Chapter 10

DANTZIG-WOLFE DECOMPOSITION FOR JOB SHOP SCHEDULING

Sylvie Gélinas
François Soumis

Abstract This chapter presents a formulation for the job shop problem based on Dantzig-Wolfe decomposition with a subproblem for each machine. Each subproblem is a sequencing problem on a single machine with time windows. The formulation is used within an exact algorithm capable of solving problems with objectives $C_{\max}$, $T_{\max}$, as well as an objective consistent with the Just-In-Time principle. This objective involves an irregular cost function of operation completion times. Numerical results are presented for 2 to 10 machine problems involving up to 500 operations.

1. Introduction

The job shop problem is a classical scheduling problem (French, 1982) that consists of scheduling n jobs on m machines. A single machine processes one job at a time and a job is processed by one machine at a time. Processing of one job on one machine is called an operation. The length of an operation is fixed; once begun, an operation may not be interrupted. The sequence of machines is known for each job. This sequence defines precedence constraints between the operations. The sequence of jobs on each machine must be determined so as to minimize a function of the operation completion times. This chapter considers problems in which jobs do not necessarily involve operations on all machines, but only on a subset of machines. This problem is occasionally referred to as the general job shop problem.

The problems considered in this chapter can be classified using the classical notation such as $(J|r_i, d_i|C_{\max})$, $(J|r_i, d_i|T_{\max})$ and $(J|r_i, d_i|\text{JIT})$. The job shop problem is NP-hard in the strong sense (Rinnooy Kan, 1976; Garey and Johnson, 1979) and is one of the hardest problems to solve in practice. The most successful exact algorithms use a branch and bound procedure. See McMahon and Florian (1975); Lageweg, Lenstra and Rinnooy Kan (1977), Barker and McMahon (1985); Carlier and Pinson (1989), and Brucker, Jurisch and Sievers (1994).

Three branching schemes are generally used: conflict resolution in the disjunctive graph (Roy and Sussman, 1964), generation of active schedules (Giffler and Thompson, 1960), and time oriented branching (Marten and Shmoys, 1996). The conflict resolution scheme produces a complete schedule at each node of the branch tree; schedules thus produced may, however, violate the machine constraints. Two descendants are obtained by selecting two operations in conflict and imposing an order on them. In the generation of active schedules scheme, schedules are constructed sequentially from the root of the tree to a terminal node. A node of the branch tree is associated with a feasible schedule for a subset of operations. Barker and McMahon (1985), by contrast, propose a scheme that obtains a feasible schedule for all operations at each node of the branch tree. This procedure branches by rearranging operations in a critical block to yield an improved schedule. The head-tail adjustment proposed by Brinkkotter and Bruckner (2001) is also of this type.

A lower bound may be obtained by relaxing the machine constraints for all machines except one, to yield m scheduling problems on a single machine with time and precedence constraints $(n|r_i, d_i, \text{prec}|C_{\max})$. While this problem is NP-hard, even large instances of it may be solved efficiently (Carlier, 1982; McMahon and Florian, 1975). The maximum value found for the m problems gives a lower bound for the job shop problem. Lageweg, Lenstra and Rinnooy Kan (1977) discuss this bound, as well as many others obtained by relaxing one or more aspects of the scheduling problem on one machine. Balas, Lenstra and Vazacopoulos (1995) propose an improvement that considers delayed precedence constraints between operations. They used the resulted bound within the shifting bottleneck heuristic of Adams, Balas and Zawack (1988). Brucker and Jurisch (1993) obtain a bound derived from two-job scheduling problems. Problems are solved in polynomial time using a graphical method. Numerical results show that the bound obtained in this way is superior to that obtained from single-machine scheduling problems if the ratio between the number of machines and the number of jobs is large.

Other authors propose a relaxation of the problem based on a mathematical formulation. Fisher (1973) uses Lagrangian relaxation to solve problems with min-sum type objectives. He dialyzes the machine constraints and conserves precedence constraints. This method is tested on eight problems involving up to five jobs and four machines. Hoitomt et al. (1993) present an augmented Lagrangian approach for the weighted quadratic tardiness job shop problem. They obtained feasible solutions within 4% of their respective lower bounds for 125–145 job problems with 1 to 3 operations per job. Fisher et al. (1983) propose two aggregate-constraint formulations for the objective $C_{\max}$, in which either the precedence or machine constraints are grouped into a linear combination. While these formulations yield a better bound than those obtained from single-machine scheduling problems or Lagrangian relaxation, the effort required discourages any search beyond the root of the search tree.

Currently available exact methods are not capable of solving large-scale problems. Carlier and Pinson (1989) solved the famous 10-machine, 10-job problem of Muth and Thompson (1963), more than 25 years after its publication. Applegate and Cook (1991) have solved some 150, 225-operation problems using many families of cutting phases and some good heuristics to find feasible solutions. However, some problems of 150, 200, 225-operations stayed unsolved. Brucker et al. (1994) have solved problems having up to 300 operations on a Sun 4/20 workstation. Optimal solutions of some of these problems require close to five days of CPU time.

In view of the difficulty of the job shop problem, algorithms should be developed to address the special structure of the problems encountered in industry. This chapter describes an efficient exact algorithm to solve problems with many jobs and few operations per job. Such problems appear more and more frequently in manufacturing, where increasingly versatile machines are capable of processing jobs with few changes per machine. The real problems from Pratt & Witney presented in Hoitomt et al. (1993) have this property (1 to 3 operations per job).

Most algorithms are designed to minimize the total length of operations ($C_{\max}$) and are poorly adapted to other objectives. In practice, however, other objectives such as minimizing inventory costs or penalties arising from delivery delays are frequently more interesting. While the objective $T_{\max}$ has been examined in the context of the one-machine scheduling problem, it has received little attention with regard to the general job shop problem. Furthermore, the literature contains virtually no discussion on exact methods for irregular functions of completion times. This chapter discusses objectives such as the total length of operations ($C_{\max}$), the maximum tardiness ($T_{\max}$) and an objective consistent with the Just-In-Time principle.

The algorithm presented here is of the branch and bound variety. A lower bound is obtained using Dantzig-Wolfe decomposition, (Dantzig and Wolfe, 1960). Unlike Fisher, we use a primal approach and relax precedence constraints rather than machine constraints. A primal approach yields faster convergence; furthermore, precedence constraints play a reduced role in problems involving few operations per job. The solution to the master problem provides a lower bound for the job shop problem, which is incorporated into a branching scheme based on conflict resolution. This formulation can be adapted to several objectives, including irregular functions of completion times. This is not the case for the aggregate-constraint formulation of Fisher et al. (1983), which is only valid for minimizing $C_{\max}$.

The Dantzig-Wolfe decomposition was used for parallel machine scheduling problems by van den Akker et al. (1995) and Chen and Powell (1999a,b, 2003). This scheduling problem does not involve precedence constraints and is simpler than the job shop scheduling problem. The column generation algorithms proposed by these authors are straightforward translations of the algorithm for vehicle routing problems presented by Desrochers et al. (1992). The present chapter proposes a column generation algorithm for a more complex problem.

More references on methods using constraint programming, metaheuristics, neural networks, can be found in surveys on the job shop scheduling problem by Blazewicz et al. (1996) and Join and Meeron (1999). More recent work using constraint propagation was presented by Dorndorf et al. (2000, 2002).

2. Mathematical formulation

We consider n jobs (index j), m machines (index i) and N operations (indices u, v). Let p_u be the time required for operation u, i_u the machine on which operation u is to be carried out, r_u the earliest time at which operation u may begin and d_u the latest time at which operation u may end.

Precedence relations are contained in the set

$$A = \{(u,v) | u \text{ and } v \text{ are successive operations of the same job, and } u \text{ preceeds } v\}$$

and pairs of operations carried out on machine i are contained in set

$$B_i = \{\{u,v\} \mid u \neq v, \text{ and } i_u = i_v = i\}, \quad i = 1, \dots, m.$$

The job shop problem is formulated using a min-max type objective function. We consider a function $g_u(C_u)$ of completion time C_u for operation u. We assume that the function $g_u(C_u)$ is piecewise linear, but not necessarily monotone. We define $g_{\max}$, a variable that takes the maximum value of the quantities $g_u(C_u)$ for all operations u.

