum_{t=1}^{k} x_{it}$, $y_{i0} \equiv 0$, $i = 1, \ldots, m$. We then obtain $x_{ij} = y_{ij} - y_{i,j-1}$, $j = 1, \ldots, n$. (The objective function of (3) can thus be stated only in terms of the y variables.) Defining $N_j = \sum_{l=1}^{j} n_l$, for $j = 1, \ldots, n$, rearranging terms, omitting constant terms, and using Remark 1 above, we obtain an equivalent formulation of Problem (3) in terms of the $m(n-1)$ y variables (for simplicity we denote $a_i \equiv (r_i + 1/2s_i)v_i$):

$$
\begin{aligned}
&\min_{y} \sum_{j=1}^{n-1} \tfrac{1}{2}(w_j - w_{j+1}) \sum_{i=1}^{m} \frac{s_i}{v_i} \left(\frac{1}{s_i} v_i y_{ij} + a_i \right)^2 \\
&\text{s.t.} \quad \sum_{i=1}^{m} y_{ij} = N_j, \qquad j = 1, \ldots, n-1, \\
&\qquad y_{i,j-1} \leq y_{ij} \leq c_i, \qquad i = 1, \ldots, m; \quad j = 1, \ldots, n-1, \\
&\qquad y_{ij} \text{ integral}, \qquad i = 1, \ldots, m; \quad j = 1, \ldots, n-1.
\end{aligned} \tag{4}
$$

Consider next the relaxation of (4) obtained by omitting the constraints $y_{i,j-1} \leq y_{ij}$, $i = 1, \ldots, m$; $j = 1, \ldots, n-1$. Since, for $j = 1, \ldots, n-1$, $(w_j - w_{j+1})$ is a nonnegative constant, the solution to the relaxed problem can be found by solving $(n-1)$ independent quadratic knapsack problems of the form

$$
\begin{aligned}
&\min_{y} \sum_{i=1}^{m} \frac{s_i}{v_i} \left(\frac{1}{s_i} v_i y_{ij} + a_i \right)^2 \\
&\text{s.t.} \quad \sum_{i=1}^{m} y_{ij} = N_j, \\
&\qquad 0 \leq y_{ij} \leq c_i, \qquad i = 1, \ldots, m, \\
&\qquad y_{ij} \text{ integral}, \qquad i = 1, \ldots, m.
\end{aligned} \tag{P_j}
$$

The right-hand side coefficients $N_1, N_2, \ldots, N_{n-1}$, corresponding to Problems (P_1), $(P_2), \ldots, (P_{n-1})$, respectively, form a monotonically increasing sequence. Therefore, it follows from the validity of the general, so-called "marginal allocation" or "incremental" algorithm (see Section 4.2 of Ibaraki and Katoh, 1988), that for each j, $j = 1, \ldots, n-2$, if $\{y_{ij}^*\}$, $i = 1, \ldots, m$, is an optimal solution to Problem (P_j), there exists an optimal solution $\{y_{i,j+1}^*\}$, $i = 1, \ldots, m$, to Problem (P_{j+1}) such that $y_{ij}^* \leq y_{i,j+1}^*$, $i = 1, \ldots, m$. In particular, for $j = 1, \ldots, n-1$, these solutions $\{y_{ij}^*\}$, $i = 1, \ldots, m$, to Problems

$(P_1), \ldots, (P_{n-1})$, respectively, satisfy the relaxed constraints and, therefore, constitute an optimal solution to (4). This result implies that the solution to Problem (4) depends on the ordering of the weights w_j, but not their magnitudes. To generate the above solutions to Problems (P_j), we define $y^*_{i0} = 0, i = 1, \ldots, m$, and inductively solve the following equivalent version of (P_j) for $j = 1, \ldots, n-1$,

$$(P_j^*) \qquad \begin{aligned} \min_y \ & \sum_{i=1}^{n} \frac{s_i}{v_i} \left(\frac{1}{s_i} v_i y_{ij} + a_i \right)^2 \\ \text{s.t.} \ & \sum_{i=1}^{m} y_{ij} = N_j, \\ & y^*_{i,j-1} \le y_{ij} \le c_i, \qquad i = 1, \ldots, m, \\ & y_{ij} \text{ integral}, \qquad i = 1, \ldots, m. \end{aligned}$$

Problem (P_j^*) can be solved in $O(m)$ time as follows. First apply the linear-time algorithm of Brucker (1984) to solve the continuous relaxation of (P_j^*). (See also Pardalos and Kovoor, 1990.) Then use the $O(m)$ rounding scheme given in Ibaraki and Katoh (1988, Theorem 4.6.2, p. 76) to obtain the optimal integral solution. The above discussion implies the following result.

