Discrete Applied Mathematics 48 (1994) 201–218
North-Holland

Scheduling identical parallel machines to minimize total weighted completion time

H. Belouadah and C.N. Potts

Faculty of Mathematical Studies, University of Southampton, Southampton SO9 5NH, UK

Received 15 November 1989
Revised 1 January 1991

Abstract

A branch and bound algorithm is proposed for the problem of scheduling jobs on identical parallel machines to minimize the total weighted completion time. Based upon a formulation which partitions the period of processing into unit time intervals, the lower bounding scheme is derived by performing a Lagrangean relaxation of the machine capacity constraints. A special feature is that the multipliers are obtained by a simple heuristic method which allows each lower bound to be computed in polynomial time. This bounding scheme, along with a new dominance rule, is incorporated into a branch and bound algorithm. Computational experience indicates that it is superior to known algorithms.

1. Introduction

The problem of scheduling jobs on identical parallel machines to minimize total weighted completion time may be stated as follows. Each of n jobs (numbered $1, \ldots, n$) is to be processed on one of m identical parallel machines (numbered $1, \ldots, m$). No machine can handle more than one job at a time. Each job i ($i = 1, \ldots, n$) becomes available for processing at time zero, requires a positive integer *processing time* p_i on the machine to which it is assigned and has a positive *weight* w_i. Preemption of jobs is not allowed. A schedule defines the *start time* S_i and the *completion time* C_i of each job i. The objective is to find a schedule which minimizes the total weighted completion time $\sum_{i=1}^{n} w_i C_i$.

For the case of one machine, the shortest weighted processing time (SWPT) rule of Smith [20] solves the problem in $O(n \log n)$ time by sequencing the jobs in nondecreasing order of p_i/w_i. Also, Conway et al. [5] show that the problem of minimizing total completion time ($w_i = 1$ for $i = 1, \ldots, n$) on identical parallel machines is solved in $O(n \log n)$ time using the following generalized SPT rule. Jobs are renumbered in

SSDI 0166-218X(92)00176-U

SPT order so that $p_1 \leq \cdots \leq p_n$ and are list scheduled: having scheduled jobs $1, \ldots, i-1$, job i is assigned to the first unfilled position on a machine that becomes available earliest. For the case of arbitrary weights, an analogous generalized SWPT rule, which is first suggested by Eastman et al. [6], may not generate an optimal solution (unless $p_i = 1$ for $i = 1, \ldots, n$). This is not surprising, since for arbitrary weights and two machines, Bruno et al. [3] show that the problem is NP-hard. Nevertheless, computational results of Baker and Merten [1] show that the generalized SWPT rule consistently produces schedules having a total weighted completion time which is close to the optimal value. Kawaguchi and Kayan [13] show that the total weighted completion time for the schedule generated by the generalized SWPT rule does not exceed $(\sqrt{2}+1)/2$ times the optimal value. An ε-approximation scheme is proposed by Sahni [18]: for any $\varepsilon > 0$, he describes an algorithm, that requires $O(n(n^2/\varepsilon)^{m-1})$ time, for which the total weighted completion time of the resulting schedule does not exceed $1+\varepsilon$ times the optimal value. Branch and bound algorithms are proposed by Elmaghraby and Park [7], Barnes and Brennan [2] and Sarin et al. [19] which are based on variants of a lower bound originally derived by Eastman et al. The algorithm of Sarin et al. appears to be the most effective.

We propose a branch and bound algorithm which is based on Lagrangean relaxation. The remaining sections of the paper are organized as follows. Section 2 reviews various properties of an optimal schedule, some of which are used as dominance rules to eliminate nodes of our branch and bound search tree. The derivation of a new lower bounding procedure using Lagrangean relaxation is given in Section 3. Special attention is given to an efficient construction whereby values of multipliers are determined at modest computational expense. A numerical example to illustrate the computation of the bound is also included. A complete description of the algorithm is given in Section 4. Section 5 reports on computational experience with the algorithm and Section 6 contains some concluding remarks.

2. Properties of an optimal schedule

In this section, we review various results of Elmaghraby and Park. These results are each used, either implicitly or explicitly, in our branch and bound algorithm.

Firstly, we observe that no optimal schedule can have machine idle time between jobs: if there were idle time, it could be removed by scheduling some jobs earlier, thereby reducing the total weighted completion time. Secondly, in any optimal schedule, the jobs on each machine are sequenced in SWPT order since any other ordering permits the total weighted completion time to be reduced through an adjacent job interchange. We have established the following result.

Theorem 1 (Elmaghraby and Park). *In any optimal schedule, there is no machine idle time between jobs and, on each machine, jobs are sequenced in SWPT order.*

Theorem 1 shows that once jobs are assigned to machines, they are sequenced in SWPT order. Thus, a schedule is specified by an assignment of jobs to machines. We discuss later how Theorem 1 is used in our branch and bound algorithm.

Our next result is based on the observation that the total weighted completion time of a schedule can be reduced if $S_i > C_h$ for any job i, where job h is the last job scheduled on a machine: this reduction is achieved by reassigning job i to the machine which processes job h and scheduling it to start at time C_h.

Theorem 2 (Elmaghraby and Park). *In any optimal schedule, if job h is scheduled last on a machine, then* $C_h \geq S_i$ *for all jobs i.*

Theorem 2 is used later to justify the branching rule which is adopted in our branch and bound algorithm. It is also used to derive an upper bound on the maximum completion time.

The final result in this section is a dominance theorem which is used in most branch and bound algorithms to eliminate search tree nodes.

Theorem 3 (Elmaghraby and Park). *If* $p_h \leq p_i$ *and* $w_h \geq w_i$ *for any jobs h and i, then there exists an optimal schedule in which* $S_h \leq S_i$.

3. The proposed lower bound

3.1. Lagrangean relaxation

In this section, we derive a lower bounding scheme based on Lagrangean relaxation [9, 10]. Our general approach resembles that adopted by Christofides et al. [4] for a resource allocation problem in project networks and by Fisher [8] for the problem of scheduling a single machine to minimize total tardiness. However, these algorithms use subgradient optimization [12] to determine values of the multipliers, whereas we propose a noniterative method which allows lower bounds to be computed at modest computational expense.

Before giving our problem formulation, it is convenient to derive an upper bound on the maximum completion time of any job in an optimal schedule. If job i is completed last in an optimal schedule, then using Theorem 2, no machine can complete its processing before time S_i. Thus, $S_i \leq (\sum_{h=1}^{n} p_h - p_i)/m$, since there is no machine idle time between processing jobs. If

$$D = \left\lfloor \sum_{h=1}^{n} p_h/m + (m-1) \max_{h=1,\ldots,n} \{p_h\}/m \right\rfloor,$$

it is apparent from the integrality of processing times that $C_i = S_i + p_i \leq D$. Thus, we may regard D as a common deadline on job completion times.

