Annals of Operations Research 26(1990)257–268

MINIMIZING THE SUM OF THE JOB COMPLETION TIMES IN THE TWO-MACHINE FLOW SHOP BY LAGRANGIAN RELAXATION

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

Abstract

A branch-and-bound algorithm is presented for the two-machine flow shop problem with the objective of minimizing the sum of the job completion times. Lower bounds and precedence constraints result from a Lagrangian relaxation of this problem. The Lagrangian subproblem turns out to be a linear ordering problem that is polynomially solvable for appropriate choices of the Lagrangian multipliers. The best choice within this class yields a lower bound that dominates previous bounds. In fact, the existing bounds correspond to particular choices of the multipliers. Several dominance criteria are given to restrict the search tree. Computational experiments show that the proposed algorithm outperforms the previously best method.

Keywords

Flow shop scheduling, Lagrangian relaxation, linear ordering problem, branch-and-bound, computational experience.

1. Introduction

An m-machine flow shop is described as follows. There are m machines, each of which can handle at most one job at a time. There are n independent jobs, each consisting of a chain of m operations. The hth operation of job i has to be scheduled on machine h during a positive uninterrupted processing time ($h = 1, \ldots, m; i = 1, \ldots, n$). Note that the jobs pass through the machines in the same order. A *schedule* defines a job order for each machine.

The bulk of flow shop research in the last decades has been focused on the minimization of the maximum of the job completion times, i.e. the length or makespan of a schedule. However, Gupta and Dudek [9] pleaded that criteria in which the costs of each job are reflected have a better economic interpretation than the makespan objective has.

This paper deals with the minimization of the sum of the job completion times in a two-machine flow shop. Following the notation of Graham et al. [8], we refer to this problem as $F2||\sum C_i$. It is well known that for this problem it suffices to optimize over all permutation schedules (Conway et al. [4]). A *permutation schedule* is a schedule in which every machine has the same job sequence. Ignall and Schrage [11]

were the first to study this problem. They presented a branch-and-bound scheme, based on two lower bounds. The heuristics presented by Krone and Steiglitz [14] were applied by Kohler and Steiglitz [13] in further developing and testing the Ignall and Schrage algorithm. Garey et al. [7] proved the problem to be NP-hard in the strong sense.

Szwarc [19] developed some properties for $F \| \sum C_i$ and identified a class of well-solvable cases. A more elaborate treatment of well-solvable cases can be found in Adiri and Amit [1]. Bansal [2] extended the lower bounds proposed by Ignall and Schrage to the m-machine case.

We will develop a branch-and-bound procedure that uses lower bounds obtained with Lagrangian relaxation techniques. Although the concept of Lagrangian relaxation has shown its merits for many types of combinatorial optimization problems (see Fisher [5] for a survey), its use in scheduling theory, outside the area of single-machine problems with minsum criteria, is limited. Scheduling problems dealing with multiple machines, and especially flow shop and job shop problems, seldom give way to promising relaxations. Fisher et al. [6] confirm this observation in their (computationally unsuccessful) attempt to apply the related technique of *surrogate relaxation* to the notorious job shop scheduling problem. A notable exception is the paper by Hariri and Potts [10] for the two-machine flow shop problem with the objective of minimizing makespan subject to precedence constraints.

The organization of this paper is as follows. In section 2, we give a formulation of $F \| \sum C_i$, followed by a relaxation. The resulting subproblem is a linear ordering problem that, although it is known to be NP-hard, is efficiently solvable for some special situations. There appears to be a class of Lagrangian multipliers that converts the subproblems into polynomially solvable linear ordering problems. The two Ignall and Schrage lower bounds correspond to two particular choices within this class. It is shown how the new lower bounds can be strengthened, and the last subsection is concerned with the derivation of precedence constraints between jobs. Section 3 presents some dominance criteria to restrict the search tree. In section 4, there is a complete description of the algorithm and a presentation of some computational results. Section 5 concludes with a few remarks and some directions for possible extensions.