We define the following variables,

C_u: completion time for operation u, $u = 1, \ldots, N$,

$g_{\max}$: cost of the solution.

The job shop problem can be formulated as follows.

$$\begin{aligned}
& && \min g_{\max} && && (10.1)\\
\text{s.t.} & && g_{\max} \geq g_u(C_u) && u = 1, \ldots, N, && (10.2)\\
& && C_u \leq C_v - p_v && (u,v) \in A, && (10.3)\\
& && r_u + p_u \leq C_u \leq d_u && u = 1, \ldots N, && (10.4)\\
& && C_u \leq C_v - p_v \vee C_v \leq C_u - p_u && \{u,v\} \in B_i,\ i = 1, \ldots, m. && (10.5)
\end{aligned}$$

We minimize a min-max type function of the operation completion times (10.1)–(10.2) under precedence constraints (10.3), time constraints (10.4) and machine constraints (10.5). The machine constraints are disjunctive and require that two operations carried out on the same machine may not occur at the same time. These constraints make the problem non-linear and hence difficult to solve. The next section reformulates the job shop problem using Dantzig-Wolfe decomposition.

3. Decomposition

We have formulated the job shop problem as a non-linear problem with disjunctive constraints. Relaxing the precedence constraints yields a problem that is separable by machine. Each problem corresponds to a single-machine scheduling problem whose solution provides a lower bound on the job shop problem. This idea is applied within the framework of Dantzig-Wolfe decomposition.

Constraints used to compute the objective (10.2), as well as the precedence constraints (10.3), are left in the master problem. The time constraints (10.4) and machine constraints (10.5) are transferred to the subproblems. Each subproblem generates schedules for one machine. The master problem selects a convex combination of the generated schedules that satisfies the precedence constraints. The Dantzig-Wolfe decomposition provides the optimal solution for a linear problem, and for the job shop problem, it provides a lower bound. This bound is better than that obtained by ignoring the precedence constraints, because of the exchange of information between the master problem and the subproblems.

3.1 Master problem

The following notation is used in the master problem formulation.

Ω_i: set of schedules that satisfy the time constraints (10.4) and machine constraints (10.5) for machine i, $i = 1, \ldots, m$,

C_u^h: completion time for operation u in schedule h, $u = 1, \dots, N$, $h \in \Omega_{i_u}$,

y_h : variable associated with schedule h, $\forall h \in \Omega_i$, $i = 1, \dots, m$.

The master problem is then:

$$\min g_{\max} \tag{10.6}$$

$$\text{s.t.} \quad g_{\max} \geq \sum_{h \in \Omega_{i_u}} y_h g_u(C_u^h) \quad u = 1, \dots, N, \tag{10.7}$$

$$\sum_{h \in \Omega_{i_u}} y_h C_u^h \leq \sum_{h \in \Omega_{i_v}} y_h C_v^h - p_v \quad \forall (u, v) \in A, \tag{10.8}$$

$$\sum_{h \in \Omega_i} y_h = 1 \quad i = 1, \dots, m, \tag{10.9}$$

$$y_h \geq 0, \quad \forall h \in \Omega_i, \qquad i = 1, \dots, m. \tag{10.10}$$

The master problem is formulated using variables associated with schedules for a single machine. Its objective is to find a convex combination of schedules for each machine such that the precedence constraints (10.8) are satisfied and the cost (10.6)–(10.7) is minimized. The convex combination constraints are given by (10.9) and (10.10).

The set of schedules for machine i is not convex, because of the machine constraints (which are disjunctive). This set contains a polytope for each ordering; each point in a polytope corresponds to a schedule for the ordering associated with the polytope. Polytopes are bounded, because of the time windows and have a finite number of extreme points

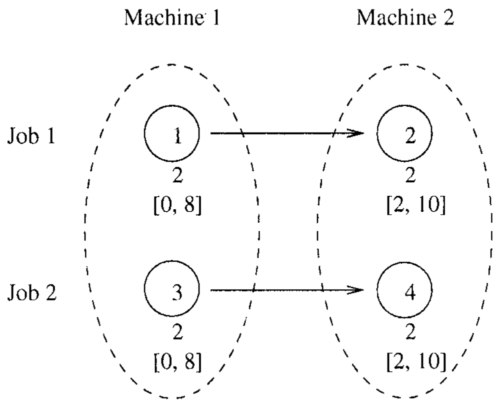

Figure 10.1. A job shop problem.

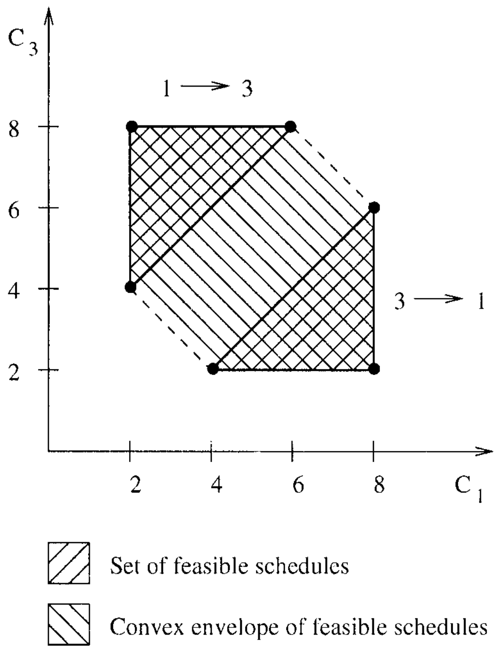
C_3
8
6
4
2
$1 \longrightarrow 3$
$3 \longrightarrow 1$
2
4
6
8
C_1
Set of feasible schedules
Convex envelope of feasible schedules

Figure 10.2. Schedules for machine 1.

and no extreme rays. The master problem obtains schedules as convex combinations of points in the set Ω_i. These schedules belong to the convex envelope of Ω_i, but not necessarily to the set Ω_i itself. Conv(Ω_i), the convex envelope of Ω_i is the convex combination of a finite set of extreme points; the union of the finite number of orderings on machine i of the finite set of extreme points of the polytope for this ordering. Because $\Omega_i \subseteq \text{Conv}(\Omega_i)$, Dantzig-Wolfe decomposition provides a lower bound for the job shop problem.

A job shop problem with two jobs, two machines and four operations is illustrated in Figure 10.1. All operations last two units of time. Job processing may begin at time 0, and must be completed by time 10. Figure 10.2 illustrates all feasible schedules for machine 1, which make up two polytopes associated either with the ordering 1-3 or the ordering 3-1. Each point in the set corresponds to a schedule for these two operations. The convex envelope of this set contains schedules that may be selected in the solution of the Dantzig-Wolfe decomposition.

Since there is a large number of sequences for each machine, it is impossible to enumerate all of them in a job shop problem unless the number of operations on each machine is very small. There is an even greater number of extreme schedules, which represent extreme points of schedule polytopes for all sequences. The master problem considers only a subset of schedules for each machine. New schedules are generated by subproblems as necessary.

3.2 Subproblems

Subproblem S_i finds the schedule at minimum marginal cost for machine i. Dual variables for the master problem are denoted as follows:

$\gamma_u \geq 0 \; u = 1, \ldots, N,$	Constraints for computing the objective (10.7),
$\alpha_{uv} \geq 0 \; (u, v) \in A,$	Precedence constraints (10.8),
$\lambda_i \; i = 1, \ldots, m.$	Convexity constraints (10.9).