Our formulation of the problem relies on partitioning into unit time intervals the period $[0, D]$ during which processing is possible in an optimal schedule. We use a variable C_i to represent the completion time of job i ($i = 1, \ldots, n$) and a zero-one variable x_{it} ($i = 1, \ldots, n$; $t = 1, \ldots, D$), where

$$x_{it} = \begin{cases} 1, & \text{if job } i \text{ is processed during the interval } [t-1, t]; \\ 0, & \text{otherwise.} \end{cases}$$

Our formulation is

$$\text{minimize} \quad \sum_{i=1}^{n} w_i C_i,$$

$$\text{subject to} \quad C_i \in \{p_i, \ldots, D\} \qquad (i = 1, \ldots, n), \tag{1}$$

$$x_{it} = 1 \text{ for } t = C_i - p_i + 1, \ldots, C_i \qquad (i = 1, \ldots, n), \tag{2}$$

$$x_{it} = 0 \text{ for } t = 1, \ldots, C_i - p_i, C_i + 1, \ldots, D \qquad (i = 1, \ldots, n), \tag{3}$$

$$\sum_{i=1}^{n} x_{it} \leq m \qquad (t = 1, \ldots, D). \tag{4}$$

Constraints (1) specify the possible completion times of each job i, while constraints (2) and (3) define the values of x_{it} for $t = 1, \ldots, D$ in terms of C_i. The machine capacity constraints (4) specify that a maximum of m jobs can be processed concurrently in any unit time interval.

We now obtain a lower bound by performing a Lagrangean relaxation of the machine capacity constraints (4). Let $\lambda = (\lambda_1, \ldots, \lambda_D)$ be a vector of nonnegative multipliers associated with constraints (4). The resulting Lagrangean problem is

$$L(\lambda) = \min \left\{ \sum_{i=1}^{n} w_i C_i + \sum_{t=1}^{D} \lambda_t \left(\sum_{i=1}^{n} x_{it} - m \right) \right\},$$

subject to (1), (2) and (3).

Substituting the values of the variables x_{it} obtained from constraints (2) and (3) into the Lagrangean function, we obtain the equivalent problem

$$L(\lambda) = \min \left\{ \sum_{i=1}^{n} \left(w_i C_i + \sum_{t=C_i-p_i+1}^{C_i} \lambda_t \right) \right\} - m \sum_{t=1}^{D} \lambda_t, \tag{5}$$

subject to (1).

Standard theory of Lagrangean relaxation [9, 10] shows that $L(\lambda)$ is a lower bound on the minimum total weighted completion time for any $\lambda \geq 0$. In a natural economic interpretation of multipliers, we regard λ_t as a price for using a machine during the time interval $[t-1, t]$. At the end of a schedule when many jobs are already

processed, there is less competition for machine capacity than at the beginning of the schedule. Thus, based on our economic interpretation, a method which selects multipliers so that $\{\lambda_t\}$ for $t = 1, \ldots, D$ forms a nonincreasing sequence is intuitively appealing.

For a given λ, the lower bound $L(\lambda)$ is obtained by solving the Lagrangean problem. The Lagrangean problem decomposes into n subproblems, so that for each job i ($i = 1, \ldots, n$), a search over all the possible completion times given by (1) enables $L(\lambda)$ to be computed in $O(nD)$ time. The best lower bound that can be obtained from our Lagrangean relaxation approach is $L(\lambda^{\text{OPT}}) = \max_{\lambda \geq 0}\{L(\lambda)\}$. A conventional approach to determine λ^{OPT} is to use subgradient optimization. However, this requires pseudopolynomial time per iteration and, therefore, might entail much computation without guarantee of a tight enough lower bound to restrict substantially the size of the branch and bound search tree. We prefer instead to use a heuristic method to find, in polynomial time, a vector of multipliers λ^* which approximates λ^{OPT}. This polynomial bound on the time complexity cannot be achieved by the explicit specification of all multipliers of λ^*. Thus, our method computes λ_t^* for certain values of t and specifies other multipliers in functional form only. Subsequent analysis shows that the complete lower bounding computation is performed in polynomial time.

3.2. Determination of the multipliers

Our multipliers $\lambda_1^*, \ldots, \lambda_D^*$ are determined from the heuristic schedule generated by the generalized SWPT rule of Conway et al. which, henceforth, is referred to as the SWPT Heuristic. Let S_i^H and C_i^H denote the start and completion time of job i ($i = 1, \ldots, n$) in this heuristic solution. During the application of the SWPT Heuristic, certain unit time intervals are selected as follows. For each t ($t = 1, \ldots, D$), if $S_i^H = t - 1$ for some job i, then we define $[t-1, t]$ as a *changeover interval*. Furthermore, i is the *changeover job* for interval $[t-1, t]$ if $p_i/w_i = \max_{h=1,\ldots,n}\{p_h/w_h \mid S_h^H = t - 1\}$. If it is not already defined by a start time, we create an additional changeover interval $[t-1, t]$, where $t = \lceil \sum_{i=1}^{n} p_i/m \rceil$. We explain later how multipliers are defined in terms of changeover intervals and jobs.

The procedure below provides a formal description of the SWPT Heuristic and the method by which changeover intervals and jobs are determined. We use T_j to denote the total processing time currently assigned to machine j ($j = 1, \ldots, m$), WC to denote the total weighted completion time for jobs which are scheduled, i to specify which job is to be scheduled next, $t_1, \ldots, t_k$ to denote the endpoints of those changeover intervals which are currently known and $i_1, \ldots, i_k$ to denote the corresponding changeover jobs.

SWPT Heuristic.

Step 1. Number jobs in SWPT order so that $p_1/w_1 \leq \cdots \leq p_n/w_n$, set $T_j = 0$ for $j = 1, \ldots, m$, set WC = 0, set $i = 1$ and set $k = 0$.

Step 2. Find a machine j with T_j as small as possible and schedule job i on machine j. Set $S_i^H = T_j$, set $C_i^H = T_j + p_i$, set $T_j = T_j + p_i$ and set WC = WC + $w_i T_j$.

Step 3. If $k = 0$ or if $k > 0$ and $t_k < S_i^H + 1$, set $k = k + 1$, set $t_k = S_i^H + 1$ and set $i_k = i$; otherwise set $i_k = i$. If $i < n$, set $i = i + 1$ and go to Step 2.

Step 4. If $t_k < \lceil \sum_{i=1}^{n} p_i/m \rceil$, set $k = k + 1$ and set $t_k = \lceil \sum_{i=1}^{n} p_i/m \rceil$. Stop with a schedule having been generated which has total weighted completion time WC; k changeover intervals and jobs have also been defined.