2. The relaxation

Let p_{hi} denote the processing time of job i ($i = 1, \ldots, n$) on machine h ($h = 1, 2$). The problem of minimizing the sum of the job completion times in a two-machine flow shop, referred to as problem (P), can then be formulated as follows: determine completion times C_{hi} ($h = 1, 2$; $i = 1, \ldots, n$) that minimize

$$\sum_{i=1}^{n} C_{2i} \qquad \text{(P)}$$

subject to

the precedence constraints between the operations of job i ($i = 1, \ldots, n$), (1)

the capacity constraints of machine h ($h = 1, 2$). (2)

Conditions (1) are formulated as

$$C_{2i} \geq C_{1i} + p_{2i} \qquad (i = 1, \ldots, n),$$

while in the sequel of this paper, condition (2) is assumed to be implicitly present.

A vector of multipliers $\lambda = (\lambda_1, \ldots, \lambda_n)$ is introduced for dualizing conditions (1). Lagrangian relaxation of those constraints yields the Lagrangian problem (LR):

$$L(\lambda) = \min \sum_{i=1}^{n} (\lambda_i C_{1i} + (1 - \lambda_i) C_{2i} + \lambda_i p_{2i}). \qquad \text{(LR)}$$

From standard Lagrangian theory [5], it is known that for any given $\lambda \geq 0$, the value $L(\lambda)$ provides a lower bound to (P). In order to prevent that $L(\lambda)$ becomes arbitrarily small, we require $\lambda \leq 1$.

In the Lagrangian problem, the operations of a job can be processed simultaneously. Hence, the primal problem decomposes into two single-machine problems that are easily solved by Smith's [18] shortest weighted processing time rule. In concreto, this implies that jobs are scheduled on machine 1 and machine 2 in order of non-increasing ratios λ_i/p_{1i} and $(1 - \lambda_i)/p_{2i}$, respectively.

However, the gist of our approach lies in imposing the restriction that (LR) is solved over all permutation schedules. This is a redundant condition for the primal problem, but it may increase the value $L(\lambda)$. We will choose the multiplier vector λ in such a way that solving (LR) over all permutation schedules can still be accomplished in polynomial time.

To this end, we will first reformulate the problem of solving (LR) for a given λ over all permutation schedules as a *linear ordering problem.* The linear ordering problem is the following: given an $n \times n$ matrix $A = (a_{ij})$ of weights, find a permutation σ of $\{1, \ldots, n\}$ that maximizes the sum

$$\sum_{(i,j):\, \sigma(i) < \sigma(j)} a_{ij},$$

where $\sigma(i)$ denotes the position of element i in the sequence σ. In our application, we identify $\sigma(i)$ with the position of job i. Since in problem (LR) we have

$$C_{hi} = \sum_{j:\, \sigma(j) \leq \sigma(i)} p_{hj}, \qquad (3)$$

it follows that

$$\sum_{i=1}^{n}(\lambda_i C_{1i} + (1-\lambda_i)C_{2i}) = \sum_{i=1}^{n}\left(\lambda_i \sum_{j:\sigma(j)\le\sigma(i)} p_{1j}\right) + \sum_{i=1}^{n}\left((1-\lambda_i) \sum_{j:\sigma(j)\le\sigma(i)} p_{2j}\right)$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i p_{1j} + (1-\lambda_i)p_{2j})$$

$$- \sum_{i=1}^{n} \sum_{j:\sigma(i)<\sigma(j)} (\lambda_i p_{1j} + (1-\lambda_i)p_{2j}).$$

Hence, solving (LR) over all permutation schedules is equivalent to finding a permutation σ that maximizes

$$\sum_{(i,j):\sigma(i)<\sigma(j)} (\lambda_i p_{1j} + (1-\lambda_i)p_{2j}). \tag{4}$$

