Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

S.L. van de Velde

Minimizing total completion time in the two-machine flow shop by Lagrangian relaxation

Department of Operations Research and System Theory Report OS-R8808 June

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Minimizing Total Completion Time in the Two-Machine Flow Shop by Lagrangian Relaxation

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

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. Earlier published lower bounds are shown to coincide with two of these choices. Several dominance criteria are given to restrict the search tree. Computational experiments show that the proposed algorithm outperforms the previously best method.

1980 Mathematics Subject Classification (1985): 90B35, 90C27.
Key Words & Phrases: flow shop scheduling, Lagrangian relaxation, linear ordering problem, branch-and-bound, computational experience.
Note: This paper has been submitted for publication.

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 h-th operation of job i has to be scheduled on machine h during a positive uninterrupted processing time $(h = 1,...,m; i = 1,...,n)$. 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 (1971) 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 completion times in a two-machine flow shop. It is well known that for this problem it suffices to optimize over all permutation schedules (Conway 1967). A *permutation schedule* is a schedule in which every machine has the same job sequence. Ignall and Schrage (1965) 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 (1974) were applied by Kohler and Steiglitz (1975) in further developing and testing the Ignall and Schrage algorithm. Garey and Johnson (1976) proved the problem to be NP-hard.

Szwarc (1983) developed some properties for the m-machine flow shop problem with the total completion time criterion and defined a class of well-solvable cases. A more elaborate treatment of well-solvable cases can be found in Adiri and Amit (1984). Bansal (1977) extended the branch-and-bound algorithm 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

Report OS-R8808
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

types of combinatorial optimization problems (see Fisher (1981) 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, seldomly give way to promising relaxations. Fisher et al. (1983) confirm this observation in their (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 (1984) 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 a formulation of the problem will be given, 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 that 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 ($i = 1,...,n$) on machine h ($h = 1,2$). The problem of minimizing the sum of the job completion times in a two-machine flow shop can then be formulated as follows: determine completion times C_{hi} ($h = 1,2; i = 1,...,n$) that minimize

$$\sum_{i=1}^{n} C_{2i} \tag{P}$$

subject to

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

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

Condition (1) will be formulated as

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

In the sequel of this paper, condition (2) will be 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):

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

From standard Lagrangian theory (Fisher 1981), it is known that for any given $\lambda \geqslant 0$, the value $L(\lambda)$ provides a lower bound to (P). In order to prevent that $L(\lambda)$ becomes arbitrarily small, we require that $\lambda \leqslant 1$.

In the Lagrangian problem, the operations of a job can be processed simultaneously. Hence, the problem decomposes into two single-machine problems, that can easily be solved by Smith's (1956) 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 (LR) can be solved in polynomial time.

To that 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 $\{1,...,n\}$ that maximizes the sum

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

In our application, we identify $\sigma(i)$ with the job that is put in the i-th position. Since in problem (LR) we have that

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

it follows that

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

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

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

Kolen (1986) proved, by an adjacent pairwise interchange argument, that the linear ordering problem is polynomially solvable for two special cases. 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 $\lambda_j = c$ for each j, for some constant c ($0 \leqslant c \leqslant 1$), converts (4) into an even simpler polynomially solvable case of the linear ordering problem: we get 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 $\lambda_j = c$ for each j. $L(c)$ 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 ($0 \leqslant c \leqslant 1$) that maximizes $L(c)$:

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

We assert that ($L(c)$) is a continuous, concave and piecewise-linear function in c. 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 $cp_{1i} + (1-c)p_{2i} < cp_{1j} + (1-c)p_{2j}$. If job i is c-preferable to job j for all c ($0 \leqslant c \leqslant 1$), then job i is *strongly preferable.* 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 $o+\epsilon$ or $o-\epsilon$, with $\epsilon > 0$ 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 a binary search over all breakpoints.

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 perturbated in the i-th 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

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

Define $a_{ij} = \lambda_i p_{1j} + (1-\lambda_i)p_{2j}$. If λ_i would be perturbated by Δ_i, then the i-th row in the weight matrix A for the linear ordering problem would turn into $a_{ij} + \Delta_i(p_{1j} - p_{2j})$, for $j = 1,...,n$. The issue now is to determine the range for Δ_i such that the optimal solution to the perturbated problem is the same as to (LR(c^*)). A sufficient condition for this is that for each j ($j = 1,...,n$, $j \neq i$)

$$a_{ji} \geqslant a_{ij} + \Delta_i(p_{1j} - p_{2j}) \quad \text{if } \sigma(i) > \sigma(j),$$
$$a_{ji} \leqslant a_{ij} + \Delta_i(p_{1j} - p_{2j}) \quad \text{if } \sigma(i) < \sigma(j).$$