Although the interpretation of most statements in this procedure is clear, some explanation of Step 3 is helpful. The first of the two possibilities deals with the case where a new changeover interval is found because no previously scheduled job has start time S_i^H. For the alternative case that a previously scheduled job has start time S_i^H, no new changeover interval is found, but the changeover job now becomes i. Also, we note that if a new changeover interval is defined in Step 4, there is no corresponding changeover job. For $m = 1$, we observe that $k = n$; alternatively, for $m > 1$, since the $\min\{m, n\}$ jobs which are scheduled first by the SWPT Heuristic each have a start time of zero, there are less than n changeover jobs and hence $k \leq n$.

Let us define $\lambda_{t_k}^* = 0$, $t_{k+1} = D$ and $\lambda_{t_{k+1}}^* = 0$. Having applied the SWPT Heuristic, multipliers corresponding to the changeover intervals with endpoints t_l $(l = 1, \ldots, k-1)$ are determined from the backward recursion

$$\lambda_{t_l}^* = \lambda_{t_{l+1}}^* + (t_{l+1} - t_l) w_{i_l}/p_{i_l}. \tag{6}$$

The multiplier for a nonchangeover interval is computed under the assumption that λ_t^* decreases linearly with t between successive changeover intervals. Thus, when $\lambda_{t_1}^*, \ldots, \lambda_{t_{k+1}}^*$ are known, the multiplier for any relevant interval $[t-1, t]$, where $t_l \leq t \leq t_{l+1}$ for some l $(l = 1, \ldots, k)$, is defined by

$$\lambda_t^* = ((t_{l+1} - t)\lambda_{t_l}^* + (t - t_l)\lambda_{t_{l+1}}^*)/(t_{l+1} - t_l). \tag{7}$$

(We note that l always exists because $1 = t_1 < \cdots < t_k \leq t_{k+1} = D$.) It is clear from these definitions that $\lambda_1^* > \cdots > \lambda_{t_k}^* = \lambda_{t_k+1}^* = \cdots = \lambda_D^* = 0$, which is consistent with our economic interpretation that the sequence $\{\lambda_t^*\}$ for $t = 1, \ldots, D$ should be nonincreasing. In our lower bounding scheme, the multipliers $\lambda_{t_1}^*, \ldots, \lambda_{t_{k+1}}^*$ are computed first from the initial conditions and (6). When they are needed in subsequent computations, multipliers for nonchangeover intervals are found from (7).

In Subsection 3.5, a theoretical justification for this choice of multipliers is given. We show that the solution of the Lagrangean problem generated by these multipliers is close to the solution obtained from the SWPT Heuristic and that an exact lower bound is obtained for the case of a single machine and for the case of unit processing times.

3.3. Solution of the Lagrangean problem

In this subsection, we discuss how the Lagrangean problem is solved when the multipliers are determined using (6) and (7). Firstly, using algebraic manipulations of (5),

we express the Lagrangean function as

$$L(\lambda^*) = \min\left\{\sum_{i=1}^{n}\sum_{t=C_i-p_i+1}^{C_i}(w_i t/p_i + \lambda_t^*)\right\} + \sum_{i=1}^{n} w_i(p_i-1)/2 - m\sum_{t=1}^{D}\lambda_t^*.$$

For each job i ($i = 1, \ldots, n$), let us define $c_{it} = w_i t/p_i + \lambda_t^*$ for $t = 1, \ldots, D$. Also, let $K_{it} = \sum_{t'=t-p_i+1}^{t} c_{it'}$ denote the cost in the Lagrangean problem of scheduling job i to be completed at time t ($t = p_i, \ldots, D$). It is apparent that the Lagrangean problem is solved by choosing, for each job i, $C_i \in \{p_i, \ldots, D\}$ such that K_{i,C_i} is minimized. Let $C_i = C_i^*$ define an optimal solution of the Lagrangean problem. To restrict the search for C_i^*, it is helpful to explore some properties of the sequences $\{c_{it}\}$ and $\{K_{it}\}$.

Theorem 4. *For each job i ($i = 1, \ldots, n$), the sequence $\{c_{it}\}$ for $t = 1, \ldots, D$ is nonincreasing for $t \le S_i^H + 1$ and is nondecreasing for $t \ge S_i^H + 1$, where S_i^H is the start time of job i in the SWPT Heuristic.*

Proof. Assume that jobs are numbered in SWPT order. Consider any t ($t = 2, \ldots, t_k$) and suppose that $t_l + 1 \le t \le t_{l+1}$ for some l ($l = 1, \ldots, k-1$). Thus, by the definition of changeover jobs, we deduce that if interval $[t_{l+1} - 1, t_{l+1}]$ has a changeover job i_{l+1}, then

$$S_{i_l}^H + 2 \le t \le S_{i_{l+1}}^H + 1; \tag{8}$$

otherwise, $l = k - 1$ and there is no changeover job corresponding to the interval $[t_k - 1, t_k]$. Using the definition of c_{it} and equations (6) and (7), we obtain

$$c_{it} - c_{i,t-1} = w_i/p_i - (\lambda_{t-1}^* - \lambda_t^*) = w_i/p_i - w_{i_l}/p_{i_l}. \tag{9}$$

We use the nondecreasing property $S_1^H \le \cdots \le S_n^H$ of the start times generated by the SWPT Heuristic to establish the required result.

Firstly, consider the case that $2 \le t \le S_i^H + 1$. If $i < i_l$, then using the nondecreasing start time property we obtain $t \le S_i^H + 1 \le S_{i_l}^H + 1$, which contradicts (8). Thus, $i \ge i_l$. Since jobs are numbered in SWPT order, we use $w_{i_l}/p_{i_l} \ge w_i/p_i$ in (9) to deduce that $c_{i,t-1} \ge c_{it}$. Thus, the sequence $\{c_{it}\}$ is nonincreasing for $t \le S_i^H + 1$.

We now analyze the alternative case that $S_i^H + 2 \le t \le t_k$. Suppose that $i > i_l$. From the nondecreasing start time property, we have $S_i^H \ge S_{i_l}^H$. However, if $S_i^H = S_{i_l}^H$, the SWPT Heuristic would choose i rather than i_l as the changeover job for the interval $[S_{i_l}^H, S_{i_l}^H + 1]$. Thus, $S_i^H > S_{i_l}^H$, and since i_l cannot be the final changeover job, we observe that a changeover job i_{l+1} exists. Furthermore, if $S_{i_l}^H < S_i^H < S_{i_{l+1}}^H$, then there would be a changeover interval between $[S_{i_l}^H, S_{i_l}^H + 1]$ and $[S_{i_{l+1}}^H, S_{i_{l+1}}^H + 1]$ for which S_i^H is the start point. Since this is not the case, we have $S_i^H \ge S_{i_{l+1}}^H$ which, when combined with $t \ge S_i^H + 2$, contradicts (8). Therefore, our supposition that $i > i_l$ is

incorrect, so we have established that $i \leq i_l$. Again, from the SWPT numbering, we use $w_i/p_i \geq w_{i_l}/p_{i_l}$ in (9) to deduce that $c_{it} \geq c_{i,t-1}$ for $t = S_i^H + 2, \ldots, t_k$.