Bergmans [3] and Pratt [17] showed, by an adjacent pairwise interchange argument, that the linear ordering problem is polynomially solvable for two special cases; see also Picard and Queyranne [16]. If the weights are in product form, i.e. $a_{ij} = x_i y_j$, the linear ordering problem is solved by ordering according to non-increasing ratios x_i/y_i. This ordering is exactly induced by Smith's rule. The linear ordering problem can also efficiently be solved if the weights are in sum form, i.e. $a_{ij} = x_i + y_j$. In that case, an optimal permutation is obtained by ordering the elements according to non-increasing values $x_i - y_i$. The choice $\lambda_i = c$ for each i, for some constant c ($0 \le c \le 1$), converts (4) into an even simpler polynomially solvable case of the linear ordering problem: we obtain the form $a_{ij} = y_j$, solved by ordering according to non-decreasing values y_j. Hence, for those particular values of λ, solving problem (LR) over all permutation schedules amounts to scheduling the jobs in order of non-decreasing values $cp_{1j} + (1-c)p_{2j}$. The values $c = 0$ and $c = 1$ render exactly the Ignall and Schrage lower bounds, and in fact these bounds result from applying Smith's rule to each of the machines separately.

In the sequel of this paper, the notation (LR(c)) refers to problem (LR) with $\lambda_i = c$ for each i. $L(c)$ denotes the optimal objective value of problem (LR(c)).

2.1. SOLVING THE LAGRANGIAN DUAL

Of course, we are particularly interested in solving the (restricted) Lagrangian dual (D), that is, in finding that value of c ($0 \le c \le 1$) that maximizes $L(c)$:

$$\max_{0\le c\le 1} \min \sum_{i=1}^{n} \left(C_{2i} + c(C_{1i} + p_{2i} - C_{2i})\right). \tag{D}$$

We assert that $L(c)$ is a continuous, concave and piecewise-linear function in c (see Fisher [5]). Hence, an optimal solution is achieved in a point of non-differentiability or breakpoint. These breakpoints can be characterized in the following way.

Job i is called *c-preferable* to job j if $cp_{1i} + (1 - c)p_{2i} < cp_{1j} + (1 - c)p_{2j}$, which means that job i is scheduled before job j in the solution to problem (LR(c)). If job i is c-preferable to job j for all c ($0 \leq c \leq 1$), then job i is *strongly preferable* to job j. For each pair of jobs (i, j) without a strong preference relation, a *critical value* is defined as the value of c for which both jobs are equally preferable, i.e. $cp_{1i} + (1 - c)p_{2i} = cp_{1j} + (1 - c)p_{2j}$. These critical values are precisely the points of non-differentiability.

The procedure to solve (D) is the following. Find the $O(n^2)$ critical values and sort them in non-decreasing order. From (D), one can tell for each critical value o whether $o + \varepsilon$ or $o - \varepsilon$, with $\varepsilon > 0$ and ε sufficiently small, is the direction of ascent. In case o has no direction of ascent, then of course o is the breakpoint at which the optimal solution is attained. So, the optimal breakpoint can be achieved by binary search over all breakpoints. The Lagrangian dual is solved in $O(n^2 \log n)$ time. As $\max_{0 \leq c \leq 1} L(c) \geq \max\{L(0), L(1)\}$, it produces a lower bound that dominates the Ignall and Schrage lower bounds.

2.2. STRENGTHENING THE LOWER BOUND

Let c^* be the value of c that solves problem (D). Suppose now that the multiplier vector λ is perturbed in the ith component by a term Δ_i, i.e. $\lambda_i = c^* + \Delta_i$. Suppose further that this perturbation does not change the processing order. Obviously, the lower bound would be affected by the term

$$\Delta_i(C_{1i} + p_{2i} - C_{2i}). \tag{5}$$

Let $a_{ij} = \lambda_i p_{1j} + (1 - \lambda_i)p_{2j}$. If λ_i were perturbed by Δ_i, then the ith row in the weight matrix A for the linear ordering problem would become $a_{ij} + \Delta_i(p_{1j} - p_{2j})$, for $j = 1, \ldots, n$. The issue now is to determine the range for Δ_i such that the optimal solution to the perturbed problem is the same as to (LR(c^*)). Recall that the choice $\lambda_i = c$ for each i implies for the solution of problem (LR(c)) that