THEOREM 1. *Problem* (4) *can be solved in* $O(mn)$ *time. Moreover, the solution depends on the order of the weights* $\{w_j\}$ *but not on their magnitudes. If we include the preprocessing time, i.e., the time to sort the weights* $\{w_j\}$ *in a nonincreasing order, the total computational complexity is* $O(mn + n \log n)$.

2.1. *The Identical Processing Rate Case.* In the instance discussed here all the machines have identical processing rates and machine-independent weights (i.e., $p_{ij} = 1, w_{ij} = w_j; i = 1, \ldots, m, j = 1, \ldots, n$). Following the above notation (see Problem (P_j)), and letting $\bar{N}$ be a nonnegative integer, consider the uniform parametric integer quadratic effort allocation problem:

$$(5) \qquad \begin{aligned} f(\bar{N}) = \min_z \ & \sum_{i=1}^{m} (z_i + a_i)^2 \\ \text{s.t.} \ & \sum_{i=1}^{m} z_i = \bar{N}, \\ & 0 \le z_i \le c_i, \qquad i = 1, \ldots, m, \\ & z_i \text{ integral}, \qquad i = 1, \ldots, m. \end{aligned}$$

Let $\{z_i^*(\bar{N})\}, i = 1, \ldots, m$, denote an optimal solution yielding an optimal value $f(\bar{N})$. From the discussion above we may assume that for each i, $i = 1, \ldots, m$, $z_i^*(\bar{N})$ is a monotonically nondecreasing function of $\bar{N}$. Moreover, an optimal solution to the identical rates version of (4) is given by $z_i^*(N_j)$ for $i = 1, \ldots, m$; $j = 1, \ldots, n-1$. We now construct in $O(m \log m)$ time a compact $O(m)$-space representation of the parametric function $f(\bar{N})$. The representation is in terms of (at most $2m$) critical values of the integer parameter $\bar{N}$, and it enables us to compute f efficiently for any given sequence

of n values of $\bar{N}$, say, $N_1 < N_2 < \cdots < N_n$, in $O(\min(m^2 + n; m \log m + n \log m))$ time.

To facilitate the discussion substitute $y_i = z_i + a_i, i = 1, \ldots, m$, to obtain the equivalent problem:

$$
\begin{aligned}
f(\bar{N}) \;=\; & \min_{y} \sum_{i=1}^{m} y_i^2 \\
& \text{s.t.} \quad \sum_{i=1}^{m} y_i = \bar{N} + \sum_{i=1}^{m} a_i, \\
& \qquad a_i \le y_i \le c_i + a_i, \qquad i = 1, \ldots, m, \\
& \qquad y_i - a_i \text{ integral}, \qquad i = 1, \ldots, m.
\end{aligned} \tag{6}
$$

Let $\{y_i^*(\bar{N})\}$ denote an optimal solution of (6), that is $f(\bar{N}) = \sum_{i=1}^{m} (y_i^*(\bar{N}))^2$. The following algorithm generates the compact representation of the function $f(\bar{N})$. It finds a sequence of $2m$ critical values $0 = N(1) \le N(2) \le \cdots \le N(2m) = \sum_{i=1}^{m} c_i$, and for each interval $[N(k), N(k+1)], k = 1, \ldots, 2m-1$, it produces the necessary information to interpolate and compute $f(\bar{N})$ and the corresponding optimal solution for any integer $\bar{N}$ in this interval.

The algorithm has (at most $2m$) stages. At each stage k the index set of the variables, $M = \{1, \ldots, m\}$, is partitioned into three subsets: $I_1(k)$ which consists of all indices of variables y_i that are already fixed at their upper bounds, $I_2(k)$ which contains the indices of the variables that are fixed at their lower bounds, and $I_3(k)$ which consists of the remaining indices in M. The critical value $N(k+1)$ is determined at state k.