To complete the proof, we analyze the sequence $\{c_{it}\}$ for $t = t_k, \ldots, D$. For $t_k < t \leq D$, we recall that $\lambda_{t-1}^* = \lambda_t^* = 0$ from which we obtain $c_{it} - c_{i,t-1} = w_i/p_i > 0$. We have now established that the sequence $\{c_{it}\}$ is nondecreasing for $t \geq S_i^H + 1$. □

Theorem 5. *For each job i $(i = 1, \ldots, n)$, $C_i^* \in \{C_i^l, \ldots, C_i^u\}$, where $C_i^l = \max\{S_i^H + 1, p_i\}$ and $C_i^u = C_i^H = S_i^H + p_i$. Furthermore, the sequence $\{K_{it}\}$ for $t = p_i, \ldots, D$ is nonincreasing for $t \leq C_i^*$ and is nondecreasing for $t \geq C_i^*$.*

Proof. From the definition of K_{it}, we note that

$$K_{it} - K_{i,t-1} = c_{it} - c_{i,t-p_i} \tag{10}$$

for $t = p_i + 1, \ldots, D$. Applying Theorem 4 to equation (10) shows that the sequence $\{K_{it}\}$ for $t = p_i, \ldots, D$ is nonincreasing for $t \leq S_i^H + 1$ and is nondecreasing for $t \geq S_i^H + p_i = C_i^H$. Thus, $S_i^H + 1 \leq C_i^* \leq C_i^H$. Combining these bounds with the inequality $C_i^* \geq p_i$, which is obtained from (1), yields $C_i^* \in \{C_i^l, \ldots, C_i^u\}$.

To complete the proof, we analyze the sequence $\{K_{it}\}$ for $\max\{S_i^H + 1, p_i\} \leq t \leq S_i^H + p_i$. Firstly, for $\max\{S_i^H + 1, p_i\} < t \leq C_i^*$, we have from Theorem 4 that $c_{i,t-p_i} \geq c_{i,C_i^*-p_i}$, since $C_i^* - p_i \leq C_i^u - p_i < S_i^H + 1$, and $c_{it} \leq c_{i,C_i^*}$. Substituting these inequalities into (10), we obtain

$$K_{it} - K_{i,t-1} \leq c_{i,C_i^*} - c_{i,C_i^*-p_i}. \tag{11}$$

The optimality of the solution C_i^* for the Lagrangean problem provides the inequality $K_{i,C_i^*} \leq K_{i,C_i^*-1}$, which, on substitution in (10), yields $c_{i,C_i^*} \leq c_{i,C_i^*-p_i}$. We deduce from (11), therefore, that $K_{it} \leq K_{i,t-1}$. Thus, the sequence $\{K_{it}\}$ is nonincreasing for $t \leq C_i^*$. Alternatively, for $C_i^* < t \leq S_i^H + p_i$, we use an analogous argument. Substituting the inequalities $c_{i,t-p_i} \leq c_{i,C_i^*-p_i+1}$ and $c_{it} \geq c_{i,C_i^*+1}$, which are deduced from Theorem 4, into (10) yields

$$K_{it} - K_{i,t-1} \geq c_{i,C_i^*+1} - c_{i,C_i^*-p_i+1}. \tag{12}$$

Since C_i^* is an optimal solution for the Lagrangean problem, we obtain the inequality $K_{i,C_i^*+1} \geq K_{i,C_i^*}$, which, on substitution in (10), yields $c_{i,C_i^*+1} \geq c_{i,C_i^*-p_i+1}$. Hence, inequality (12) implies that $K_{it} \geq K_{i,t-1}$. We conclude that the sequence $\{K_{it}\}$ is nondecreasing for $t \geq C_i^*$. □

Having obtained λ^*, it is straightforward to derive from Theorem 5 a *bisection search procedure* to find C_i^* $(i = 1, \ldots, n)$. Our procedure initially uses the lower and upper limits C_i^l and C_i^u which are defined in Theorem 5. If $C_i^l \neq C_i^u$, it computes $t = \lceil (C_i^l + C_i^u)/2 \rceil$ and evaluates $c_{i,t-p_i}$ and c_{it}. Firstly, if $c_{i,t-p_i} < c_{it}$, then Theorem 5 and equation (10) show that $t - 1$ is a valid upper bound on C_i^*; thus, our procedure

sets $C_i^u = t - 1$. Secondly, if $c_{i,t-p_i} > c_{it}$, then the current lower bound on C_i^* is updated using $C_i^l = t$; again, this is justified by Theorem 5 and equation (10). Finally, if $c_{i,t-p_i} = c_{it}$, then a straightforward extension of the analysis in the proof of Theorem 5 shows that $C_i^* = t$ and $C_i^* = t - 1$ each define a solution of the Lagrangean problem. Thus, our procedure terminates with $C_i^* = t$ if $c_{i,t-p_i} = c_{it}$. Unless $c_{i,t-p_i} = c_{it}$, the procedure continues until $C_i^l = C_i^u$, at which stage it is clear that $C_i^* = C_i^l = C_i^u$.

3.4. Implementation of the lower bound

In this subsection, we show that, using a suitable implementation, the lower bound $L(\lambda^*)$ is computed in polynomial time. As pointed out in Subsection 3.1, an explicit formulation of the Lagrangean problem requires the computation of λ_t^* for $t = 1, \ldots, D$, the time complexity of which is $O(D)$. To avoid this pseudopolynomial time construction, at most n multipliers are computed explicitly using (6), whereas the others are implicit in (7).

Our lower bounding procedure first applies the SWPT Heuristic and, having found changeover intervals and changeover jobs, sets $\lambda_{t_k}^* = 0$, $t_{k+1} = D$ and $\lambda_{t_{k+1}}^* = 0$, and uses (6) to compute $\lambda_{t_l}^*$ for $l = 1, \ldots, k-1$. The SWPT Heuristic requires $O(n \log n)$ time, whereas the application of (6) requires $O(n)$ time, since $k \leq n$.

The next stage in the computation of $L(\lambda^*)$ is to apply the bisection search procedure of the previous subsection to find C_i^* for $i = 1, \ldots, n$. Since Theorem 5 shows that $C_i^u - C_i^l < p_i$, at most $\lceil \log_2 p_i \rceil$ bisection search iterations are necessary. For each iteration, it is necessary to compute $\lambda_{t-p_i}^*$ and λ_t^* so that $c_{i,t-p_i}$ and c_{it} can be evaluated. A prerequisite for using (7) to find λ_t^* is a value l, where $1 \leq l \leq k$, such that $t_l \leq t \leq t_{l+1}$. A bisection search determines l in $O(\log n)$ time, since $k \leq n$. Thus, c_{it}, and in a similar way $c_{i,t-p_i}$, is found in $O(\log n)$ time. It is now apparent that the complete bisection search procedure requires $O(\log n \sum_{i=1}^{n} \log p_i)$ time.