$$a_{ji} > a_{ij} \Leftrightarrow \sigma(i) > \sigma(j).$$

Hence, a sufficient condition to ensure that the optimal solution remains the same is that for each j ($j = 1, \ldots, n$, $j \neq i$) we have

$$a_{ji} \geq a_{ij} + \Delta_i(p_{1j} - p_{2j}) \qquad \text{if } \sigma(i) > \sigma(j), \tag{6}$$

$$a_{ji} \leq a_{ij} + \Delta_i(p_{1j} - p_{2j}) \qquad \text{if } \sigma(i) < \sigma(j). \tag{7}$$

The next step is to compute for each j, $j \neq i$, the value δ_{ij} of Δ_i such that the values a_{ji} and $a_{ij} + \Delta_i(p_{1j} - p_{2j})$ coincide, if such a value exists. From this, we obtain

$$\delta_{ij} = (a_{ji} - a_{ij})/(p_{1j} - p_{2j}) \qquad \text{if } p_{1j} \neq p_{2j}.$$

Defining $\Delta_i^+ = \min_j \{d_{ij} \mid d_{ij} \geq 0 \text{ and } p_{1j} \neq p_{2j}\}$ and $\Delta_i^- = \max_j \{d_{ij} \mid d_{ij} \leq 0 \text{ and } p_{1j} \neq p_{2j}\}$, respectively, we conclude that as long as λ_i is perturbed by Δ_i with $\Delta_i^- \leq \Delta_i \leq \Delta_i^+$, the optimal solution to (LR(c^*)) is also optimal to the perturbed problem. Therefore, the current lower bound can be improved by maximizing (5) subject to and $\Delta_i^- \leq \Delta_i \leq \Delta_i^+$ and $0 \leq \lambda_i + \Delta_i \leq 1$. Hence, the Lagrangian weights are perturbed in the following way:

$$\text{(a)} \quad \lambda_i \leftarrow \min\{c^* + \Delta_i^+, 1\} \qquad \text{if } C_{1i} + p_{2i} > C_{2i},$$

$$\text{(b)} \quad \lambda_i \leftarrow \max\{c^* + \Delta_i^-, 0\} \qquad \text{if } C_{1i} + p_{2i} < C_{2i}.$$

This analysis can consecutively be performed for each job i and takes $O(n^2)$ time altogether. It is important to note that the ultimate strength of the lower bound depends on the order in which the multipliers have been adjusted.

2.3. PRECEDENCE CONSTRAINTS

A job i is said to have *precedence* to job j, denoted by $i \rightarrow j$, if there is an optimal solution in which job i precedes job j. The technique of deriving precedence constraints is based upon the following concept. Let (LR($c, i \rightarrow j$)) denote problem (LR(c)) to which we added the constraint $i \rightarrow j$, while job j is c-preferable to job i. Clearly, we have $L(c, i \rightarrow j) > L(c)$. If $L(c, i \rightarrow j)$ exceeds a known upper bound, then obviously there is an optimal solution to (P) in which $j \rightarrow i$. We only have to deal with the question whether (LR($c, i \rightarrow j$)) is polynomially solvable. Fortunately, this is the case. A single-machine result from Monma and Sidney [15] for objective functions that possess the adjacent pairwise interchange property applies to problem (LR($c, i \rightarrow j$)). This result clears the way for solving (LR($c, i \rightarrow j$)) in a quite straightforward way.

THEOREM 1

For problem (LR($c, i \rightarrow j$)) with job j c-preferable to job i, there is an optimal permutation with job j immediately succeeding job i.

Again, this can be demonstrated by an interchange argument.