We assume without loss of generality that $c_i, i = 1, \ldots, m$, is integral. (Note, however, that $a_i, i = 1, \ldots, m$, is not assumed to be integral, since the release time r_i might be any real number in this model.) For $i, i = 1, \ldots, m$, let $\bar{a}_i$ be defined by $\bar{a}_i \equiv a_i \pmod 1$, $0 \le \bar{a}_i < 1$. We show that if $N(k) < N(k+1)$, then $N(k+1) = N(k) + \delta |I_3(k)|$, for some positive integer δ. Moreover, for all $i \in I_3(k)$, $y_i^*(N(k+1)) = \alpha(k) + \delta + \bar{a}_i$ for some positive integer $\alpha(k)$.

THE PARAMETRIC ALGORITHM

To initiate the algorithm set: $I_1(1) = \emptyset$, $I_3(1) = \{i : a_i \le a_j \text{ for all } j \in M\}$, $I_2(1) = M - I_3(1)$, and $\alpha(1) = a_i - \bar{a}_i$ for some index $i \in I_3(1)$. Let $N(1) = 0$ and $y_i^*(N(1)) = a_i, i = 1, \ldots, m$. (Note that $y_i^*(N(1)) = \alpha(1) + \bar{a}_i$ for any $i \in I_3(1)$.)

Stage $k, k = 1, 2, \ldots$

Step 1

If $I_2(k) = \emptyset$ set $\beta = \infty$ and go to Step 2.

If $I_2(k) \neq \emptyset$ select $i_2 \in I_2(k)$ such that $a_{i_2} = \min\{a_i : i \in I_2(k)\}$.

If $a_{i_2} < \alpha(k) + 1$, then set $N(k+1) = N(k)$, $\alpha(k+1) = \alpha(k)$, $y_i^*(N(k+1)) = y_i^*(N(k))$, $i \in M$, $I_3(k+1) = I_3(k) + \{i_2\}$, $I_2(k+1) = I_2(k) - \{i_2\}$, $I_1(k+1) = I_1(k)$ and return to Step 1.

If $a_{i_2} > \alpha(k) + 1$, then set $\beta = a_{i_2} - \bar{a}_{i_2} - \alpha(k)$, and go to Step 2. ($\beta$ is the largest possible simultaneous increase in the variables y_i, $i \in I_3(k)$, before we start increasing the variable y_{i_2}, i.e., move the index i_2 from $I_2(k)$ to $I_3(k)$.)

Step 2
Select $i_3 \in I_3(k)$ satisfying $c_{i_3} + a_{i_3} = \min\{c_i + a_i : i \in I_3(k)\}$.
Let $\gamma = (c_{i_3} + a_{i_3}) - (\alpha(k) + \bar{a}_{i_3})$. ($\gamma$ is the largest possible simultaneous increase in the variables y_i, $i \in I_3(k)$, due to the upper bounds $\{c_i + a_i\}$, $i \in I_3(k)\}$.) Note that $\gamma \geq 1$.

Step 3 (Increasing the variables y_i, $i \in I_3(k)$, by $\delta \geq 1$)
Let $\delta = \min(\beta, \gamma)$. Set $N(k+1) = N(k) + \delta|I_3(k)|$,

$$y_i^*(N(k+1)) = \begin{cases} y_i^*(N(k)) + \delta, & i \in I_3(k), \\ y_i^*(N(k)), & \text{otherwise.} \end{cases}$$

Set $\alpha(k+1) = \alpha(k) + \delta$.

Step 4
Set $J_2 = J_3 = \emptyset$.
If $\delta = \gamma$, let $J_3 = \{i \in I_3(k) : c_i + a_i = y_i^*(N(k+1))\}$. If $\delta = \beta$, let $J_2 = \{i_2\}$. Set $I_1(k+1) = I_1(k) + J_3$, $I_2(k+1) = I_2(k) - J_2$, $I_3(k+1) = I_3(k) - J_3 + J_2$.

Step 5
If $I_1(k+1) = M$, stop. If $I_3(k+1) = \emptyset$ set $I_3(k+1) = \{i_2\}$ and $\alpha(k+1) = a_{i_2} - \bar{a}_{i_2}$.
Return to Step 1.