Having found C_i^* for $i = 1, \ldots, n$, our lower bound $L(\lambda^*)$ is evaluated using (5). We now describe an efficient method to compute $\sum_{t=C_i^*-p_i+1}^{C_i^*} \lambda_t^*$ for $i = 1, \ldots, n$. Firstly, an $O(\log n)$ bisection search is used to find l' and l'', where $1 \leq l' \leq l'' \leq k$, such that $t_{l'} < C_i^* - p_i + 1 \leq t_{l'+1}$ and $t_{l''} < C_i^* \leq t_{l''+1}$. Then, we use the following expressions which are derived from (7). If $l' = l''$, then

$$\sum_{t=C_i^*-p_i+1}^{C_i^*} \lambda_t^* = p_i((t_{l'+1} - C_i^* + (p_i - 1)/2)\lambda_{t_{l'}}^* + (C_i^* - t_{l'} - (p_i - 1)/2)\lambda_{t_{l'+1}}^*)/(t_{l'+1} - t_{l'}).$$

Alternatively, if $l' < l''$, then

$$\sum_{t=C_i^*-p_i+1}^{C_i^*} \lambda_t^* = \sum_{t=C_i^*-p_i+1}^{t_{l'+1}} \lambda_t^* + \sum_{l=l'+1}^{l''-1} \sum_{t=t_l+1}^{t_{l+1}} \lambda_t^* + \sum_{t=t_{l''}+1}^{C_i^*} \lambda_t^*,$$

where

$$\sum_{t=C_i^*-p_i+1}^{t_{l'+1}} \lambda_t^* = (t_{l'+1} - C_i^* + p_i)((t_{l'+1} - C_i^* + p_i - 1)\lambda_{t_{l'}} + (C_i^* - p_i + 1 + t_{l'+1} - 2t_{l'})\lambda^*_{t_{l'}+1})/(2(t_{l'+1} - t_{l'})),$$

$$\sum_{l=l'+1}^{l''-1} \sum_{t=t_l+1}^{t_{l+1}} \lambda_t^* = (t_{l'+2} - t_{l'+1} - 1)\lambda_{t_{l'+1}}/2 + \sum_{l=l'+2}^{l''-1} (t_{l+1} - t_{l-1})\lambda_{t_l}/2 + (t_{l''} - t_{l''-1} + 1)\lambda_{t_{l''}}/2,$$

$$\sum_{t=t_{l''}+1}^{C_i^*} \lambda_t^* = (C_i^* - t_{l''})((2t_{l''+1} - t_{l''} - C_i^* - 1)\lambda_{t_{l''}} + (C_i^* + 1 - t_{l''})\lambda_{t_{l''}+1})/(2(t_{l''+1} - t_{l''})).$$

Thus, $\sum_{t=C_i^*-p_i+1}^{C_i^*} \lambda_t^*$ is computed in $O(n)$ time. We also deduce from (7) that $\sum_{t=1}^{D} \lambda_t^*$ is computed in $O(n)$ time using

$$\sum_{t=1}^{D} \lambda_t^* = (t_2 - t_1 + 1)\lambda_{t_1}/2 + \sum_{l=2}^{k-1} (t_{l+1} - t_{l-1})\lambda_{t_l}/2.$$

It is now evident that, having computed C_i^* for $i = 1, \ldots, n$, we obtain $L(\lambda^*)$ in $O(n^2)$ time.

The analysis of this section is summarized in the following result.

Theorem 6. *The computation of* $L(\lambda^*)$ *requires* $O(n^2 + \log n \sum_{i=1}^{n} \log p_i)$ *time.*

3.5. Analysis of the lower bound

In this subsection, we justify the use of multipliers λ^* by first showing that an exact lower bound is generated for the case of a single machine and for the case of unit processing times. We also aim to establish the maximum deviation of the lower bound from the optimal solution value.

Theorem 7. *If* $m = 1$, *then* $L(\lambda^*) = \mathrm{WC}$, *where* $\mathrm{WC} = \sum_{i=1}^{n} w_i C_i^H$ *is the total weighted completion time of the SWPT sequence* $(1, \ldots, n)$.

Proof. Firstly, we show that there exists a solution to the Lagrangean problem for which $C_i^* = C_i^H$ $(i = 1, \ldots, n)$. Using Theorem 4, it is sufficient to show that each of the values c_{it} for $t = S_i^H + 1, \ldots, C_i^H$ are identical. The endpoints of the changeover intervals found in Step 3 of the SWPT Heuristic are $S_1^H + 1, \ldots, S_n^H + 1$, and the corresponding changeover jobs are $1, \ldots, n$ respectively. Noting that $S_i^H = C_{i-1}^H$ for $i = 2, \ldots, n$, we deduce from equations (6) and (7) that $\lambda_{t-1}^* - \lambda_t^* = w_i/p_i$ for

$t = S_i^H + 2, \ldots, C_i^H$. Thus, $c_{it} - c_{i,t-1} = 0$ for $t = S_i^H + 2, \ldots, C_i^H$, which establishes that $C_i^* = C_i^H$ $(i = 1, \ldots, n)$. We observe that $\sum_{i=1}^{n} \sum_{t=C_i^H - p_i + 1}^{C_i^H} \lambda_t^* = \sum_{t=1}^{D} \lambda_t^*$ since $D = C_n^H$. Substituting in (5), we obtain $L(\lambda^*) = \sum_{i=1}^{n} w_i C_i^H = \text{WC}$, as required. □

Theorem 8. *If* $p_i = 1$ *for* $i = 1, \ldots, n$, *then* $L(\lambda^*) = \text{WC}$, *where* $\text{WC} = \sum_{i=1}^{n} w_i C_i^H$.

Proof. Theorem 5 shows that $C_i^* = C_i^H$ $(i = 1, \ldots, n)$. Since the SWPT Heuristic generates a schedule in which m unit time jobs are scheduled in each of the intervals $[t-1, t]$ for $t = 1, \ldots, t_k - 1$ and since $\lambda_t^* = 0$ for $t \geq t_k$, we deduce that $\sum_{i=1}^{n} \sum_{t=C_i^H - p_i + 1}^{C_i^H} \lambda_t^* = m \sum_{t=1}^{D} \lambda_t^*$. The required result that $L(\lambda^*) = \sum_{i=1}^{n} w_i C_i^H = \text{WC}$ is obtained by substitution in (5). □

Theorem 7 shows that, as is the case for the lower bound of Eastman et al., our proposed bound is exact when there is a single machine. For the problem of scheduling jobs with unit processing times on m machines, however, our proposed lower bound is guaranteed from Theorem 8 to be exact, whereas the Eastman et al. bound may deviate from the optimal solution value. Even though we have no proof of uniform superiority, our proposed lower bound is tighter than that of Eastman et al. for at least one class of problems.