Column h^* of minimum reduced cost $\bar{c}_{h^*}$ for subproblem S_i is such that

$$\begin{aligned}
\bar{c}_{h^*} &= \min_{h \in \Omega_i} \left\{ \sum_{u|i_u=i} \gamma_u g_u(C_u^h) + \sum_{u|i_u=i} \sum_{v|(u,v)\in A} \alpha_{uv} C_u^h \right. \\
&\qquad\qquad \left. - \sum_{u|i_u=i} \sum_{v|(v,u)\in A} \alpha_{vu} C_u^h - \lambda_i \right\} \\
&= \min_{h \in \Omega_i} \left\{ \sum_{u|i_u=i} \gamma_u g_u(C_u^h) + \sum_{u|i_u=i} \left(\sum_{v|(u,v)\in A} \alpha_{uv} - \sum_{v|(v,u)\in A} \alpha_{vu} \right) C_u^h \right\} - \lambda_i \\
&= \min_{h \in \Omega_i} \left\{ \sum_{u|i_u=i} \left(\gamma_u g_u(C_u^h) + w_u C_u^h \right) \right\} - \lambda_i \\
&= \min_{h \in \Omega_i} \left\{ \sum_{u|i_u=i} g'_u(C_u^h) \right\} - \lambda_i
\end{aligned}$$

where $w_u = \sum_{v|(u,v)\in A} \alpha_{uv} - \sum_{v|(v,u)\in A} \alpha_{vu}$ and $g'_u(C_u^h) = \gamma_u g_u(C_u^h) + w_u C_u^h$, $u = 1, \ldots, N$.

Subproblem S_i is formulated as follows:

$$\min \sum_{u|i_u=i} g'_u(C_u) \tag{10.11}$$

$$\text{s.t.} \qquad r_u + p_u \leq C_u \leq d_u \quad u \mid i_u = i, \tag{10.12}$$

$$C_u \leq C_v - p_v \vee C_v \leq C_u - p_u \quad \{u, v\} \in B_i. \tag{10.13}$$

This subproblem is a sequencing problem on a single machine with time constraints and an objective of minimizing a piecewise linear function of the completion times: $n|r_u, d_u| \sum g'_u(C_u)$. The problem is difficult because the cost function is irregular (the weights w_u may be positive or negative and the function $g_u(C_u)$ is an irregular function). This problem is solved using a dynamic programming algorithm. A dynamic programming state is associated with a set of operations X and cost function $G_X(t)$. The function $G_X(t)$ gives the minimum cost of a feasible schedule that carries out all operations of X and ends at the latest at time t. The functions $G_X(t)$ are evaluated by stages. At stage k, functions are evaluated for sets having k operations, using function values for sets containing $(k-1)$ operations. Details of the algorithm may be found in Gélinas and Soumis (1997).

At each iteration of the Dantzig-Wolfe algorithm, the master problem is solved using the simplex algorithm. The solution provides the values of the dual variables, which are then used in the subproblems to obtain new schedules, that is, new columns for the master problem. Columns are added to the master problem if their marginal cost is negative, giving rise to a new iteration. The procedure terminates when each subproblem generates a column with nonnegative marginal cost. The solution to the master problem is then the optimal solution for all columns, whether or not they are considered explicitly.

3.3 Branching

Dantzig-Wolfe decomposition provides a lower bound for the job shop problem. Although the solution satisfies the precedence and time constraints, it may violate the machine constraints because $\Omega_i \subseteq \mathrm{Conv}(\Omega_i)$. If all machine constraints are satisfied, the solution is optimal for the job shop problem. Otherwise, there are operations carried out concurrently on the same machine. In this case, a pair (u, v) of operations that conflict on one machine is selected and two new problems are created by imposing an order on these operations: either operation u is carried out before operation v, or operation v is carried out before operation u. The new problems are solved using Dantzig-Wolfe decomposition and this process continues until the branching tree has been thoroughly explored. The lower bound may be used to prune branches from the tree.

To respect the order imposed by the branching, precedence constraints are added between operations carried out on one machine. These constraints are easily handled in the subproblem solution. Dynamic programming states that do not satisfy constraints are not constructed. The subproblem then becomes a sequencing problem on a single machine with time and precedence constraints $\left(n \mid r_u, d_u, \text{prec} \mid \sum g'_u(C_u)\right)$.

An optimal schedule for the job shop problem of Figure 10.1, with the objective $C_{\max}$, is illustrated in Figure 10.3. The optimal schedule ends

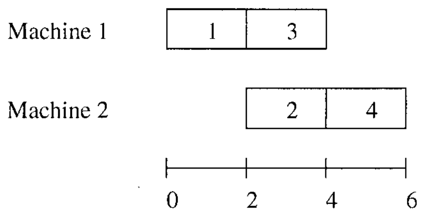

Figure 10.3. Optimal solution for objective $C_{\max}$.

at time 6. For this example, the Dantzig-Wolfe solution combines two schedules for each machine

$$\begin{pmatrix} C_1 \\ C_3 \end{pmatrix} = 0.5 \begin{pmatrix} 2 \\ 4 \end{pmatrix} + 0.5 \begin{pmatrix} 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix},$$

$$\begin{pmatrix} C_2 \\ C_4 \end{pmatrix} = 0.5 \begin{pmatrix} 4 \\ 6 \end{pmatrix} + 0.5 \begin{pmatrix} 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \end{pmatrix}.$$

This solution is infeasible because operations 1 and 3 are carried out concurrently on machine 1, and operations 2 and 4 are carried out concurrently on machine 2. An optimal, feasible solution is then obtained using the branching tree. Finally, note that, for this example, the Dantzig-Wolfe decomposition provides a lower bound of 5, while an approach ignoring the precedence constraints would provide a bound of 4.

3.4 $C_{\max}$ objective

The present formulation can be used to model problems with objectives of type $C_{\max}$ if we set $g_u(C_u) = C_u$. The master problem then becomes:

$$\min C_{\max} \tag{10.14}$$

$$\text{s.t.} \quad C_{\max} \geq \sum_{h \in \Omega_{i_u}} y_h C_u^h \quad u = 1, \ldots, N, \tag{10.15}$$

$$\text{constraints (10.8), (10.9) and (10.10).}$$

Another formulation, providing a better bound in the linear relaxation of the Dantzig Wolfe decomposition, has been developed for this specific objective. In this formulation, the objective is a function of the weighted mean completion time for each operation. The order of operations is not the same in the generated schedules. Some schedules for machine i may have operation u as their final operation, while others may have operation $v \neq u$ as their final operation. The mean completion time for operations u and v may be far smaller than the mean completion time for all operations on machine i.

It would be better to express makespan constraints as a function of the completion time for each machine. To accomplish this, a fictitious operation is added for each machine, i.e., $u = N+1, N+2, \dots, N+m$. We set $p_{N+i} = 0$, $r_{N+i} = \max_{u|i_u=i} r_u + p_u$, $d_{N+i} = \max_{u|i_u=i} d_u$, $i = 1, \dots, m$ and require that the operation $N+i$ be the last carried out on machine i.

Constraints (10.15) of the master problem are then replaced by

$$C_{\max} \geq \sum_{h \in \Omega_{i_u}} y_h C_u^h \quad u = N+1, \dots, N+m. \tag{10.16}$$

A dual variable γ_i is defined for each machine. The objective of subproblem S_i is to minimize the weighted sum of the operation completion times $(n \mid r_u, d_u, \text{prec} \mid \sum w_u' C_u)$ where

$$w_u' = \begin{cases} w_u, & u \mid i_u = i \quad \text{and} \quad u \leq N, \\ \gamma_i, & u = N+i. \end{cases}$$

The cost function is also irregular in the operation completion times and the same dynamic programming algorithm described in Section 3.2 can be applied.

With this formulation, the Dantzig-Wolfe solution for the problem illustrated in Figure 10.1 has a cost of 6.

$$\begin{pmatrix} C_1 \\ C_3 \\ C_5 \end{pmatrix} = 0.5 \begin{pmatrix} 2 \\ 4 \\ 4 \end{pmatrix} + 0.5 \begin{pmatrix} 4 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix},$$

$$\begin{pmatrix} C_2 \\ C_4 \\ C_6 \end{pmatrix} = 0.5 \begin{pmatrix} 4 \\ 6 \\ 6 \end{pmatrix} + 0.5 \begin{pmatrix} 6 \\ 4 \\ 6 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 6 \end{pmatrix}.$$

This solution, which is optimal for the Dantzig-Wolfe decomposition, has the same cost as the optimal solution of the job shop problem.