We claim that the above algorithm has at most $2m$ stages. Furthermore, an optimal solution yielding $f(N(k+1))$, $k = 1, 2, \ldots$, is defined in Step 3. (Note that $N(2m) = \sum_{i=1}^m c_i$.) To establish the $2m$ bound on the number of stages note that $I_1(k) \subseteq I_1(k+1)$ and $I_2(k+1) \subseteq I_2(k)$, for $k = 1, 2, \ldots$. Also $\sum_k |I_3(k+1) - I_3(k)| \leq 2m$, since each $i \in M$ enters I_3 at some iteration and departs to I_1 at a later iteration. At each iteration k, $k = 1, 2, \ldots$,

$$y_i^*(N(k)) = \begin{cases} \alpha(k) + \bar{a}_i, & i \in I_3(k), \\ a_i, & i \in I_2(k), \\ c_i + a_i, & i \in I_1(k). \end{cases}$$

To validate the algorithm consider an integer $\bar{N}$, $\bar{N} < \sum_{i=1}^m c_i$. Let $\bar{y}_i(\bar{N})$, $i \in M$, be an optimal solution yielding $f(\bar{N})$. Define $i_0 \in M$ by $\bar{y}_{i_0}(\bar{N}) = \min\{\bar{y}_i(\bar{N}) : \bar{y}_i(\bar{N}) < c_i + a_i\}$. Since the objective is to minimize the symmetric function $\sum_{i=1}^m y_i^2$ the "marginal allocation" algorithm (Ibaraki and Katoh, 1988) guarantees that an optimal solution resulting with $f(\bar{N}+1)$ is obtained by increasing $\bar{y}_{i_0}(\bar{N})$ by one unit. More generally, define

$$I_3 = \{i : \bar{y}_i(\bar{N}) - \bar{y}_{i_0}(\bar{N}) < 1, \bar{y}_i(\bar{N}) < c_i + a_i\}.$$

Let δ be any positive integer such that $\bar{y}_i(\bar{N})+\delta \leq c_i+a_i$, for all $i \in I_3$, and $\bar{y}_{i_0}(\bar{N})+\delta \leq \bar{y}_j(\bar{N})$ for all $j \in M$ with $\bar{Y}_{i_0}(\bar{N}) + 1 \leq \bar{y}_j(\bar{N})$. Then an optimal solution yielding $f(\bar{N} + \delta|I_3|)$ is defined by

$$\bar{y}_i(\bar{N} + \delta|I_3|) = \begin{cases} \bar{y}_i(\bar{N}) + \delta, & \text{if} \quad i \in I_3, \\ \bar{y}_i(\bar{N}), & \text{otherwise.} \end{cases}$$

The later observation justifies the claim that the algorithm presented above does indeed produce the optimal solution for all values of $N(k), k = 1, 2, \ldots$.

The parametric algorithm can be implemented in $O(m \log m)$ time and in $O(m)$ space by applying conventional data structures. We use one priority queue (see Aho *et al.*, 1974) to store the elements $\{a_i : i \in I_2(k)\}$, and a second one for the elements $\{c_i + a_i : i \in I_3(k)\}$. Since the total number of insertions and deletions performed by the algorithm is $O(m)$, the total effort of computing the minima in Steps 1 and 2 is $O(m \log m)$. For each $k, k = 1, 2, \ldots$, the algorithm outputs $\alpha(k)$ and the sets $I_1(k) - I_1(k-1)$, $I_2(k) - I_2(k-1)$, and $I_3(k) - I_3(k-1)$. The additional effort needed to compute $f(N(k))$, for all values of $k, k = 1, 2, \ldots$, is only $O(m)$.

Suppose that $N(k+1) = N(k) + \delta|I_3(k)|$ for some positive integer δ. Then

$$\begin{aligned} f(N(k+1)) &= f(N(k)) + \sum_{i \in I_3(k)} ((y_i^*(N(k)) + \delta)^2 - (y_i^*(N(k)))^2) \\ &= f(N(k)) + 2\delta \sum_{i \in I_3(k)} y_i^*(N(k)) + \delta^2 |I_3(k)| \\ &= f(N(k)) + 2\delta \sum_{i \in I_3(k)} \bar{a}_i + (2\delta\alpha(k) + \delta^2)|I_3(k)|. \end{aligned}$$

Returning now to the identical rates version of Problem (4), that is, where $1/s_i = v_i = 1$, $i = 1, \ldots, m$, we need to evaluate $f(\bar{N})$ for each element of the sequence $N_1 < N_2 < \cdots < N_{n-1}$, where $N_j = \sum_{l=1}^{j} n_l$. We present two approaches to perform this latter task.

Suppose without loss of generality that $\bar{a}_1 \leq \bar{a}_2 \leq \cdots \leq \bar{a}_m$. We first merge the sequence $\{N_1, \ldots, N_{n-1}\}$ with the sequence of critical values $N(1), N(2), \ldots, N(2m)$ produced by the parametric algorithm. The effort needed is clearly $O(m + n)$.