It is now convenient to extend our problem definition to allow jobs to have zero weight. In an SWPT order, it is assumed that any jobs with zero weight appear after all jobs with positive weight. We now establish the maximum deviation from the optimal solution value of the lower bound $L(\lambda^*)$, under the assumption that at least one job has zero weight. It is apparent that a dummy job with zero weight can be added to the problem, if necessary, without affecting how other jobs are scheduled. Such a dummy job may, however, influence λ^*.

Theorem 9. *If jobs are numbered in SWPT order and* $w_n = 0$, *then* $\text{WC} - L(\lambda^*) \leq \sum_{i=m+1}^{n} w_i (p_i - 1)$, *where* $\text{WC} = \sum_{i=1}^{n} w_i C_i^H$.

Proof. From (5) we have

$$L(\lambda^*) = \sum_{i=1}^{n} \left(w_i C_i^* + \sum_{t=C_i^* - p_i + 1}^{C_i^*} \lambda_t^* \right) - m \sum_{t=1}^{D} \lambda_t^*. \tag{13}$$

We now derive a lower bound on $L(\lambda^*)$ by using suitable bounds on C_i^* and λ_t^* in (13). Firstly, applying Theorem 5, we obtain $C_i^* = C_i^H = p_i$ for $i = 1, \ldots, \min\{m, n\}$ and $C_i^* \geq C_i^H - p_i + 1$ for $i = m + 1, \ldots, n$. Secondly, since $C_i^* \leq C_i^H$ from Theorem 5 and the sequence $\{\lambda_t^*\}$ for $t = 1, \ldots, D$ is nonincreasing, we have $\lambda_t^* \geq \lambda_{t'}^*$ for $t = C_i^* - p_i + 1, \ldots, C_i^*$ and $t' = t + C_i^H - C_i^*$ $(i = 1, \ldots, n)$. Substituting in (13) yields

$$L(\lambda^*) \geq \sum_{i=1}^{n} \left(w_i C_i^H + \sum_{t=C_i^H - p_i + 1}^{C_i^H} \lambda_t^* \right) - \sum_{i=m+1}^{n} w_i (p_i - 1) - m \sum_{t=1}^{D} \lambda_t^*. \tag{14}$$

In the schedule given by the SWPT Heuristic, there is no idle time on any machine during a time interval $[t-1, t]$, where $t = 1, \ldots, S_n^H$. Also, $[S_n^H, S_n^H + 1]$ is the final changeover interval defined in Step 3 of the SWPT Heuristic and n is the corresponding changeover job. Since $w_n = 0$, we have from (6) and (7) that $\lambda_t^* = 0$ for $t = S_n^H + 1, \ldots, D$. Thus,

$$\sum_{i=1}^{n} \sum_{t=C_i^H - p_i + 1}^{C_i^H} \lambda_t^* = m \sum_{t=1}^{D} \lambda_t^*. \tag{15}$$

From (14) and (15), we obtain

$$L(\lambda^*) \geq \sum_{i=1}^{n} w_i C_i^H - \sum_{i=m+1}^{n} w_i(p_i - 1) = \mathrm{WC} - \sum_{i=m+1}^{n} w_i(p_i - 1),$$

which is the required inequality. □

Our initial experiments show that better results are obtained if a dummy job with zero weight is not included. Nevertheless, it is satisfying to have the performance guarantee given by Theorem 9 for this type of lower bound. As a by-product of this analysis, we obtain a bound on the worst-case performance of the SWPT Heuristic. If WC* denotes the total weighted completion time for an optimal schedule, then we deduce from Theorem 9 that $\mathrm{WC} - \mathrm{WC}^* \leq \sum_{i=m+1}^{n} w_i(p_i - 1)$ if jobs are numbered in SWPT order.

3.6. *A numerical example*

In this subsection, we illustrate the computation of the lower bound $L(\lambda^*)$ with a numerical example. There are three machines and six jobs. Processing times and weights are given in Table 1. The SWPT Heuristic schedules jobs 1 and 4 on machine 1, jobs 2 and 5 on machine 2 and jobs 3 and 6 on machine 3 to give $S_1^H = 0$, $S_2^H = 0$, $S_3^H = 0$, $S_4^H = 3$, $S_5^H = 3$, $S_6^H = 4$ and WC = 1225. Also, the SWPT Heuristic detects four changeover intervals which are defined by $t_1 = 1$, $t_2 = 4$, $t_3 = 5$ and $t_4 = 7$; changeover jobs are $i_1 = 3$, $i_2 = 5$ and $i_3 = 6$ (there is no changeover job corresponding to t_4). Also, $D = 10$, while (6) yields multipliers $\lambda_{10}^* = 0$, $\lambda_7^* = 0$, $\lambda_5^* = 2$, $\lambda_4^* = 7$ and $\lambda_1^* = 67$. Values of the other multipliers obtained from (7), together with all values of c_{it}, are shown in Table 2 (although not all of these values are computed explicitly if the

Table 1
Data for the example

i	1	2	3	4	5	6
p_i	3	3	4	5	2	4
w_i	75	66	80	50	10	4

Table 2
Values of λ_t^* and c_{it}

t	1	2	3	4	5	6	7	8	9	10
λ_t^*	67	47	27	7	2	1	0	0	0	0
c_{1t}	92	97	102	107	127	151	175	200	225	250
c_{2t}	89	91	93	95	112	133	154	176	198	220
c_{3t}	87	87	87	87	102	121	140	160	180	200
c_{4t}	77	67	57	47	52	61	70	80	90	100
c_{5t}	72	57	42	27	27	31	35	40	45	50
c_{6t}	68	49	30	11	7	7	7	8	9	10

implementation of Subsection 3.4 is adopted). Application of the bisection search procedure using the values in Table 2 yields $C_1^* = 3, C_2^* = 3, C_3^* = 4, C_4^* = 6, C_5^* = 5, C_6^* = 8$, which gives $L(\lambda^*) = 1198$. We note that, for this example, the SWPT Heuristic generates an optimal schedule.

4. The branch and bound algorithm

In this section, we give a complete description of our branch and bound algorithm. Special attention is given to a description of our branching rule and to the implementation of various dominance rules which are used to eliminate nodes of the search tree.

We use a *forward scheduling branching rule* in which each node of the search tree corresponds to an initial partial sequence of jobs on each machine. More precisely, the current initial partial schedule is examined and the machine j which becomes available earliest is selected; ties are broken by choosing j as small as possible. Search tree nodes are constructed which correspond to appending unscheduled jobs to the current partial sequence on machine j. Theorems 1 and 2 show that we need not consider schedules which have no job appearing next on machine j.