4. Implementation issues

An exact algorithm for the job shop problem has been implemented using three types of objectives:

$C_{\max}$: We set $g_u(C_u) = C_u$ and modify the formulation as described in Section 3.4.

$T_{\max}$: We set $g_u(C_u) = \max\{C_u - d'_u, 0\}$ where d'_u is the latest time at which operation u may terminate and not be late.

JIT **(Just-In-Time):** We set $g_u(C_u) = |C_u - d'_u|$ where d'_u is the desired termination time for operation u. A penalty is paid if operation u terminates before or after time d'_u. Different penalty costs can be used for earliness and tardiness without increasing the complexity of the solution approach.

The algorithm uses Dantzig-Wolfe decomposition and a branching strategy based on conflict resolution. It implements several exact and heuristic rules to accelerate the solution process.

4.1 Overview

An upper bound z_{sup} is provided as an input to the optimization algorithm. This bound may be obtained using heuristic methods. A three-step procedure is executed at each branching node.

The first step tightens time windows $[r_u, d_u]$ using the upper bound, the precedence constraints of the problem, those imposed by the branching procedure, and others that can be deduced from rules. Problem feasibility tests are carried out. If such tests conclude that the problem is infeasible, the node is abandoned.

The second step computes a solution to the relaxed job shop problem obtained from the Dantzig-Wolfe decomposition. Two cases are possible:

- The solution process is completed with the proof that no solution for the relaxation is possible. In this case, there is no solution to the job shop problem at the current node, and the node is abandoned.
- A solution is found for the relaxation; the process is halted, although optimality is not necessarily obtained. While a solution of the relaxation satisfies time and precedence constraints of the job shop problem, it may violate the machine constraints. If the process is stopped prior to optimality, the cost of the solution is not a lower bound for the current node. A lower bound, however, is of little interest because its cost is necessarily inferior to the z_{sup} and is not sufficient to eliminate the node at this stage. In fact, any solution to the relaxation has a cost less than z_{sup} as it satisfies the time constraints that were tightened using the value of $z_{\text{sup}} - 1$.

The third step applies a heuristic to calculate a solution that satisfies all constraints of the job shop problem, using the solution obtained in step 2. Once again, two cases are possible:

- A solution is found for the job shop problem. In this case, the upper bound is adjusted. Furthermore, branching nodes that have been solved to optimality and that have a lower bound greater than or equal to the new upper bound are eliminated. If the current node cannot be eliminated, the three-step procedure for processing a branching node is restarted with the new value of z_{sup}.

- No solution to the job shop problem is found. In this case, a pair of operations that conflict on one machine is selected. Two new problems are created by imposing an order on these two operations.

The branching tree is explored depth-first to find feasible solutions as quickly as possible. The advantage of proceeding in this way is that the operation time windows can be tightened, reducing the number of dynamic programming states. The following sections contain further details on the steps of the algorithm.

4.2 Preprocessing of the branching nodes

Before starting the solution process at a branching node, rules are applied to find precedence constraints and tighten time windows. An efficient implementation of these rules is described in Brucker, Jurisch and Sievers (1994); Carlier and Pinson (1990). In addition, the feasibility of each single-machine sequencing problem is verified, using calls to subproblems if necessary.

Precedence constraints

Let $\text{Succ}(u)$ denote the set of operations that must be carried out on machine i_u after operation u, and $\text{Prec}(u)$ the set of operations that must be carried out on machine i_u before operation u. Operation $v \in \text{Prec}(u)$ if and only if $u \in \text{Succ}(v)$.

Precedence relations may be deduced from simple rules. In particular, $v \in \text{Succ}(u)$ if $u \neq v$, $i_u = i_v$ and if one of the following conditions holds:

- The relation $u \to v$ is imposed by the branching.

- By the time constraints, operation v cannot be carried out before operation u: $r_v + p_v + p_u > d_u$.

- The relation $u \to v$ may be obtained by transitivity: $\exists w \mid u \in \text{Succ}(w)$ and $w \in \text{Succ}(v)$.

Other precedence constraints are deduced from more complex rules involving blocks of operations carried out on the same machine. Let X

be a subset of operations to be carried out on the same machine. Let $r_X = \min_{v \in X} r_v$, $d_X = \max_{v \in X} d_v$ and $p_X = \sum_{v \in X} p_v$.

- If there is a set X of operations that must be carried out on machine i_u, such that $u \notin X$ and

  $$\min\{r_X, r_u\} + p_u + p_X > d_X,$$

  then all operations in X must precede operation u, that is, $u \in \text{Succ}(v)$ for all $v \in X$.

- If there exists a set X of operations that must be carried out on machine i_u, such that $u \notin X$ and

  $$r_X + p_u + p_X > \max\{d_X, d_u\},$$

  then operation u must precede all operations in X, that is, $v \in \text{Succ}(u)$ for all $v \in X$.

The problem is not feasible if the precedence constraints induce a cycle, that is, if there exists u, v such that $u \in \text{Succ}(v)$ and $v \in \text{Succ}(u)$.

Time constraints

The time intervals $[r_u, d_u]$ are tightened using the upper bound z_{sup}, the precedence constraints of the job shop problem and the precedence constraints among operations carried out on the same machine.

The new earliest time r_u to begin operation u is the largest of the following quantities:

- r_u,
- $d'_u - p_u - z_{\text{sup}} + 1$, (JIT objective),
- $r_v + p_v$, $\forall v\colon (v, u) \in A$,
- $\min_{v \in X} r_v + \sum_{v \in X} p_v$, $\forall X : X \subseteq \text{Prec}(u)$.

The new latest time d_u to terminate operation u is the smallest of the following quantities:

- d_u,
- $z_{\text{sup}} - 1$, ($C_{\max}$ objective),
- $d'_u + z_{\text{sup}} - 1$, (JIT, $T_{\max}$ objectives),
- $d_v - p_v$, $\forall v\colon (u, v) \in A$,
- $\max_{v \in X} d_v - \sum_{v \in X} p_v$, $\forall X : X \subseteq \text{Succ}(u)$.

The problem is infeasible if an operation u can be found such that $r_u + p_u > d_u$.

4.3 Processing at a branching node

Dantzig-Wolfe decomposition is applied to the job shop problem associated with the current node.

Master problem

At each iteration of the Dantzig-Wolfe algorithm, the master problem calls the subproblems to receive columns with negative marginal cost. In the case at hand, subproblems are solved using a computationally intensive dynamic programming algorithm. It is not necessary to solve subproblems exactly to obtain columns with negative marginal cost, especially during the initial iterations. Subproblems are solved heuristically by limiting the number of dynamic programming states.

The limit on the number of states is controlled by a parameter passed to the subproblems from the master problem. The subproblem returns a boolean value indicating whether the state space has been explored completely or only partially. The master problem increases the limit if no further columns are generated or if the objective does not increase sufficiently. The optimal solution is found when all subproblems are solved exactly and generate no further columns.

As discussed in Section 4.1, the problem is not necessarily solved to optimality. Before raising the limit on the number of dynamic programming states, the feasibility of the relaxed master problem is verified. If the problem is feasible, column generation terminates and the heuristic search for a feasible solution to the job shop problem begins immediately (Section 4.4).

Subproblem

The subproblem is a sequencing problem on a single machine, $\big(n \mid r_u, d_u, \text{prec} \mid \sum g'_u(C_u)\big)$, and is solved by dynamic programming. States are eliminated using both exact and heuristic criteria.

Exact criteria ensure that eliminated states cannot lead to an optimal solution. Only states that satisfy the precedence constraints are constructed. Several of these states are eliminated using rules based on the time constraints. These rules are given in Gélinas and Soumis (1997). Other states are eliminated using bounds. The dual variable λ_i of the master problem provides an upper bound on the cost of a schedule on machine i that may improve the solution to the master problem. A lower bound is computed for the cost of schedules constructed from a dynamic programming state. The state is eliminated if the lower bound is not promising.