Consider now some $j, 1 \leq j \leq n-1$, and suppose that $N(k) < N_j < N(k+1)$. Let b, d be nonnegative integers such that $N_j - N(k) = b|I_3(k)| + d$ and $0 \leq d < |I_3(k)|$. Let $J^d(k)$ be a subset of $I_3(k)$ consisting of the smallest d indices in $I_3(k)$. With our supposition that $\bar{a}_1 \leq \bar{a}_2 \leq \cdots \leq \bar{a}_m$, we have

$$y_i^*(N_j) = \begin{cases} y_i^*(N(k)) + b, & \text{if} \quad i \in I_3(k) - J^d(k), \\ y_i^*(N(k)) + b + 1, & \text{if} \quad i \in J^d(k), \\ y_i^*(N(k)), & \text{otherwise.} \end{cases}$$

Therefore,

$$\begin{aligned} f(N_j) &= f(N(k)) + 2b \sum_{i \in I_3(k) - J^d(k)} y_i^*(N(k)) + 2(b+1) \sum_{i \in J^d(k)} y_i^*(N(k)) \\ &\quad + b^2 |I_3(k) - J^d(k)| + (b+1)^2 |J^d(k)| \end{aligned}$$

$$= f(N(k)) + 2b \sum_{i \in I_3(k) - J^d(k)} (\alpha(k) + \bar{a}_i) + 2(b+1) \sum_{i \in J^d(k)} (\alpha(k) + \bar{a}_i)$$
$$+ b^2 |I_3(k) - d| + (b+1)^2 d.$$

Hence,

$$f(N_j) = f(N(k)) + 2b \sum_{i \in I_3(k)} \bar{a}_i + 2 \sum_{i \in J^d(k)} \bar{a}_i + |I_3(k)|(b^2 + 2b\alpha(k)) + d(2b + 2\alpha(k) + 1). \tag{7}$$

The first solution procedure to compute $f(N_j)$, $j = 1, \ldots, n-1$, requires $O(m^2)$ time of preprocessing. Since $|I_3(k)| \leq m$, for any k it takes a total of $O(m)$ time to compute $\sum_{i \in J^d(k)} \bar{a}_i$ for all $d = 0, 1, \ldots, |I_3(k)| - 1$. Thus, the total processing time of computing $\sum_{i \in J^d(k)} \bar{a}_i$ for all $k = 1, \ldots, 2m$, and $d = 0, 1, \ldots, |I_3(k)| - 1$ is $O(m^2)$. It then follows from (7) that $f(N_j)$ can be computed in constant time for each j, $j = 1, \ldots, n-1$. Thus, the total time to solve this model using the above approach is $O(m \log m + m^2 + n) = O(m^2 + n)$.

We note that if $\bar{a}_i = e$ for some constant e for all i, $i = 1, \ldots, m$ (e.g., when r_i, the release time of the ith machine, is integral for i, $i = 1, \ldots, m$), then no preprocessing is required since $\sum_{i \in J^d(k)} \bar{a}_i = de$ for any integer d. The total time in this case reduces to $O(m \log m + n)$.

The second solution procedure is based on an application of a data structure described in Galil *et al.* (1986). Throughout the execution of the parametric algorithm we dynamically maintain the set $I_3(k)$ as follows: If $i \in I_3(k)$ is the dth smallest index in $I_3(k)$, it is associated with the real number $\Theta^d(k) \equiv \sum_{q \in J^d(k)} \bar{a}_q$. Suppose that at some state of the algorithm an index i has to be deleted from $I_3(k)$. If i is the dth smallest index in $I_3(k)$, then, in addition to deleting i, we have to subtract the real $\bar{a}_i$ from $\Theta^q(k)$ for all $q = d+1, \ldots, |I_3(k)|$. Similarly, when an index i is to be inserted into $I_3(k)$ the following update is needed. Suppose that i is to become the dth smallest index in $I_3(k)$. Then i is associated with $\Theta^{d-1}(k) + \bar{a}_i$, and for each q, $q = d+1, \ldots, |I_3(k)| + 1$, we have to add $\bar{a}_i$ to $\Theta^q(k)$.