We now explain how dominance rules are used to eliminate nodes of the search tree. Assume that jobs are numbered in SWPT order so that $p_1/w_1 \leq \cdots \leq p_n/w_n$. Our *first elimination test* discards a node of the search tree if job i is scheduled first on machine j_1 and job h is scheduled first on machine j_2, where $h < i$ and $j_1 < j_2$. We justify the elimination of this node by noting that an equivalent partial schedule is obtained by interchanging the partial sequences on machines j_1 and j_2; clearly, it is sufficient to retain just one of these partial schedules. A consequence of this first test is that job 1 must be sequenced first on machine 1 in an optimal schedule: job 1 is assumed to be sequenced first on some machine according to Theorem 1 and our test eliminates the possibility that it is sequenced first on any other machine.

Our *second elimination test* is a straightforward application of Theorem 1. If some job i appears out of SWPT order when it is appended to the partial sequence on machine j, then the corresponding search tree node is discarded.

We use Theorem 3 in our *third elimination test*. If h and i are unscheduled jobs, where $p_h \leq p_i$, $w_h \geq w_i$ and $h < i$, then a node which corresponds to scheduling job i on machine j, but leaving h unscheduled, is discarded. To justify the elimination of this node, we note that it must lead to a schedule in which $S_i \leq S_h$. If $S_i < S_h$, then Theorem 3 shows that the schedule is dominated. On the other hand, if $S_i = S_h$ and h is scheduled on some machine j' $(j' \neq j)$, then interchanging the final partial sequences from jobs h and i onwards on machines j and j' respectively yields a schedule with the same total weighted completion time. Clearly, it is not necessary to consider both of these schedules which have the same total weighted completion time: the one which assigns job i to machine j is discarded.

Our *fourth elimination test* is also derived from Theorem 3. Let $h_1, \ldots, h_r$ be jobs which are sequenced in adjacent positions on some machine and let $i_1, \ldots, i_s$ be a group of adjacent jobs on another machine in the partial schedule at some node of the search tree. Using Lawler's theory [14], a group of consecutively sequenced jobs can be treated as a composite job. In our case, these composite jobs h and i have processing times and weights given by $p_h = p_{h_1} + \cdots + p_{h_r}$, $w_h = w_{h_1} + \cdots + w_{h_r}$, $p_i = p_{i_1} + \cdots + p_{i_s}$ and $w_i = w_{i_1} + \cdots + w_{i_s}$. If Theorem 3 is applied to composite jobs h and i, then the search tree node is eliminated if $p_h \leq p_i$, $w_h \geq w_i$ and $S_{h_1} > S_{i_1}$. Our implementation of this fourth test is as follows. Based on the results of initial computational experiments, we perform elimination tests with composite jobs h and i which contain at most three of the last jobs on a machine: the investment in computation time is too great if larger composite jobs are considered. Suppose that the current search tree node is created by scheduling a job on machine j. For each machine j' $(j' \neq j)$, attempts are made to eliminate this current node, firstly by selecting h and i to be the composite jobs formed from the last jobs on machines j and j' respectively, and secondly by forming composite jobs h and i from the last jobs on machines j' and j respectively. Thus, for each of the $m - 1$ possible choices of machine j', tests are undertaken by forming composite jobs h and i for each value $r \in \{1, 2, 3\}$ and $s \in \{1, 2, 3\}$ (where r and s define the number of jobs in h and i respectively), although no test is performed when $r = s = 1$ since our third elimination test deals with this case. When $p_h = p_i$ and $w_h = w_i$ in this fourth test, care must be taken that the relevant pair of partial schedules are not used to dominate each other, thereby risking the elimination of all nodes which lead to an optimal schedule.

Whenever a node is not eliminated by one of these tests, a lower bound is computed. If $L(\lambda^*)$ is used, then the SWPT Heuristic is applied at each node of the search tree to generate an upper bound. However, the SWPT Heuristic is applied only at the root node if the lower bounding scheme of Eastman et al. replaces $L(\lambda^*)$. A *newest active node search* selects a node from which to branch.

Having described our algorithm, a discussion of its relationship with other branch and bound algorithms is appropriate. Our branching rule is, essentially, the same as

that used by Elmaghraby and Park, and Barnes and Brennan. Sarin et al., however, claim that a more efficient rule is to select an unscheduled job i, with p_i/w_i as small as possible, and create m branches which correspond to assigning job i to each machine. Sarin et al. argue that there is redundancy in the rule which assigns jobs to a specified position in a partial sequence, i.e., some schedules are generated more than once. We counter this by claiming that redundancy is removed through the use of our first two elimination tests. Thus, the performance of the algorithm described above, which uses the first three elimination tests and the lower bounding scheme of Eastman et al., is expected to closely match that of Sarin et al.

5. Computational experience

Our aim in this section is to evaluate the proposed branch and bound algorithm. In particular, the performance of our Lagrangean based lower bound $L(\lambda^*)$ relative to that of Eastman et al. is of interest. Also, the effectiveness of using composite jobs in Theorem 3 to eliminate search tree nodes (our fourth elimination test), which is not attempted in previous algorithms, should be assessed. We compare algorithms A(EEI, ET3), A(LAG, ET3), A(EEI, ET4) and A(LAG, ET4), where the first parameter indicates which lower bound is used and the second parameter indicates which elimination tests are applied. Thus, EEI indicates that the bound of Eastman, Even and Isaacs (as defined by equation (3) of [19]) is used, whereas LAG refers to our Lagrangean based bound; ET3 indicates that only the first three elimination tests are used, whereas the fourth test is additionally applied under ET4.

Test problems having 2, 3, 4, 5 and 8 machines and up to 40 jobs were generated as follows. For each job i, an integer processing time p_i from the uniform distribution [1, 10] and an integer weight w_i from the uniform distribution [1, 100] were generated. For each selected pair of values of m and n, 20 problems were generated.

The algorithms were coded in FORTRAN 77 and run on a CDC 7600 computer. Whenever a problem was not solved within the time limit of 60 seconds, computation was abandoned for that problem. Computational results are given in Table 3. For each algorithm, average computation times in seconds (with unsolved problems contributing 60 seconds in the computation of the average), numbers of unsolved problems and average numbers of search tree nodes (with the number of nodes for unsolved problems at the time of abandonment used in the computation of the average) are listed. Averages are not given, however, when all problems are unsolved.

Our first observation from Table 3 is that, except for some of the small problems where A(EEI, ET3) gives the lowest computation times, algorithm A(LAG, ET4) is the most efficient in terms of average computation times, numbers of unsolved problems and average number of search tree nodes. A more detailed analysis of how our Lagrangean based lower bound and the fourth elimination test individually affect efficiency is given below.