The next step is then to calculate for each j, $j \neq i$, the value δ_{ij} such that a_{ji} and $(a_{ij} + \delta_{ij}(p_{1j} - p_{2j}))$ coincide, if such a value exists. From this, we get

$$\begin{aligned}
\delta_{ij} &= (a_{ji} - a_{ij})/(p_{1j} - p_{2j}) && \text{if } p_{1j} \neq p_{2j}, \\
&= 1 - \lambda_i && \text{if } p_{1j} = p_{2j},\ a_{ij} \neq a_{ji},\ C_{1i} + p_{2i} - C_{2i} > 0, \\
&= -\lambda_i && \text{if } p_{1j} = p_{2j},\ a_{ij} \neq a_{ji},\ C_{1i} + p_{2i} - C_{2i} < 0, \\
&= 0 && \text{if } p_{1j} = p_{2j},\ a_{ij} = a_{ji}.
\end{aligned}$$

Defining $\Delta_i^+ = \min_{j \mid \delta_j \geqslant 0} \delta_j$ and $\Delta_i^- = \max_{j \mid \delta_j \leqslant 0} \delta_j$, respectively, we conclude that as long as λ_i is perturbated by Δ_i with $\Delta_i^- \leqslant \Delta_i \leqslant \Delta_i^+$, the optimal solution to (LR(c^*)) is also optimal to the perturbated problem. Therefore, the current lower bound can be improved by perturbating the Lagrangian weights in the following way:

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

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

This analysis can consecutively be performed for each job i. It is important to note that the ultimate strengthened 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* over 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 that $L(c, i \rightarrow j) > L(c)$. 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 (1979) for objective functions that possess the adjacent pairwise interchange property applies to problem (LR(c)). 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 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 for 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 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:

$$TC(\pi^*) \leqslant TC(\pi), \tag{6}$$

$$C_2(\pi^*) \leqslant \max\{ C_2(\pi), C_1(\pi) + \min_{i \notin \{\pi\}} p_{1i} \}. \tag{7}$$

In that case we say that the sequence π is *dominated* by π^*. Condition (7) ensures that the unscheduled jobs can start on machine 2 at least as soon with π^* as with π as an initial sequence. Of course, finding out whether a given permutation π is dominated or not is as hard as the original problem. A *dominance rule* gives an easy to check sufficient condition for the existence of dominance.

The next result comes forth from this dominance concept, but it boils down to a rule to generate a priori precedence constraints.

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

PROOF. We shall compare two sequences $\pi_1 i \pi_2 j \pi_3$ and $\pi_1 j \pi_2 i \pi_3$: the orders are identical except for jobs i and j. Then we have that $C_1(\pi_1 j) = C_1(\pi_1 i) + p_{1j} - p_{1i}$ and therefore $C_2(\pi_1 j) \geqslant C_2(\pi_1 i)$. Furthermore, for every job $k \in \pi_2$ it holds that $C_2(\pi_1 j \pi_k k) \geqslant C_2(\pi_1 i \pi_k k)$, where π_k denotes those jobs of subsequence π_2 that are scheduled before job k. Consequently, $C_2(\pi_1 j \pi_2) \geqslant C_2(\pi_1 i \pi_2)$ and from this and from $C_1(\pi_1 j \pi_2 i) = C_1(\pi_1 i \pi_2 j)$ we derive that $C_2(\pi_1 i \pi_2 j) \leqslant C_2(\pi_1 j \pi_2 i)$. This is exactly condition (7). Totalling all processing times yields $TC_2(\pi_1 i \pi_2 j) \leqslant TC_2(\pi_1 j \pi_2 i)$. Since these arguments are irrespective of the subsequences π_1, π_2 and π_3, the conclusion that job i precedes job j is warranted. □

The next rules should be checked as soon as we are about to add a new job j to the current initial sequence. The *dynamic programming dominance* criterion is probably the most obvious one. A node that adds job j to the sequence πi can be eliminated if it is dominated by the sequence πji. The second one reschedules πj into π^* according to Johnson's rule (1954) for minimizing makespan for a two-machine flow shop. Then certainly, condition (7) is satisfied. It is left to find out whether $TC_2(\pi^*) \leqslant TC_2(\pi j)$. The third rule compares the new sequence $\pi_1 i \pi_2 j$ with $\pi_1 j \pi_2 i$, where job i is selected such that $p_{1i} \leqslant p_{1j}$ and $p_{2i} < p_{2j}$. From the arguments we used for Theorem 2 we know that $TC(\pi_1 i \pi_2 j) \leqslant TC(\pi_1 j \pi_2 i)$. It can rapidly be verified whether condition (7) also holds. It should be noted that in case ties occur implementation of these rules should be carried out carefully in order not to eliminate both sequences.