The data structures described in Galil *et al.* (1986) can be used dynamically to maintain the elements $\Theta^q(k)$, $q = 1, \ldots, |I_3(k)|$. These data structures need only $O(\log m)$ time to perform any deletion or insertion of the above nature. Also, for any value of q, $q = 1, \ldots, |I_3(k)|$, it takes $O(\log m)$ time to compute the entry $\Theta^q(k)$.

We now return to the computation of $f(N_j)$, $j = 1, \ldots, n-1$. Suppose that $N(k) < N_j < N(k+1)$ and consider (7). Clearly, using the above data structures each $f(N_j)$ can be computed in $O(\log m)$ time at the end of the kth stage of the parametric algorithm.

Thus, the total time to solve the problem with the second procedure is $O(m \log m + n \log m)$. Including the $O(n \log n)$ time for initially ordering the n job types by their weight factors w_j, $j = 1, \ldots, n$, the total complexity bound of the first procedure is $O(m^2 + n \log n)$, while the respective bound for the second scheme is only $O(m \log m + n \log m + n \log n) = O(m \log m + n \log n)$. The above discussion can thus be summarized in the following theorem.

THEOREM 2. *The total time needed to solve the identical rates version of Problem*

(4) (*i.e.*, $p_i = 1, w_{ij} = w_j$), *is* $O(m \log m + n \log n)$. *If the job types are already sorted by the weight factor, the identical rates version of Problem* (4) *is solvable in* $O(\min(m^2 + n; (m + n) \log m))$ *time.*

Finally, using Remark 2 we note that the complexity bounds of Theorems 1 and 2 can also be obtained for related unweighted models with variable processing requirements in which all the machines are available at time zero.

Acknowledgment. The authors wish to thank the referees for helpful suggestions on improving this paper.

References

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.

Brucker, P. 1984. An Algorithm for Quadratic Knapsack Problems, *Operations Research Letters* **3**(3), 163–166.

Bruno, J. L., E. G. Coffman, Jr., and R. Sethi. 1974. Scheduling Independent Tasks to Reduce Mean Finishing Time, *Communications of the ACM* **17**, 382–387.

Galal, Z., S. Macali, and H. Gabow. 1986. Priority Queues with Variable Priority and an $O(EV \log V)$ Algorithm for Finding a Maximal Weighted Matching in General Graphs, *SIAM Journal on Computing* **15**, 120–130.

Garey, M. R., and D. S. Johnson. 1979. *Computers and Intractability*: *a Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA.

Granot, F., and J. Skorin-Kapov. 1993. On Polynomial Solvability of the High Multiplicity Total Weighted Tardiness Problem, *Discrete Applied Mathematics* **41**, 139–146.

Granot, F., J. Skorin-Kapov, and A. Tamir. 1994. On Solvability of High Multiplicity Scheduling Problems on Parallel Machines, Working Paper, SUNY, Stony Brook, NY.

Hochbaum, D. S., and R. Shamir. 1991. Strongly Polynomial Algorithms for the High Multiplicity Scheduling Problem, *Operations Research* **39**, 648–653.

Hochbaum, D. S., and J. G. Shanthikumar, 1990. Convex Separable Optimization is Not Much Harder than Linear Optimization, *Journal of the ACM* **37**(4), 843–862.

Hochbaum, D. S., R. Shamir, and J. G. Shanthikumar. 1992. A Polynomial Algorithm for an Integer Quadratic Non-Separable Transportation Problem, *Mathematical Programming* **55**, 359–371.

Horn, W. A. 1973. Minimizing Average Flow Time with Parallel Machines, *Operations Research* **21**, 846–847.

Ibaraki, T., and N. Katoh. 1988. *Resource Allocation Problems*: *Algorithmic Approaches*, MIT Press, Cambridge, MA.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. 1993. Sequencing and Scheduling: Algorithms and Complexity, in *Handbooks in Operations Research and Management Science*, Volume 4, eds. S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin, North-Holland, Amsterdam, pp. 445–522.

Minoux, M. 1986. Solving Integer Minimum Cost Flows with Separable Convex Objective Polynomially, *Mathematical Programming Study* **26**, 237–239.

Pardalos, P. M., and N. Kovoor. 1990. An Algorithm for a Singly Constrained Class of Quadratic Programs Subject to Lower and Upper Bounds, *Mathematical Programming* **46**, 321–328.

Tamir, A. 1993. A Strongly Polynomial Algorithm for Minimum Convex Separable Quadratic Cost Flow Problems on Two-Terminal Series-Parallel Networks, *Mathematical Programming* **59**, 117–132.