Finally, states are eliminated using a heuristic criterion if their number exceeds the limit passed from the master problem. This criterion is based on the quality and feasibility of a state. States with a good lower bound and those that appropriately place operations that must terminate early are retained. When states are eliminated using the heuristic criterion, the subproblem solution may not be optimal, and the master problem is so notified.

4.4 Branching node post-processing

The Dantzig-Wolfe solution satisfies time and precedence constraints at a cost less than the upper bound z_{sup}. This solution will be used as a starting point for another solution that also satisfies the machine constraints.

The disjunctive graph $G = (V, C \cup D)$ associated with the job shop problem will be used in this regard. The nodes of the graph correspond to operations, including two fictitious operations representing the beginning and end of operations, $V = \{0, 1, \ldots, N, *\}$. Execution times p_u are associated with nodes of the graph. Arcs of the graph fall into two types. The set C of conjunctive arcs includes precedence arcs; arcs $(0, u)$ where u is the first operation of a job; and arcs $(u, *)$ where u is the last operation of a job. The set D contains disjunctive arcs representing pairs of operations processed on the same machine. The arc pair $\{(u, v), (v, u)\}$ is said to be resolved if one of the two arcs is selected and the other rejected. In selecting (u, v), we require that operation u be performed before operation v, which corresponds to the addition of a precedence constraint (conjunctive arc) in the graph G.

Some disjunctive arc pairs are resolved at the current node using rules stated in Section 4.2. To obtain a feasible solution, the rest of the disjunctive arcs are resolved temporarily, according to the order of the operations in the relaxed solution.

$$u \rightarrow v \quad \text{if} \quad \sum_{h \in \Omega_{i_u}} y_h C_u^h \leq \sum_{h \in \Omega_{i_v}} y_h C_v^h.$$

A longest path problem with time windows is then solved from node 0 to all other nodes. If the arrival time at each node is such that the operation may be carried out within the specified time interval, then a feasible solution has been found with a cost below the upper bound. The upper bound is then updated.

Local search

The solution obtained to the longest path problem is then improved by inverting a disjunctive arc that has been resolved temporarily. Of interest are the arcs that belong to the longest path (if the solution is feasible) or to an infeasible path (if the solution is infeasible). The time gain obtained locally by inverting the disjunctive arcs is calculated. Let (u^*, v^*) be the arc yielding the maximal gain. If this gain is positive, the arc is inverted and the longest path problem is solved on the new graph. The process terminates when there is no further local improvement.

JIT objective

A further stage occurs when a feasible solution is found for the objective JIT. The solution to the longest path algorithm places the operations as early as possible within the time windows $[r_u, d_u]$. A better solution can be obtained by delaying operations so that they end as close as possible to the desired termination time. The maximum tardiness $T_{\max}$ of an operation with respect to its desired termination time is calculated in the feasible solution. The time windows are then temporarily tightened in such a way that this maximum tardiness is not exceeded: $[r_u, \min\{d_u, d'_u + T_{\max}\}]$; and a longest path problem is solved by pulling node $*$ back toward the other nodes in the graph. The upper bound for the job shop problem is adjusted using this new solution.

4.5 Branching strategies

If no solution is found using the procedure described in Section 4.4, branching occurs on a pair of operations that are carried out on the same machine and in conflict in the relaxed solution.

When there are many candidates when selecting a pair, we use the following rules:

- reduce the set of candidates to a set of pairs in conflict on the longest path found in Section 4.4 if this set is not empty,
- select the earliest scheduled pair in the set of candidates.

5. Experimentation

Numerical experiments were conducted using the Dantzig-Wolfe algorithm implemented in *C* on a HP9000/735 computer. The following sections describe the test problems and present the results obtained.

5.1 Test problems

Problems with ten machines and up to 500 operations were generated and solved using $C_{\max}$, $T_{\max}$ and JIT objectives. Problem sizes are described in Table 10.1.

Table 10.1. Sizes of the job shop problem instances.

Number of machines	Number of jobs	Number of operations per job	Total number of operations
10	250	2	500
10	100	3	300
10	30	5	150
10	10	10	100

A problem is constructed as follows. First, the number of machines, the number of jobs, and the number of operations per job are established. For each operation, a machine is selected at random in such a way that no job has two operations on the same machine. The length of an operation is generated uniformly in the interval $[1, 100]$. The times r_u are initialized to zero; the times d_u are initialized to a large value (∞). To select the desired completion times for jobs and their mutually compatible operations, a feasible schedule is constructed using decision rules. The operation that can begin earliest is placed first. In the case of a tie, the operation in that job having the most outstanding work is selected. The completion time T for this schedule is used to generate times d'_u. Let $u_1, u_2, \ldots, u_{n_j}$ be the operations in job j, in order. Times $d'_{u_{n_j}}$ are generated in the interval $[\sum_{k=1}^{n_j} p_{u_k}, T]$; in addition, we set

$$d'_{u_k} = d'_{u_{k+1}} - p_{u_{k+1}}, \quad k = n_j - 1, \ldots, 1.$$

The data d'_u are ignored for the objective $C_{\max}$. If $T_{\max}$ is to be minimized, we desire that the processing of job j terminate no later than time $d'_{u_{n_j}}$. If JIT is to be minimized, we desire that the processing begin at time $d'_{u_{n_j}} - \sum_{k=1}^{n_j} p_{u_k}$ and continue without stopping until time $d'_{u_{n_j}}$.

Ten problems are generated for each problem size, for a total of 40 job shop problems. Of particular interest are the job shop problems with many jobs and few operations per job. Such problems are easy to solve with the objective $C_{\max}$ because machines can operate without stopping. The schedule constructed using decision rules is optimal for all problems having 30 jobs or more. Therefore, we only present results for 10-job problems using the objective $C_{\max}$.

An upper bound is provided for the optimization algorithm. The schedule obtained with decision rules provides an upper bound for the objective $C_{\max}$. This bound is from 7% to 21% greater than the cost of the optimal solution for 10-job problems. For $T_{\max}$ and JIT, the upper bound is taken to be the optimal value plus 20%. (The optimal value is known because the algorithm has already been executed once using a large value as upper bound.) Future applications of the algorithm will require a heuristic procedure to produce a feasible solution for the objectives $T_{\max}$ and JIT. The cost of this solution will be an upper bound that should be no more than 20% from the optimal solution.

5.2 Numerical results

This section first presents results for specific steps of the algorithm. It then presents results for the job shop problems and analyzes the behavior of the algorithm with different initial upper bounds.

Solution of the Dantzig-Wolfe decomposition

The relaxed job shop problem is generally not solved to optimality for the first branching node. The value of the lower bound that could be obtained from Dantzig-Wolfe decomposition was obtained in a separate calculation. Table 10.2 gives the lower bound and the cost of the optimal solution for 10- and 30-job problems. Since the dynamic programming algorithm requires too much memory for the 100- and 200-job problems, optimal solutions are not obtained for them.

The lower bound is fairly distant from the optimal solution at the top of the branching tree, which provides justification for the solution approach presented here. We don't use much effort to get exact solutions to the Dantzig-Wolfe relaxation at the top of the tree. The job shop problem becomes more highly constrained at the lower level of the tree, as branching decisions are taken and feasible solutions are found. So, lower bounds become easier to get by exactly solving the Dantzig-Wolfe relaxation and are of better quality. This bound eliminates nodes associated with these more constrained problems.

On the other hand, the Dantzig-Wolfe solution is very useful for finding feasible solutions at each node of the branching tree, and is used to establish the order of operations in a schedule constructed heuristically.

Dynamic programming

Subproblems are solved using a dynamic programming algorithm. Two statistics are particularly germane as measures of the problem difficulty: the number of states and the number of labels. A state is asso-

Table 10.2. Lower bound from Dantzig-Wolfe decomposition.