By use of theorem 1, an optimal permutation for (LR($c, i \rightarrow j$)) can be found in the following way. Start by scheduling all jobs as in the solution of problem (LR(c)) and remove the jobs i and j from this sequence. Call this permutation π. The

module $\{i,j\}$ is then inserted just before the first job $k \in \pi$ for which $2(cp_{1k} + (1-c)p_{2k}) > c(p_{1i} + p_{1j}) + (1-c)(p_{2i} + p_{2j})$. If no such job exists, then $\{i,j\}$ is scheduled last. This condition stems from evaluating the objective values for (LR(c)) for the sequences ijk and kij, respectively. The lower bound resulting from (LR($c, i \rightarrow j$)) can be strengthened in the same spirit as was outlined in section 2.2.

3. Dominance criteria

A node at level k of the branch-and-bound procedure corresponds to an initial partial sequence π in which k jobs have been put in the first k positions. For each node at level k, at most $n-k$ descendant nodes are created, one for every job without unscheduled predecessors. Let $C_h(\pi)$ be the completion time of the last job in sequence π on machine h. The sum of the job completion times on machine 2 of the jobs in π is denoted by $TC(\pi)$. Then there is no need to branch from a node having π as an initial sequence if there is permutation π^* of the jobs in π, $\pi^* \neq \pi$, that satisfies the following conditions:

$$TC(\pi^*) \leq TC(\pi), \tag{8}$$

$$C_2(\pi^*) \leq \max\{C_2(\pi), C_1(\pi) + \min_{i \in \pi} p_{1i}\}. \tag{9}$$

In that case, we say that the sequence π is *dominated* by π^*. Condition (9) ensures that the unscheduled jobs can start on machine 2 at least as soon with π^* as with π as initial sequence. Of course, finding out whether a given permutation π is dominated or not is as difficult as the original problem. A *dominance rule* gives an easy to check sufficient condition for the existence of dominance.

The following three rules should be checked as soon as we are about to add a new job i to the current initial sequence. The *dynamic programming dominance* criterion is probably the most obvious one: a node that corresponds with the sequence $\pi = \rho ji$ can be eliminated if the sequence π is dominated by the sequence $\pi^* = \rho ij$. Here, ρ is a subsequence of jobs.

The second rule reschedules the jobs in $\pi = \rho i$ into π^* according to Johnson's rule [12] for minimizing maximum completion time in the two-machine flow shop. Then certainly, condition (9) is satisfied. It is not difficult to find out whether $TC(\pi^*) \leq TC(\pi)$. Note that if job i appears before job j in π^*, while we have derived in section 2.3 that $j \rightarrow i$, we can still eliminate the node associated with π if conditions (8) and (9) are satisfied.

The third rule looks for a job $j \in \pi$ such that $p_{1i} \leq p_{1j}$ and $p_{2i} \leq p_{2j}$. Thus, π can be written as $\pi = \rho_1 j \rho_2 i$, where ρ_1 and ρ_2 are subsequences. If we let $\pi^* = \rho_1 i \rho_2 j$, then condition (8) for the existence of dominance of π by π^* is satisfied. This is stated in the following lemma.

LEMMA

If we have $p_{1i} \leq p_{1j}$ and $p_{2i} \leq p_{2j}$, then $TC(\rho_1 i \rho_2 j) \leq TC(\rho_1 j \rho_2 i)$.