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 get some upper bound, say, *UB*. Furthermore, we store 21 permutations that solve the problems (LR(c)) with $c = x/20, x = 0,...,20$ respectively. Preliminary calculations showed that the function (LR(c)) is very flat around the optimum. This storage enables a significant reduction in lower bound calculation time, since we only have to sort the jobs for each value of c once. This storage enables the calculation of ($L(c)$) in a node of the tree in only linear

time.

In order to derive additional precedence constraints, the best c, say, c^* among these 21 values is achieved by a binary search. The completion times on both machines can easily be calculated from (3), taking linear time, albeit that we can put $C_{2i} \leftarrow C_{2i} + \min_{1 \leq j \leq n} p_{1j}$ for each job i ($i = 1,...,n$), 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, as soon as we find a better upper bound, new precedence constraints can possibly be derived.

Furthermore, for each value of c we determine and store the maximum perturbation values Δ_i^+ and Δ_i^-, for each job i ($i = 1,...,n$). Actually, these values depend on the set of unscheduled jobs and, consequently, these values are likely to increase if we go down the search tree. However, this storage reduces the cost of lower bound strengthening from $O(n^2)$ to $O(n)$ time per call. The deficiency of this weakened strengthening procedure was more that compensated by the reduction in computation time.

Table 1: Computational results on a VAX-780 computer

	IGNALL and SCHRAGE ALGORITHM			PROPOSED ALGORITHM	
data set	max. # active nodes	total # nodes	time sec	total # 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

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 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 on 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 for only one non-fathomed node at level k its descendant nodes, of which there are at most $n-k$, are generated. These descendant nodes are stored in a separate list, sorted according to a branching rule, and we pick the node on top of this list to branch from. This node becomes the active node at level $k+1$. Such a procedure only requires $O(n^2)$ space, since at each level k we have a list of at most $n-k$ jobs. The only thing

that remains to explain 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 $\sum_{i \notin \{\pi\}} x_{ji}$. This sum is supposed to reflect some notion of 'costs' if we schedule job j before the other unscheduled jobs.

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 (1975) 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 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 10000 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

The computational tests show that the $F2||\sum C_j$ problem remains difficult to solve. Nevertheless, the presented approach proved to be very useful. Most of the results obtained here carry over to the more general $F2||\sum w_j C_j$ problem. In this problem, each job j has got some weight w_j 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 efficiently be solved in case that for each j, $\lambda_j = c$, with $c = 0$, w_j, or $w_j/2$. For this last choice of λ the weights of the linear ordering problem are in product form.

References

I. Adiri, N. Amit (1984). Openshop and flowshop scheduling to minimize sum of completion times. *Computers and Operations Research 11*, 275-284.

S.P. Bansal (1977). Minimizing the sum of completion times of n jobs over m machines in a flowshop - a branch and bound approach. *AIIE Transactions 9*, 306-311.

R.W. Conway, W.L. Maxwell, L.W. Miller (1967). *Theory of Scheduling*. Addison-Wesley, Reading, Mass.

M.L. Fisher (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science 27*, 1-18.

M.L. Fisher, B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan (1983). Surrogate duality relaxation for job shop scheduling. *Discrete Applied Mathematics 5*, 65-75.

M.R. Garey, D.S. Johnson, R. Sethi (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*, 117-129.

J.N.D. Gupta, R.A. Dudek (1971). Optimality criteria for flowshop schedules. *AIIE Transactions 3*, 199-205.

A.M.A. Hariri, C.N. Potts (1984). Algorithms for two-machine flow-shop sequencing with precedence constraints. *European Journal of Operational Research 17*, 238-248.

E. Ignall, L. Schrage (1965). Application of the branch and bound technique for some flow-shop scheduling problems. *Operations Research 13*, 400-412.

S.M. Johnson (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly 1*, 61-68.

W.H. Kohler, K. Steiglitz (1975). Exact, approximate and guaranteed accuracy algorithms for the flow-shop problem $n/2/F/\bar{F}$. *Journal of the Association for Computing Machinery 22*, 106-114.

A.W.J. Kolen (1986). A polynomial algorithm for the linear ordering problem with weights in product form. *Report 8622/A, Econometric Institute*, Erasmus University, Rotterdam.

M.J. Krone, K. Steiglitz (1974). Heuristic programming solution of a flowshop-scheduling problem. *Operations Research 22*, 629-638.

C.L. MONMA, J.B. SYDNEY (1979). Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research 3*, 215-224.

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

W. SZWARC (1983). The flow-shop problem with mean completion time criterion. *IIE Transactions 15*, 172-176.