	$C_{\max}$		$T_{\max}$		JIT	
Oper	DW	Opt	DW	Opt	DW	Opt
30× 5	-	-	164.0	174	135.4	235
	-	-	127.0	156	109.3	197
	-	-	99.6	205	117.7	220
	-	-	285.1	346	285.2	346
	-	-	67.0	67	133.2	196
	-	-	159.1	188	159.1	188
	-	-	72.7	121	87.1	152
	-	-	112.7	199	114.5	199
	-	-	91.6	218	127.6	278
	-	-	134.0	146	134.0	153
10×10	717.9	792	14.5	80	69.2	172
	792.0	867	53.1	69	106.6	172
	750.0	810	0.0	6	79.0	155
	742.8	845	2.4	99	75.7	167
	825.0	885	78.6	221	97.2	266
	625.6	728	14.8	90	60.9	162
	689.3	811	10.9	109	68.6	166
	743.2	840	26.1	108	70.5	190
	841.0	855	24.9	95	79.6	207
	747.0	766	36.6	119	102.7	208

ciated with a set of operations. A cost function is associated with each state. The cost function is piecewise linear and represented by a list of labels, one label per piece. At iteration k of the dynamic programming algorithm, all states associated with sets of k operations are considered. In a job shop problem with 10 machines, 100 jobs and 3 operations per job, there are a total of 300 operations and an average of 30 operations per machine. Consider a subproblem with 30 operations. At iteration 10 of the dynamic programming algorithm, there are a possible C_{10}^{30} states, that is more than 30 million states. While exact criteria can eliminate states, the number of them that remain to be considered in an exact procedure may be very large. The proposed algorithm uses heuristic criteria to eliminate states.

Table 10.3 gives the average and maximum number of states constructed in one iteration of the dynamic programming algorithm during the solution of the job shop problems. In all dynamic programming iter-

Table 10.3. Dynamic programming statistics.

	$C_{\max}$				$T_{\max}$				JIT			
	States		Labels		States		Labels		States		Labels	
Oper	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
250× 2	-	-	-	-	11.8	16	2.8	51	11.3	16	2.6	59
100× 3	-	-	-	-	11.0	233	2.7	42	9.3	120	2.7	47
30× 5	-	-	-	-	6.5	133	2.0	27	5.2	109	2.2	34
10×10	3.9	15	2.1	23	3.6	16	2.1	24	3.5	18	2.4	31

ations in all problems solved, no more than 233 states (i.e., a very small number) were constructed. This was sufficient, however, to find a feasible solution to the relaxed job shop problem or prove that no solutions exists.

To prove that there are no solutions to the relaxed problem, the subproblems must be solved exactly. The implementation of the algorithm increases the limit on the number of states until no further elimination occurs using heuristic criteria. While such a procedure may require that a large number of states be considered, this did not occur in the numerical experiments conducted for this study. Two reasons may explain this. First, the relaxed job shop problem is almost always feasible when Dantzig-Wolfe decomposition is applied. Rules applied in the preprocessing stage help to identify infeasible job shop problems. If the node is not eliminated using these rules, a solution is usually found to the decomposition. Second, when the relaxation has no solution, the subproblems are highly constrained and states are eliminated using exact criteria that are highly effective under the circumstances.

The table also gives the mean and maximum number of labels required to represent the cost function attached to a state. The number of labels increases with the number of operations and with the width of the time windows (Gélinas and Soumis, 1997). Larger numbers of labels imply greater calculations and manipulations in the dynamic programming algorithm. The average number of labels per state was low in problems solved for this study.

Elimination of nodes in branching tree

Figure 10.4 illustrates the branching tree obtained for one of the 250-job problems using the objective $T_{\max}$. Nodes are numbered in the order in which they were explored. The value of the initial upper bound is 432. The relaxed job shop problem is not solved to optimality for the initial nodes; a lower bound of 360 is obtained at the fifth node. Going down

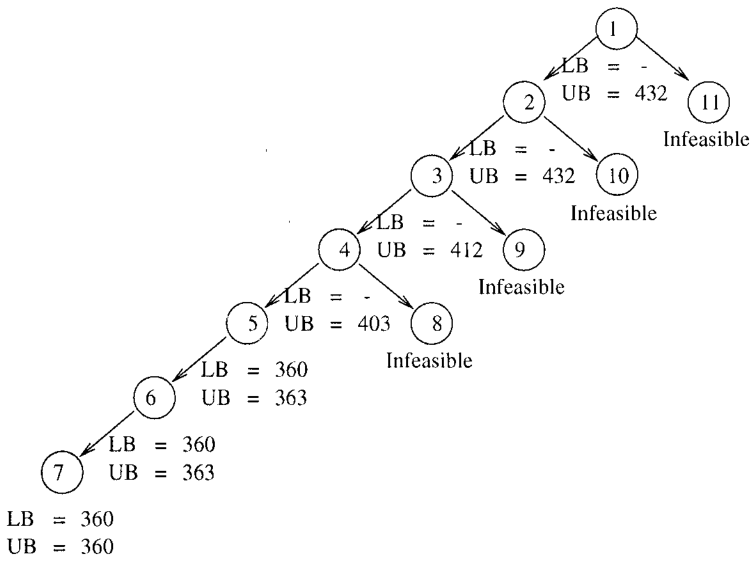

Figure 10.4. Branching tree.

the tree, feasible solutions are found with respective costs of 412, 403, 363 and finally 360. Nodes 7, 6 and 5 are then eliminated using the lower bound. While exploration continues from node 4, the new nodes are found to be infeasible when the rules from the pre-processing stage are applied. This tree contains 11 explored nodes, four nodes eliminated in the preprocessing stage and three nodes eliminated by the Dantzig-Wolfe solution.

Table 10.4 gives the percentage of nodes eliminated during solution of the various job shop problems using the three objectives.

Tot: percentage of the nodes eliminated.

Pre: percentage of the nodes eliminated in the pre-processing stage.

Table 10.4. Percentage of eliminated branching nodes.

	$C_{\max}$			$T_{\max}$			JIT		
Oper	Tot	Pre	DW	Tot	Pre	DW	Tot	Pre	DW
250× 2	-	-	-	82	21	61	57	43	14
100× 3	-	-	-	78	21	57	53	41	12
30× 5	-	-	-	63	35	28	54	43	11
10×10	51	47	4	51	48	3	51	48	3

DW: percentage of the nodes eliminated by the Dantzig-Wolfe solution.

Percentages of the eliminated nodes, both overall and by the Dantzig-Wolfe solution, were observed to be higher for problems having few operations per job. The decomposition proposed here is more appropriate for problems of this type. The master problem selects convex combinations of schedules for a single machine so as to satisfy precedence constraints. It ignores disjunctive constraints for problems of sequencing on a single machine. A solution to the relaxed problem may differ significantly from a feasible solution to the job shop problem if there are many precedence constraints.

Results for the job shop problems

The algorithm was used to solve all job shop problems using the three objectives. Only one problem, with objective JIT, was not solved to optimality. Results are presented in Tables 10.5, 10.6 and 10.7. These tables contain the following information:

MaxWork: maximum total processing time on a single machine $= \max_{i=1,\ldots,m} \left\{ \sum_{u|i_u=i} p_u \right\}$.

UB: upper bound.

Opt: cost of the optimal solution, or of the best solution if optimality is not attained.

UtilMach: percentage utilization of machines. Let T be the value of $C_{\max}$ in the optimal solution. The mean percentage utilization (Avg) and maximum percentage utilization (Max) of the machines are calculated for the interval $[0, T]$.

IterDW: total number of iterations in the Dantzig-Wolfe algorithm.

BB: total number of nodes explored in the branching tree.

Cpu: CPU time in seconds. It indicates the time required to solve the master problem (TM), the subproblems (TS), and the total time (TT).

All problems were solved in less than 20 minutes for the objectives $C_{\max}$ and $T_{\max}$; most problems were solved in less than one hour for the objective JIT. Two 100-job problems required much more CPU time for the JIT objective; one of these problems was not solved to optimality. In fact, the JIT objective is more difficult to optimize as it is sensitive

Table 10.5. Numerical results for the objective $C_{\max}$.