Proof

We have $C_1(\rho_1 i) = C_1(\rho_1 j) + p_{1i} - p_{1j} \leq C_1(\rho_1 j)$; this implies

$$C_2(\rho_1 i) \leq C_2(\rho_2 j) + p_{2i} - p_{2j}. \tag{10}$$

Furthermore, it holds that $C_1(\rho_1 i \rho_k k) = C_1(\rho_1 j \rho_k k) + p_{1i} - p_{1j} \leq C_1(\rho_1 j \rho_k k)$ for every job $k \in \rho_2$, and hence that

$$C_2(\rho_1 i \rho_k k) \leq C_2(\rho_1 j \rho_k k) \qquad \text{for every } k \in \rho_2, \tag{11}$$

where ρ_k denotes the jobs of subsequence ρ_2 that are scheduled before k. In addition, we have $C_1(\rho_1 i \rho_2 j) = C_1(\rho_1 j \rho_2 i)$. Because of this and since $C_2(\rho_1 i \rho_2) \leq C_2(\rho_1 j \rho_2)$, we have

$$C_2(\rho_1 i \rho_2 j) \leq C_2(\rho_1 j \rho_2 i) + p_{2j} - p_{2i}. \tag{12}$$

Totalling all completion times with the help of expressions (10), (11), and (12) yields the desired result. □

As can be seen from (12), there is no guarantee beforehand that condition (9) is satisfied as well. It has yet to be verified if this is the case; only then is $\pi = \rho_1 j \rho_2 i$ dominated by $\pi^* = \rho_1 i \rho_2 j$. We may fathom the node associated with π even if some of the precedence relations obtained in section 2.3 are violated in the sequence π^*. In that case, we have $TC(\rho_1 j \rho_2 i \rho_3) \geq TC(\rho_1 i \rho_2 j \rho_3) > UB$, where $\rho_1 j \rho_2 i \rho_3$ and $\rho_1 i \rho_2 j \rho_3$ are complete schedules.

Conway et al. [4] claim that there is an optimal solution in which job i precedes job j if $p_{1i} \leq p_{1j}$ and $p_{2i} \leq p_{2j}$. As can be seen from expression (12), this cannot be established by the interchange argument used in the proof of the lemma. Szwarc [19] shows the claim to be faulty by giving a counterexample.

Under a more stringent condition, however, we deduce the following result, which can be used to generate a priori precedence constraints.

THEOREM 2

If for jobs i and j it holds that $p_{2i} = p_{2j}$ and $p_{1i} \leq p_{1j}$, then there is an optimal permutation in which job i precedes job j.

Proof

We have to show that under these conditions any subsequence of the type $\rho_1 i \rho_2 j$ is dominated by $\rho_1 j \rho_2 i$ in terms of conditions (8) and (9). Condition (8) is

satisfied, as can be seen from the lemma. Since $p_{2i} = p_{2j}$, expression (12) reduces to $C_2(\rho_1 i \rho_2 j) \leq C_2(\rho_1 j \rho_2 i)$, which implies that $C_2(\pi^*) \leq C_2(\pi)$; hence, condition (9) is satisfied too. □

Of course, if for jobs i and j we have $p_{2i} = p_{2j}$ and $p_{1i} = p_{1j}$, we allow either $i \rightarrow j$ or $j \rightarrow i$ in order to avoid the inconsistency to have both $i \rightarrow j$ and $j \rightarrow i$. Note that the combination of the precedence relations from theorem 2 and the precedence relations generated as described in section 2.3 cannot result in inconsistencies.

4. The algorithm

Before starting the actual branch-and-bound procedure, we do some preprocessing in order to find an upper bound, to derive precedence constraints, and to accelerate the calculations in a node of the tree. As far as an upper bound is concerned, we begin with a random permutation and we try to improve its sum of the job completion times by local interchanges. In this way, we obtain some upper bound, say, UB. It turned out to be a robust procedure that provided us with satisfactory initial upper bounds.

In addition, we approximate the search over the $O(n^2)$ points, as described in section 2.1, by a search over 21 points. Therefore, we store 21 permutations that solve the problems (LR(c)) with $c = x/20$, $x = 0, \ldots, 20$, respectively. This search works sufficiently well due to the flatness of (LR(c)) around the optimum. The storage implies a significant reduction in lower bound calculation time, since we have to sort the jobs for each of these values of c only in the preprocessing phase; it then takes only linear time to compute ($L(c)$) in a node of the tree.