Table 3
Computational results (ACT: average computation time in seconds, NU: number of unsolved problems, if any, ANN: average number of search tree nodes)

		A(EEI, ET3)		A(LAG, ET3)		A(EEI, ET4)		A(LAG, ET4)	
m	n	ACT:NU	ANN	ACT:NU	ANN	ACT:NU	ANN	ACT:NU	ANN
2	15	0.13	1934	0.14	173	0.19	1139	0.14	149
2	20	0.55	7557	0.45	436	0.60	3219	0.41	349
2	30	33.14: 4	389620	7.90	4511	22.57: 1	102899	5.60	2989
2	40	–:20	–	43.44: 9	21624	–:20	–	33.53: 5	15685
3	15	0.60	8840	0.51	723	0.91	4409	0.50	587
3	20	6.91	95116	3.46	3530	6.58	27459	2.88	2475
3	30	–:20	–	48.95:12	37499	–:20	–	42.21: 9	28129
4	15	0.73	10840	0.89	1502	1.38	6347	0.93	1247
4	20	12.00: 2	162216	8.63	9516	12.00: 1	42577	7.34	6461
4	30	–:20	–	58.30:19	50011	–:20	–	57.42:17	38994
5	15	0.51	7438	0.85	1441	1.17	5674	0.93	1317
5	20	10.63: 1	141893	10.23	11315	12.69: 1	43653	9.16: 1	8310
8	15	0.06	716	0.27	438	0.14	703	0.31	430
8	20	1.11	12681	2.96	3449	2.59	10536	3.20	3180
8	25	47.28:11	506204	50.31:13	48527	53.80:13	155416	49.26:12	38324

We discuss now the effect of replacing the bound EEI of Eastman et al. by our Lagrangean based bound LAG. The reduction in the size of the search tree is striking. For instance, there is a 97% reduction in size when LAG replaces EEI for $m = 2$ and $n = 30$. Even though the computation time required to compute LAG is much larger than that for EEI, computation times for algorithms A(LAG, ET3) and A(LAG, ET4) are significantly smaller than their counterparts in which EEI replaces LAG, except for some of the smaller problems for which the elimination tests by themselves are sufficient to restrict the search. The replacement of EEI by LAG also produces a significant reduction in numbers of unsolved problems.

Comparing computation times for algorithms A(EEI, ET3) and A(EEI, ET4), it is not clear that there is any advantage in applying the fourth elimination test. Although this extra test is effective in reducing search tree size, its computational requirements are large compared with those of the lower bound EEI. On the other hand, computation times for A(LAG, ET4) are generally smaller than those for A(LAG, ET3). In this case, the elimination of a node generates a much larger saving due to the more substantial computational requirements of LAG.

As is expected for $n \geq m$, problems become much harder to solve as n increases. For fixed n, problems are relatively easy for $m = 2$, but become harder as m increases until a stage is reached when m and n are sufficiently close that they start to become easier (the case $m = n$ is trivial).

As observed above, the best of the previously published algorithms is that of Sarin et al. and its performance is likely to be similar to that of A(EEI, ET3). Computational

results show that our algorithm A(LAG, ET4) is superior. In spite of its superiority, A(LAG, ET4) experiences difficulty in solving 30 job problems (except for $m = 2$). A partial explanation is that there appear to be many near-optimal solutions for the majority of problems.

6. Concluding remarks

Using Lagrangean relaxation, we have developed a new lower bound for the problem of scheduling jobs on identical parallel machines to minimize total weighted completion time. In common with lower bounds for other scheduling problems [11, 15, 16, 17], multipliers are found from a simple construction rather than using the computationally expensive subgradient optimization technique. Our lower bound is used in a branch and bound algorithm which, according to the results of extensive computational tests, is superior to previous algorithms. A contributory factor to the success of the algorithm is the use of a new dominance rule (the fourth elimination test).

Computational results indicate that there is scope for further research into the development of an algorithm which can successfully solve problems with 30 or more jobs. Possibly, an approach which uses adjustments to the multipliers given by our approach may be fruitful, although pseudopolynomial time may be required if a very tight lower bound is to be obtained from this relaxation.

Acknowledgement

The research by the first author was supported by a grant from the Algerian government.

References

[1] K.R. Baker and A.G. Merten, Scheduling with parallel processors and linear delay costs, Naval Res. Logist. Quart. 20 (1973) 793–804.

[2] J.W. Barnes and J.J. Brennan, An improved algorithm for scheduling jobs on identical machines, AIIE Trans. 9 (1977) 25–31.

[3] J. Bruno, E.G. Coffman Jr and R. Sethi, Scheduling independent tasks to reduce mean finishing time, Comm. ACM 17 (1974) 382–387.

[4] N. Christofides, R. Alvarez-Valdes and J.M. Tamarit, Project scheduling with resource constraints: a branch and bound approach, European J. Oper. Res. 29 (1987) 262–273.

[5] R.W. Conway, W.L. Maxwell and L.W. Miller, Theory of Scheduling (Addison-Wesley, Reading, MA, 1967).

[6] W.L. Eastman, S. Even and I.M. Isaacs, Bounds for the optimal scheduling of n jobs on m processors, Management Sci. 11 (1964) 268–279.

[7] S.E. Elmaghraby and S.H. Park, Scheduling jobs on a number of identical machines, AIIE Trans. 6 (1974) 1–12.

[8] M.L. Fisher, A dual algorithm for the one-machine scheduling problem, Math. Programming 11 (1976) 229–251.
[9] M.L. Fisher, The Lagrangian relaxation method for integer programming problems, Management Sci. 27 (1981) 1–18.
[10] A.M. Geoffrion, Lagrangean relaxation for integer programming, Math. Programming Stud. 2 (1974) 82–114.
[11] A.M.A. Hariri and C.N. Potts, An algorithm for single machine sequencing with release dates to minimize total weighted completion time, Discrete Appl. Math. 5 (1983) 99–109.
[12] M. Held, P. Wolfe and H.P. Crowder, Validation of subgradient optimization, Math. Programming 6 (1974) 62–88.
[13] T. Kawaguchi and S. Kyan, Worst case bound of an LRF schedule for the mean weighted flow-time problem, SIAM J. Comput. 15 (1986) 1119–1129.
[14] E.L. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, in: Annals of Discrete Mathematics 2 (North-Holland, Amsterdam, 1978) 75–90.
[15] C.N. Potts, A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time, Management Sci. 31 (1985) 1300–1311.
[16] C.N. Potts and L.N. Van Wassenhove, An algorithm for single machine sequencing with deadlines to minimize total weighted completion time, European J. Oper. Res. 12 (1983) 379–387.
[17] C.N. Potts and L.N. Van Wassenhove, A branch and bound algorithm for the total weighted tardiness problem, Oper. Res. 33 (1985) 363–377.
[18] S. Sahni, Algorithms for scheduling independent tasks, J. ACM 23 (1976) 116–127.
[19] S.C. Sarin, S. Ahn and A.B. Bishop, An improved branching scheme for the branch and bound procedure of scheduling n jobs on m parallel machines to minimize total weighted flowtime, Internat. J. Production Res. 26 (1988) 1183–1191.
[20] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logist. Quart. 3 (1956) 59–66.