				UtilMach				Cpu (secs)		
Oper	MaxWork	UB	Opt	Avg	Max	IterDW	BB	TM	TS	TT
10×10	627	960	792	61.9	79.2	775	169	107	7	120
	652	982	867	60.4	75.2	1340	310	194	12	216
	750	960	810	62.1	92.6	1218	286	164	10	184
	673	937	845	62.8	79.6	2568	622	529	28	574
	679	1016	885	56.8	76.7	1195	248	235	16	258
	599	780	728	66.1	82.3	555	122	73	4	82
	610	896	811	61.5	75.2	2138	530	329	20	366
	706	975	840	63.8	84.0	6048	1555	927	50	1026
	677	945	855	63.9	79.2	3039	852	387	20	435
	682	873	776	64.0	87.9	311	66	38	3	42

to operations that end late or begin too early in the schedule. The cost of the solution using $T_{\max}$ is a lower bound for JIT.

Table 10.5 indicates the maximum operation time on a single machine (MaxWork). In fact, this number is a lower bound for $C_{\max}$. The precedence constraints cause waiting times on the machines, so that the cost of the optimal solution is well above this bound. The percentage utilization of the machines decreases when changing the objective from $C_{\max}$ to $T_{\max}$ and to JIT. The objective $C_{\max}$ produces better machine utilization because the optimal schedule compresses operations on the bottleneck machine as much as possible. When the objective JIT is used, it may be advantageous to create waiting times on machines, so that operations begin and end at the desired times.

The algorithm spends most of its time solving the master problem. Little time is spent solving subproblems because the number of dynamic programming states is restricted. The difference between the total time (TT) and time spent solving the master problem (TM) and the subproblems (TS) is accounted for by pre- and post-processing at the branching nodes.

Behavior of the algorithm using different starting values

Table 10.8 presents results obtained using various starting values of the upper bound for the objective JIT. The algorithm was executed once using an upper bound of 1000. Later executions used tighter upper bounds, at 20%, 10% and 5% of the optimal value. With an improved

Table 10.6. Numerical results for the objective $T_{\max}$.

			UtilMach				Cpu (secs)		
Oper	UB	Opt	Avg	Max	IterDW	BB	TM	TS	TT
250× 2	594	495	77.7	87.2	17	1	8	10	18
	269	224	72.8	98.4	10	1	4	3	8
	129	107	78.0	99.2	45	11	46	15	65
	225	187	77.3	97.2	62	9	62	21	86
	410	341	77.4	99.9	45	3	40	17	61
	432	360	76.5	95.8	91	11	84	48	138
	219	182	75.7	99.4	61	10	106	22	136
	250	208	78.0	95.5	202	37	723	90	835
	542	451	76.7	98.2	28	1	26	13	41
	480	400	80.1	92.0	71	1	53	32	87
100× 3	250	208	72.7	96.9	339	103	639	50	715
	236	196	69.9	93.1	470	163	839	61	938
	276	230	74.6	92.7	297	77	474	45	532
	300	250	73.3	91.3	91	18	170	18	190
	128	106	83.5	98.1	693	318	878	60	1031
	255	212	76.0	98.0	161	37	393	29	429
	249	207	75.2	92.6	105	18	153	17	174
	203	169	72.1	96.3	89	29	182	14	200
	254	211	66.8	94.2	35	2	28	6	35
	98	81	68.6	98.8	148	31	346	23	375
30× 5	209	174	71.5	90.1	417	177	113	12	136
	188	156	68.9	91.6	406	173	103	10	122
	246	205	65.2	94.4	123	42	61	5	68
	416	346	52.9	77.0	132	51	58	5	65
	81	67	64.9	95.6	377	126	66	7	83
	226	188	70.1	97.2	2175	877	315	41	469
	146	121	74.9	97.8	406	138	101	9	120
	239	199	74.0	85.4	221	67	69	7	81
	262	218	70.8	90.3	1001	280	244	31	293
	176	146	61.5	92.5	391	102	89	10	107
10×10	96	80	53.7	68.6	2206	442	481	22	518
	83	69	53.3	66.3	82	16	8	1	10
	8	6	54.3	81.0	116	15	10	1	12
	119	99	52.3	66.4	2204	375	513	26	550
	266	221	50.8	68.6	1925	364	514	31	556
	108	90	58.6	73.0	1426	279	247	12	269
	131	109	54.8	67.0	753	149	147	8	159
	130	108	51.8	68.2	3184	703	586	33	641
	114	95	55.5	68.8	1478	326	240	14	264
	143	119	55.7	76.5	303	50	51	3	56

Table 10.7. Numerical results for the objective JIT.

			UtilMach				Cpu (secs)		
Oper	UB	Opt	Avg	Max	IterDW	BB	TM	TS	TT
250× 2	594	495	76.3	85.6	286	30	749	176	959
	269	224	70.5	95.3	271	9	107	70	242
	329	274	72.6	92.3	88	13	99	29	149
	321	267	73.5	92.3	366	41	392	117	579
	410	341	70.3	90.8	105	5	137	43	193
	432	360	72.1	90.3	380	66	1052	206	1333
	255	212	72.2	94.8	824	183	448	137	960
	305	254	75.8	92.8	992	163	1234	306	1753
	542	451	69.1	88.4	171	10	367	88	475
	480	400	76.9	88.4	293	25	726	153	911
100× 3	320	266	66.6	88.7	8453	2156	16629	1397	18926
	294	245	66.8	89.0	723	176	1339	106	1506
	305	254	72.1	89.5	665	139	1203	100	1353
	326	271	70.8	88.1	423	79	992	83	1096
	280	233	75.6	88.8	485	121	680	64	774
	255	216*	70.2	90.5	21148	5000	23159	1958	30542
	249	207	73.9	91.1	466	111	793	64	897
	249	207	68.3	91.2	2341	645	2240	202	2758
	254	211	65.3	92.0	363	46	241	36	298
	336	280	62.6	90.2	357	7	328	65	404
30× 5	282	235	64.2	80.8	11310	3281	2903	322	3437
	237	197	66.2	88.0	976	300	200	25	247
	264	220	58.7	85.0	219	50	117	8	128
	416	346	53.8	78.3	349	103	102	14	121
	236	196	58.0	85.5	429	108	61	7	77
	226	188	62.0	86.0	1338	503	242	28	329
	183	152	71.7	93.6	755	172	163	14	200
	239	199	68.9	79.5	642	165	133	14	162
	334	278	63.5	80.9	937	280	237	27	283
	184	153	58.7	88.3	721	157	134	13	163
10×10	172	143	48.5	62.0	4240	766	1102	44	1173
	172	143	47.9	59.6	723	131	125	6	137
	155	129	46.6	69.5	59	3	38	1	40
	167	139	50.3	63.9	722	129	132	7	144
	266	221	41.8	56.5	1766	313	495	26	531
	162	135	53.1	66.0	2136	416	507	21	542
	166	138	48.2	59.0	601	91	126	5	135
	190	158	48.0	63.2	2187	338	549	29	590
	207	172	50.5	62.6	5054	1081	1157	59	1253
	208	173	48.1	66.0	671	106	88	6	98

* : Not optimal

Table 10.8. Behavior of the algorithm using different starting values for the objective JIT.

	$z_{\text{sup}} = 1000$		$z_{\text{sup}} = 1.20 z_{\text{opt}}$		$z_{\text{sup}} = 1.10 z_{\text{opt}}$		$z_{\text{sup}} = 1.05 z_{\text{opt}}$	
Oper	BB	Cpu	BB	Cpu	BB	Cpu	BB	Cpu
250× 2	43	1368	30	959	15	443	45	591
	15	388	9	242	59	429	19	100
	7	343	13	149	5	46	5	29
	67	1286	41	579	45	343	19	137
	3	499	5	193	1	68	1	28
	134	2337	66	1333	55	746	51	535
	176	4082	183	960	113	701	49	288
	163	7073	163	1753	231	1976	98	774
	4	294	10	475	10	279	6	143
	1	773	25	911	24	524	15	339
100× 3	401	4375	2156	18926	346	1642	273	1784
	276	5478	176	1506	97	748	329	4308
	216	3385	139	1353	170	990	70	486
	99	1752	79	1096	104	770	75	715
	107	1073	121	774	25	245	135	486
	921	7434	-	-	811	5235	165	1002
	189	2336	111	897	69	457	51	407
	332	3606	645	2758	220	1223	199	967
	96	1346	46	298	58	375	21	123
	25	1150	7	404	27	305	18	150
30× 5	418	482	3281	3437	226	258	217	240
	247	372	300	245	108	100	121	66
	158	324	50	128	111	221	36	57
	59	147	103	121	44	76	11	28
	91	173	108	77	57	55	148	103
	600	528	503	329	151	105	178	121
	572	679	172	200	241	262	43	55
	289	319	165	162	65	79	109	86
	305	430	280	283	233	258	144	142
	305	664	157	163	66	61	15	15

bound, time windows may be narrower. Furthermore, the branching tree is smaller and solution time is faster.