In a similar fashion, we store the maximum perturbation values Δ_i^+ and Δ_i^-, for each job i ($i = 1, \ldots, n$), which are computed as described in section 2.2. Actually, these values depend on the set of unscheduled jobs and should be computed in each node of the tree. Although they are likely to increase if we go down the search tree, the loss in strength was more than compensated for by the reduction in computation time. The storage reduces the cost of lower bound strengthening in a node of the tree from $O(n^2)$ to $O(n)$ time.

In order to derive precedence constraints, the best value of c with $c = x/20$, $x = 0, \ldots, 20$, is achieved by binary search. The completion times on both machines can easily be calculated from (3), taking linear time, albeit we can alternatively put $C_{2i} \leftarrow C_{2i} + \min_{1 \leq j \leq n} p_{1j}$ for each job i ($i = 1, \ldots, n$), since the second machine is surely idle until $\min_{1 \leq j \leq n} p_{1j}$. For problem (LR($c^*$)), we try to derive precedence constraints as described in section 2.3. For that purpose, we introduce an $n \times n$ matrix X with elements $x_{ij} = L(c^*, i \rightarrow j)$ and $x_{ii} = 0$. It is necessary to store this matrix, since new precedence constraints can possibly be derived as soon as we find a better upper bound.

The Ignall and Schrage algorithm follows a *best bound* strategy. For each of the new nodes, the corresponding lower bound is calculated and, if this lower bound is smaller than the current upper bound, this new node is inserted in a list of *active*

nodes. That list is sorted in order of non-decreasing lower bounds. The node at the top of this list is chosen to branch from. A significant advantage of such a list is that it facilitates dominance checking. However, in the worst case, the size of this list is exponential in the number of jobs. Computational experiments made it clear to us that this dominance checking was only advantageous for instances with n up to 10.

In contrast to the Ignall and Schrage procedure, we use an *active node* strategy. This means that we generate descendant nodes, of which there are at most $n - k$, for only one non-fathomed node at level k. These descendant nodes are stored in a separate list and sorted according to a branching rule. We then branch from the node at the top of this list. Such a procedure requires only $O(n^2)$ space, since at each level k we have a list of at most $n - k$ jobs. The only thing that remains to be explained is the branching rule. The new nodes that add some job j without unscheduled predecessors to an initial sequence π are sorted in non-decreasing order of $\sum_{i \in \{\pi\}} x_{ji}$. This sum is supposed to reflect some notion of "costs" if we schedule job j before the other unscheduled jobs.

Table 1

Computational results on a VAX-780 computer.

	Ignall and Schrage algorithm			Proposed algorithm	
Data set	Max number of active nodes	Total number of nodes	Time [sec]	Total number of nodes	Time [sec]
10.1	5	53	0.86	9	1.54
10.2	13	84	0.88	10	1.30
10.3	18	152	0.96	14	1.52
10.4	117	728	3.10	57	1.86
10.5	135	957	3.94	169	2.70
15.1	1462	13718	92.99	693	9.48
15.2	2097	11156	116.86	388	7.44
15.3	1721	17712	142.36	603	9.66
15.4	676	2946	18.58	169	5.04
15.5	4280	35442	958.74	380	6.02
20.1	5213	(98.81%)	336.72	963	18.98
20.2	6411	(95.28%)	281.02	9235	95.45
20.3	5266	(97.12%)	182.19	1282	21.66
20.4	8909	(90.43%)	489.98	8846	102.61
20.5	8184	(96.72%)	422.38	4913	56.28

Both algorithms were coded in C, implemented on a VAX-780 computer, and tested on problems with 10, 15 and 20 jobs. The processing times for each job were taken from the uniform distribution [1, 10], as Kohler and Steiglitz [13] did in carrying out their experiments. Table 1 presents the results. The entries in the column "maximum

number of active nodes" give an indication of the space required by the Ignall and Schrage algorithm. Data inspection shows that the new algorithm outperforms the Ignall and Schrage procedure, although in the case $n = 10$ it is sometimes slower. The main reason for this lies in the preprocessing phase. For instance, the derivation of precedence constraints takes $O(n^3)$ time, and is consequently relatively expensive for smaller instances.

As to the Ignall and Schrage algorithm with 20 jobs, computation was terminated after 10 000 nodes. An entry within brackets represents the ratio in percentage upon termination between the lower bound of the first node in the list and the current upper bound.

5. Conclusions

Although the presented approach shows a significant improvement with respect to the Ignall and Schrage algorithm, the $F2 \mid\mid \sum C_i$ problem remains difficult to solve. From additional experiments, it appeared that major difficulties are encountered for instances beyond 25 jobs. Most of the results obtained here carry over to the more general $F2 \mid\mid \sum w_i C_i$ problem. In this problem, each job i has got some weight w_i attached to it, expressing its importance relative to other jobs. Performing an analysis along the lines of section 2, one can find out that the resulting linear ordering problem can be efficiently solved in the case that for each i, $\lambda_i = c$, with $c = 0$, w_i, or $w_i/2$. For this last choice of λ, the weights of the linear ordering problem are in product form.

Acknowledgements

The author wishes to express his thanks to Han Hoogeveen, Jan Karel Lenstra, Maurice Queyranne, and to two anonymous referees whose suggestions led to a significant improvement of an earlier version of the paper.

References

[1] I. Adiri and N. Amit, Openshop and flowshop scheduling to minimize sum of completion times, Comput. Oper. Res. 11(1984)275–284.

[2] S.P. Bansal, Minimizing the sum of completion times of n jobs over m machines in a flowshop – a branch and bound approach, AIIE Trans. 9(1977)306–311.

[3] P.P. Bergmans, Minimizing expected travel time on geometrical patterns by optimal probability rearrangements, Information and Control 20(1972)331–350.

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

[5] M.L. Fisher, The Lagrangian relaxation method for solving integer programming problems, Manag. Sci. 27(1981)1–18.

[6] M.L. Fisher, B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, Surrogate duality relaxation for job shop scheduling, Discr. Appl. Math. 5(1983)65–75.

[7] M.R. Garey, D.S. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling, Math. Oper. Res. (1976)117–129.
[8] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Ann. Discr. Math. 5(1979)287–326.
[9] J.N.D. Gupta and R.A. Dudek, Optimality criteria for flowshop schedules, AIIE Trans. 3(1971) 199–205.
[10] A.M.A. Hariri and C.N. Potts, Algorithms for two-machine flow-shop sequencing with precedence constraints, Eur. J. Oper. Res. 17(1984)238–248.
[11] E. Ignall and L. Schrage, Application of the branch and bound technique for some flowshop scheduling problems, Oper. Res. 13(1965)400–412.
[12] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Naval Res. Logist. Quart. 1(1954)61–68.
[13] W.H. Kohler and K. Steiglitz, Exact, approximate and guaranteed accuracy algorithms for the flow-shop problem $n/2/F/\bar{F}$, J. Assoc. Comput. Mach. 22(1975)106–114.
[14] M.J. Krone and K. Steiglitz, Heuristic programming solution of a flowshop-scheduling problem, Oper. Res. 22(1974)629–638.
[15] C.L. Monma and J.B. Sidney, Sequencing with series-parallel precedence constraints, Math. Oper. Res. 3(1979)215–224.
[16] J.-C. Picard and M. Queyranne, On the one-dimensional space allocation problem, Oper. Res. 29(1982)371–391.
[17] V.R. Pratt, An $O(n \log n)$ algorithm to distribute n records in a sequential access file, in: *Complexity of Computer and Computations*, ed. R.E. Miller and J.W. Thatcher (Plenum, New York, 1972), pp. 111–118.
[18] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logists. Quart. 3(1956)59–66.
[19] W. Szwarc, The flow-shop problem with mean completion time criterion, AIIE Trans. 15(1983) 172–176.