Some exceptions, however, can be observed. The algorithm presented here searches the branching tree using a depth first strategy and com-

Table 10.8. continued

	$z_{\text{sup}} = 1000$		$z_{\text{sup}} = 1.20 z_{\text{opt}}$		$z_{\text{sup}} = 1.10 z_{\text{opt}}$		$z_{\text{sup}} = 1.05 z_{\text{opt}}$	
Oper	BB	Cpu	BB	Cpu	BB	Cpu	BB	Cpu
10×10	706	1266	766	1173	648	1146	619	987
	91	135	131	137	85	96	33	36
	36	124	3	40	23	41	21	14
	145	233	129	144	140	184	119	162
	645	1112	313	531	383	691	125	243
	890	989	416	542	239	265	407	505
	158	242	91	135	114	198	64	75
	1332	1610	338	590	159	183	349	443
	1072	1243	1081	1253	904	968	1322	1326
	61	136	106	98	70	48	18	18

putes feasible solutions using a heuristic. If good solutions are found along the initial branches explored, the algorithm gives good results regardless of the starting value. The converse is true as well: even with a good starting value, it is possible to select a bad search direction and explore many nodes before finding good solutions. This seems to have been the case for the two 100-job problems that are very difficult to solve starting from a value situated 20% from the optimal value. These two problems were solved much more easily using a different starting value.

Such exceptions apart, better bounds generally yield better results. One possible fruitful approach could be to develop effective heuristic methods for finding good upper bounds before the optimization methods are called upon. Interestingly, however, the algorithm managed to solve all problems in reasonable times, even with very poor bounds.

6. Conclusion

This study has presented a formulation of the job shop problem that uses Dantzig-Wolfe decomposition. This approach breaks down the problem of coordination between machines and procedures to construct a schedule for each machine. In this way, an efficient algorithm may be applied to each problem component. Exchange of information between the master problem and the subproblems produces better lower bounds than approaches that treat each machine independently.

The algorithm presented here uses Dantzig-Wolfe decomposition and a branching strategy based on conflict solution procedure. We measure the effort provided at each stage of the resolution. While the optimal value of the Dantzig-Wolfe decomposition is a lower bound for the job shop problem, this bound is not efficient (especially for the initial branching nodes). Importantly, good bounds for the job shop problem are difficult to obtain, even with other formulations. Therefore, rather than putting considerable effort into finding lower bounds, this approach settles for a solution whose cost falls below the upper bound. Such a solution forms the basis of a heuristic schedule construction method for the job shop problem.

Subproblems in the decomposition are solved using dynamic programming. This technique is rarely used in sequencing problems because the number of states grows too quickly. The approach presented here applies this technique successfully by controlling the size of the state space to be explored. Even when only a small number of states are explored, the dynamic programming algorithm produces good schedules for the master problem.

The algorithm has been tested on 10-machine problems using three objectives: $C_{\max}$, $T_{\max}$ and an objective consistent with the Just-In-Time philosophy. Interesting problems for the objective $C_{\max}$ have as many jobs as machines. There exist methods that are better than ours at solving such problems. Nevertheless, the present algorithm has solved problems involving 10 jobs and 10 operations per job in less than 20 minutes each.

This algorithm is particularly efficient for problems involving many jobs and few operations per job. Such problems arise frequently in industry, as machines are increasingly versatile and jobs are processed with few changes of machine. Objectives other than $C_{\max}$ (such as minimization of delivery delays or storage periods) in fact appear to be more interesting in practice. Most existing methods consider the objective $C_{\max}$ and are poorly adapted to other objectives, especially when these objectives involve irregular functions of the operation completion times. The algorithm described here can handle such objectives. Finally, problems of up to 500 operations were solved using an objective consistent with a Just-In-Time approach.

References

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401.

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *Journal on Computing*, 3(2):149–157.

Balas, E., Lenstra, J.K., and Vazacopoulos, A. (1995). One machine scheduling with delayed precedence constraints. *Management Science*, 41(1):94–109.

Barker J.R. and McMahon, G.B. (1985). Scheduling the general job-shop. *Management Science*, 31(5):594–598.

Blazewicz, J., Domscke, W., and Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93:1–33.

Brinkkotter, W. and Bruckner, P. (2001). Solving open benchmark instances for the job-shop by parallel head-tail adjustment. *Journal of Scheduling*, 4(1):53–64.

Brucker, P. and Jurisch, B. (1993). A new lower bound for the job-shop scheduling problem. *European Journal of Operational Research*, 64:156–167.

Brucker, P., Jurisch, B., and Sievers, B (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

Carlier, J. and Pinson, E. (1990). A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26:269–287.

Chen, Z.-L. and Powell W.B. (1999a). A column generation based decomposition algorithm for parallel machine just-in-time scheduling problem. *European Journal of Operational Research*, 116:220–232.

Chen, Z.-L. and Powell W.B. (1999b). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, 11(1):79–94.

Chen, Z.-L. and Powell W.B. (2003). Exact algorithms for scheduling multiple families of jobs on parallel machines. *Naval Research Logistics*, 50(7):823–840.

Dantzig, G.B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8:101–111.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354.

Dorndorf, V. Pesch, E., and Phan-Huy, T. (2000). Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122(1–2):189–240.

Dorndorf, V. Pesch, E., and Phan-Huy, T. (2002). Constraint propagation and problem decomposition: A preprocessing procedure for the job-shop problem. *Annals of Operations Research*, 115(1–4):125–145.

Fisher, M.L. (1973). Optimal solution of scheduling problems using Lagrange multipliers—Part 1. *Operations Research*, 21:1114–1127.

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

French, S. (1982). *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* Wiley, New York.

Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability.* W.H. Freeman and Co., San Francisco.

Gélinas, S. and Soumis, F. (1997). A dynamic programming algorithm for single machine scheduling with ready times and deadline to minimize total weighted completion time. MIS Collection in the *Annals of Operation Research*, 69:135–156.

Giffler, B. and Thompson, G.L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503.

Hoitomt, D.J., Luh, P.B., and Pattipati, K.R. (1993). A practical approach to job-shop scheduling problems. *IEEE Transactions on Robotics and Automation*, 9(1):1–13.

Join, A.S. and Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390–434.

Lageweg, B.J., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1977). Job-shop scheduling by implicit enumeration. *Management Science*, 24(4): 441–450.

Martin, P. and Shmoys, D.B. (1996). A new approach to computing optimal schedule for the job-shop scheduling problem. *Proceedings of the* 5th *International IPCO conference*, pp. 389–403.

McMahon, G. and Florian, M. (1975). On scheduling with ready times and due dates to minimize maximum lateness. *Operations Research*, 23(3):475–482.

Muth, J.F. and Thompson, G.L. (1963). *Industrial Scheduling*. Englewood Cliffs, New Jersey, Prentice-Hall.

Rinnooy Kan, A.H.G. (1976). *Machine Scheduling Problems: Classification, Complexity and Computations*. The Hague, The Netherlands, Martinus Nijhoff, 39.

Roy, B. and Sussman, B. (1964). Les problèmes d'ordonnancement avec contraintes disjonctives. *NoteDS* No.9 bis, SEMA, Paris.

van den Akker, J.M., Hoogeveen, J.A., and van de Velde, S.L. (1995). Parallel machine scheduling by column generation. *Technical Report*, Center for Operations Research and Econometrics, Université Catholique de Louvain, Belgium.